Technische Universität München
TUM School of Engineering and Design

A Deep Reinforcement Learning Model for Combinatorial Optimization and Fleet Dispatching in Mobility On-Demand Services

Sascha Sirus Jakob Hamzehi

Vollständiger Abdruck der von der TUM School of Engineering and Design der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz:	Prof. Dr. Constantinos Antoniou
Prüfer*innen der Dissertation:	1. Prof. Dr.-Ing. Klaus Bogenberger
	2. Prof. Dr. Maximilian Schiffer

Die Dissertation wurde am 17.06.2022 bei der Technischen Universität München eingereicht und durch die *TUM School of Engineering and Design* am 16.01.2023 angenommen.

Foreword and Acknowledgements

The underlying thesis summarizes my research during the last three and a half years at the Institute of Traffic Engineering of the Technical University Munich. First and foremost, I would like to thank and express my gratitude to all my colleagues for the exciting and wonderful time we shared at the institute and BMW Group. Besides this, I would like to mention some colleagues and friends who contributed to this thesis. Clearly, this thesis would not have been possible without my first Ph.D. advisor Prof. Dr. Klaus Bogenberger, who offered me this Ph.D. position. His excellent knowledge of Operations Research and mobility systems, his personal encouragement, and his motivational, professional, and technical advice has always led to progress and success within my thesis. Equally, I would like to express my gratitude to Prof. Dr. Maximilian Schiffer who additionally provided his excellent knowledge within the field of Operations Research and Supply Chain Management for this thesis. Equally, I am very grateful to Prof. Dr. Bernd Kaltenhäuser, who always provided excellent constructive criticism for all publications. Besides, I would also like to thank my advisors at the BMW Group, Philipp Franeck for his great profession, orchestrational efforts, machine learning and programming advice, discussions, and personal trust throughout these years. Without him, this thesis would not have been possible. I want to express my gratitude to Dr. Urlich Fastenrath, who showed great trust in my abilities and provided very valuable discussions, especially in the beginning of my research. Many thanks go to my colleagues Tian Jilei, Dr. Alvin Chin, Tanja Niels, Florian Dandl, Roman Engelhardt, Fynn Terhar, Arslan Ali Syed, Aledia Bilali, Marvin Erdmann, and Cao Yang for always helping me with discussions, traveling, and all other things that kept me running. I would like to thank the support of David Gackstetter with his Master's thesis and a big thank you to all proofreaders: Patrick Malcolm, Julius Tutz, Mauricio Platteau, and Julia Blank. I am especially delighted that Tian Jilei, Alvin Chin, and Cao Yang provided with me funds for improving my research at their machine learning research department abroad in Chicago, USA. I also have to mention the great atmosphere and funny moments in the office that I have shared with Katharina Gompf, Dr. Felix Rempe, and Dr. Melike Güler in the course of time. A very special thank-you goes to my family and friends. I am in depth with my parents Ingeborg and Hossein, who always supported me, who showed great sympathy, and understood when I had hard times. Particularly, I am deeply grateful to my girlfriend Monika who always shared my most difficult and also my best moments. Her unprecedented empathy and trust helped me to stay positive, healthy, and energetic. The same thank you goes to my friends Julius Tutz, Daniel Urbinati, and Mauricio Platteau, who always provided hope, cheerfulness, and an open door. Finally, there is my beloved brother Sebastian, who always enlightens my mood with his positivity, intellectual challenges, and heartful jokes. All of you are awesome and have my utmost respect and gratitude.

Thank You!

Sascha Sirus Jakob Hamzehi

Contents

Abstract

This thesis is about the development and benchmarking of a deep reinforcement learning model using a pointer generation neural network with an Actor-Critic Monte-Carlo Policy Gradient training algorithm. Within this thesis, novel architectural solutions are developed for the solution of unbalanced, asymmetric, and constraint combinatorial optimization problems for Traveling Salesman Problems (TSPs) and Minimum Weighted Bipartite Matching (MWBM) graphs by learning a competitive near-optimal and efficient heuristic automatically. Classical approaches for solving Minimum Weighted Bipartite Matching (MWBM) problems compute mathematical solutions from a naive greedy heuristic in order to obtain efficient solutions, or classical approaches focus on exact Mixed Integer Programming (MIP) solvers for optimal but less time competitive solutions. In practice, approximate algorithms are used to handle the rapidly increasing complexity of combinatorial optimization problems. The concept of existing neural network architectures for solving TSP graphs is extended to learn and solve different variants of MWBM graphs. Graphs as such, are more complex with respect to their structural variability and context-feature-sensitivity and thus require more contextual features for modeling. Herefore, the pursued DRL approach develops a Sequence-to-Sequence (Seq2Seq) - Encoder-to-Decoder neural network architecture based on the knowledge of machine learning, Deep Reinforcement Learning (DRL), and machine translation. The approach analyzes, evaluates, and transfers the existing Mixed Integer Programming (MIP) formulations to a Markov Decision Process (MDP) formulation in order to represent the problem structure in a machine-learnable manner. The developed model called PGN can be applied to a variety of combinatorial optimization problems such as optimal transport problems (Operations Research), scheduling, tracking, collaborative fleet routing, knapsack, and similar tasks (Supply-Chain Management) where the least cost assignments between two or more sets of nodes must be found efficiently. After the PGN network has been trained, the network state is saved and subsequently can be used to solve even different variants of MWBM efficiently. The thesis finally includes qualitative and quantitative comparisons with respect to solution quality, solution times, and solution reliability. For this purpose, a set of additional suited algorithm baselines are implemented to evaluate the PGN performance to well-known combinatorial optimization algorithms of the MIP and graph theory research. The results show that for symmetric, asymmetric, balanced, and unbalanced MWBM graphs the developed PGN can achieve a solution quality of 3.2% MAPE deviation from the global optimal solution provided by CPLEX. However, the PGN approach provides much faster solution times which results in approximately 66 times faster solution times compared to CPLEX. Hence, the results show that the approach provides the most efficient method currently known compared with all other tested state-of-the-art methods.

Zusammenfassung

Die vorliegende Dissertation behandelt die Entwicklung sowie die anschließende Evaluation einer Neuronalen Netz Architektur und eines Lern-Algorithmus des bestärkenden Lernens. Die Architektur wird als Ansatz zur effizienten kombinatorischen Lösung von Fahrzeug-Kunden Zuweisungen im Anwendungsbereich der intelligenten Transport Steuerung für hoch-performante Server-basierte Mobilitätssysteme benötigt. Innerhalb dieses Ansatzes werden bereits existierende mathematische MIP Repräsentierungen als Markov-Entscheidungs-Problem (MDP) reformuliert, um eine Maschinen-erlernbare Aufgabenstellung zu erhalten. Der entwickelte Lernansatz genannt Zeiger-Generations-Netzwerk (vgl. Eng. Pointer Generation Network (PGN)) kann bei der effizienten Lösung einer Vielzahl von kombinatorischen Optimierungsproblemen, wie optimale Transport Probleme, Termin und Zeitplanung, Kollaborative Flotten-Navigation, Aufgabenverteilung, kombinatorische Rucksack-Pack-Probleme (Operations Research), sowie ähnliche Aufgabenstellungen in der Lieferkettenoptimierung effizient angewendet werden. Hierbei muss die kombinatorische Zuweisung mit den geringsten Kosten zwischen allen Knoten schnellst-möglich gefunden werden. Nachdem das Zeiger-Generations-Netzwerk (vgl. Eng. Pointer Generation Network (PGN)) trainiert wurde, kann dieses im gespeicherten Zustand verschiedenste Varianten von Zuweisungsproblemen effizient lösen. Die Dissertation zeigt und stützt dies mit beinhalteten qualitativen und quantitativen vergleichen hinsichtlich Lösungsgüte, Lösungszeit und Lösungs-Zuverlässigkeit. Zu diesem Zweck, wird eine Menge von bekannten kombinatorischen Optimierungsalgorithmen aus der Literatur implementiert, um die beinhalteten Vergleiche zwischen der Zeiger-Generations-Netzwerk (vgl. Eng. Pointer Generation Network (PGN)) Architektur und den anderen Algorithmen aus den wissenschaftlichen Bereichen der Mixed-Integer Programmierung (MIP) und Graph Theorie stochastisch aufzustellen, zu interpretieren und zu bewerten. Hierbei werden die Anwendungsmöglichkeiten und die in der Literatur bestehenden Architekturen, um die Möglichkeit der Lösung für unbalancierte, asymmetrische und eingeschränkte kombinatorische Optimierungsprobleme erweitert. Um die Lösungsgüte und die statistische Lösungseffektivität zu belegen, wird hierfür ein spezieller Testaufbau mit anschließender stochastischer Auswertung entwickelt. Der Test-Aufbau umfasst dabei statische bestehende kombinatorische Optimierungsprobleme eines realen operativen serverbasierten Ride-Hailing Management und Flottenmanagement Systems. Um bestehende kombinatorische Probleme wie dem Traveling Salesman Problem (TSP) oder gewichtete Zuweisungsprobleme (MWBM) für “Ride Hailing” Applikationen noch effizienter zu lösen, wird eine Datengenerationsmethodik entwickelt, um große Beispielmengen von kombinatorischen Optimierungsproblemen zu generieren. Nach der Datengeneration soll das Neuronale Netz eine möglichst optimale kombinatorische Heuristik (Strategie) selbstständig erlernen, um die gestellten kombinatorischen Optimierungsprobleme möglichst akkurat zu lösen. Anschließend wird die Qualität und Quantität der zurück-gegebenen Lösungen des Neuronalen Netzes nach der Trainings- und Validierungsphase mit Vergleichslösungen anderer Algorithmen und mathematischen MIP Lösungsverfahren verglichen. Die Tests des entwickelten PGN Ansatzes für symmetrische, asymmetrische, balancierte, unbalancierte und limitierte MWBM Graph-Probleme zeigen, dass der Ansatz Lösungsgüten von 3.2% MAPE Abweichung vom globalen Optimum, verglichen mit der CPLEX Lösung, erreichen kann. Allerdings, bietet der entwickelte PGN Ansatz in etwa 66-mal schnellere Lösungszeiten und ist dahingehend viel effizienter als die bis dahingehend bekannten Lösungsmethoden. Die erarbeiteten stochastischen Ergebnisse dienen final als Grundlage für die abgeleiteten Schlussfolgerungen und der Ableitung von weiteren wissenschaftlichen Arbeiten in diesem Zusammenhang und angrenzendem Kontext.

1. Introduction

1.1 On-Demand Mobility (ODM)

With the recent boom of digital connectivity in Intelligent Transportation Systems (ITS) (see Figure 1.1) and the development of On-Demand Mobility (ODM) services for Ride Hailing (RH) and Ride Sharing (RS), urban mobility is transformed to provide comfortable transportation to anybody, anywhere, and anytime.

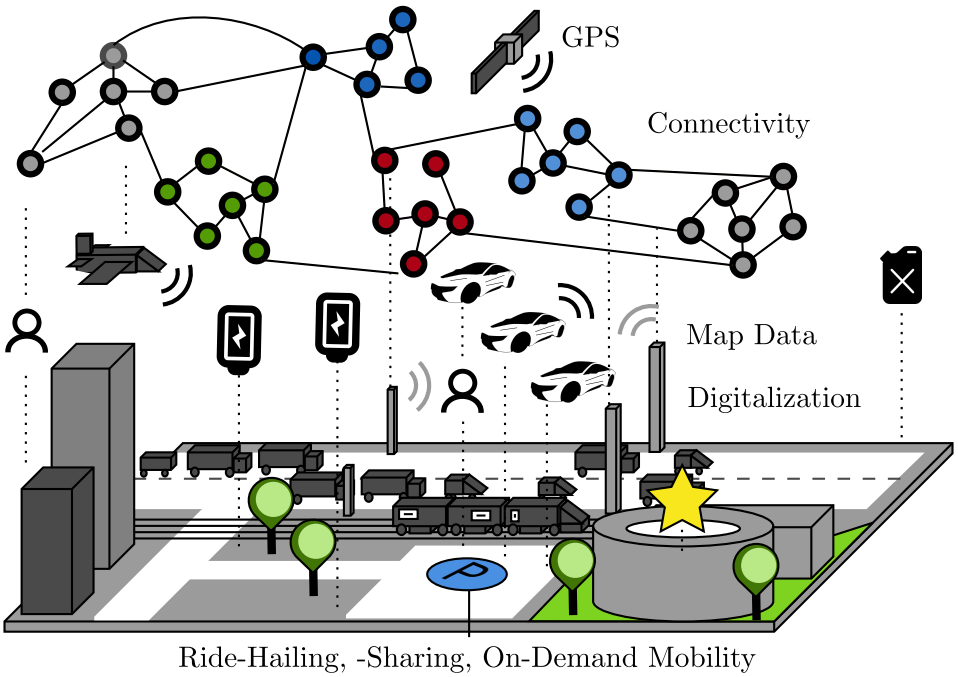


Figure 1.1: A smart and connected city.

ODM services like Didi Chuxing, Uber, Lyft, and Share Now present tremendous potential and advantages for human impacts with respect to greenhouse gas emissions, energy consumption, particulate matter, pollution, and congestion. On the other hand, user-centric mobility has become a large-scale contributor to negative impacts on environmental sustainability [Nations, p. 3][1], which may further reinforce the development of future ODM services. For instance,

[1] https://www.un.org/en/climatechange/assets/pdf/cas_report_11_dec.pdf

in the United States, user-centric mobility is the reason for tremendous congestion costs, and consequently, user waiting times. The annual cost of congestion is approximately $ 121 billion per year, which is 1% of the Gross Domestic Product (GDP). Additionally, this includes 5.5 billion hours of waiting time and an extra 2.9 billion gallons of fuel caused by traffic jams [Schrank et al., 2012, p. 1]. In Europe, the transportation sector causes 20% of the accumulated European greenhouse emissions, and thus, is one of the major hazardous environmental emission effects [Schiffer and Walther, 2018b].

On-Demand Mobility (ODM) Ride Sharing (RS) systems could help to decrease such negative impacts due to increased vehicular capacity utilization. RH can also benefit especially from the increase of available information by using it to avoid costly situations. The advantage of platform-based ODM services is that large databases enable the aggregation of large-scale information for planning trips, optimizing routes, and managing humanly-driven or autonomous fleets. Each trip and decision-making process is completed by efficient algorithms, which can be used to save resources, increase profit, increase fleet utility, and for other use cases. Future efficient large-scale fleet routing and dispatching algorithms will have more and more significant positive and maybe also negative impacts on environmental sustainability. While many big companies focus on saving operational costs and increasing the fleet utility to increase profit, officials and government are interested in how ODM services could improve human and environmental impacts. All programmed algorithms on each ODM platform process and make context-sensitive decisions between user-vehicle requests in a matter of milliseconds. Thus, the used algorithms have a significant influence with respect to the operational transport efficiency (e.g. the trips completed, required CO_2 emissions), functionality (e.g. energy and resource consumption of platform and fleet), and reliability (e.g. algorithm recommendations may influence if traffic jams occur), etc. Due to the availability of additional mobile phone data, ODM and ITS present an ideal research domain for Artificial Intelligence (AI), Machine Learning (ML), and data mining applications.

1.2 Applications of ODM

ODM operational environments (i.e. urban districts in different cities) are usually characterized by stochastic, continuous, discrete, static, dynamic, and combinatorial processes. Hence, ODM usually deals with a challenging, complex, and interactive system. On a time scale, the strategic requirements within ODM for dynamic and context-based (i.e. situation-depending) decision-making processes can change by the hour, the minute, or even the second. During operation, the main goal of an ODM platform is to support, recommend, plan, and manage client journeys and short trips in the most comfortable way. Applications of ODM are related to fleet and supply chain logistics. For example, there are

- RS / RH companies that usually assign the best routes for drivers to pick up and drop-off passengers,
- package delivery companies that aim to assign routes for drivers to complete their deliveries,
- mobile planning applications that aim to find an optimal context-based sightseeing tour,
- and maintenance companies or applications, where the best routes for technicians and

logistic vehicles are desired in order to offer and absolve maintenance services, deliver items and goods, or to complete other tasks.

In order to provide efficient planning and control mechanisms, ODM, RS/RH platforms typically depend on a set of sub-mechanisms to ensure quality services to their clients. Typically, the most critical components include

- supply and demand forecasts, where the trends of client demand, vehicle stock supplies, and traffic data are predicted [Dandl et al., 2019],
- booking [Dandl and Bogenberger, 2018a], reservation [Bilali et al., 2019b; Bilali et al., 2019a], and payment components for RH/RS client requests [Hardt and Bogenberger, 2016],
- RS and pooling mechanisms [Engelhardt et al., 2019; Bilali et al., 2019c; Bilali et al., 2020],
- planned and managed vehicle-client assignments [Syed et al., 2019b; Erdmann et al., 2019],
- fleet relocation mechanisms to re-balance vehicle distributions within city hotspots [Weikl and Bogenberger, 2013],
- collaborative routing components used for planning and adapting vehicle routes within the city [Dandl and Bogenberger, 2018b], thus requiring routing algorithms for calculating the shortest routes or paths between departure and destination locations,
- or the calculation of roundtrips such as TSPs, Vehicle Routing Problems (VRPs), and other Pickup-and-Delivery Problems (PDPs).

The next section 1.3, will provide more technical details on a given sample ODM platform for RS / RH and vehicle-client trip assignment purposes. The platform will provide the basis framework for the upcoming chapters by providing the technical background for developing a new DRL-based approach for vehicle-client trip planning and assignments.

1.3 ODM Platform

Figure 1.2 shows a more detailed example of an RS/RH platform-based system architecture. For fleet planning and control, the system requires a set of functional parameters, such as a description of a road-graph network, information about the availability of each vehicle within the operational fleet, multiple communication channels between the clients, a central planning and control component and the operational fleet where the tasks are updated via automated messages. Further examples of automated messages are explained in the following.

An ODM and Fleet Logistic System: A general fleet planning and control system, for example, shown in Figure 1.2, involves continuously updated and exchanged real-time data flows in the following:

1. *Information about the fleet availability and activity,* such as vehicle arrivals, departures, trip cancellations, and operational interruptions, requiring repeatedly updated location data of the vehicles,

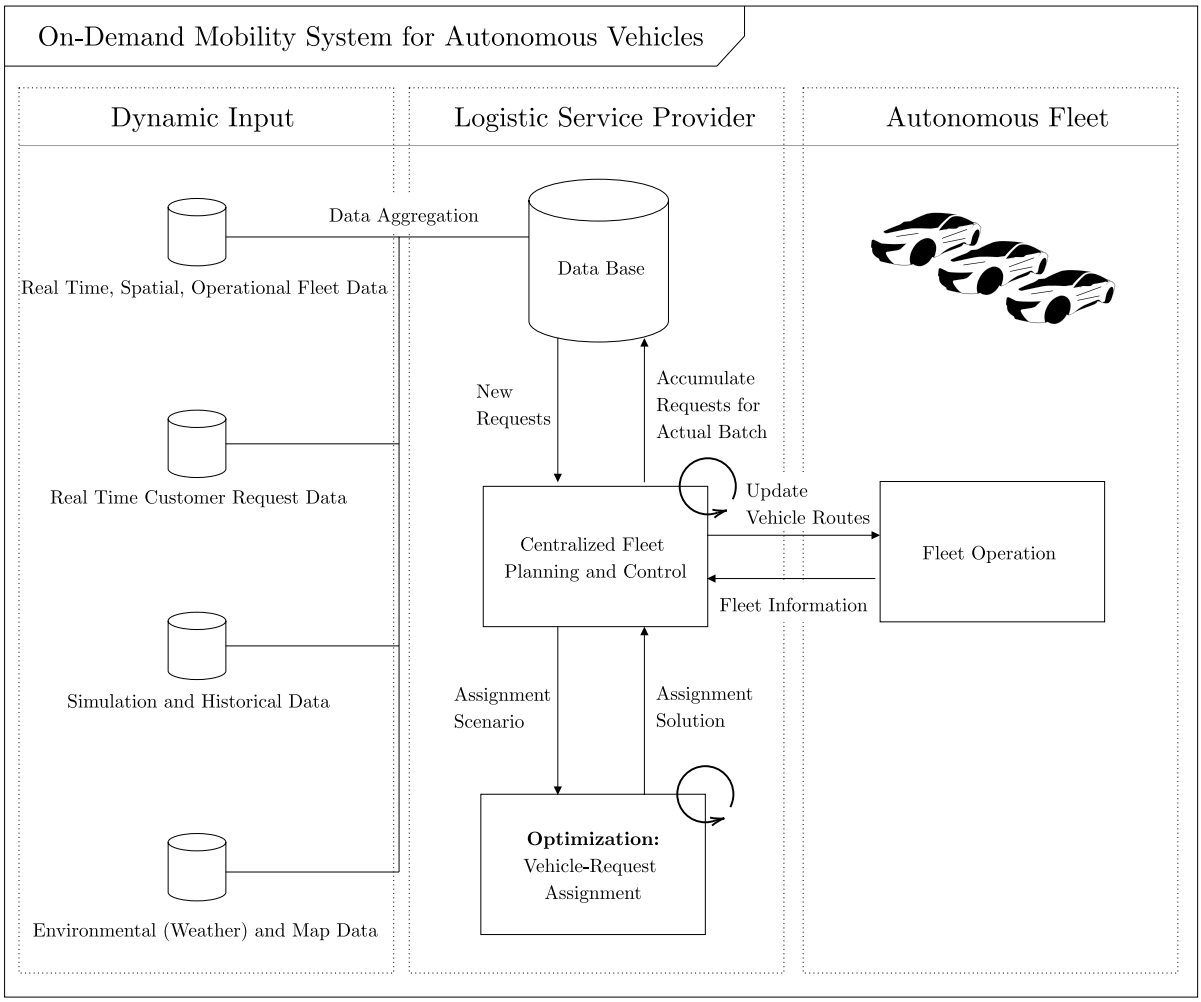


Figure 1.2: A centralized fleet planning and control architecture of an On-Demand Mobility (ODM) system ensuring demand responsiveness for an operational autonomous or non-autonomous fleet.

2. *Information about the client or system users,* where each incoming request may require information about additional passengers, geographical locations, time windows, a pre-set pickup/drop-off location, where the conglomerate of information is processed by an automated client request broker,
3. *Information about current booking confirmation,* where only the sub-set of clients that have shown commitment to finishing the booking confirmation are considered for the trip planning and scheduling process,
4. *Information about booking cancellations,* where the currently confirmed booking is made obsolete and excluded from the trip planning process,
5. *Information about dispatched vehicles,* where a specified vehicle driver receives messages in order to get directed to reach a client within a specified time and specified pickup location,
6. *Information about the verification of trip-requests,* where the sent messages indicate if a trip has been a success, failure, or another type of attribute for recording and accumulating

historical data, and

7. *Informative and contextual messages* about the system state and offers for new bookings, trip restarts, or re-bookings of stranded passengers [Horn, 2002, p. 38].

Within the fleet planning and control component, visualized in Figure 1.3, the accumulated valid bookings are temporally gathered in a preprocessing process called *batching* (visualized in Figure 1.4). Furthermore, batching is necessary to ensure a time-concluded combinatorial optimization setting for the fleet planning and control process, which proceeds to start the scheduling process. After each batch has been optimized, the vehicles are subsequently dispatched via automated messages from the fleet dispatching component [Pavone et al., 2020; Potts and Kovalyov, 2000, pp. 1–32, 228–249].

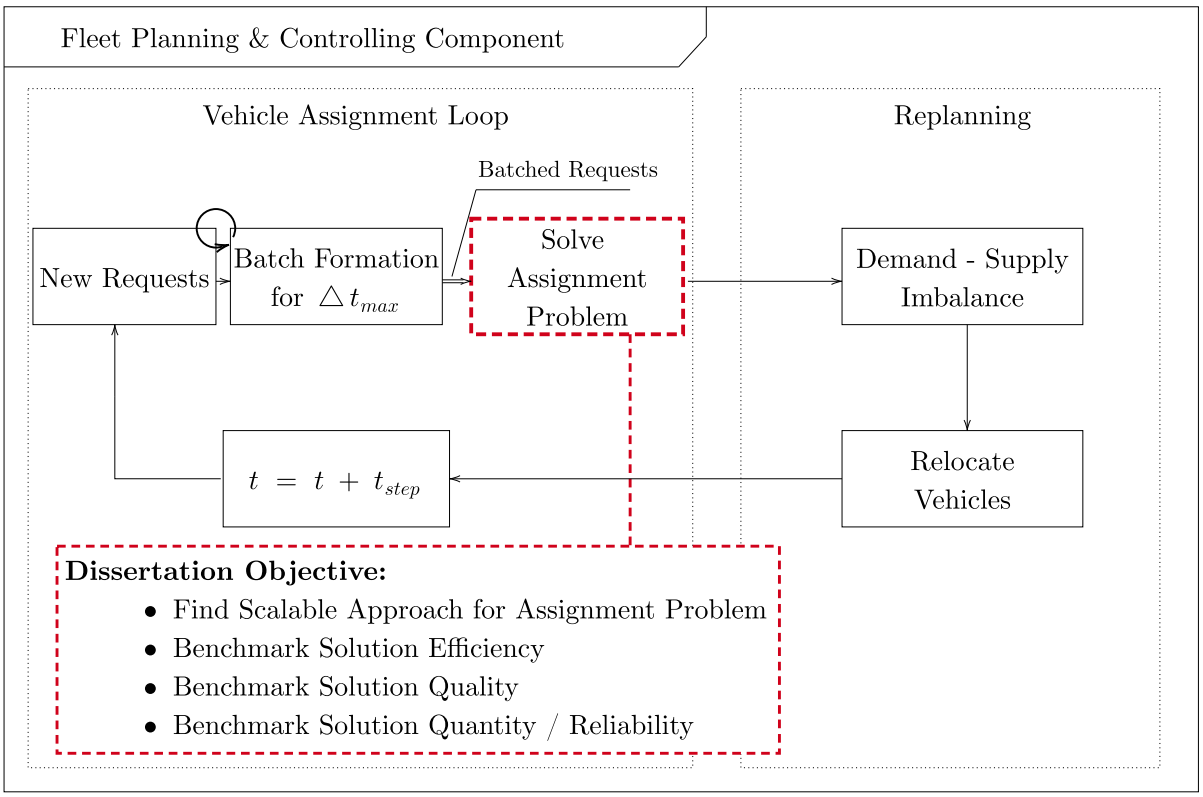


Figure 1.3: The core fleet planning and control component for bipartite assignment or matching of client requests to vehicles.

During the scheduling process, the fleet planning component must decide which vehicle (supply) should pickup which client (demand). The current group of client requests is therefore referred to as *demand*, where the goal of the scheduling process is to satisfy as much demand as possible for RS. For RH, which we will consider in later sections, a client requests a ride for himself only. For RH, each currently available vehicle has to be assigned to an unknown number of client requests for each batch. Clearly, the problem setup results in a Combinatorial Optimization Problem (COP), where the planning component has to select the right vehicle for each customer. Technically, the problem setting can be modeled as a discrete mathematical optimization problem

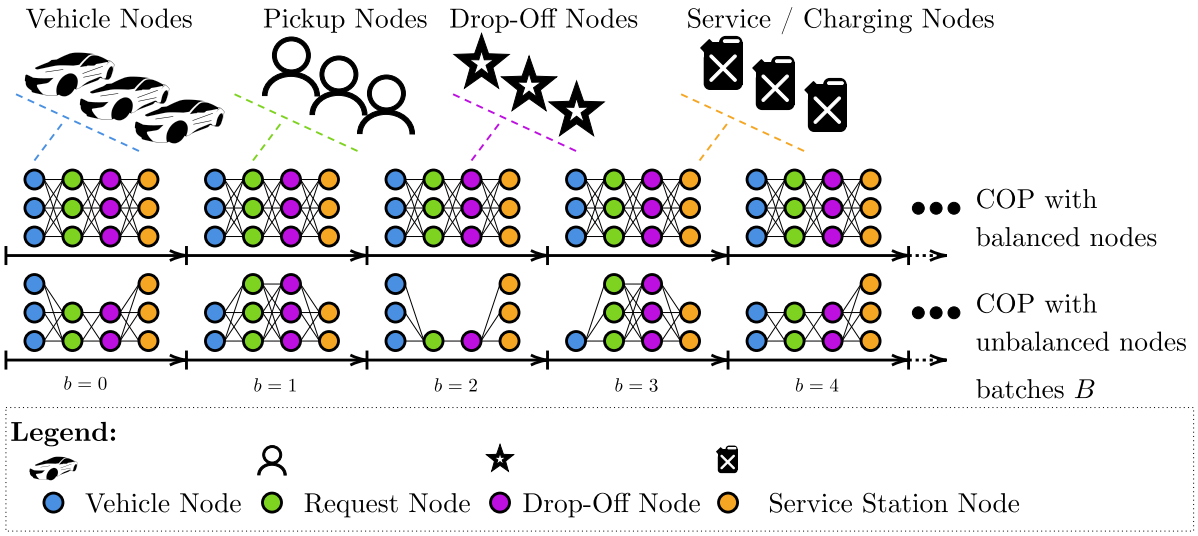


Figure 1.4: The Temporal Batching Process.

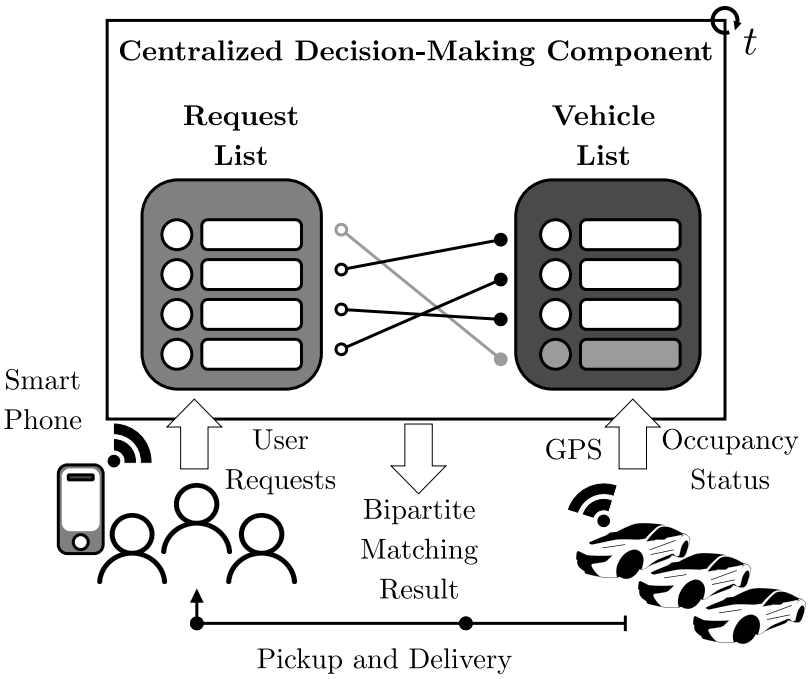


Figure 1.5: The vehicle-client assignment, matching, fleet dispatching as Combinatorial Optimization Problem (COP) decision-making process.

with a cost matrix and multiple constraints, as MDP representation, which is explained later, and using a graph theory-based (network) representation. Using graph theory, each entity (e.g. vehicle, client, and service stations) can be modeled as nodes and intermediary assignments. The cost for assigning a vehicle to a client can be modeled via adhering costs and profits as links between vehicle, client, and service station nodes. Such graph-model structure further

results in a bipartite graph (see Figure 1.6) where such problem settings are well-known under the domain of Minimum Weighted Bipartite Perfect Matching (MWBPM). The Operations Research (OR) domain also deals with similar problems under the name of Pickup-and-Delivery Problems (PDPs). Within OR, the formulation of PDPs is usually proposed using a discrete Mixed Integer Programming (MIP) formulation. Generally, the whole optimization domain also can be seen as a specific instance of the general maximum flow problem [Wayne and Tardos, 1999]. However, differences in objective functions and constraints often occur if the desired result is a minimum-cost path instead of a maximum-cost (capacity) flow. After the vehicle-client assignment process (see Figure 1.3) the vehicles have served a client trip. Subsequently, the vehicles often end up distributed within the operating environment (i.e. the city). Hence, there is an appended optimization process, called *vehicle relocation*, which has the goal to relocate the vehicles to higher-demand areas within the city. This problem setting another time results in a maximum flow COP, which can be solved equally to the vehicle-client assignment problem, (i.e. as Mixed Integer Programming (MIP) or using graph algorithms, etc.). Note, that in Chapter 2, "Fleet Dispatching and Combinatorial Optimization", more details about the mathematical perspective on the optimization problem are provided.

1.4 Focus of the Thesis

In the following chapters, this thesis will focus on the vehicle-client assignment problem as MWBPM (see Figure 1.5) with DRL in an offline learning process. The DRL solver module is employed as a central fleet planning component (see Figure 1.6) and is used to solve the vehicle-client assignment problem. Generally, multiple algorithms exist that could handle the same purpose. The goal of this thesis is to provide the development of a DRL-based learning algorithm that is suited to efficiently solving shortest path problems, such as the aforementioned the vehicle-client assignment problem. Also, the thesis provides a custom-tailored and sophisticated benchmarking framework for comparing the solution quality and quantity (reliability) to other existing scheduling algorithms, i.e. for assigning vehicles to clients.

In order to accomplish this, an aggregation of multiple data sources to a general database is required initially, which ideally should contain all necessary information as follows. It is assumed that all necessary information is known at the decision time within the core fleet planning component. On one hand, this assumption enables a transparent algorithm benchmarking process, however, on the other hand, it may not always be fulfilled in reality (see next Section 1.5 for further details). In other words, it is assumed that enough contextual information for decision-making and enough strategic information can be aggregated within a certain (real-) time. Such information further entails data structures and data attributes about vehicles, client requests, service or charging stations, and environmental data. In more detail, such data structures and attributes should include

- large client request and fleet databases, used for tracking and understanding the environmental processes,
- request pickup and drop-off Global Positioning System (GPS) locations,

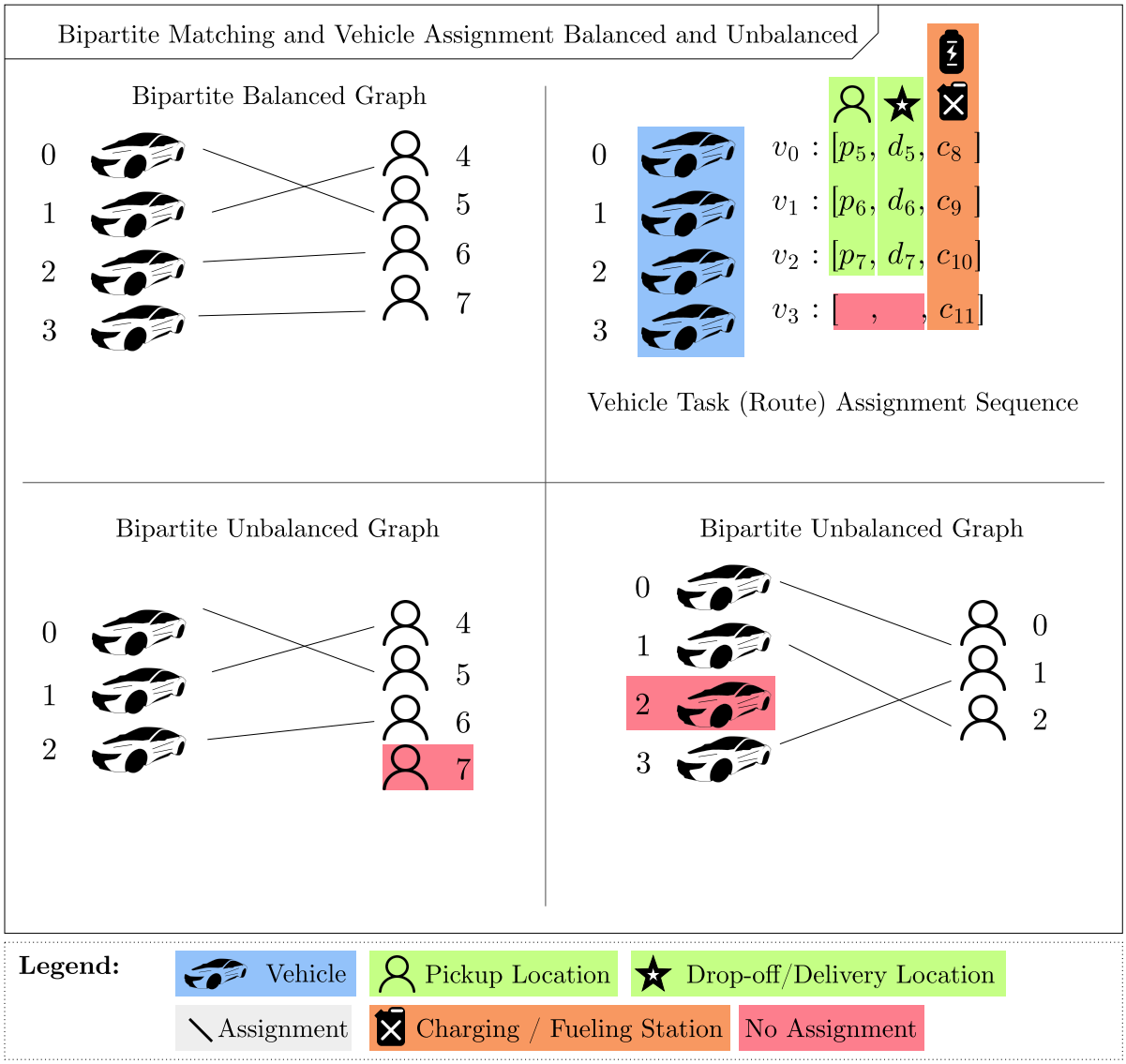


Figure 1.6: The generic algorithmic bipartite assignment or matching process of the core fleet planning and control in order to assign client requests to vehicles [Hamzehi et al., 2021a].

- vehicle (fleet) GPS locations,
- service station or charging station GPS locations,
- accurate traveling distances, e.g. from a routing engine or Application Programming Interfaces (APIs),
- accurate traveling times from a routing engine or APIs,
- accurate vehicle State-Of-Charge (SOC) information, and
- additional information about anomalies such as construction site locations, tidal lanes or one way roads.

Apart from the provided information sources, functional requirements must also be provided. In order to provide meaningful DRL optimization, one requires functional features such as:

- an objective function capturing the strategic goals for the clients and fleet in the environment (city),
- constraints that capture which decisions for planned trips are infeasible,
- high-performance computing power w.r.t. large-scale application, and
- sufficient evaluation and monitoring mechanisms for observing the quality and efficiency of decisions, and

additional information that is helpful for modeling contextual decisions such as

- average velocity on traveled roads, speed limits,
- the seat occupancy of every vehicle, etc.

When optimizing the schedule of vehicles to clients or vice versa, the strategic goals to achieve are defined as mathematical objectives, e.g. serving as many customers as possible or reducing fleet operating costs. Using mathematical formulations different strategic objectives can be defined and also combined. It is assumed that each objective is formulated either using an accumulated cost or a reward (profit) structure, which is well-known in Mixed Integer Programming (MIP). Within this thesis, the main objective is to find the most optimal sets of vehicle-to-client assignments (i.e. ordered integer sequences with minimal cost, also called solution trajectories or paths) for all available vehicles (vehicle supply), originating from client requests (demand). As well, this thesis aims to incorporate assignments to service stations, which presents an additional challenge by inflicting adverse effects (conflicting objectives) to satisfy as much demand as possible.

1.5 Challenges of the Thesis

As Figure 1.6 illustrates, the task of the fleet planning component is to find the best assignments given multiple sets of entities: vehicles, client pickup and drop-off requests, and additional service station requests. Different entities must be distinguished via an identifier (ID), which is represented by subscript numbers in Figure 1.6. Obviously, if more and more entities for possible assignments exist, it is more and more difficult to find the best assignments among all vehicles, requests and service stations (graph complexity increases). In addition, the number of possible decisions increases if some assignments are infeasible. Moreover, the number of active vehicles and active requests can change, which leads to balanced or unbalanced sets of entities. This further leads to the challenge of gracefully denying to assign either vehicles or requests depending on the availability of vehicles (supply) and demand (amount of client requests). Another challenge is that the costs for each assignment may vary depending on the city road network topology (tidal lanes, one-way roads, etc.). This is termed as an asymmetric cost if the visiting order of request and service station assignments may result in different costs, if individual costs do not change with respect to assignment order. Depending on the current demand (i.e. the number of current client requests) the number of possible decisions can increase to very large numbers, making it very difficult for exact and search algorithms to provide fast solutions periodically (see Chapter 2, Section A.4). For this reason, it has been a traditional practice, widely used in large RH companies, to use simple greedy heuristic methods[2],

[2]The term heuristic refers to finding the most optimal way imaginable with the least amount of effort.

solely based on the locations of vehicles and clients. The task of a fleet operator hereby is to dispatch the nearest vehicle to serve a new client [Liao, 2003]. But, there also have been other strategies. Some other central taxi dispatching systems have applied variations of queuing strategies, e.g. First-Come-First-Serve (FCFS) [Alonso-Mora et al., 2017]. The advantage of such simple scheduling strategies is that they are easy to implement by programmers, but also easy to manage and understand by clients. However, they typically result in user-centric and uncoordinated vehicle behavior which prioritizes immediate client satisfaction over optimal vehicle supply utilization. A consequence is that given optimization potentials are not addressed completely, which is particularly important for large-scale Ride-Hailing / -Sharing companies and logistic dispatching systems. The application of simple dispatching strategies also leads to adverse effects on environmental sustainability, citizen health factors, and fleet sustainability, since the given resources are utilized sub-optimally.

Since heuristics most often provide efficient but suboptimal results, the most efficient solution would be a mathematical model. However, since operational environments, fleet tasks, and client patterns differ greatly for different cities and countries, strategic requirements may also completely change by the hour or the minute. This property makes it a tedious effort to develop and tune transferable mathematical models for large-scale RH applications. Similar to heuristics, mathematical models often can be used for a specific task only, and thus are often even less flexible. In order to yield optimal results, exact mathematical solvers can be used for a given cost function. However, such solvers do not re-use any experience of spatiotemporal contextual patterns such as traffic congestion, stochastic demand and supply patterns, and re-occurring events such as vehicle charging. Exact solvers have been used widely in OR, however, for large-scale applications of 40 billion routing requests, and 15 billion location points per day [Ye, 2018], even the fastest mathematical solvers may not be fast enough. This approach also by design would have major disadvantages with respect to platform resources. With this being said, how can a solution method be designed that is characterized by superior flexibility and scalability? This question is further addressed in Section 1.6.

1.6 Solution Approach of the Thesis

In order to research modern methods for achieving advanced fleet planning strategies, this thesis focuses on algorithms for efficient *offline* fleet planning and dispatching optimization. The term offline refers to the characteristic, that all information of clients and vehicles is known in advance. Given large-scale datasets of client orders and fleet data, the goal is to optimize over a short temporal horizon and to benchmark the developed algorithm solutions with other state-of-the-art algorithms. Since modern dispatching and matching systems use *batching* methods to group temporally similar client requests, this thesis focuses on developing a scalable method for repeated parallel computation of batches. Within each batch, groups of similar fleet planning problems are solved as discrete Combinatorial Optimization Problems (COPs). Each fleet planning problem is further represented by small MWBM graphs. Since combinatorial optimization problems are classified among the most difficult existing optimization problems

computation-wise, inefficient algorithms can lead to high energy consumption on a server platform during the service operation. Generally, COPs are hard to scale, which is caused by the enormous amount of solution possibilities (search space size), from which the best combination of tasks has to be found automatically.

For this purpose, this thesis aims to provide a scalable method for repeated parallel computation of similar and small to medium-sized fleet planning problems as MWBM graphs. One of the biggest challenges for the presented optimization component is to handle different node classes at once, where multiple optimization objectives must be achieved for clients and vehicles. Furthermore, this thesis focuses on the development of approximation-based algorithms using DRL, since exact and globally optimal solutions would result in very long system response delays, which increase exponentially for large quantities of client requests, and many operating vehicles. In such a situation, an optimal algorithm may not be practical with respect to short real-time applications. Importantly, this thesis does not focus on finding optimal solvers for large fleet planning problems individually, since such large problems do not occur regularly when using temporal clustering and batching methods.

Although a tremendous amount of work has been dedicated to traditional transportation problems, the scientific research currently available is far from being complete or satisfactory for the increasing and constantly changing requirements resulting from the rapidly evolving digitalization trends. In particular, the decision-making or core planning process for transportation has attracted a surge of interest in machine learning-based methods for large-scale transportation and route networks. Here, the application of available mathematical models for fleet operation, management, and routing is usually highly problem specific and typically cannot be generalized or transferred to other problems. The comparably complex objective diversity in fleet dispatching and pickup- or delivery problems originates from many different system goals that have to be satisfied or for which the best consensus must be found (visualized in Figure 1.7). Continuously changing system objectives lead to very dynamic optimization requirements, especially when the mathematical model must be changed for different applications. The development of mathematical models for each slightly different fleet routing or dispatching application results in a repetitive and tedious process, where each newly developed model has to be tuned with parameters and validated for deployment on a platform. Usually, for deployment, a clear definition of the mathematical function is required, but what can be done if such a definition does not exist or cannot be defined covering all scenarios from the start? What if such a definition additionally needs to be changed frequently, as previously mentioned?

An example of this case is the algorithmic application to different urban environments or cities which may be structurally and inherently different. In order to be applicable and scaleable to different environments, a more flexible approach is required which is capable of adapting, or rather of learning new models from scratch and from the actual data basis.

With this all being said, what if a mathematical model could be learned automatically and could generalize (i.e. learn to adapt autonomously) to many or even all fleet operating environments? This would obviously simplify the algorithm development process significantly and speed up the development process for real-world application. Such a mechanism of automated model learning

will be fundamentally system-relevant for the successful implementation of future fleet management solutions, and this is precisely why this thesis focuses on this aspect. Note, the next section describes the domain of fleet dispatching, routing, management, and operation research processes in more detail, in order to provide more technical context for this thesis.

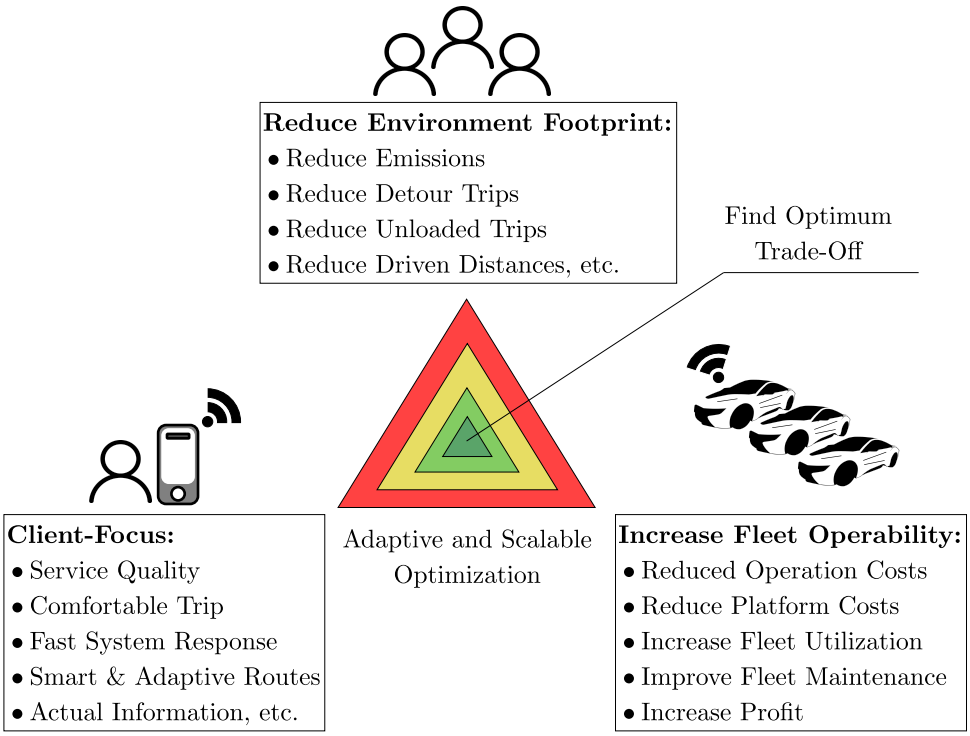


Figure 1.7: Three stake-holding entities influencing the objective function for designing optimal system behavior within mobility-services and fleet management domains.

1.7 Research Questions and Outline of the Thesis

After providing the research context and introducing the thesis' objectives, this section further details the objectives of specific research questions that will be answered at the end of this thesis. The research questions are further based on the publications made within a 4-year time frame from 2017 to 2021. As outlined before, the platform requires a flexible and efficient algorithm for processing batches for planning and managing vehicular-request assignments. The choice of DRL is based on past work and the current state-of-the-art where such algorithms have already been used and have shown very good results [Syed et al., 2019b; Feng et al., 2020; Lin et al., 2018]. From this state, the following research questions are proposed:

- *Research Question 1: Can a Deep Reinforcement Learning approach learn near-optimal fleet routing and assignment controls for generally occurring graph structures in collaborative fleet routing and dispatching?*
- *Research Question 2: If Research Question (RQ) 1 is yes, does this assumption hold for*

symmetric and asymmetric graphs?

- *Research Question 3: If RQ 1 and RQ 2 are yes, does this assumption hold for balanced and unbalanced graphs?*
- *Research Question 4: If RQ 1, RQ 2, and RQ 3 are yes, does this assumption hold for mathematical problem constraints?*
- *Research Question 5: If RQ 1, RQ 2, RQ 3, and RQ 4 are yes, can a DRL-based approach be applied to real fleet data?*

This thesis covers one major aspect of developing a DRL architecture for fleet planning as a discrete Combinatorial Optimization Problem (COP) in the ODM domain. The developed architecture comprises a deep Recurrent Neural Network (RNN)-based Pointer Network and a Reinforcement Learning (RL) algorithm called Advantage Actor-Critic Monte-Carlo Policy Gradient (AACMCPG) for training the RNN. The Actor network includes an Attention Mechanism for learning contextual information from the environment (urban map data) and a mask function in order to satisfy given optimization constraints. Moreover, the thesis contains experimental, and computational simulations as well as analytic discussions and a final applied implementation instance tested with real client and fleet data. In order to provide a benchmarking framework (see Figures 1.8 and 1.9), the thesis includes multiple sub-parts for comparing the algorithm solution quality, efficiency, and reliability. Initially, chapter 2 introduces the necessary mathematical formulations, optimization objectives, and fundamental definitions. In order to research different capable approaches for fleet planning, Chapter 3 summarizes the current state-of-the-art from multiple research domains of On-Demand Mobility (ODM), Machine Learning (ML), Natural Language Processing (NLP), discrete Combinatorial Optimization Problem (COP), and Operations Research (OR). Note, that only the state-of-the-art with respect to combinatorial DRL and MIP regarding combinatorial optimization problems with the focus of operations research are considered within Chapter 3. Since both fields of research are very vast, this thesis only introduces the state-of-the-art in a very abstract, and punctual way. The Fundamentals of Graph Theory or Maximum Flow Optimization can be found in Chapter 2, while the basics of Deep Reinforcement Learning (DRL) can be found in Chapter 4. The main development phase within this thesis starts in Chapter 5, where further detailed Benchmarks with different state-of-the-art algorithms are shown in Chapter 6. Finally, Chapter 7 presents a summary of conclusions and aspects about future work and mechanisms for possible improvements.

1.8 Contributions and Outcomes

This thesis advances theoretical and practical knowledge of machine-learning-based algorithms in a real-world platform-based application of autonomous fleet operation and mobility services. The value of the outcomes will be important to one or more communities (mobility service managers, environmental engineers, machine learning algorithm developers, and computer scientists). The thesis therefore contains outcomes about algorithm learnability, efficiency (scalability), solution quality (optimality), and solution quantity (reliability) which heavily influence operation costs of an autonomous-operated vehicle fleet in urban environments. In particular, the thesis benefits

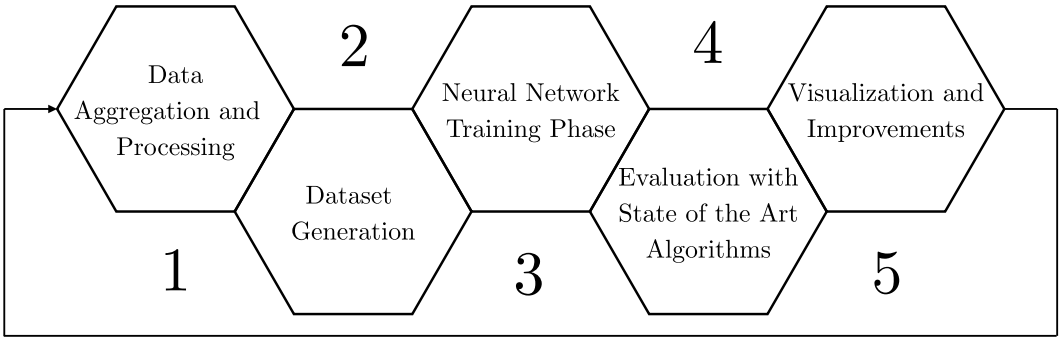


Figure 1.8: The individual methodological phases of this thesis.

the domain of Combinatorial Optimization (CO) for complex mathematical problems. Generally, COPs are hard to scale which is caused by the enormous amount of possibilities where the task is to automatically find and select the best solution. In practice, there are even more difficulties added by modeling a number of variations which is characterized by real-world application. This thesis further contains practical insights about the development process of applied machine learning algorithms in a real working environment, i.e. the computation platform often called the "cloud" and in the presence of real data. The thesis furthermore touches on multiple areas in intelligent transportation systems, machine learning, robotics, traffic engineering, operations research, computer science, statistics, informatics, game theory, network modeling, and graph theory.

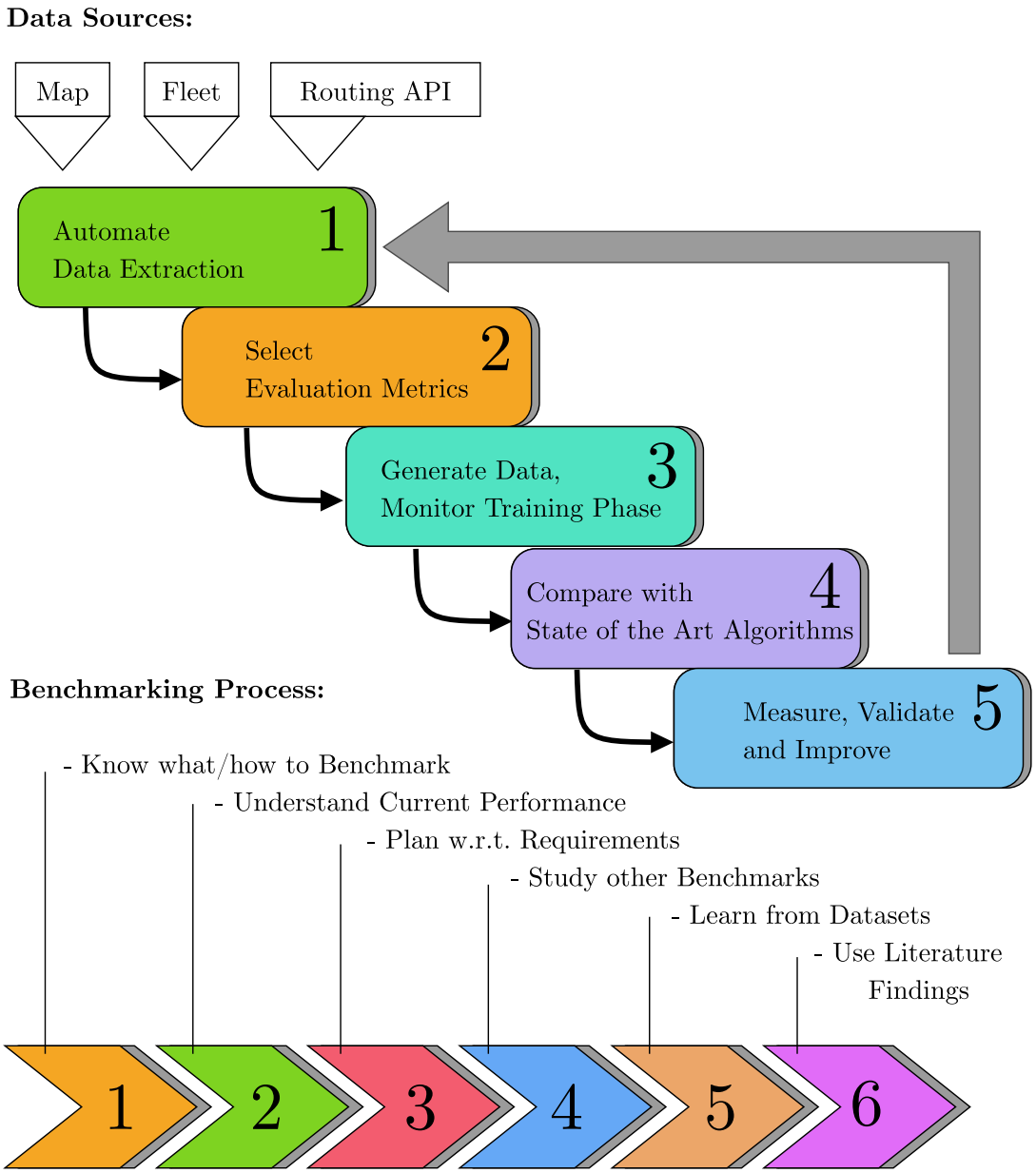


Figure 1.9: The Process of Algorithm Benchmarking.

2. Fleet Dispatching and Combinatorial Optimization

The biggest challenge in fleet planning with DRL is the "Curse of Dimensionality" [Bellman, 2015, p. 94]. On one hand, there is the size of the feature space dimensionality and the combinatorial complexity on the other hand. When using DRL for fleet planning, it is crucial to use a representation of the mathematical fleet planning problem graph that scales well with the dimension of state space features, which can be for example travel times, distances, vehicle ranges, and other physical entities. Since this thesis deals with approximate computations of state spaces (called "Embeddings"), it is possible to over-approximate or under-approximate complex state and action spaces. Another challenge is that more data to consider increases the training time, which can lead to the exhaustion of computation capacities, even for modern high-performance computation platforms. Problem constraints are hard to learn for any ML-based method, but without constraints, the solution to a given fleet planning problem is less valuable because of missing solution guarantees (e.g. also called conflict-freeness). Another important aspect is that the most important computations for predicting fleet planning solutions must be computed efficiently. This property limits the quantity of optimization methods that can be used for large-scale fleet planning purposes during the prediction phase with DRL. In order to provide more details about the fleet planning problem as discrete Combinatorial Optimization Problem (COP) this chapter continues to introduce important mathematical definitions, formulations, and objective descriptions. This chapter is organized as follows: Section 2.4 introduces the PDP problem formulations with representations of general network graphs, combinatorial optimization objectives via MIP and MWBPM representations, the representation of individual solutions, and further fundamental definitions. Next, it is shown in Section 2.6, how an MIP representation can be converted to an MDP representation.

2.1 Introduction to Graph-Networks

Different types of networks including information flows can be encountered in our daily lives. For instance, electrical circuits, telephone communication, cables, manufacturing, computer networks, supply-demand logistics, railroads, and road networks. Using existing formulations in graph theory, such a network or graph structure can be described as a network graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$. The short form, commonly just called a *graph*, consists of special points called *nodes* $\mathcal{N}$ and their connections called *links* $\mathcal{L}$. Some general networks examples are listed in Table 2.1.

Network	Nodes	Links	Flow
communication	data exchange, phone, computers, satellites	cables, optics, microwaves	messages: voice, text, video, data
financial	currencies, stocks, demand, supply	transactions	money
hydraulic	lakes, reservoirs, pump stations	pipelines	hydraulic fluids: water, gas, oil, etc.
transportation, logistics	airports, rail hubs, road intersections	roads, highways, railroads, flight routes	vehicles, passengers, cargo

Table 2.1: Common Examples of Networks [Wayne and Tardos, 1999, pp. 2–4].

2.2 Set and Node Representations

The most general entities considered in automated fleet planning are sets of nodes that represent a location on a map (see Figure 2.1). However, in order to build more context, different classes of nodes for vehicles, client requests, and $\langle e, g \rangle$ are considered within this thesis. First, a vehicle set that specifies the number of available vehicles within an environment (i.e. the operation area of the fleet) is considered. In particular, a vehicle node class can have multiple physical attributes such as an Identifier (ID) that enables to distinguish all vehicles, location coordinates, a number of passengers or capacities, a State-Of-Charge (SOC) for electric vehicles, a State-Of-Fuel (SOF) for combustion engines, limited driving ranges, and other status messages.

Definition 1 (*Vehicle Set*)**:** The *Vehicle Set* denotes a set of vehicles $\mathcal{V} = \{v_0, ..., v_n\}$ where each vehicle has a location p_v and a passenger capacity $c_v = 1$ at any given time t.

The major task of an ODM fleet is to satisfy a client demand, which is represented by a quantity of client requests. Similar to the vehicle nodes, a client request requires an ID and location for any pickup and drop-off trip on demand.

Definition 2 (*Request Set*)**:** The *Request* Set consists of multiple client-requests (vehicle orders) $\mathcal{R} = \{r_0, ..., r_m\}$ which may also be referred to as request schedule.

The difference to the vehicle node is that a client request additionally provides the trip destination (drop-off location) as secondary information.

Definition 3 (*Request*)**:** A *Request* is denoted by multiple attributes $r^{[p,d]} = \{p^o, p^d\}$ such as a client pickup r^p and drop-off request r^d which entails the trip routes from origin position p^o, to the drop-off destination position p^d.

For RH it usually is assumed that each client request is characterized by a passenger capacity c^v of 1. Both pickup and drop-off location tuples (i.e. two values for *lat* and *lon* or x and y coordinates) are considered as nodes on the map (visualized in Figure 2.1). During demand satisfaction, the operating fleet consumes energy which has to be recharged or refilled. Hence, a set of fixed service, fuel, and charging stations as another class of location nodes with different attributes is considered.

Definition 4 (*Service Station Set*)**:** The *Service Station Set* consists of multiple electrical e or gasoline g service station locations $\mathcal{S} = \{s_0^{\langle e,g \rangle}, ..., s_o^{\langle e,g \rangle}\}$ with ID o, an individual location tuple p^s and number of free slot capacity c^s, that can be used to recharge or refuel a vehicle.

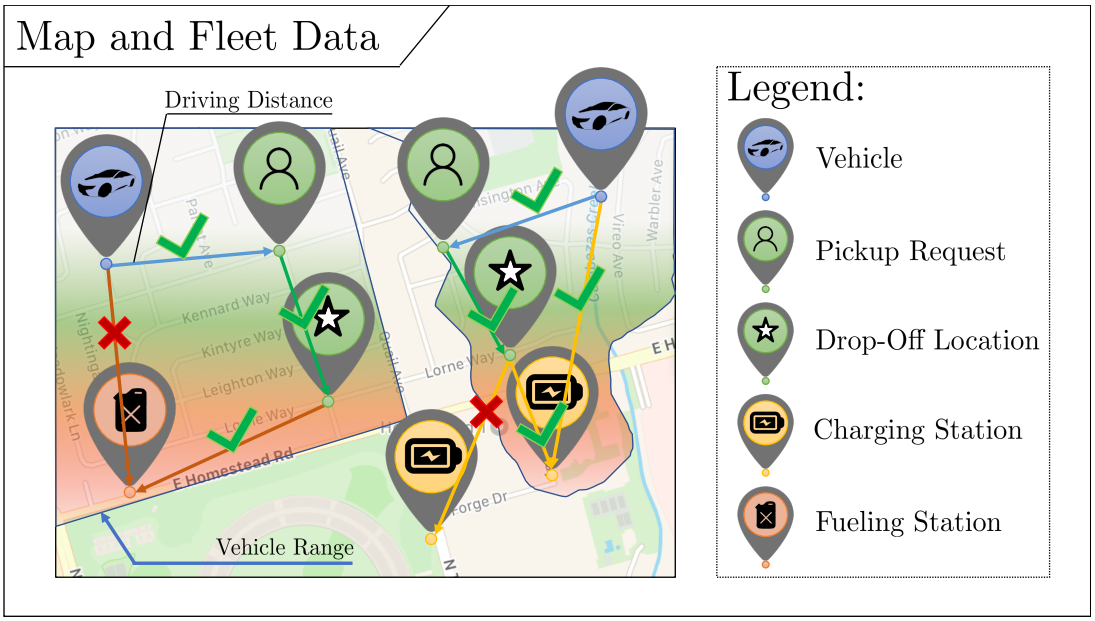


Figure 2.1: The pickup and delivery/drop-off problem with a collaborative fleet.

2.3 Graph-Network Representations

Modeling a whole city complexity as an operational environment would exceed even modern memory requirements and running cost of any ODM platform. Luckily, it is sufficient to use abstract models to represent the most important environment characteristics. In order to represent the city as operational environment, this thesis considers a given road graph-network, and more specifically, a constructed assignment network which is denoted as bipartite graph $\mathcal{G}$. The graph $\mathcal{G}$ further comprises all node sets and so-called links $\mathcal{L}$ that represent a trip between given location nodes. In order to distinguish a large quantity of links, each link is denoted by a subscript tuple $\langle i, j \rangle$, which represents the direction of trip traversal from location i to location j.

Definition 5 (*Graph-Network*)**:** A general *Graph-Network* is denoted by a weighted bipartite and asymmetrical graph $\mathcal{G} = \langle \mathcal{N}, \mathcal{L}, \mathcal{C} \rangle$ with a set of nodes $\mathcal{N}$ and a set of links $\mathcal{L}$ with corresponding costs or weights $\mathcal{C}$. Each link $\langle i, j \rangle \in \mathcal{L}(n_i, n_j \in \mathcal{N})$ corresponds to a traveling cost along the link which is determined by a cost function using time, distance, price values or other similar utility entities. If the vehicle driving speeds are available, both metrics can be transformed interchangeably.

As Definition 5 shows, all graph link weights are characterized by individual trip costs $\mathcal{C}$. Each link cost thereby can bootstrap multiple profits, costs and other attributes at once. If multiple

costs are bootstrapped, the whole link weight is either represented as a final cost or profit, i.e. a real value $c \in \mathbb{R}$.

Definition 6 (*Travel Cost*)**:** Given at least two nodes in a graph-network $\mathcal{G}$, the travel cost is indicated by $C(n_i, n_j)$ as the link cost between any two nodes $n_i \in \mathcal{N}$ and $n_j \in \mathcal{N}$.

If multiple trips between locations are considered, the concatenation of locations results in costs for traveling a route or multiple routes. The path or route cost further is an ordered sequence of costs which can be accumulated to a singular cost value, representing the costs of the whole travel path.

Definition 7 (*Cost Path*)**:** A travel cost path $\mathcal{P}$ is a vehicle route which is defined by the sum of all travel costs over the traversed nodes and links starting from the sequence: vehicle nodes to request pickup locations, to drop-off locations and to service station nodes $\langle v_n^{\langle e,g \rangle}, r_m^p, r_m^d, s_o^{\langle e,g \rangle} \rangle$. The travel cost path,

$$\mathcal{P} = \langle v_0^{\langle e,g \rangle}, r_0^p, r_0^d, s_0^{\langle e,g \rangle} \rangle, ..., \langle v_{|\mathcal{N}|-1}^{\langle e,g \rangle}, r_{|\mathcal{R}|-1}^p, r_{|\mathcal{R}|-1}^d, s_{|\mathcal{S}|-1}^{\langle e,g \rangle} \rangle$$

is represented by an *ordered* sequence of vehicle, origin and destination of requests and charging stations. There is also the possibility that the ordered list may be reduced to

$$\mathcal{P} = \langle v_0^{\langle e,g \rangle}, s_0^{\langle e,g \rangle} \rangle, ..., \langle v_{|\mathcal{N}|-1}^{\langle e,g \rangle}, s_{|\mathcal{S}|-1}^{\langle e,g \rangle} \rangle$$

which results in a vehicle trip to the service station directly.

Obviously, since multiple vehicles are considered simultaneously, every vehicle can be assigned with diverse routes and designated route costs individually. Hence, most optimal route assignment is crucial for any ODM fleet and plays a major role for an operational success. In particular, the optimization of assignments furthermore saves costs and thus increases fleet operational profits. The next section continues to describe how strategic and collaborative objectives can be represented.

2.4 Objective and Constraint Formulations

Ideally, a fleet always is provided with collaborative routes to minimize the total effort or maximize the total profit during operation. Any ODM service requires to satisfy strategic goals, where different goals exist to increase client comfort, reduce fleet operational costs or reduce the environment footprint. Different mathematical ways and formulations of modeling collaborative objectives exist. Within OR, traditionally PDPs are modeled as MIP comprising a decision matrix $\mathbf{X}$ where any residing decision value can take integer values of either 0 or 1. General MIP formulations also come with at least one cost matrix $\mathbf{C}$ that models individual trip or route assignment costs from origin to destination locations as follows:

$$\mathbf{X}^{[N\times M]} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{NM} \end{bmatrix} \quad (2.1) \qquad \mathbf{C}^{[N\times M]} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1m} \\ c_{21} & c_{22} & \dots & c_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{NM} \end{bmatrix} \quad (2.2)$$

Both matrices have N rows and M columns, where N represents the number of origin locations and M is the number of destination locations. Such structures are called Origin-Destination (O-D) matrices, where origins are represented by vehicle nodes and destinations are represented by the next pickup locations or service station location nodes. When using O-D matrices to describe the transportation flow of vehicles to requests and $\langle e, g\rangle$. The decision structure results in an MWBM graph which can be solved via an MIP assignment problem formulation, as shown in Equation (2.5).

Definition 8 (*MWBM*)**:** Given a Minimum Weighted Bipartite Matching (MWBM) graph, a set of vehicles $\mathcal{V}$, requests $\mathcal{R}$, and $\langle e, g\rangle$ $\mathcal{S}$, the optimization objective is to find the optimal set of route assignments, such that for each vehicle $v \in \mathcal{V}$ the optimal travel route cost $\mathcal{P}^*$ is minimal, and such that the objective cost function $C(\mathcal{V}, \mathcal{R}, \mathcal{S})$ is minimized

$$C(\mathcal{V}, \mathcal{R}, \mathcal{S}) = \min \sum_{b=0}^{B-1} \sum_{h=0}^{\mathcal{P}_{i,j}-1} \mathbf{C}_{i,j}\mathbf{X}_{i,j} \quad (2.3)$$

$$\sum_{h=0}^{\mathcal{P}_{i,j}-1} = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \quad (2.4)$$

$$[B,\, \mathcal{P}_{i,j}] \in \mathbb{Z}_0^+ \quad \forall\, \{b, h, i, j\} \quad (2.5)$$

subject to the constraints

$$\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \mathbf{D}_{i,j} < \mathbf{r}_n^v \quad (2.6)$$

$$\sum_{j=1}^{m} \sum_{h=1}^{p_{i,j}} \mathbf{X}_{i,j} = 1 \quad \forall i, \quad (2.7)$$

$$\sum_{i=1}^{n} \sum_{h=1}^{p_{i,j}} \mathbf{X}_{i,j} = 1 \quad \forall j, \quad (2.8)$$

$$\mathbf{X}_{i,j} \in \{0, 1\} \quad \forall \{i, j\} \quad (2.9)$$

$$\mathbf{X}_{i,j} \in \mathbb{Z}^+ \quad \forall \{i, j\} \quad (2.10)$$

within the given constraints: i) *route feasibility constraint*: The solution path $\mathcal{P}$ must be a feasible route in the sense that the vehicle driving range exceeds the driving distance; ii) *unique assignment constraint*: each vehicle must serve a different request and thus must result in a minimal-cost perfect matching/assignment. The objective cost function is defined by multiple

cost matrices (batches) $\mathbf{C}_{i,j}$ and multiple decision or also called adjacency matrices $\mathbf{X}_{i,j}$.

Each individual solution to a given objective cost function can be represented as two sequences of IDs between vehicle nodes and destination nodes (client pickup nodes or service station nodes). Since every assignment between any ID tuple results in an assignment cost, equally there exist cost sequences for multiple assignments.

Definition 9 (*Assignment Cost*)**:** Given at least two nodes in a graph-network $\mathcal{G}$, the assignment cost is calculated by a function $D(n_i, n_j)$ as the distance between any two nodes $i \in \mathcal{N}$ and $j \in \mathcal{N}$.

Multiple vehicles, may have multiple assignment cost paths, which are as previously mentioned, represented by ID sequences of nodes that are visited per vehicle.

Definition 10 (*Assignment Cost Path*)**:** An assignment cost path $\mathcal{P}$ can include multiple nodes per vehicle, which is represented by the sum of all travel costs over the traversed nodes and links starting from the sequence: vehicles to pickup location, to drop-off location, and to a final service station location node $\langle v_n^{\langle e,g \rangle}, r_m^p, r_m^d, s_o^{\langle e,g \rangle} \rangle$. Technically, a vehicle path such as

$$\mathcal{P} = \langle v_0^{\langle e,g \rangle}, r_0^p, r_0^d, s_0^{\langle e,g \rangle} \rangle, ..., \langle v_{|\mathcal{V}|-1}^{\langle e,g \rangle}, r_{|\mathcal{R}|-1}^p, r_{|\mathcal{R}|-1}^d, s_{|\mathcal{S}|-1}^{\langle e,g \rangle} \rangle$$

can be represented by an *ordered* integer sequence of vehicle, request pickup/drop-off, and service station Identifiers (IDs). There is also the possibility that the ordered list may be reduced to

$$\mathcal{P} = \langle v_0^{\langle e,g \rangle}, s_0^{\langle e,g \rangle} \rangle, ..., \langle v_{|\mathcal{V}|-1}^{\langle e,g \rangle}, s_{|\mathcal{S}|-1}^{\langle e,g \rangle} \rangle \,,$$

which results in a vehicle trip to the service station directly.

Multiple cost paths of multiple vehicles are called assignment policy. The assignment policy hereby is a strategic decision over all vehicles to minimize the objective costs via collaboration.

Definition 11 (*Assignment Policy*)**:** An "*Assignment Policy*" of a vehicle v is denoted by multiple assignment sequences $\mathcal{A}_\pi = \langle \mathcal{P}_0, ..., \mathcal{P}_{|\mathcal{N}|} \rangle$ with $\mathcal{N} = \mathcal{V} \cup \mathcal{R} \cup \mathcal{S}$, which is represented by multiple sets of ordered sequences, that exist for each vehicle.

As previously mentioned, each among the sequences consist of IDs that represent a trip from vehicle origin nodes $v_n^{\langle e,g \rangle}$ to either request pickup nodes r_m^p and request drop-off nodes r_m^d, or from vehicle origin nodes $v_n^{\langle e,g \rangle}$ to charging or service station $s_o^{\langle e,g \rangle}$ location nodes directly. Note, that the trip cost for assigning a vehicle to a client request or service station can be calculated via a variety of metrics and distances. A common case within ODM is to represent the assignment costs with the travel distance or travel time for traveling between any O-D pairs of locations. It is crucial to distinguish the used distances or metrics via terms such as *symmetric* or *asymmetric*. Distances by default may not be symmetric for real road networks with the existence of one way roads or tidal lanes. Generally, real road distances rarely satisfy the triangle inequality. On one hand, this limits the combinatorial complexity for a given dispatching problem, however, this also makes it harder for a learning algorithm to learn a combinatorial strategy that solves the

dispatching problem. If a distance function is called on any pair of nodes and the calculated distances satisfy the triangle inequality, the distance function is called a metric.

Definition 12 (*Triangle Inequality*)**:** The triangle inequality states that for any triplet of nodes i, j and k the edge weights or costs between satisfy

$$c_{i,j} \leq c_{i,k} + c_{k,j} \quad \forall i, j, k \ \in \ \mathcal{N} \ . \tag{2.11}$$

Note, that the term symmetric also refers to the cost values above and below the matrix diagonal of any given cost matrix, for example, see in Equation (2.5). A metric produces costs that equal, regardless of the assignment direction between any origin-destination node pair. On the other hand, a normal distance function can calculate costs that are different depending on the assignment direction.

Definition 13 (*Symmetry*)**:** The symmetry between any tuple of nodes i, j yields the same cost or distance values for $D(n_i, n_j) = D(n_j, n_i)$.

Apart from reaching objectives in combinatorial optimization, assigning vehicles for client requests without constraints would result in unpleasant consequences, particularly if the number of vehicles currently available, do not equal the number of client requests currently booked. If a vehicle would be assigned to two clients, this would not result in a proper solution. Such effects are resolved via modeling constraints. The consequence of constraint modeling in Equations (2.7) and (2.8), ideally ensures a solution that is called a *perfect* assignment or matching. Another case where constraints are required is if the current vehicle ranges do not suffice to carry out specific long trips.

Definition 14 (*Vehicle Range Constraint*)**:** A vehicle node is defined to have limited range R^v attributes while traveling within the road network and while serving customer pick-up and drop-off requests.

Additionally, constraints ensure that only vehicles are assigned to client requests that ensure feasible trips.

Definition 15 (*Feasible Trip*)**:** A feasible trip is valid only if the vehicle range is sufficient to execute the trip successfully, i.e. if the current vehicle range exceeds the trip's total travel distance denoted by $\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} D_{i,j} < r_n^v$.

The travel costs further can be either distances or an average Estimated Time of Arrival (ETA), which can be obtained by querying a Routing Engine, such as the open-source OSRM Router. If the assumption of traffic is ignored, the travel distance and travel time can be used interchangeably.

2.5 Graph Variants

To summarize, Figure 2.2 shows the different MWBM problem graph variants considered in this thesis. As previously mentioned, one can distinguish

- "Balanced MWBM Graphs", where the "cardinality", i.e. the numbers of elements between the matched sets, such as vehicles V and requests R must be equal,
- "Unbalanced MWBM Graphs", which is the counter-example such that both sets may have unequal numbers of elements,
- "Symmetric MWBM Graphs", which refers to the definition that bidirectional assignment directions $(v_0, r_1), (r_1, v_0)$ do not result in different assignment costs or weights $C(v_0, r_1) = C(r_1, v_0)$ for a symmetric MWBM graph,
- "Asymmetric MWBM Graphs", which are characterized by different costs or graph weights depending on the assignment direction $C(v_0, r_1) \leq C(v_1, r_0)$, and finally,
- "MWBM Graph Constraints", required for modeling limited vehicle ranges or distances, which require modeling infinite costs for infeasible graph link weights $C(v_0, r_1) = \infty$.

Generally, such properties have an influence on the solution structure, -efficiency, and -quality and thus are crucial to consider. Clearly, the presented problem description is well-known in OR

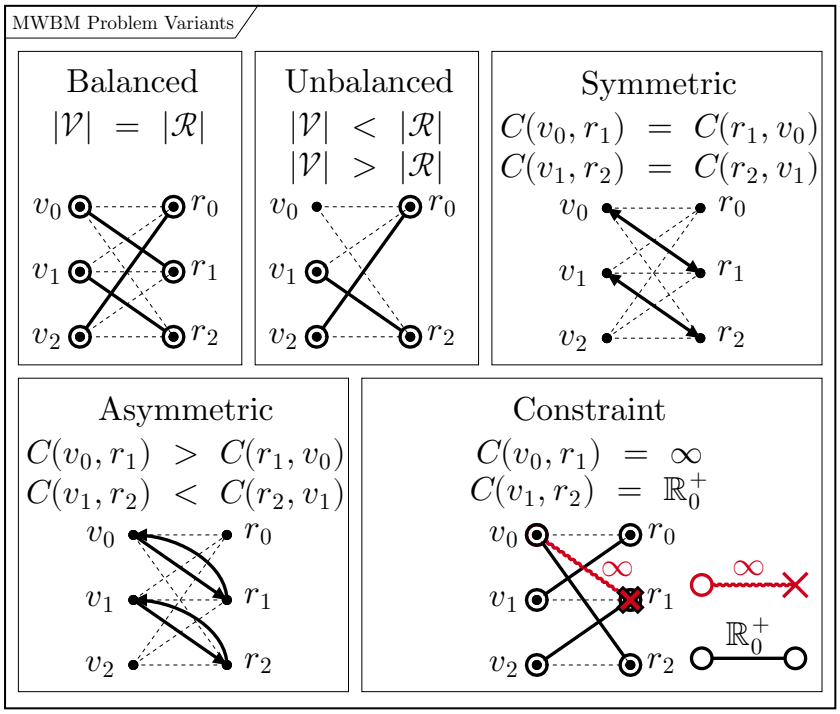
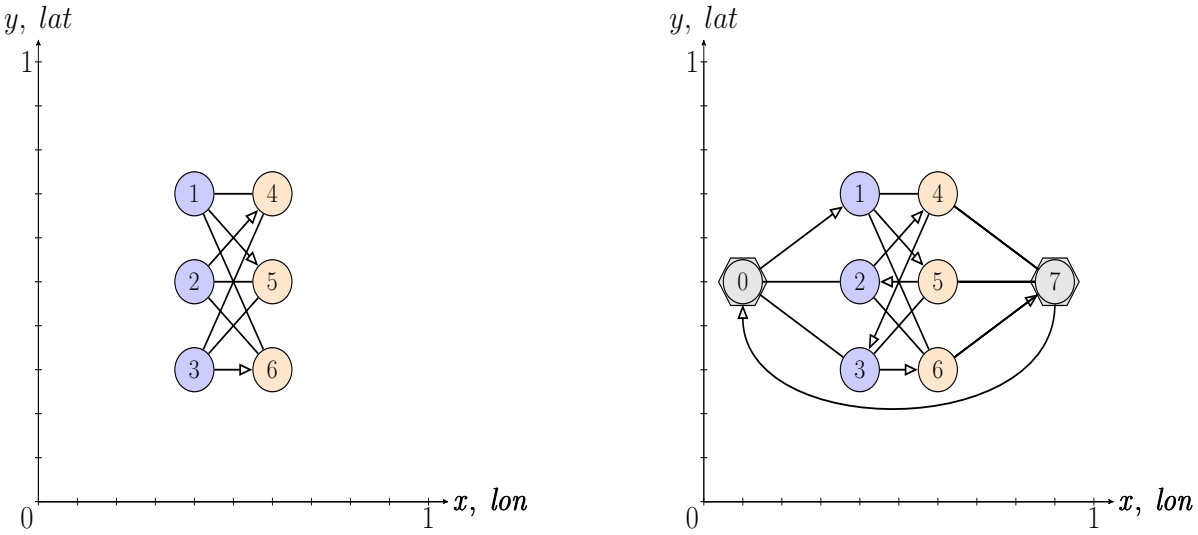


Figure 2.2: Different problem variants of Minimum Weighted Bipartite Matching (MWBM) graphs that have an impact on solution structure, quality and efficiency.

and discrete combinatorial optimization. However, in order to solve such a problem with DRL, a method must be found to reformulate the MWBM problem structure as an MDP. Hence, the next section will address one way how to deal with this challenge.

2.6 Conversion of Objective Formulations

A general MWBM problem setup can be typically solved optimally via the Hungarian method [Kuhn, 1955], where the decision graph is visualized in Figure 2.3 and 2.5. The presented objective conversions in Figures 2.3a and 2.3b show that the problem setting in Equation (2.5) can be exactly solved using other mathematical solvers as well. This includes MIP assignment, CP-SAT, Integer Linear Program (ILP), TSP, VRP, Auction Algorithms, Minimum Cost Flow solvers, and MWBPM algorithms, which are well-known in discrete combinatorial optimization. Also, the relation between the MWBM problem and the minimum flow problem in Figure



(a) A MWBM Vehicle-Request Assignment Problem.

(b) The same graph but solved as Traveling Salesman Problem (TSP).

Figure 2.3: A MWBM Graph to converted to a TSP with shortest augmenting path.

2.3 explains why a DRL approach that solves Traveling Salesman Problems (TSPs) can be used additionally to solve the MWBM problem as well. The main reason is that by adding a source and a sink node to an existing MWBM problem becomes a more general minimum flow problem formulation. In order to solve the special instance (i.e. the MWBM problem) with a DRL-based approach, the mathematical formulation has to be transferred to a Markov Decision Process (MDP) formulation. The transfer to a MDP is visualized in Figure 2.4, which shows that each individual trip among all vehicles has to be reduced to individual assignment actions.

Definition 16 (*Markov Decision Process (MDP)*)**:** An MDP is denoted by a tuple of a state space, action space, reward function, and state transition function denoted by $\mathbb{S}$, $\mathbb{A}$, $R(\cdot)$, and $T(\cdot)$. A MDP that provides mathematical processes for sequential decision-making problems.

Generally, Figure 2.1 visualizes an example of a RH planning problem, where optimal routes for a fleet of vehicles is desired with respect to driving range constraints. If a client request can be served, the final stop is again a service station. If the vehicle range does not suffice, the vehicle is

redirected to a service station directly. The optimization algorithm has to consider vehicle ranges and the distances for incoming client requests (pickup and drop-off trip), and the follow-up route to a given service station. The optimization algorithm furthermore has to decide if such a trip is feasible within the driving range limits and which combinatorial route assignments yield the minimal cost based on the objective in Definition 8 (also see Equation (2.5)). The vehicle, the request, and the services station locations are given via coordinates (latitudes and longitudes). It is assumed that each vehicle has a capacity of 1, whereas a vehicle must be able to follow the route sequence called *Action 1* (vehicle to, pickup, drop-off, and service station locations) to serve a client request or *Action 2* (vehicle to service station location) in order to recharge/refuel the vehicle state of charge.

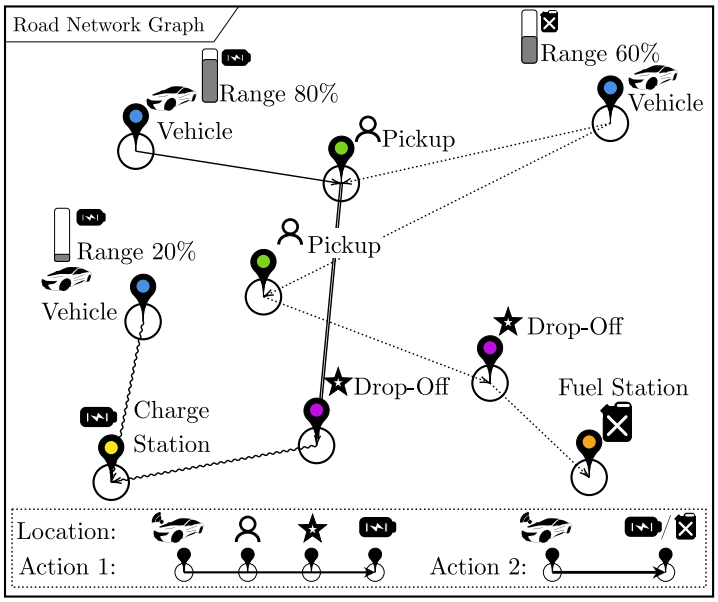


Figure 2.4: State and action space model, by transforming the MIP to a MDP.

Definition 17 (*MDP Action*)**:** The MDP actions include two main types of actions in the presented fleet planning setting.

By reducing the route node assignments to bootstrapped actions, the MWBM is transformed to an MDP. This yields the advantage that the problem setting of state transitions, actions, and rewards can be simplified. With this measure, the whole fleet planning problem can be seen as a centralized decision-making component, i.e. also called a single agent MDP. Moreover, the environment (i.e. the city network, the fleet, the client-request, and service station data) must contain the complete information required for the decision-making process, which is called MDP state.

Definition 18 (*MDP State*)**:** The MDP state must satisfy the Markov property and includes all information required to solve the MDP.

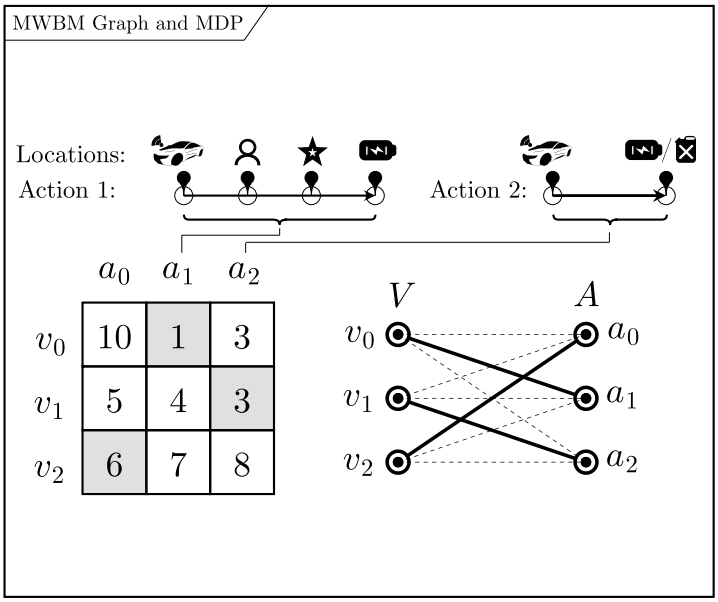


Figure 2.5: A fleet planning problem modeled as Markov Decision Process with cost matrix.

Importantly, the MDP state serves as the basis input mechanism for the DRL architecture and further developments in Chapter 5. The MDP state is defined to include all current node features such as latitudes, longitudes, the vehicle ranges and the distance and time metrics to currently available request pickup and drop-off and service station nodes.

3. Combinatorial Optimization in On-Demand Mobility

The introduced problem formulations (static fleet planning problem, MIP, PDP, MWBM, and MDP formulation) in the previous chapter are used for reviewing different solution approaches and results in this chapter. All formulations belong to the category of mathematical discrete combinatorial optimization problems, which also include shortest-path problems such as TSPs and VRPs. An introduction to combinatorial optimization with respect to fleet planning problems is presented below.

3.1 Introduction and State of the Art

Basically, discrete combinatorial optimization techniques determine an optimal set of solution indices (integers) that represent assignment IDs for vehicle, request, and service station nodes among a large set of possible assignment combinations. Using a graph model, the assignments can be also represented as graph links and thus, the same problem can be solved by finding an optimal unique set of graph-links with the least global cost. When using a PDP formulation, which is a special instance of the VRP, shortest-path techniques (TSP Solvers) can be used as well to find optimal collaborative routes [Gamache et al., 2005, pp.1]. The most simple model to use for exact solutions, however, is to use an MIP or MWBPM formulation as assignment problem. For problem structures as such, a variety of solution methods exist, where a well-known optimal algorithm is the Hungarian algorithm (when using the MWBPM formulation). Due to the high problem complexity, however, it has been a wide practice to use simpler heuristics (see Section 3.3). Heuristics are interpreted as naive rules which are similar to a policy (strategy) to make good decisions [Kool et al., 2018, p. 2]. When formulating the problem as MIP, subsequently also well-known MIP assignment, CP-SAT, ILP, and Auction Algorithms can be used (see Section 3.2, "Exact Algorithms"). An even more general formulation can be achieved when formulating the problem as maximum flow problem with minimum costs. Maximum flow solvers algorithms are well known in OR. All mentioned methods have their own advantages, disadvantages, and characteristics when it comes to optimality, scalability (efficiency), and reliability. With respect to the aforementioned algorithms, a lot of literature can be found, which makes it relatively straight forward to asses which algorithm might be the best for the given problem. Since this thesis aims to build a DRL-based approach, which is a fundamentally different approach by definition. Thus, the question now is how competitive an MDP formulation, i.e. being solved

by a DRL approach, might be. With respect to ODM applications, DRL approaches have been widely and successfully used (see in Section 3.4 "DRL Applications in ODM"). However, with respect to graph-based fleet planning and combinatorial optimization, comparably little closely related literature (compared to exact solution methods) is available. One important relation however is found, if the problem formulation is considered a sequential-decision making process. Thus, Section 3.5 "DRL Algorithms" will review the contributions of other authors and their achievements with respect to combinatorial sequential-decision making (see Section 3.5, "DRL Algorithms").

3.2 Exact Algorithms

At a glance, Table 3.1 shows the history of exact MWBPM or shortest augmenting path algorithms is dated back to the year of publication in 1955-1965, when the first exact shortest-augmenting path (Hungarian Method) and MWBPM have been published [Kuhn, 1955; Edmonds, 1965]. Kuhn further noted that their algorithm even was latent in the work of the Hungarian mathematicians Köning and Egerváry [Kuhn, 1955; Duan et al., 2018].

The history can be traced back even further, where recent rediscovery has stated that Jacobi had described a variant of the Hungarian algorithm in 1865 [Ollivier, 2009; Duan et al., 2018]. Munkres showed that the running time of the presented Hungarian method is $\mathcal{O}(n^4)$ for quadratic matrices at the time. Here, n means the number of nodes serving as origins and m is the number of nodes serving as destinations for rectangular matrices. In the case of quadratic matrices $n = m$ holds, i.e. the number of origin nodes equals the number of destination nodes. Additionally, Gleyzal published a polynomial time cycle-canceling (min-cut) algorithm for a generalized assignment problem (transportation problem) [Gleyzal, 1955]. The same problem later was addressed by Brown and Von Neumann who provided a solution for the general assignment problem by finding optimal strategies in a zero-sum Bi-Matrix game, where the algorithm is stated to run in polynomial time [Brown and Von Neumann, 1950]. Faster algorithms have been published later in the late 1960s, where Dinic and Kronrod provided an $\mathcal{O}(n^3)$ run-time algorithm. Two years later, it was stated by Edmonds and Karp; Tomizawa, that the general assignment problem can be reduced to n single-shortest path computations, given a non-negatively weighted directed graph. However, also given arbitrary weighted graphs, different reductions are known to n single-shortest path computations, e.g. see [Ford Jr and Fulkerson, 1962; Hoffman and Markowitz, 1963; Desler and Hakimi, 1969]. For this algorithm, the computation time could be reduced to $\mathcal{O}(nm + n^2 \text{loglog} n)$, by using Fibonacci heaps and n sequential executions of Dijkstra's shortest path algorithm [Dijkstra et al., 1959]. In 2002, it was reported that this algorithm can be implemented faster in $\mathcal{O}(nm)$ time for given random integer weighted graphs, instead of floating point weights [Han, 2002; Thorup, 2004] and regardless of the maximum integer link weight [Andersson et al., 1998]. Another algorithm published by Gabow and Tarjan was improved by the same authors and provides a weight scaling algorithm for the assignment problem in $\mathcal{O}(m\sqrt{n}\log(nI_w))$ time, $\mathcal{O}(\log(nI_w))$ slower than the Hopcroft-Karp algorithm [Hopcroft and Karp, 1973; Karzanov, 1973] which

No.	Year	Complexity $\mathcal{O}(\cdot)$	Reference
1	1955	n^4, n^3	Kuhn
2	1962	$n\lvert f_{max}\rvert$	Ford Jr and Fulkerson
3	1965	nm^2	Edmonds
4	1965	nm^2	Witzgall and Zahn
5	1969	nm^2	Balinski
6	1969	n^3	Dinic and Kronrod
7	1971	$m\sqrt{n}$	Hopcroft and Karp
8	1973	$m\sqrt{n}$	Dinic; Karzanov
9	1976	n^3	Gabow
10	1976	n^3	Lawler
11	1976	$n^3 + mn\log n$	Karzanov
12	1978	$poly(n)$	Cunningham and Marsh
13	1980	$m\sqrt{n}$	Micali and Vazirani
14	1982	$mn\log n$	Galil et al.
15	1985	$mn^{3/4}\log n$	Gabow and Tarjan
16	1987	$n^2 - n^3$	Jonker and Volgenant
17	1989	$mn\log\log\log_d n + n^2\log n$	Gabow et al.
18	1990	$mn + n^2\log n$	Gabow
19	1991	$m\sqrt{n\alpha(n,m)\log n}\log(nN)$	Gabow and Tarjan
20	1991	$m\sqrt{n}/\kappa$	Feder and Motwani
21	1996	$m\sqrt{n^2 + n^{5/2}/\omega}$	Cheriyan et al.
22	1997	$m\sqrt{n}/\kappa$	Goldberg and Kennedy
23	2004	$m\sqrt{n}/\kappa$	Goldberg and Karzanov
24	2004	$m\sqrt{n}/\kappa$	Mucha and Sankowski
25	2006	n^ω	Harvey
26	2012	Nn^ω	Cygan et al.
27	2013	nm	Orlin
28	2014	$approx. \leq m\sqrt{n}$	Duan and Pettie

Table 3.1: Well-known perfect Maximum Weighted Bipartite Matching (MWBM) algorithms and their complexities [Duan and Pettie, 2014; Cook and Rohe, 1999].

solves maximum flow problems. A few years later, even faster algorithms could be developed for small and dense graphs [Cheriyan et al., 1996; Kao et al., 2001] or when fast solutions of a specific cost matrix size are desired [Ramshaw and Tarjan, 2012]. Interestingly, there is an algorithm that solves the MWBPM problem in $\mathcal{O}(I_w n^\omega)$ time, where I_w is the maximum integer link weight and ω is the exponent of square matrix multiplication. To summarize, for a general MWBPM problem, it can be assumed currently, that the solution time of the fastest *exact* solving algorithms is $\mathcal{O}(nm)$. However, faster solution times can be achieved by approximating the search space. The aforementioned publications can be accessed in Table 3.1. Note, that it is also possible to exactly solve MWBPM problems with MIP solvers, see Table 3.2. However, it is stated that algorithms such as the simplex method [Nelder and Mead, 1965] or interior point methods [Dantzig, 1955; Karmarkar, 1984; Nesterov and Nemirovskii, 1994; Boyd et al., 2004] practically scale at least in quadratic $\mathcal{O}(n^2)$ [Gudmundsson et al., 2007] up to super-cubic $\mathcal{O}(n^3\log n)$ time complexity for square matrices and equal numbers of node sets [Pele and Werman, 2009]. Thus, using the simplex method would not lead to a comparably

No.	Year	Method	Complexity $\mathcal{O}(\cdot)$	Reference	MIP-model
1	1965	Simplex	(n^2)-$(n^3 \log n)$	Nelder and Mead	√
2	1988	Branch and Cut	(n^2)-$(n^3 \log n)$	Padberg and Rinaldi	√

Table 3.2: MIP algorithms and MWBM time complexity.

scalable approach, especially if the current state-of-the-art algorithm has been stated to achieve a $\mathcal{O}(nm)$ time complexity [Orlin, 2013; Duan and Pettie, 2014].

3.3 Heuristics and Meta-Heuristic Algorithms

According to the literature research so far, the most efficient *exact* methods can solve the special instances of a maximum flow problem, called MWBPM in $\mathcal{O}(nm)$ time, for n row and m column entries. To yield faster algorithms, there is an additional possibility to leverage heuristics and other approximations for optimal transport problems [Duan and Pettie, 2014; Villani, 2008]. When looking at the Operations Research (OR) domain, apart from the already introduced methods, commonly Heuristic (HR) and Meta-Heuristic (MHR) have been used to yield efficient approximate solutions to large-scale problems. This is shown by Table 3.4, where the first well-known heuristic, i.e. the Clarke and Wright Savings heuristic was proposed [Clarke and Wright, 1964] in 1964. The heuristic is still integrated in modern solvers such as OR-Tools [*Google OR-Tools*, 2016], and can be used to approximately solve a variety of optimal transport or combinatorial optimization problems, e.g. Minimum Weighted Bipartite Matchings (MWBMs), Traveling Salesman Problems (TSPs), and Vehicle Routing Problems (VRPs). In 1966, Von Neumann, Burks, et al. proposed a Meta-Heuristic (MHR) which is an algorithm inspired by the natural selection of biological operators such as mutation, crossover, and selection [Mitchell, 1998]. Seven years later, Lin and Kernighan published the well-known Lin-Kernighan Heuristic (LKH) local search heuristic, which has achieved very good results for optimizing TSPs and VRPs [Lin and Kernighan, 1973]. For the same purpose, Christofides extended the work of Lin and Kernighan, but with the extension to asymmetric TSPs. They published the extended local search heuristic in the year 1976, where importantly some variants like guided local search are currently used in modern TSP solvers like OR-Tools [Voudouris and Tsang, 1999]. Due to the characteristic, that many problems in transportation are of stochastic and sequential decision-making nature, Powell and Sheffi proposed multiple approaches using Approximate Dynamic Programming (ADP), which similar to RL, makes use of the Bellman update equation to express a notion of systematic utility (of the whole transportation system), that has to be maximized or minimized [Bellman, 1966]. For this optimization process, the complete system state variables, the decision variables, random or stochastic process variables, a transition function, and the utility or target function to optimize must be known. Every solution to a multi-stage stochastic optimization process is represented by a rule-based form expression called policy [Sutton et al., 2000; Powell, 2007]. ADP furthermore uses basic principles of stochastic dynamic programming, such as *value iteration* and *policy iteration*. Both also are well-known in the RL domain [Sutton et al., 2000], where the approach uses efficient

generation of predictions for representing the parametric value function. The generation is done via generating multiple state space samples via Monte-Carlo (MC) simulation and updates of the predictions by stochastic gradient descent. It is commonly known that approximate dynamic programming enables to efficiently generate good solutions for a given optimization problem, however, usually with the solution being suboptimal or non-exact. One year later, in 1983, Kirkpatrick et al. published another method known as Simulated Annealing (SA), which similar to ADP, is a stochastic or probabilistic approximative technique for approximating the global optimum of a given cost function. The approach requires an objective cost function and leverages a slow cooling parameter, also called the temperature parameter, to control the initial search exploration and final optimum selection phase. Simulated Annealing (SA) is considered to be powerful for solving combinatorial optimization problems such as TSPs, VRPs, and MWBMs, when finding an approximate solution is more important than finding an exact local optimum within a given time duration. In the year 1989, Glover proposed another MHR method called Tabu-Search, for solving TSPs, VRPs, and MWBMs. Tabu-Search again is an iterative approach to approximate the solution of complex optimization problems. Furthermore, the approach is trajectory-based, in the sense that the algorithm keeps track of the previously visited trajectories during the search for the global optimum. Several strategies for adding solutions to the Tabu-list exist, where a well-known strategy is to add the complement of the currently followed solution by the Meta-Heuristic (MHR) [Glover, 1989].

Another quite different approach called Particle Swarm Optimization (PSO) is an MHR inspired by the biological behavior of swarms in order to look for an efficient solution. The algorithm was published in 1995, by Eberhart and Kennedy. The algorithm further imitates the behavior of animals to control particles that are employed to search for improved or exploit already achieved results. The search and exploitation strategy thus is naturally balanced, and the algorithm terminates after the algorithm stops improving for a given time. In combination with Q-Learning (i.e. a RL method) the algorithm has been applied to solve symmetric and asymmetric TSPs [Dorigo and Gambardella, 1997]. Interestingly, PSO has been reported to achieve high qualitative results, i.e. a Mean Absolute Percentage Error (MAPE) of $0 - 3.79\%$. In 1997, two interesting approaches to tackling TSPs were published by Mladenović and Hansen; Dorigo and Gambardella. The first method, called Variable Neighborhood Search (VNS) again belongs to the category of MHR algorithms and is based on local search-based (LCS) methods [Voudouris and Tsang, 1999]. The algorithm furthermore explores distant neighboring solutions of the current solution by deciding to transfer to the new solution if an improvement can be made by consequence. The method generally explores neighboring local optima, in order to achieve an approximate solution of a given discrete and continuous combinatorial optimization problem. Later in 2006 VNS has been extended to an even more powerful extension called Adaptive Large Neighborhood Search (ALNS) by Ropke and Pisinger. More recently, this approach has been optimized and applied to spatial customer distribution, time windows and dynamic demand applications by Schiffer and Walther, where the authors present the MHR under requirements of large-sized instances and real-world application ready circumstances [Schiffer et al., 2021]; for electric logistics fleets [Schiffer et al., 2018], robust location routing

[Schiffer and Walther, 2018b], and green logistics with intermediate stops [Schiffer et al., 2019], solved by an ALNS approach [Schiffer and Walther, 2018a].

Generally, this MHR uses an implicitly defined solution destroy and repair method and has shown interesting results for various transportation and scheduling problems, e.g. the TSP [Syed et al., 2019b] and VRP [Ropke and Pisinger, 2006]. Interestingly, ALNS in combination with Neural Network (NN) selection methods has shown promising results for MWBM for client-vehicle matching and Ride Hailing (RH) purposes. Since additionally, Machine Learning (ML) methods have shown promising results, i.e. when combined with traditional optimization algorithms [Syed et al., 2019b] , the next paragraphs will continue with past publications in the domain of Machine Learning (ML) for solving combinatorial optimization problems.

3.4 DRL Applications in ODM

In 2019 companies such as Uber, Didi Chuxing, Share Now have tried to leverage the qualities of efficient ML algorithms for large-scale fleet applications [Lin et al., 2018]. For this purpose, large-scale databases of data are aggregated to leverage ML-based methods via cloud-based high-computing platforms [Chuxing, 2019]. The overall challenge originates from the large number of vehicles and billions of customer requests that may occur daily. The goal thereby is to automate the management of given fleet operations and logistics, when given actual vehicle telematics in addition to maintenance tasks [Aljaafreh et al., 2011; Vujanović et al., 2012]. Such tasks furthermore include applications of autonomous pick-up, drop-off as well as charging and fueling strategies to ensure operational responsiveness and to increase fleet availability.

Interestingly, more and more stochastic approaches [Lin et al., 2018] have been tried where DRL is used for efficient strategy planning. Currently, the challenge is that only small numbers of zones can be managed. Each zone thereby is a discretization with respect to the client geolocations and an approximation of the original cost for trips from A to B. In practice, more complex planning problems cannot be solved efficiently, which leaves room for further improvements. Other applications further include vehicle supply and demand forecasting, the classification of suited pick-up and drop-off locations, the learning of accurate ETAs and to optimize the number of required vehicles and capacities for fleet management decisions and operations [Uber, 2019].

Moreover, the application of NN has already reduced operation costs of vehicle fleets, e.g. by increasing also the profitability through efficient demand prediction [Ke et al., 2017; Geng et al., 2019; Yao et al., 2018; Hauschild et al., 2013] for different cities world-wide. NN-based predictions play an important role for the implementation of efficient and adaptive pricing strategies [Hardt and Bogenberger, 2016; Hardt and Bogenberger, 2018; Schmöller et al., 2019], adaptive control strategies [Dandl et al., 2019; Dandl and Bogenberger, 2018b; Dandl et al., 2020] and other conditional service quality factors [Bilali et al., 2019c; Bilali et al., 2020][1]. Furthermore, service quality factors and parameters such as system response times heavily influence the perceived quality of a given mobility service [Bilali et al., 2019c]. In order to provide fast response times in a dynamic and stochastic city environment, very flexible and

[1] Hereby, the programming language python plays an important role in academia such as Technical University Munich, Stanford University, Massachusetts Institute of Technology, and Harvard University.

efficient optimization methods are required. The focus generally is set on the approximation of the environment states, decision states, and feature spaces [Powell et al., 1995]. Other authors rather focus on the efficiency of exact algorithms [Syed et al., 2019b; Syed et al., 2019a; Syed et al., 2019b; Engelhardt et al., 2020] or efficient meta-heuristics [Erdmann et al., 2019; Erdmann et al., 2020]. In summary, this illustrates the potential of efficient algorithms. Furthermore, this illustrates that efficient planning and decision-making control strategies are crucial, especially for economical but also environmental and human factors. In particular, well-designed algorithms play an important role to avoid congestion/pollution patterns, to save sustainability resources, and to reduce system energy requirements [Alonso-Mora et al., 2017; Schwarting et al., 2018; Engelhardt et al., 2019; Gurumurthy et al., 2019; Alonso-Mora et al., 2017].

3.5 DRL Algorithms

With recent advances in ML in planning and decision-making processes, techniques such as NN and RL have been successfully applied in real-world decision-making games (Atari, Go, Chess) [Mnih et al., 2015; Silver et al., 2016; Silver et al., 2017]. Such games range from 3D navigation through labyrinths, and physical system control to user interaction. In this domain, the success of NN originates from RL training which is a general-purpose decision-making algorithm for unknown environment tasks. By incorporating a reward signal as a feedback mechanism, algorithms that explore all possible and hopefully good actions can learn to solve such a sequential decision-making task. An RL learning algorithm interacts with the environment (e.g. map, route costs, vehicle positions), thereby iteratively learns its characteristics, and searches for better decisions and actions incrementally via episodes. By steadily improving its experience or utility, such an algorithm can achieve very competitive results as well as very fast decision times.

Since, Neural Networks (NNs) have been proven to be an universal function approximators [Hornik et al., 1989], the use of Embeddings [Cuturi, 2013] might be the key to yield efficient solution methods for complex combinatorial problems. While NNs have to be trained based on given labeled datasets, RL enables to also use unlabeled datasets for training. It is well known that the performance of NNs usually increases for large datasets, hence the generation of large unlabeled datasets may result in a challenge that has to be considered. For this reason, many researches in the fleet planning domain have rather focused on Deep Reinforcement Learning (DRL) instead on supervised-learning approaches with Deep Learning (DL) only.

The first time that a Deep Reinforcement Learning (DRL) method has shown a major breakthrough in real-world decision-making applications has been in 2013, where the work of Mnih et al. demonstrated the capabilities and potentials of this method within ATARI games. Previously, Deep Neural Networks (DNNs) also have outperformed classic approaches for application such as speech recognition, machine translation, image captioning by learning from data [LeCun, Bengio, et al., 1995, pp. 1–14]. Although DNNs usually are used to make *predictions* for regression and classification, Reinforcement Learning (RL) has enabled DL-based methods to make decisions by interacting with a provided game environment (e.g. when playing

ATARI games). The objective here is to efficiently learn the control strategy (called a policy) of one or more entities (e.g. called agents) by observing high-dimensional sensor input data. For instance, such data includes images, vision, sound, and speech which is the main challenge RL algorithms could provide new breakthroughs.

Since the real-world is not a game, wrong decisions can have very severe consequences for the environment, interacting humans, and interfacing systems. Thus, the ITS domain equally requires algorithms that also provide reliability, safety, and provide guarantees when solving real-world problems. For each wrong solution, a client may consequently be very dissatisfied or even hurt depending on the type of mobility system. Hence, algorithms as such have to prove their applicability to real-world problems with respect to reliability. Especially, additional monitoring means are required if ML-based algorithms have large-scale responsibilities and influences on the system stability. With respect to combinatorial optimization for TSPs, VRPs, and Minimum Weighted Matchings (MWMs) the history of OR is shown in Tables 3.3 and 3.5. All aforementioned algorithms have been part of past research. In 2013, one can observe that DRL techniques have become more and more popular, which majorly can be explained by the progress and availability of computational resources.

The attempts to apply NNs to combinatorial optimization problems (combinatorial decision making) dates back to the work of [Hopfield and Tank, 1985, pp. 1–12], where a Hopfield-Network was implemented for solving small TSP instances. However, due to the lack of computation power at this time, the benefit was not large enough to be successful in real-world examples. However, 30 years later, the development of powerful Graphics Processing Units (GPUs) and high-performance platform computing platforms has led to a revival of NNs. Tremendous advances in technology have enabled new possibilities and more efficient training methods, that have paved the way for DRL techniques. At the same time, and due to increasing digitalization, large amounts of datasets have become available open-source. New benefiting circumstances as such, have been the basis for modern research as follows.

3.5.1 DRL Approaches for TSPs

One of the first modern approaches to tackle the TSP (e.g. see Figure 3.1a) with an RNN solution has been proposed by [Vinyals et al., 2015, pp. 1–12] in 2015. For the first time, [Vinyals et al., 2015] have proposed an RNN-based Seq2Seq model-based [Sutskever et al., 2014, pp. 1–9] PN architecture [Vinyals et al., 2015, pp. 1–12] with "Attention Mechanism" [Vaswani et al., 2017, pp. 5998–6008] to overcome prior architectural limitations of fixed length input and output sequences. This is the so-called dictionary length or "Embedding Size", where the new architecture allows for variable input and output dictionary sequence lengths. This invention is fundamental for combinatorial optimization problems, where the length of the output sequences depends on the length of the prior input sequences. The whole architecture is called PN, and initially was trained via supervised learning (without RL). In more detail, the authors used a single layer Long-Term Short-Term Memory (LSTM) with 256 up to 512 hidden units and trained the NN with stochastic gradient descent. Furthermore, they used a learning rate of 1.0, while training 128 graph problems with a batch size of 128 in parallel. The initial network

Acronym	Explanation
HR	Heuristic
SL	Scheduling
ILP	Integer Linear Programming
NETs	Networks
NN	Neural Networks
DCO	Discrete Combinatorial Optimization
DRL	Deep Reinforcement Learning
TSP	Traveling Salesman Problem
RL	Reinforcement Learning
ADP	Approximate Dynamic Programming
DP	Dynamic Programming
AP	Assignment Problems
MWM	Maximum Weighted Matching
MWBM	Maximum Weighted Bipartite Matching
MWBPM	Maximum Weighted Bipartite Perfect Matching
MHR	Meta-Heuristic
SA	Simulated Annealing
VNS	Variable Neighborhood Search
PSO	Particle Swarm Optimization
ALNS	Adaptive Large Neighborhood Search
QL	Q-learning
PN	Pointer Network
GCN	Graph Convolutional Network
GCAN	Graph Convolutional Attention Network
CVRP	Capacitated Vehicle Routing Problem
ATTN	Attention
MCTS	Monte-Carlo Tree Search
mTSP	Multi-Traveling Salesman Problems
mVRP	Multi-Vehicle Routing Problems

Table 3.3: Acronyms for Table 3.5.

weights were initialized from -0.08 to 0.08, where additionally, ℓ_2-norm-gradient clipping with a magnitude of 2.0 was used to protect the network from divergence[2] during the training phase. Upon the test phase, [Vinyals et al., 2015] beam search was applied to filter invalid TSP tours. Moreover, 1 Million training examples (graph problems) were generated, where over-fitting the network was only observed for small training dataset sizes. Finally, tests were shown that the training generally converged after 10 up to 20 training epochs. The authors conducted more tests for 20-50 TSP nodes, where, compared to state-of-the-art algorithms, the Pointer Network could achieve close-to-optimal for larger problem sizes and optimal results for small problem sizes. The results of Vinyals et al. showed that for small TSP problem sizes with 5 nodes, the optimal tour with an optimal tour length of 2.12 could be found [Vinyals et al., 2015, p. 8]. When training and testing with 10 TSP nodes, the computed results by the PN showed a slight deviation, i.e. 2.88 from the optimal tour length 2.87. Computing the MAPE [De Myttenaere et al., 2016, pp. 1–33] estimate from this deviation would result in a MAPE of 0.34%. When training with 20 TSP nodes and testing only 10 TSP nodes, however, the optimal tour length of

[2] i.e. the exploding gradient phenomenon, e.g. see Chapter 4, Section 4.7 for further details;

No.	Year	Method	Task	Reference	Model-based
1	1964	HR	SL	Clarke and Wright	√
2	1966	Genetic MHR	ILP	Von Neumann, Burks, et al.	√
3	1973	HR	TSP	Lin and Kernighan	√
4	1976	HR	TSP	Christofides	√
5	1980	Local Search	TSP	Kanellakis and Papadimitriou	√
6	1982	ADP	MWBM/AP	Powell and Sheffi	√
7	1983	SA MHR	DCO	Kirkpatrick et al.	√
8	1989	Tabu-Search MHR	SL	Glover	√
9	1995	PSO MHR	TSP	Eberhart and Kennedy	√
10	1997	VNS MHR	TSP	Mladenović and Hansen	√
11	2006	ALNS MHR	TSP	Ropke and Pisinger	√

Table 3.4: Heuristic and approximate combinatorial optimization techniques.

No.	Year	Method	Task	Reference	Model-based
1	1958	NN	Classification	Rosenblatt	√
2	1981	RL	NETs	Sutton and Barto	√
3	2002	α, β Search	Chess	Campbell et al.	√
4	2013	Conv.DRL/QL	Atari	Mnih et al.	×
5	2015	Seq.DRL/PN	TSP	Vinyals et al.	×
6	2016	Seq.DRL/PN	TSP	Bello et al.	×
7	2016	Conv.DRL/MCTS/Minimax	Alpha Go	Silver et al.	×
8	2017	Conv.DRL/Struc2Vec	TSP	Khalil et al.	×
9	2017	Conv.DRL/GCN	QAP/MWM	Nowak et al.	×
10	2018	Seq.DRL/PN	TSP	Deudon et al.	×
11	2018	Conv.DRL/GCAN	TSP	Kool et al.	×
12	2018	Seq.DRL/RNN+Attn.	CVRP	Nazari et al.	×
13	2018	Conv.DRL/GCN+Pooling	mTSP	Kaempfer and Wolf	×
14	2018	Seq./Conv.DRL/SPG	TSP/MWM	Emami and Ranka	×
15	2019	Conv.DRL/GCN	TSP	Joshi et al.	×
16	2019	Conv.DRL/AC	CVRP	Malazgirt et al.	√

Table 3.5: Neural Network-based approximation techniques for combinatorial optimization.

2.87 could be found. For TSP node sets above 10, the computed results by the Pointer Network showed an increasing degradation in solution quality [Vinyals et al., 2015, p. 8]. For 20 trained TSP and 20 tested TSP the MAPE shows a magnitude of 1.3%. For 20 trained TSP nodes, but 30 tested TSP nodes, the MAPE rate increases to 2.6%, for 40 tested TSP nodes the MAPE rate increases to 13% and for 50 TSP nodes the MAPE rate increases to 32%. In other words, this shows that with their model, optimal TSP tours could be learned by the PN up to 20 TSP nodes [Vinyals et al., 2015, p. 8].

One year later, [Bello et al., 2016, pp. 1–15] showed an improvement when tackling the same 2D Euclidean TSP graph setup as shown by [Vinyals et al., 2015]. As a major difference, [Bello et al., 2016, pp. 1–15] showed that RL instead of supervised learning could be used to learn close-to-optimal routes up to even 100 TSP nodes. Bello et al. used an architecture called Actor-Critic with Attention Mechanism which comprised two RNN modules. Both RNN modules further included Encoder and Decoder modules which respectively, consisted of LSTM cells [Bello et al., 2016, p. 2]. In order to train the PN, Bello et al. proposed to use a Policy Gradient RL algorithm, since computing labeled data for supervised learning is impractical, especially for the training of large graph problem instances. In order to improve the learning stability, or to reduce the probability of exploding gradients, a critic module was used as a baseline function. This further enabled to reduce the variance of the computed network gradients. A baseline function was leveraged to compute an exponential moving average of the achieved network rewards over the training time. During the training phase, the critic network parameters were additionally optimized via stochastic gradient descent, using a mean squared error objective (loss) function between the Actor and Critic network. In order to compute multiple candidate solutions from the learned Actor policy[3], Bello et al. used Monte-Carlo Sampling and Active Search, which further led to improvements in the PN's solution quality. Subsequent benchmarks were conducted for Euclidean TSPs with 20, 50, and 100 TSP nodes per graph instance, where equally to the work of Vinyals et al., all node coordinates were uniformly and randomly sampled within unit square $[0, 1]^2$. During the tests of the actor network, 1000 sampled TSP graphs were used and were divided into 128 mini-batches. Further, the batches were sequentially processed by the actor network [Bello et al., 2016, p. 5]. The actor module consisted of LSTM 128 hidden units, that embedded two coordinates for each node in a 128-dimensional hidden unit space. The models were trained with the Adam optimizer [Kingma and Ba, 2014, pp. 1–15], where an initial learning rate of 10^{-3} was used for testing the 20 and 50 TSP node datasets. A smaller learning rate 10^{-4} was used for the larger graph problems, consisting of 100 TSP nodes. The learning rate was decayed every 5000 steps by a factor of 0.96, where the network parameters were initialized uniformly and random within $[-0.08, 0.08]$ [Bello et al., 2016, p. 8]. In particular, the gradients were clipped to 1.0 using the ℓ_2-norm. Both authors, Vinyals et al. and Bello et al. used one attention glimpse, that allowed the attention mechanism to update or rather focus on only one specific input (node assignment) at a decoding time step. Compared to the work of Vinyals et al., Bello et al. introduced an extra model depth in the decoder module, where an additional attention glimpse at the Embeddings was used. During the learning phase, the authors used a mask function for masking the TSP nodes already visited.

During the networks' search for optimal TSP solutions, the training mini-batches consisted of a subset of TSP graphs with replications of the test data set sequences and its permutations. A baseline decay α was used and was set to 0.99. Whereas a validation set of 10000 randomly generated TSP graph instances was used for hyper-parameter tuning. The Critic module consisted of an encoder network, which was architecturally similar to the policy network, but with the difference of having exactly 3 processing steps and 2 fully connected layers. The Logit values of

[3] e.g. see Chapter 4, Section 4.9

the actor network were clipped to magnitudes of $[-10, 10]$ but also a $tanh(\cdot)$ activation function has been applied[4], which helped to balance exploration and exploitation and resulted in slight performance gains during the MC search for good TSP solutions. Bello et al. furthermore tested a simple greedy decoding search strategy during the decoding steps, that was needed to find the largest probability of a node assignment in order to complete a whole TSP tour. Their results showed, that the combination of RL pretraining in combination with Active Search yielded the best performance [Bello et al., 2016, p. 8] compared to past attempts [Vinyals et al., 2015]. They noted, that for each test instance, an Active Search method was called for 10.000 training steps, while sampling and applying batch sizes of 128 TSP graphs, out of 1,280,000 TSP candidate solutions in parallel [Bello et al., 2016, p. 9]. The Actor-Critic PN approach was compared against 3 different increasingly qualitative and complex baselines, such as i) the Christofides algorithm [Christofides, 1976], ii) Heuristic-based algorithms as implemented in the OR-Tools [*Google OR-Tools,* 2016][5] Vehicle Routing Solver, which includes simple local search operators as 2-opt [Johnson, 1990], a version of the LKH [Lin and Kernighan, 1973], simulated annealing [Kirkpatrick et al., 1983; Reinelt, 1994], Tabu-Search [Glover et al., 2018] or Guided Local Search [Voudouris and Tsang, 1999] and iii) optimal solutions via the Concorde Solver [Applegate et al., 2006a]. They deployed their DRL method on a single Nvidia Tesla K80 GPU, whereas other heuristics with Or-Tools were executed on an Intel Haswell CPU. Further, Concorde and the LKH were executed on an Intel Xeon CPU E5-1650 v3 3.50GHz CPU. The learning time of the PN approach with Active Search was stated to require a considerable amount of time, i.e. 7.25 hours per experiment with 50 and 100 TSP nodes. Note, that for this benchmark various implementations in Pytorch [Paszke et al., 2016][6] and Tensorflow [Abadi et al., 2016][7] can be found [Github, 2020a; Github, 2020b].

Also note, that in the work of Bello et al., a comparison between the work of Vinyals et al. is included. It is reported that training the PN with RL significantly improved the solution quality over the previously published supervised learning approach by Vinyals et al. and that all proposed methods generally surpassed the tested Christofides heuristic [Christofides, 1976]. With respect to solution quality, the PN further is stated to be slightly less competitive compared to Tabu-Search and even much less than Guided Local Search. For the training and testing datasets that are characterized by 20 TSP the PN approach of Bello et al. yields the optimal tour length computed by the Concorde solver. For 50 trained and tested TSP nodes the results showed a slight degradation yielding a MAPE of 0.35%. The optimal TSP tour length was calculated to be 5.68 whereas the achieved TSP tour length was 5.70 by the proposed PN approach. For 100 trained and tested TSP nodes the MAPE further increased to 0.77, achieving a tour length of 7.83 by the PN and an optimal tour length of 7.77 by the Concorde solver. Finally, this illustrated that the proposed PN could solve the TSP with a MAPE of less than 1% which is an accuracy above 99%. The proposed PN was compared with Concorde, which is stated to be the most accurate TSP solver currently known [Applegate et al., 2006a].

[4] e.g. see Chapter 4, Section 4.8
[5] https://developers.google.com/optimization/routing
[6] https://pytorch.org/
[7] https://www.tensorflow.org/

In 2017, other authors proposed a different model based on Convolutional Neural Networks (CNNs) [Khalil et al., 2017]. Instead of using separate Encoder and Decoder modules based on the Seq2Seq model, Khalil et al. proposed a non-autoregressive single model solution with Struc2Vec Embeddings. The term regressive refers to the model characteristic of conditioning the network output to the qualitative partial solutions of the TSP tour. An autoregressive model is merely a feed-forward NN that predicts future values by incorporating past values, while non-autoregressive models process every prediction from current values (ignoring past computations) and produce its outputs in parallel. Such a model was trained with a 1-step Deep Q-learning Network (DQN) [Mnih et al., 2015], while RL and helper functions were required to compose the final TSP tour from the network. Hereby, the networks' output was the order in which partial TSP nodes have to be visited. Furthermore, individual partial solutions were composed to the final best possible ordering location. As an objective function, Khalil et al. used the negative of the reward, which led to the behavior that the network inserted the farthest nodes at first. This was later stated to be an effective heuristic for solving TSP problems in general [Rosenkrantz et al., 1977; Kool et al., 2018].

Other non-autoregressive approaches were proposed by [Nowak et al., 2017a], where a Graph Neural Network (GNN) was trained via Supervised Learning to learn the optimal adjacency matrix (i.e. the decision matrix $\mathbf{X}$), which entails the information of which nodes are connected within a TSP tour. The GNN response was converted into a feasible solution using Beam Search [Medress et al., 1977]. As a result, Nowak et al. reported an optimality gap of 2.7% for 20 TSP nodes, which was slightly inferior than the regressive approaches by [Bello et al., 2016, p. 8].

At the same time, Shimomura and Takashima proposed an MCTS algorithm for yielding efficient solutions to the TSP, whereas in this scheme another MCTS method was developed by [Fu et al., 2019]. Based on this work, Emami and Ranka implemented an Actor-Critic NN, and chose a different approach with SPG to learn the policy by approximating a double stochastic matrix.

In 2018, [Deudon et al., 2018] similar to [Kool et al., 2018] presented an auto-regressive Graph Multi-Head Attention Network (GMHAN) approach for solving the TSP. Deudon et al. showed a performance increase when using 2-OPT local search within the Policy Gradient training algorithm of used for training the PN. The results showed that the optimal solution could be obtained for 20 TSP nodes, whereas for 50 nodes their model achieved a total tour length of 5.77 with respect to an optimal tour length of 5.68. This resulted in a MAPE of 1.6%. For 100 TSP nodes, the GMHAN obtained a total tour length of 8.16, which compared with the optimal solution of 7.77, yielded a MAPE of 5%.

On the other hand, the GMHAN approach of Kool et al. was trained using the RL algorithm called "REINFORCE" [Williams, 1992]. The algorithm is a Policy Gradient-based technique, which iteratively was used to improve the quality of the learned and obtained TSP solutions during training. The authors showed results for 20 TSP nodes, where the Attention Mechanism (AM) yielded a MAPE of 0.3% by achieving the same tour length as OR with 3.85, compared with an optimal tour length of 3.84, calculated by Concorde. For 50 nodes, the AM achieved a MAPE of 1.8% and for 100 TSP nodes a MAPE of 4.5% was achieved. The individual solution times for

20, 50, and 100 TSP nodes were reported to be 0.1, 2 and 5 seconds. Kool et al. also showed solutions of their model to other problems such as the CVRP, Split Delivery Vehicle Routing Problem (SDVRP), Orienteering Problem (OP), Price Collecting TSP (PCTSP), and Stochastic Prize Collecting TSP (SPCTSP).

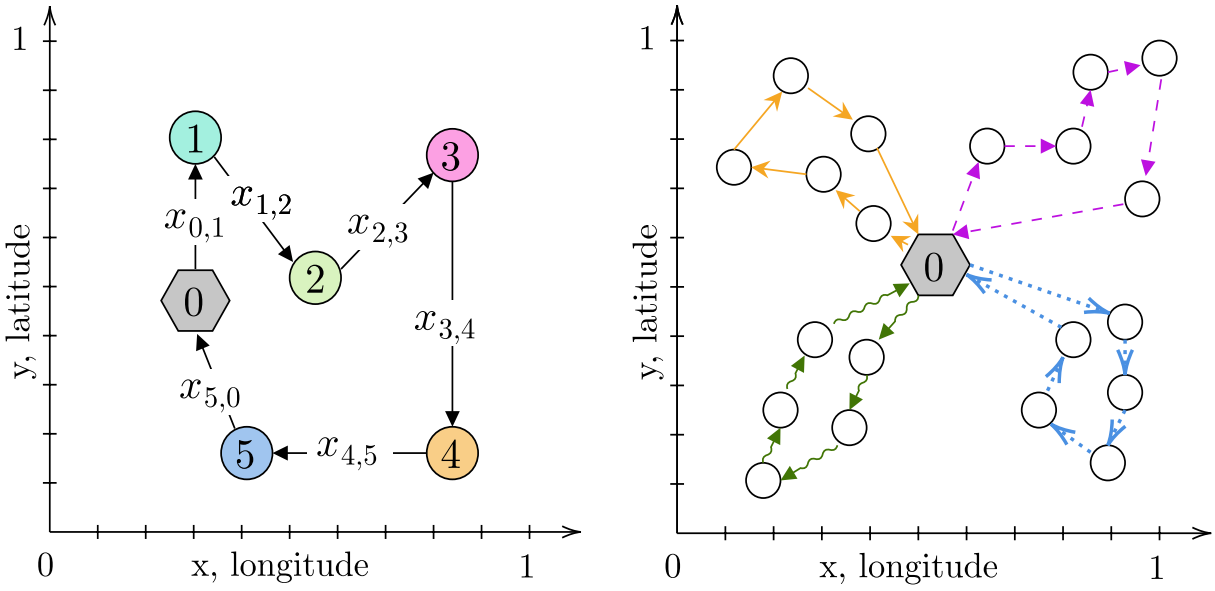
Other work by [Joshi et al., 2019] provided a non-autoregressive approach where they trained a GCN. The approach is similar to [Emami and Ranka, 2018] in the sense, that the optimal output of an adjacency matrix was targeted for the TSP with 20, 50 and 100 nodes. Similar to Nowak et al., Joshi et al. employed Beam Search to compose the finally obtained TSP tours. As a result, the authors reported that the average MAPE could be reduced from 0.52% to 0.01%, and 2.26% to 1.39% for 50, and 100 TSP nodes.

To summarize, all aforementioned models can be classified as model-free approaches, where the trained model does not require any stochastic transition model matrix $\mathbf{T}$. But also modern online model-based approaches can be found, such as by [Malazgirt et al., 2019, pp. 50–51]. Note, that the term on-line refers to the train-ability in repeated short amount of times. This however may result in a quality-scalability trade-off with respect to the achievable solution quality of many TSP nodes. While model-free approaches (e.g. Bello et al.) do not depend on a stochastic transition model matrix $\mathbf{T}$, Malazgirt et al. showed that the training time could be reduced efficiently by leveraging a transition model. Their results also provided insights into the final solution quality. As a result, their model provided total tour lengths of 2.88, 3.91, 6.37 for 10, 20, and 50 TSP nodes, where the optimal tour lengths were 2.87, 3.82, and 5.68. The resulting MAPE magnitudes can be self-calculated to 0.34%, 2.35%, and 12% deviations from the optimal tour length.

3.5.2 DRL Approaches for VRPs

In 2018, Nazari et al. were known to be the first authors to tackle VRP with a DRL approach. Briefly, VRPs are multiple extensions of TSPs from the depot node, visualized in Figures 3.1a and 3.1b, and thus are considered to be even more complex to solve. Equal to the TSP the VRP belongs to the class of combinatorial optimization problems, where a decision component must calculate the optimal round-trip routes for multiple vehicles. Here, each vehicle route may consist of several stops or pickup locations for clients or station locations. Essentially, this requires calculating multiple dependent TSPs which all have a common start at the depot node. While targeting to solve VRPs, Nazari et al. extended the approach of Bello et al. by replacing the LSTM encoder of the PN by element-wise projections, such that the updated Embeddings can be efficiently computed during state changes. Nazari et al. apply their architecture to the VRP while also considering split deliveries and stochastic variants, such as dynamic demands and supply modeling, called CVRP. The stochastic graph features as demands and supplies could be incorporated via dynamic tensor elements, whereas static features were employed to represent static VRP elements such as present node coordinates. Each input element was composed of a stacked input tensor which incorporated the static and dynamic tensor elements via $x_t^i := (s^i, d_t^i),\ t = 0, 1, ..., T$, for each input element i at each encoding-decoding time step t.

The authors argued that the RNN encoder of the approach of Bello et al. added unnecessary



(a) A general visualization of an TSP with depot node.

(b) A general visualization of an VRP with depot node.

Figure 3.1: Extension visualization from TSP graphs to VRP graphs.

complication to the encoder. Whereas by omitting the encoder, the approach could be made less complex and more general. They mentioned crucial limitations of the approach of Bello et al., where dynamic elements could not be updated concurrently while computing the updated probabilities for the next decision step. To prevent such limitations, Nazari et al. proposed a model where an embedding layer between the input and Attention Mechanism mapped the inputs to high-dimensional vector space [Nazari et al., 2018, 5, Figure 2]. The RNN encoder further stored the information on the decoded sequence, where the RNN hidden state and embedded input were used to compute the conditional probability distribution over the next input using an Attention mechanism. They reasoned this with the usual characteristic that combinatorial optimization would not provide any meaningful order within the input data [Nazari et al., 2018, p. 5]. As an example, the VRP inputs usually are sets of unordered customer locations with individual demands. Moreover, the order of the input does not matter to the network. With respect to architectural design, the model of Nazari et al. consisted of two main components, namely the embedding mechanism, which mapped the inputs to D-dimensional space to enable the input processing for the AM. The second component was the common practice to use RNNs to model the Encoder and Decoder networks [Bahdanau et al., 2014; Cho et al., 2014; Sutskever et al., 2014]. The authors reported that only the static tensor elements were used as input to the decoder, whereas the dynamic elements did not yield any significant improvements when they were being used as input. Instead, the dynamic elements were only used in the Attention Mechanism of the overall network [Nazari et al., 2018, 5, Section 3.2]. With respect to the results, Nazari et al. additionally employed a set of heuristics and OR for competitive baselines. Finally, they reported that their approach could find optimal solutions for the VRP with 10 and

20 nodes, while they used a MIP formulation [Toth and Vigo, 2002] for comparisons. The authors uploaded their work online where also their tensor flow implementation can be found[8].

As a final note, it is also worth mentioning that the recently published Transformer architecture [Vaswani et al., 2017; Kaempfer and Wolf, 2018] has been trained while targeting the mTSP or VRP. The result can be interpreted as the linear relaxation of the problem, where Kaempfer and Wolf have used beam search to obtain the final feasible TSP tour [Kool et al., 2018]. Unfortunately, no details on the accuracy of their model are shown. Also note, that NN approaches have been used to advance modern meta-heuristics ALNS [Syed et al., 2019b; Hottung and Tierney, 2019], which also have shown competitive results to the approaches aforementioned.

3.6 Summary and Research Gap

In this chapter, most related literature was analyzed to summarize the state-of-the-art within MIP and DRL for combinatorial optimization. The next Subsection 3.6.1 briefly summarizes the insights gathered from exact and heuristic combinatorial optimization algorithms with a focus on achievable run-time complexities. Secondly, Subsection 3.6.2 summarizes the insights from DRL approaches for the same purpose. Finally, the summary finishes with Subsection 3.6.3 which explains the gap of research and further defines the course of research within this thesis.

3.6.1 Summary of Exact and Heuristic Methods

The literature research of past combinatorial optimization has shown that many authors have shown very remarkable results, using a variety of very different approaches. Comparably simple approaches are dated back to the 60s, when the first heuristics were used to approximate complex Combinatorial Optimization Problems (COPs) such as TSPs, and VRPs. Whereas, for PDP and MWBMs naive greedy heuristics ($\mathcal{O}(n)$ or $\mathcal{O}(m)$ time complexity) have been a widely used practice, and are even used within some RH companies today. The reason for using such simple methods mainly originates from user-centric design, scalability and maintainability reasons. However, heuristics do not reach full optimization potential, particularly when dealing with fleet planning and scheduling applications. Heuristics furthermore are known to provide very fast solution times, however, also are stated to require tedious development and testing efforts (costs) in order to work well for many different problem scenarios.

Thus, many authors proposed MIP formulations, which can be exactly solved by well-known MIP assignment, simplex, CP-SAT, ILP, Auction Algorithms, and Max-Flow-Min-Cut (Hungarian Method) solvers, see Figure 3.2.

[8] accessed 07/30/2020: https://github.com/OptMLGroup/VRP-RL

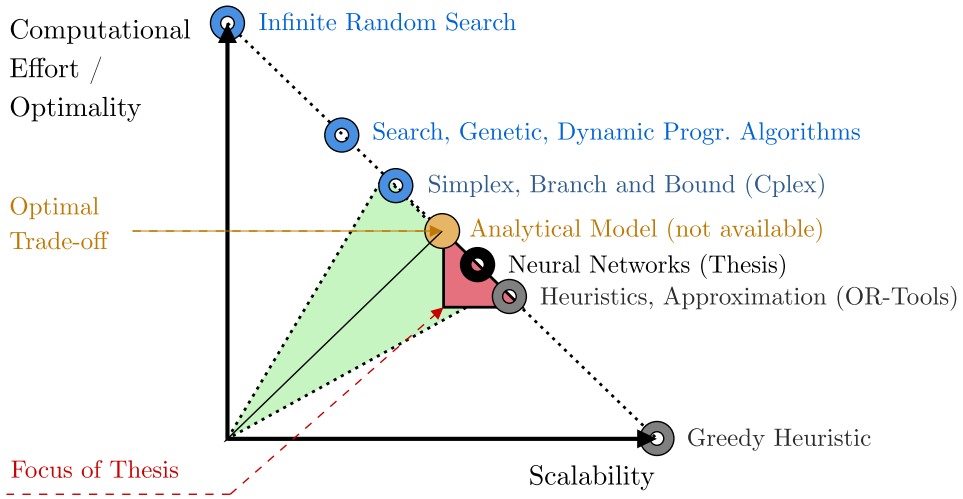


Figure 3.2: Computational Effort vs. Scalability by different MIP, Linear Sum Assignment Program (LSAP) and MWBM Methods for fleet planning problems.

On the one hand, past authors have shown optimal results and large efficiency improvements. For instance, with respect to MWBPMs, modern research has stated to achieve exact-optimal solutions in quadratic $\mathcal{O}(nm)$ (best-case) time complexity up to polynomial, e.g. $\mathcal{O}(n^3 \log\, n)$ (worst-case) time complexity. Faster efficiencies only were achieved using qualitatively suited approximations. In this case, meta-heuristics finally have become more and more popular, and have shown remarkable results with respect to a trade-off for solution efficiency and optimality.

3.6.2 Summary of DRL Methods

All presented approaches focus on the solution of large quantities of logistic requests for shortest-path problems like TSPs and VRPs. Known approaches further leverage DRL-based autoregressive and non-autoregressive approaches to efficiently solve such combinatorial problems. It can be noted that the combinatorial complexity highly depends on the problem formulation and the used features for definition [James and Lam, 2017; Lee et al., 2012; Jaillet et al., 2016, pp. 1, 1, 3]. Past surveys also showed that common Traveling Salesman Problems (TSPs), Vehicle Routing Problems (VRPs) and Minimum Weighted Matching Problems (MWMPs) were mainly solved with ILP or MIP solvers which achieved polynomial run-times. Using DRL past authors have focused on solving TSPs and VRPs via artificially generated datasets. By solving artificial Euclidean symmetric TSPs the potential of using regressive approaches was shown to achieve optimal for small TSPs below 20 nodes and near-optimal results for larger TSPs for 20-100 nodes [Bello et al., 2016]. In general, "Encoder-Decoder"-based architectures with policy gradient optimization were employed and showed remarkable results regarding solution quality while providing efficient solutions to combinatorial complex optimization problems. Secondly, an extended architecture was used to optimally solve VRPs with static and dynamic problem features, e.g. vehicular capacities [Nazari et al., 2018]. Unfortunately, Nazari et al. still used artificial data for comparisons, which makes it difficult to assess how similar approaches might

perform in real-world applications. Since the VRP is closely related to a PDP, it might be possible to develop an architecture that could solve PDPs and MWMs efficiently. However, with respect to this combinatorial problem, it is additionally unknown which problem properties could be solved and which may fail.

3.6.3 Research Gap

As aforementioned, some past authors have used DRL methods to optimally solve TSPs and VRPs. Conversely, asymmetric, unbalanced, and constraint MWMs, PDPs, and MWBM have not been solved optimally with Neural Networks at this point. Also past authors mainly have used artificial data, which commonly leaves the consequences of using real data currently unknown. Generally, the problem structure of an PDP is similar to an VRP and by omitting the routes to the depot, both problems can be transformed to MWBMs. However, by doing so, multiple problems arise. First, it is unclear, if an approach as proposed by Nazari et al. could handle symmetric or asymmetric distances (ubiquitous in real road networks). Second, it is unclear if such "Encoder-Decoder" architecture could learn to solve unbalanced problems with different sequence lengths in the input data. All past approaches have assumed balanced graph samples with the same amount of nodes for training and testing. This assumption however is comparably unrealistic for real-world fleet planning problems. Alongside this challenge, another fact occurs that client request node quantities cannot be controlled without insertion of dummy nodes. In this case, if dummy nodes would be inserted, another question arises, if DRL approaches could yield competitive solution qualities and learning reliability. Finally, it is unclear how DRL approaches might behave with the existence of problem constraints. Past approaches often have focused on reaching optimal results for equal problems, rather than optimal results for general and variable problems. Thus, this thesis will focus on such aspects to provide a complementary view to past and modern literature.

4. Deep Reinforcement Learning

4.1 Introduction to Machine Learning

This chapter provides fundamental introductions to Machine Learning (ML), Reinforcement Learning (RL) and Deep Learning (DL) methods. The following sections 4.1-4.9 provide a focus on the subfield of Machine Learning called Deep Reinforcement Learning (DRL) where the main components of Reinforcement Learning and Deep Learning are combined. The sections also aim to provide information on why and how DRL has a profound influence on data-driven optimization problems in operations research and similar domains.

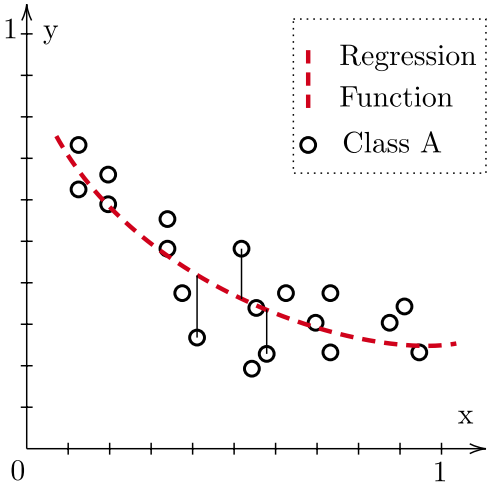


Figure 4.1: Regression.

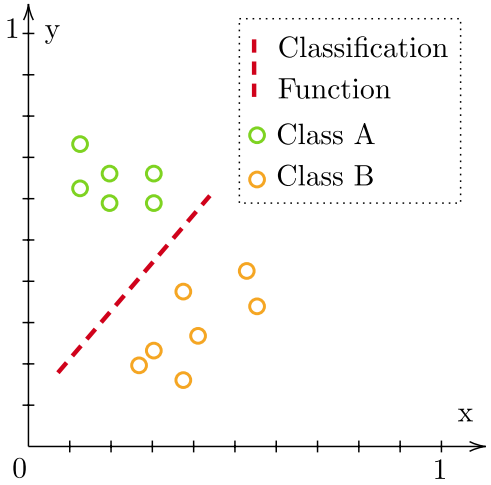


Figure 4.2: Classification.

Generally, ML is a subfield of AI that relates to teaching computers to learn from data examples. Further, ML incorporates multiple technologies where machines are enabled to perceive, recognize, understand, and decide. Such "intelligent" machines can learn from data samples, knowledge bases, and experience from reward mechanisms by interacting with an environment.

Generally, it is distinguished into three major categories as follows,

- Supervised Learning, where the learner (learning program) also called "the agent" learns to solve classification or regression tasks from labeled training and validation datasets.
- Unsupervised Learning, where the agent has to learn tasks such as clustering, anomaly detection or associations for given datasets without any pre-provided labeling data samples from the programmer.

- Reinforcement Learning (RL) which concerns with the learning of optimal actions given an environment structure. The learning machine or so-called "agent" tries to learn the correct order of actions in order to maximize its cumulative reward (given a reward function) or to reduce its cumulative collected penalties given a cost function.

Due to recent improvements in Deep Learning (DL), the aforementioned three major categories have seen major improvements, especially working with big and high-dimensional data such as time series, images, and videos. The success of Deep Learning originates from recent technological improvements: (i) the exponential increase of available computing power, also by enhancing GPUs for matrix operations, (ii) scientific breakthroughs in Deep Learning methods [Srivastava et al., 2014; Ioffe and Szegedy, 2015; He et al., 2016; Szegedy et al., 2017; Klambauer et al., 2017] and (iii) the increasingly available open libraries, datasets, and support tools which simplify the training and testing processes.

4.2 Introduction to Deep Learning

Deep Learning (DL) models are initially inspired and algorithmically adapted from the biological neural processes of the human brain. Recent scientific advances in neural biology however have led to fundamental conceptual differences between Deep Learning and neural biology [Bengio et al., 2015].

Generally, Convolutional Neural Networks (CNNs) are well known for their application for computer vision and Recurrent Neural Networks (RNNs) for the application of NLP and sequence prediction. Both neural network architectures incorporate the same fundamental components which are called neurons or also "Perceptron". Initially, invented by McCulloch and Pitts [Jain et al., 1996; Rosenblatt, 1958] the fundamental components of a NN architecture are described in Figure 4.3. Overall, the basic set of components can be described as the input state vector

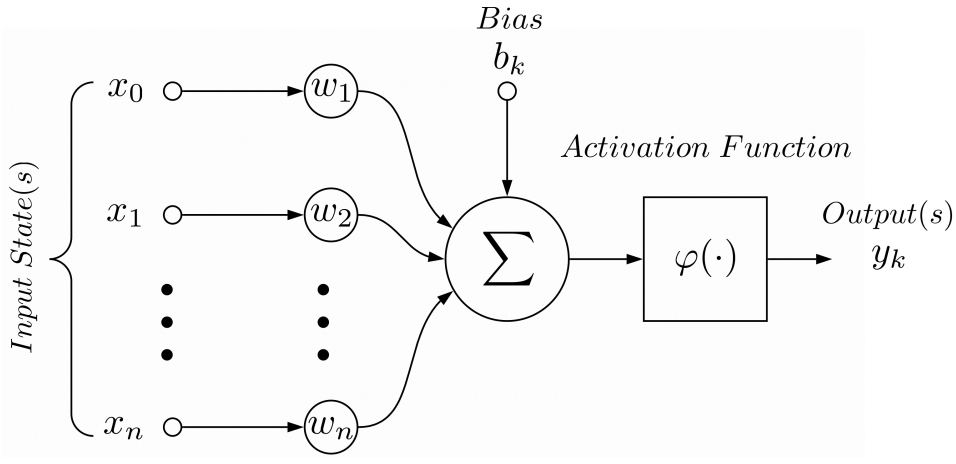


Figure 4.3: Basic Perceptron Model [Rosenblatt, 1958].

$\boldsymbol{x}_n$ (sometimes referred to as input layer), a summed junction Σ, which computes the weighted sum among all input states, the non-linear differentiable activation function $\varphi(\cdot)$ which serves as

non-linear threshold by limiting the amplitude of the Perceptron's output value and mapping it to a finite interval of values. The bias $\boldsymbol{b}_k$ is an offset value applied for increasing and decreasing the net input of the activation function, which affects the output by means of an affine transformation. Each NN introduces a set of learnable parameters which are optimized and tweaked to find the best function approximation between the input and output states. In the presented NN model, the learnable parameters θ are a set of weights $\boldsymbol{w}_n$ that are represented as links in Figure 4.3. The individual weights are multiplied with an individual input state $\boldsymbol{x}_n$, summed with all corresponding bias terms $\boldsymbol{b}_k$ and the linear weighted sum is fed to the activation function (element-wise non-linearity) as described in the following equation

$$\boldsymbol{y}_k = \varphi(\sum_{j=1}^{m} \boldsymbol{W}_{nj}\boldsymbol{x}_{nj} + \boldsymbol{b}_{kj}) \,. \tag{4.1}$$

In its most abstract description, a neural network aims to learn to model a function $f(\boldsymbol{x}, \boldsymbol{\theta})$ that represents an optimal mapping of an input vector or matrix $\boldsymbol{x}$ to some output signal $\boldsymbol{y}$ and formally denoted by $f(\boldsymbol{x}, \boldsymbol{\theta}) : \boldsymbol{x} \rightarrow \boldsymbol{y}$.

4.3 Feed-forward Neural Networks (NNs)

The first feed-forward Neural Network (NN) models with a single hidden layer were introduced in 1985 by Rumelhart et al. The first NN model, as visualized in Figure 4.4, was composed by the input layer $\boldsymbol{x}$ as state vector (a row vector with n elements), $\boldsymbol{W}_1$ the *weight matrix* containing all weights and $\boldsymbol{b}$ the *bias, offset or translation* which transforms the weighted sum via affine transformation $\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}$ [Rumelhart et al., 1985]. Generally, there are many applicable activation functions $\varphi(\cdot)$. The most common functions used are functions such as the Sigmoid (Sigm), the Tangens Hyperbolicus (TanH) or the Rectified Linear Unit (ReLU)[1], as shown in Figure 4.5. The non-linear activation functions are applied to the affine transformation output, which is called the *hidden layer* (deterministic function of the input). In the *hidden layer*, the network elements are referred to as hidden units or hidden neurons [Murphy, 2012; Bishop, 2006, pp. 564, 226]. The composition is again transformed via a second affine transformation including the weight matrix $\boldsymbol{W}_2$ which maps the hidden layer values to a finite interval, which is referred to as model output layer $\boldsymbol{y}_k$. The model output $\boldsymbol{y}_k$ hereby is a row vector with k elements. Thus the neural network model contains two *inner-product layers* where $\boldsymbol{W}_1$ is $n \times k$ and $\boldsymbol{W}_2$ is $k \times d$ dimensional. Finally, b and the network output y are k dimensional which can be summarized by

$$\bar{y} = \varphi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})\boldsymbol{W}_2 \,. \tag{4.2}$$

Note, that different activation functions mainly influence the gradient calculation during back-propagation. The gradient determines to which degree the weights are changed or updated in the learning process [Gal and Ghahramani, 2016, p. 3].

[1] $relu(x) = max(x, 0)$

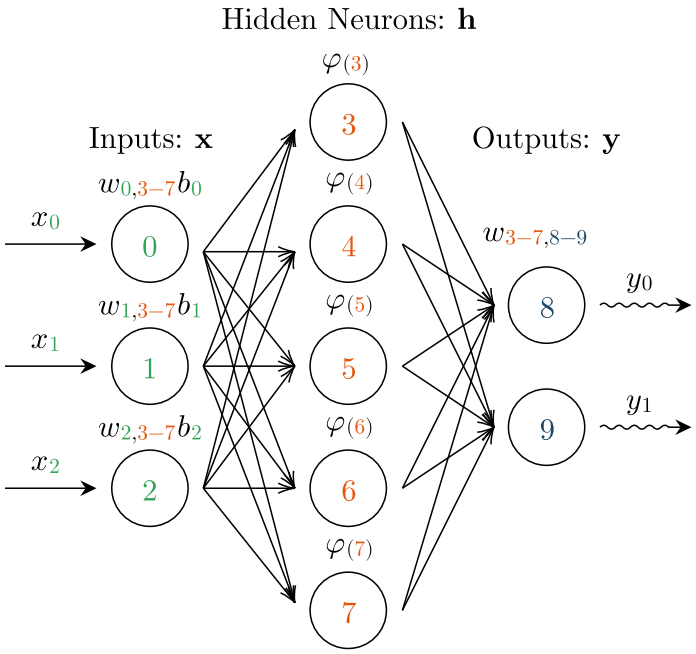


Figure 4.4: An example feed-forward neural network model with two fully-connected layers [Rumelhart et al., 1985].

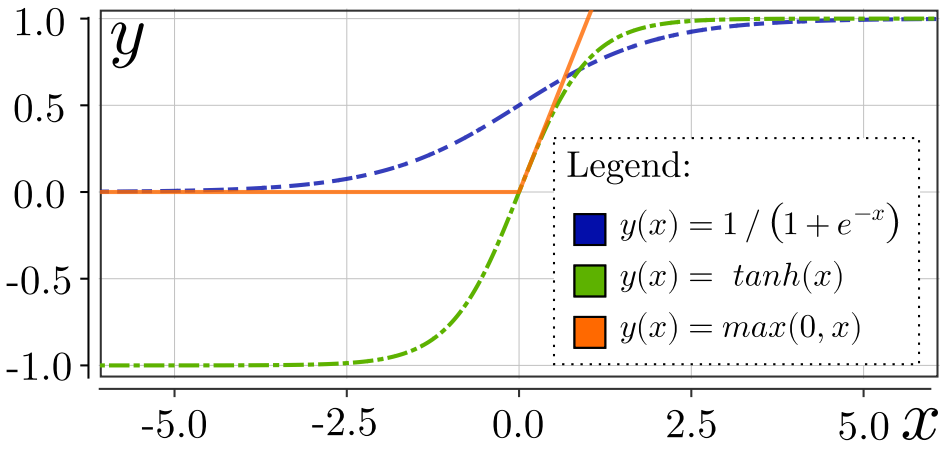


Figure 4.5: Three examples of activation functions $\varphi(x)$: (i, blue) Sigmoid, (ii, green) Tangens Hyperbolicus and (iii, orange) ReLU.

Back in 1989, it was first proved that a multi-layered neural network using Sigmoid activation functions can serve as an *Universal Function Approximator* [Cybenko, 1989, pp. 303–314]. During the first application in 1991, it was shown by [Hornik, 1991] that the topology of a multi-layered neural network is rather crucial than using different activation functions. This is meant in a deeper context, stating that a multi-layered neural network can approximate any smooth function to any desired accuracy, if enough hidden units are given [Murphy, 2012, p. 564]. For more information, the reader may see the additional literature [Bishop et al., 1995;

Bishop, 2006, pp. 16–138, 225–284].

In order to use a multi-layered neural network for statistical regression or classification applications, the learning process requires a quality metric that describes how close the approximation is learned compared to the original inputs. Given a neural network regression task, a simple example solution would be to minimize the distance between the line and all original data points during the regression. Therefore, a simple example metric function might be the Euclidean loss function

$$E^{\boldsymbol{W}_1,\boldsymbol{W}_2}(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{2N}\sum_{i=1}^{N} \parallel \boldsymbol{y}_i(\boldsymbol{x}_n) - \bar{\boldsymbol{y}}_i \parallel^2 \tag{4.3}$$

whereby the original data points are denoted by $\boldsymbol{y}_1, ..., \boldsymbol{y}_n$ and the predicted model outputs are denoted by $\bar{\boldsymbol{y}}_1, ..., \bar{\boldsymbol{y}}_n$, corresponding to the observed inputs $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$. Further, the parameters $\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{b}$ are learned such that, by learning the adaptation of weights $\boldsymbol{W}_1, \boldsymbol{W}_2$, and bias $\boldsymbol{b}$, the Euclidean error in Equation (4.3) is minimized. The loss function hereby estimates the error between the actual correct solution and the network's predicted solution. By learning the optimal parameters $\boldsymbol{\theta}$ and $\boldsymbol{W}$ the training error or loss are minimized. To do so, all learnable parameters within all layers have to be trained and updated. This is called back-propagation [Rumelhart et al., 1986, pp. 533–536], where most commonly gradient descent methods, but also evolutionary strategies can be used for updating the learnable parameters within the NN layers [Salimans et al., 2017, pp. 1–13]. By observing the network's prediction error on the validation dataset, it is possible to evaluate the network's generalization of transferability capabilities to unseen data. Similarly to the training error or training loss function, it is desired to minimize the validation error. If the validation error can be minimized, the network should perform well on test data as well and therefore generalize to an unseen test dataset $\boldsymbol{x}_{test}, \boldsymbol{y}_{test}$. In other words, the model has learned the parameters $\boldsymbol{\theta}$ and $\boldsymbol{W}$ from the training data $\boldsymbol{x}_{train}, \boldsymbol{y}_{train}$ which enables the network to successfully predict the outcomes of other test datasets $\boldsymbol{x}_{test}, \boldsymbol{y}_{test}$.

4.4 Convolutional Neural Networks (CNNs)

A well-known class of Neural Networks (NNs) are Convolutional Neural Networks (CNNs) [Rumelhart et al., 1985]. The present hidden layers are sequentially positioned and connected via operations such as a recursively executed convolution, pooled (max) layers, and inner product layers. Incorporating convolutional layers, CNNs are particularly suited for image processing, matrix filtering, and also sequential data applications [LeCun, Bengio, et al., 1995]. Using convolutional matrix operations, CNNs are capable of learning the classification and regression of data patterns. By using the linear transformation of the input information the model can preserve the input's spatial information. The max pooling layers subsequently take the convolutional output and apply the maximum of the image pixel blocks (kernels) which reduces its dimensionality [Gal and Ghahramani, 2016]. Figure 4.6 illustrates the individual terms such as Input Layer, Input Layer Height, Number of Channels, Channel Width, Kernel Width, and Kernel Height. The Output Layer consists of a number of kernels that compress the output

information. Overall, CNNs generally consist of feed-forward layers with fully-connected layers,

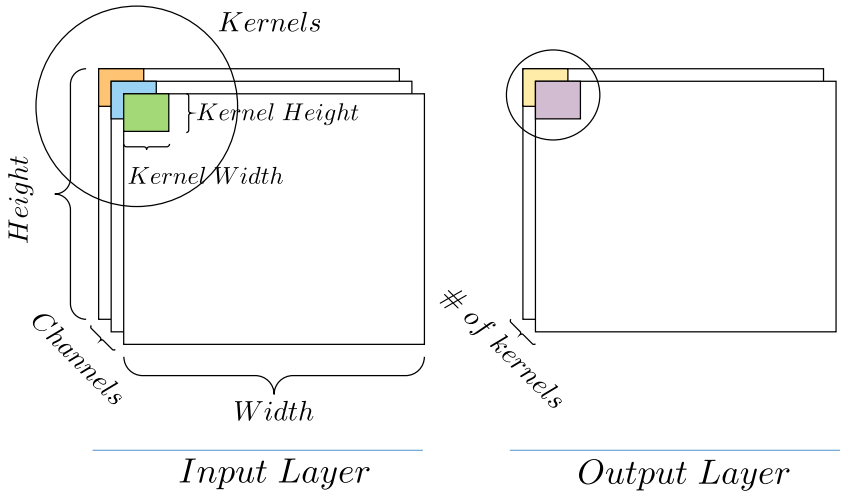


Figure 4.6: Basic CNN: For example, if the network captures an RGB image, then each kernel would capture the different blue, green, and red channel containing the colored pixels. Each kernel may be of different size, therefore capturing different locations and features of the image and serving as edge or feature detector. This can be seen as a simple convolution [Mairal et al., 2014, p. 4].

where the difference is that the weights may be set to zero, whereas other values may share the same values [LeCun, Bengio, et al., 1995]. The input features are stored as values, where concise input features are enforced and fuzzy features are filtered using the convolution operation, illustrated in Figure 4.7.

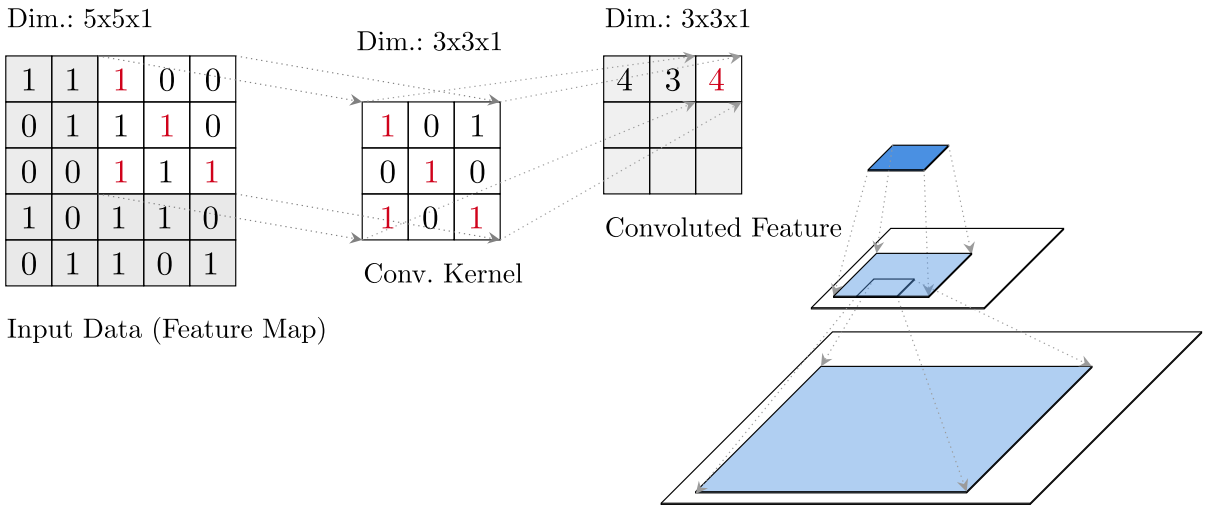


Figure 4.7: A convolutional layer with an input and output feature map for filtered feature extraction [François-Lavet et al., 2016, p. 12].

4.5 Recurrent Neural Networks (RNNs)

In comparison to Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) as shown in Figure 4.8, are particularly designed to apply for handling sequential data [Rumelhart et al., 1985; Werbos, 1988]. Many different architectural models exist, such as the LSTM

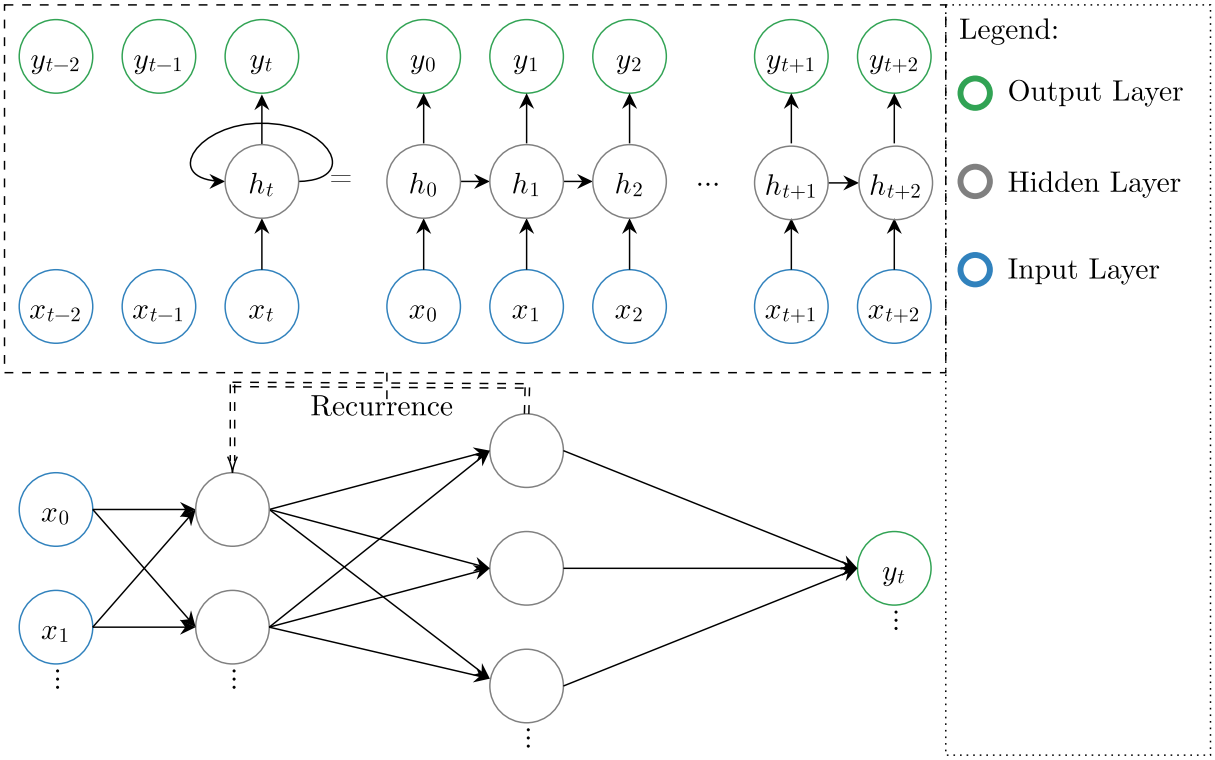


Figure 4.8: An example Recurrent Neural Network (RNN) with Input Layer, Hidden Layer, Output Layer and Hidden Unit Recurrence Connection.

[Hochreiter and Schmidhuber, 1997] and Neural Turing Machines (NTMs) [Graves et al., 2014] which got special importance for natural language processing / understanding [Sutskever et al., 2014], language generation and translation [Kalchbrenner and Blunsom, 2013; Mikolov et al., 2010; Sundermeyer et al., 2012; Bahdanau et al., 2014] and video processing [Graves, 2012; Donahue et al., 2015; Srivastava et al., 2015, p. 2]. The major difference of RNNs to CNNs is that the neuron cells have recurrent connections to propagate information to higher leveled neurons. To realize this, special neuron cells such as LSTM cells and Gated Recurrent Units (GRUs) are used [Chung et al., 2014]. Furthermore, this enables the modeling of long-term information dependency which is required for time-series prediction modeling. For each individual input sequence, the *Input Layer* is fed with a set of features where for each time step a neural network unit is applied to a single feature at a time step and added to the RNN units' output from the previous time step.

Gated Recurrent Units (GRUs): This paragraph aims to provide more detailed information regarding the use of GRUs in a multi-layered RNN which is applied to an input sequence. In order to realize the modeling of long-term information dependencies which is required for time-series prediction and combinatorial optimization, a neuron or also called neural network cell should provide certain types of characteristics:

- the used NN cells should be able to store long-term dependencies between different input sequences,
- the cell should provide a method of resetting or forgetting specific and partial parts of the input and hidden state information,
- thus, having a concurrent information propagation paradigm comes with promising characteristics for combinatorial optimization.

Requirements as such, generally are satisfied by the GRU cell but also LSTM cells provide this feat. The individual parts of a GRU cell will be explained next and are visualized in Figure 4.9. Given an input sequence feature vector $\boldsymbol{x} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_I]$ with size I and a multi-layered GRU

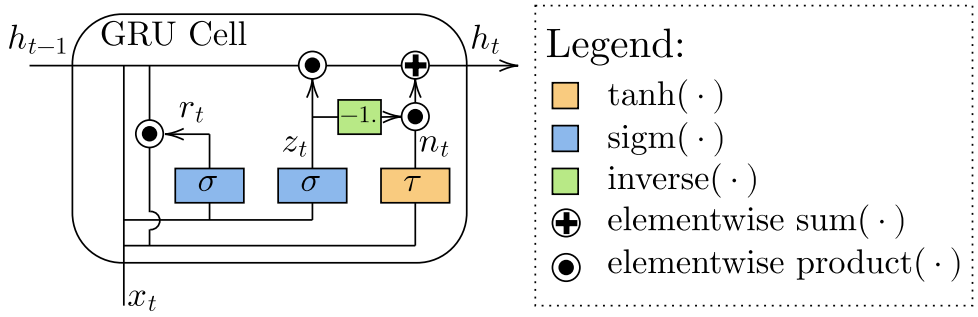


Figure 4.9: An example RNN GRU Cell with input (last hidden state), reset, update, new information, and output (next hidden state) gate [Dey and Salemt, 2017, pp. 1597–1600].

RNN, as depicted in Figure 4.10, each layer computes individual forward propagated values. The forward propagation is denoted by the following functions [Pytorch Doc., 2020a]:

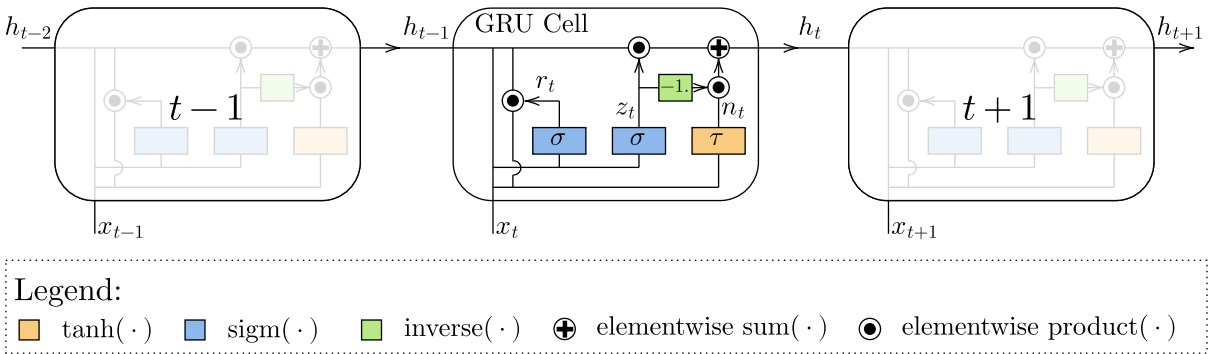


Figure 4.10: An example multi-layered GRU with input (last hidden state), reset, update, new information and output (next hidden state) gate. The repeating modules apply three individual activation layers respectively [Dey and Salemt, 2017, pp. 1597–1600].

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_{ir}\boldsymbol{x}_t + \boldsymbol{b}_{ir} + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hr}) \,, \tag{4.4}$$

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_{iz}\boldsymbol{x}_t + \boldsymbol{b}_{iz} + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hz}) \,, \tag{4.5}$$

$$\boldsymbol{n}_t = \tau(\boldsymbol{W}_{in}\boldsymbol{x}_t + \boldsymbol{b}_{in} + \boldsymbol{r}_t \odot (\boldsymbol{W}_{hn}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hn})) \,, \tag{4.6}$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{n}_t + \boldsymbol{z}_t \odot \boldsymbol{h}_{t-1} \,, \tag{4.7}$$

where h_t denotes the hidden state at a time step t and accordingly, x_t is the input feature vector at time step t. The hidden layer state tensor from the previous time step is denoted by h_{t-1}. Note, that the previous time step additionally may be the initial hidden state at time step 0. The main parts of the GRU are:

- the reset gate $\boldsymbol{r}_t$ which is used to forget certain information,
- the update gate $\boldsymbol{z}_t$, which is used to update existing information,
- the new information gates $\boldsymbol{n}_t$ for the input of new data,
- the Sigmoid and Tangens Hyperbolicus activation functions, $\sigma(\cdot) = \text{sigm}(\cdot)$, $\tau(\cdot) = \tanh(\cdot)$,
- and the final output or next RNN hidden state $\boldsymbol{h}_t$.

In a multi-layered GRU, where the number of layers must be two at minimum ($l >= 2$), each input $\boldsymbol{x}^l$ of the l-th layer yields the hidden state-value of the previous layer h_t^{l-1}, multiplied with the dropout probability δ_t^{l-1}. Here, δ_t^{l-1} is a Bernoulli random variable that takes the value 0 with the dropout probability δ_t^{l-1}. The main parameters of the GRU are called the *input_size* I, *hidden_size* H, and the amount of layers used *num_layers* L. The hidden size H thereby denotes how many hidden layers are used to define the depth of the RNN. Initially, all weights and biases are initialized uniformly via $\mathcal{U}[-\sqrt{k}, \sqrt{k}]$ where k is defined by the inverse of the hidden size $k = \frac{1}{H}$ [Dey and Salemt, 2017, pp. 1597–1600].

4.6 Learning with NNs

The learning process of a neural network or technically, the update mechanism of the neural network's parameters, is achieved by applying two main processes as follows. First, there is the

- Forward Pass, $f(\boldsymbol{x})$ which refers to the calculation of output values by propagating the input data through the network (network top-down). The input data traverses all neuron elements from the first to the last layer. Using the output values, a loss function is calculated which indicates the training error in the training dataset. Subsequently, there is the
- Backward Pass or so-called Back-propagation $\boldsymbol{J}_{\boldsymbol{x}} f(\boldsymbol{x})$ refers to the process of estimating the changes in the weights (i.e. the learning process) usually using gradient descent or similar algorithms. The computation of the Backward Pass is made from the last layer, backward to the first layer (network bottom-up). In order to compute all network parameter updates via gradient descent, all gradients of network's state (i.e. its computational graph) are required. In more detail, a gradient is the partial derivation of all residing network states denoted by:

$$y = f(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R} \,, \tag{4.8}$$

$$\frac{\partial y}{\partial \boldsymbol{x}} = \nabla_x f(\boldsymbol{x}) = \left[\frac{\partial f}{\partial \boldsymbol{x}_1}, ..., \frac{\partial f}{\partial \boldsymbol{x}_d}\right] . \tag{4.9}$$

Here, $f(\boldsymbol{x})$ is a function that calculates a d-dimensional input state $\boldsymbol{x}$ to a scalar output $\mathbb{R}$. The gradient in Equation (4.9) is the vector of partial derivatives. The positive gradient furthermore computes the directional vector which denotes the maximum increase of the original function $f(\boldsymbol{x})$. On the other hand, the negative gradient presents the directional vector which denotes the minimum increase and vice versa.
Subsequently, the gradient vectors can be stacked, which is called the Jacobian matrix with dimensions $d \times k$. The Jacobian matrix finally formulates the generalization for d-dimensional input signals and k-dimensional output signals as follows:

$$\boldsymbol{y} = f(\boldsymbol{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^k \ , \ \boldsymbol{J}^x_{i,j} = \frac{\partial \boldsymbol{y}_i}{\partial \boldsymbol{x}_j} \ , \tag{4.10}$$

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \boldsymbol{J}^x[f(\boldsymbol{x})] = \begin{bmatrix} \frac{\partial y_1}{\partial \boldsymbol{x}_1} & \cdots & \frac{\partial y_1}{\partial \boldsymbol{x}_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_k}{\partial \boldsymbol{x}_1} & \cdots & \frac{\partial y_k}{\partial \boldsymbol{x}_d} \end{bmatrix} . \tag{4.11}$$

Generally, Backward and Forward Pass can be seen as one iteration, where a subset of the training or validation dataset is passed. The subset of the individual data set is called mini-batch, whereas passing all data at once would be called a batch [Google Deep Mind Lecture, 2020].

Gradient Descent: The main learning algorithm behind Deep Learning is called Gradient Descent. Gradient Descent is a numerical algorithm that estimates to which degree the network's weights and bias parameters are adjusted [Kingma and Ba, 2014, pp. 1–3], where the gradient is calculated based on a pre-defined loss (training error) function. Generally, stochastic gradient descent finds a local minimum of any given function $f(\boldsymbol{x})$ in $\mathbb{R}^3$ three-dimensional space, if the function is sufficiently smooth.

The algorithm initially picks a random starting point and progressively uses several parameter updates and iterations to converge to the local optimum as depicted in Figure 4.11. The update rule is as follows:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) \tag{4.12}$$

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) = \nabla_{\boldsymbol{\theta}} \sum_i^I l(g(\boldsymbol{x}^i, \boldsymbol{\theta}_t), \boldsymbol{t}^i) = \sum_i^I \nabla_{\boldsymbol{\theta}} l(g(\boldsymbol{x}^i, \boldsymbol{\theta}_t), \boldsymbol{t}^i) \, . \tag{4.13}$$

In Equation (4.12), $\boldsymbol{\theta}$ is a bootstrapping parameter that includes all weights and bias hyper parameters of the network, α is the learning rate, and $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$ is the gradient of an arbitrary loss function, see for an example in Figure 4.11. The choice of the learning rate α is crucial because this influences the learning step size of the algorithm. The parameter α determines how rapidly the algorithm approaches the local minimum, and thus how urgently the network adjusts and

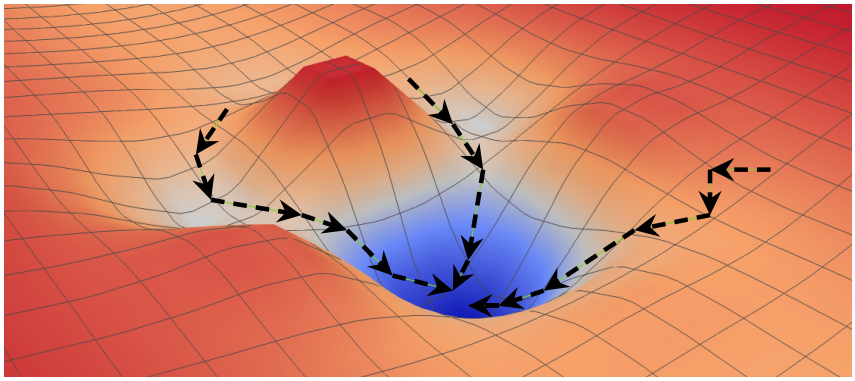


Figure 4.11: An example 3-D Loss function $L(\boldsymbol{\theta}_t)$ where the gradient stochastic descent algorithm aims to find the directions to a local minimum.

learns the weight and bias parameters. A small learning rate usually leads to slow, but robust learning, a high learning rate value leads to fast learning and therefore large parameter updates, however, this may also lead to unstable learning behavior. This is commonly called learning divergence, which should be avoided at this point since the network is not progressing to learn a stationary policy. The process of stagnating learning is called (under-fitting) which is one of the problems that may occur during the training of neural networks. Such problems are addressed in more detail in Subsection 4.7.

Finally, the gradients that are required for the main gradient descent update in Equation (4.12), are computed mathematically by using the chain rule as follows:

$$y = f(g(\boldsymbol{x})), \qquad \frac{\partial y}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}, \tag{4.14}$$

$$y = f(g(\boldsymbol{x})), \qquad \frac{\partial y}{\partial x} = \sum_{i=1}^{I} \frac{\partial f^i}{\partial g}\frac{\partial g^i}{\partial x}, \tag{4.15}$$

where $f(\boldsymbol{x})$ is the outer and $g(\boldsymbol{x})$ is the inner function of the neural network's graph. If the inputs $\boldsymbol{x}$, the outputs $\boldsymbol{y}$ and $f(\boldsymbol{x}), g(\boldsymbol{x})$ are multi-dimensional, the chain rule is applied via matrix calculus which is the sum over all possible input data paths. This computes the gradients over the whole NN computational graph by summing over the partial derivatives in Equation (4.15) and generally depends on the used network topology.

The computation of the gradients is efficiently solved by automatic differentiation optimizers. For this purpose, multiple optimizers exist that are based on gradient descent. Some optimizers such as ADAM [Kingma and Ba, 2014, pp. 1–10], RMSprop [Tieleman and Hinton, 2012] Adadelta [Zeiler, 2012, pp. 1–6] and Adagrad [Duchi et al., 2011, pp. 1–39] have been proven and practically successful. In the past, NNs have been applied to learn from discrete non-smooth or discrete datasets. It turns out, that violating the smoothness assumption, however, leads to unstable learning when using NNs for function approximation. Such behavior is commonly indicated by a non-decrease of the loss function during the learning phase and can be avoided by using the methods introduced in Subsection 4.8.

4.7 NN Training Issues

This section contains common issues when training NNs such as

- under-fitting,
- over-fitting,
- exploding gradients,
- and vanishing gradients.

A very common issue when training neural networks is the occurrence of over-fitting and under-fitting. As the neural network becomes more powerful over the training phase, the NN model can create very complex hypotheses, even if the input data is very simple. As the NN gets more complex, in the sense that it increases its capabilities of representing increasingly complex functions, the more probable it is for the NN to end up over-fitting the input data [Vapnik, 2013, pp. 512–513]. Over-fitting refers to the issue that the network learns to represent the input data in a relatively over-complicated way such as memorizing the training data without learning its patterns at all. Hereby, the capability of representing increasingly complex input data while storing more data over time is referred to as the increase of the hypothesis space $\mathcal{H}$, e.g. see Figure 4.12. While increasing the hypotheses space, the training error decreases because

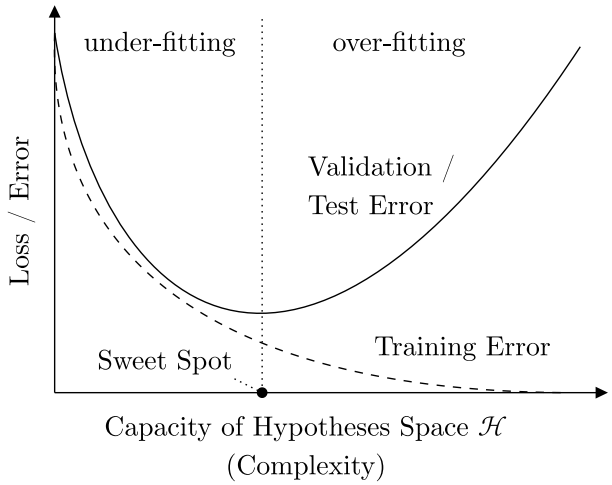


Figure 4.12: An example plot that illustrates the Risk of under- and over-fitting depending on the model complexity or also called Hypotheses Space Capacity $\mathcal{H}$ [Belkin et al., 2019].

the network parameters are tuned in order to minimize the training error as much as possible. This as mentioned earlier, is stated by the universal function approximation theorem [Cybenko, 1989, pp. 303–314]. Another common issue when training neural networks with Gradient Descent is that all values of the gradients may saturate (exploding gradient) [Pascanu et al., 2013, pp. 1310–1318] or the directional values of the gradient converge to zero (vanishing gradient) [Hochreiter, 1998, pp. 107–116] and the gradient loses its directional information. Both issues deteriorate, or in the worst-case scenario, prevent the learning progress and therefore should be avoided. Especially, the training of RNNs with GRUs is prone to run into the exploding gradient issue [Kanai et al., 2017, pp. 435–444]. In order to avoid such issues, a list of countermeasures can be found in the next Section 4.8.

4.8 Solving NN Training Issues

This section contains common methods dealing with the issues of training neural networks that have been introduced in Section 4.7. In more detail, this section addresses methods such as specific "Selection", "Dropout", "Gradient Normalization", and "Maximum Gradient Clipping". Such methods are mainly used to avoid the over-fitting, exploding, and vanishing gradient issues in the main Chapter 5. Still, there are other means worth mentioning including the previously mentioned methods such as

- Activation Function Selection [Marchisio et al., 2018],
- Dropout and Noising Training Data [Gal and Ghahramani, 2016, p. 60],
- Maximum Gradient Normalization [Chen et al., 2018, pp. 1–10] and Clipping [Graves, 2013; Pytorch Doc., 2020b, pp. 23, 1],
- Batch Normalization [Ioffe and Szegedy, 2015, pp. 1–11],
- Early Stopping [Caruana et al., 2001, pp. 1–7], and
- L_p regularization [Zaremba et al., 2014; Demyanov, 2015, pp. 1–8, 45–75].

First, the Activation Function Selection mainly addresses and solves the vanishing gradient issue, which commonly occurs when using Sigmoid and Hyperbolic Tangent activation functions within a Deep Neural Network with many layers. The vanishing gradient problem refers to the case, that large networks become harder to train when adding more and more layers. By adding more and more layers, the gradients calculated based on the loss functions tends to converge to zero. This unfortunately prevents the network from progressing to learn new valuable information. This phenomena, can be explained at the example in Figure 4.13, where the calculated derivative of the Sigmoid function $\sigma_{t+1}(x)$ converges to zero for large negative and large positive inputs x. Hereby, certain activation functions such as the Sigmoid and Tangens Hyperbolicus, squash large

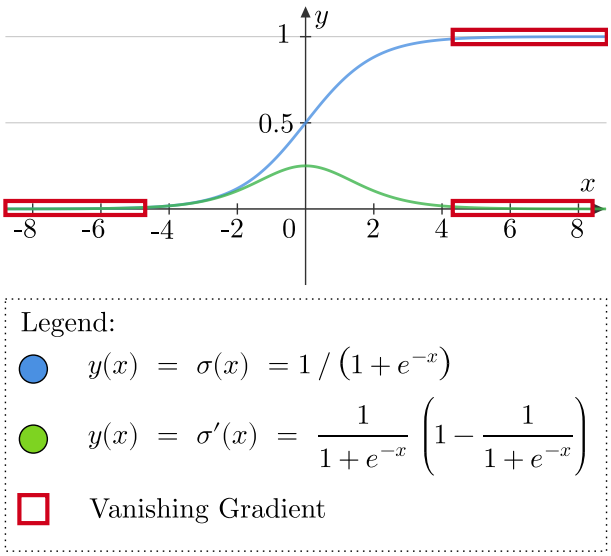


Figure 4.13: An example plot that depicts the vanishing gradient issue of using a Sigmoid activation function.

input states into a small input space between a function-specific value range, e.g. $[0, 1]$. If the derivative of the used activation function has a very flat gradient for large negative and large positive numbers, large input values will result in small gradient changes in the output of the NN. This occurrence is quite common when training large neural networks and therefore has to be considered and mitigated. The same phenomena can be observed for any Tangens Hyperbolicus activation function since the function similarly is characterized by an "s-shaped" basis function, which results in the analogous effect of the Sigmoid function. In comparison, other shapes of activation functions such as the ReLU basis function are less prone to the vanishing gradient problem [Glorot et al., 2011, pp. 315–323]. The basis shape furthermore results in a first-order derivation that does not allow for a convergence process towards zero. Even less prone to the vanishing gradient are relatively modern designed activation functions such as the Leaky ReLU [Maas et al., 2013, pp. 1–3] and the Exponential Linear Unit (ELU) [Clevert et al., 2015, pp. 5–10], which enhance the similar calculus effect. In particular, the work of Clevert et al. has illustrated, that by changing the activation functions to ReLUs, Leaky ReLUs, and especially to ELUs the learning efficiency of RNNs can be increased [Clevert et al., 2015, pp. 1–14]. On the other bookend, similarly shaped activation functions are relatively prone to face the exploding gradient issue, where the gradient values converge to infinity. Well known as the exploding gradient problem, this phenomenon can be efficiently encountered using Maximum Gradient Clipping [Graves, 2013; Pytorch Doc., 2020b, pp. 23, 1] and Gradient Normalization [Chen et al., 2018, pp. 1–10] during the training phase.

Another technique for mitigating the exploding gradient phenomenon is called "Dropout". Dropout is a regularization technique that has the effect of inflicting noise on the training process in order to counter NN over-fitting issues during the training phase. Dropout forces a number of neural network cells within a layer to vary their responsibility for computing the output from the input data. Therefore, the concept aims to break or dropout learning situations where each prior NN layer propagates mistakes to the post-NN layers. This generally makes the model more robust, especially when the model is prone to over-fitting [Srivastava et al., 2014, p. 23]. Additionally, useful for preventing a NN model to over-fit, is by using Dropout [Gal and Ghahramani, 2016; Zaremba et al., 2014, pp. 60, 1–8] and scaling or normalizing the gradient, which aims to reduce the magnitudes of large gradients. This results in the effect, that high changes in the weights are reduced, which not solely reduces over-fitting, but also reduces the risk of the exploding gradient issue [Chen et al., 2018, pp. 1–10]. Particularly, useful when dealing with exploding gradient phenomena, is the concept of "Maximum Gradient Clipping". The gradient magnitudes are clipped, if the derivative of the loss function with respect to the NN's inputs exceeds a certain value range [Graves, 2013; Pytorch Doc., 2020b, p. 23]. The objective here is to keep the gradient in a predefined range, where clipping according to the predefined bounds directly depends on the used activation function shape and its derivative. If the used NN is likely to face the exploding gradient issue, the gradient clipping method presents an additional efficient means to increase the NN's learning robustness. Batch Normalization [Ioffe and Szegedy, 2015, pp. 1–11] for instance, aims to reduce the dynamism of each NNs layer, which results in increases concerning the learning robustness. Without using Batch Normalization, the training of Deep

NNs is stated to require lower learning rates, which results in a slower learning process. The concept achieves this by normalizing the layer inputs for each trained mini-batch. The work of Ioffe and Szegedy shows that Batch Normalization achieves faster training times by achieving the same prediction accuracy for state-of-the-art image classification models, which underlines the advantages of this method. Moreover, the concept of early stopping can be explained by taking into account Figure 4.12. The concept aims to stop the learning process of a NN by stopping the learning at the point which is marked as a "Sweet Spot" in Figure 4.12. This is achieved by tracking the decreasing Training Error or Loss to the point where the Validation Loss or Error values stops decreasing. At this point, the NN model achieves the best trade-off of avoiding the under-fitting and over-fitting issues. The last method which is the simplest and most traditional method to prevent a DNN from over-fitting is called L^p regularization, which has similar effects to Maximum Gradient Clipping and Dropout.

L^p regularization similarly aims to penalize large weight updates due to large calculated gradients by adding a p-order penalty to reduce the magnitudes of the derived gradient values. The method is a common standard, where more information can be found in [Demyanov, 2015, pp. 45–75]. As a final note, it is worth mentioning, that the work of Belkin et al. proved, that under some constraints, large-scale NN models are less prone to over-fitting because as the network size increases, the learning dynamics simultaneously change [Belkin et al., 2019, pp. 303–314]. Nevertheless, the training of large NN models still requires and benefits from the outlined regularization methods. After researching the outlined regularization concepts it can be concluded, that enhancing regularization methods have desired properties with respect to increasing learning efficiency and quality.

4.9 Introduction to Reinforcement Learning

This section contains a brief and fundamental introduction to the research field of Reinforcement Learning (RL). It is advised to additionally consider fundamental literature for a deep-dive and more information [Sutton, Barto, et al., 1998, pp. 1–445].

Reinforcement Learning (RL) is a sub-area in machine learning where the main focus lies on sequential decision-making processes under stochastic uncertainty assumptions. RL algorithms mainly use the trial-and-error paradigm for gaining experience in order to learn problem-specific beneficial behavior. Thus the area of RL is fundamentally and methodologically different to other optimization paradigms such as genetic algorithms, simulated annealing, and linear programming. The named paradigms differ in the sense that no experience/learning phase is used to compute a desired optimal solution to a given problem. An RL framework as visualized in Figure 4.14 is mathematically formalized by introducing the terms

- "environment" (i.e. the dynamic process, surroundings or habitat) which is affected and is changed by the agent's decisions and actions,
- "reward or cost mechanism"(i.e. the notion of desired and undesired agent behavior), and finally,
- the learning "agent",

which symbolizes a learning entity that has to make decisions in order to achieve its goals or in

other words to optimize its utility. This is possible given the notion of rewards or penalties/costs

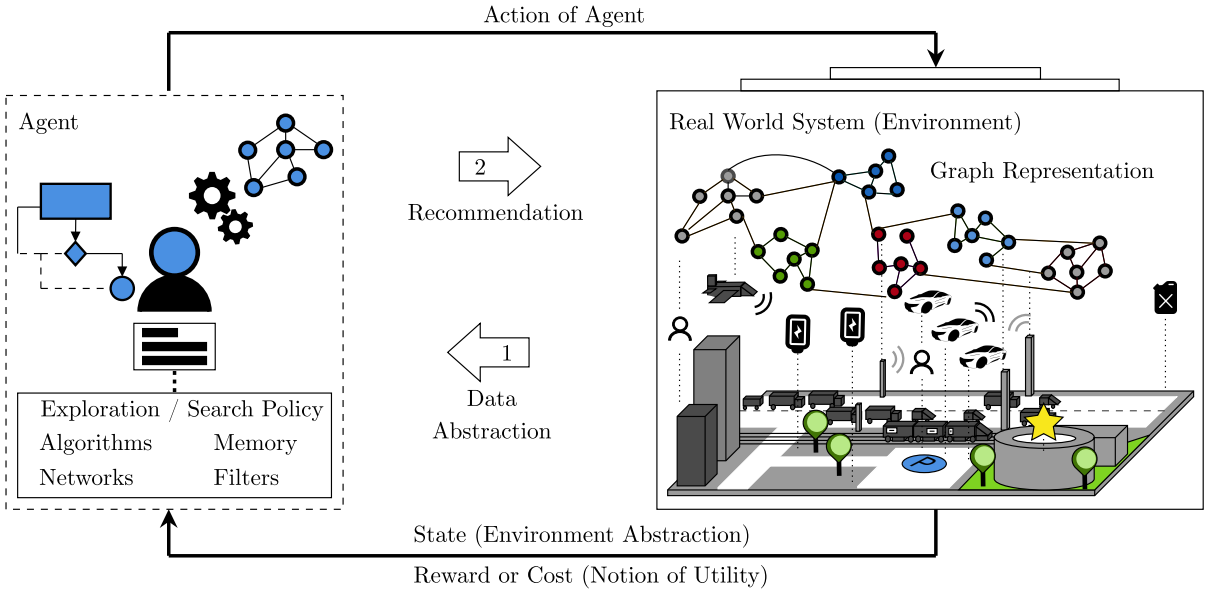


Figure 4.14: A basic reinforcement learning framework with the essential parts such as the environment, a learning agent as a decision entity, a reward mechanism, the environment state representation, and the selected action. The term agent thereby is an intelligent software component that learns a strategy to solve problems of the environment. The software component capabilities thereby may include data exploration, search and optimization algorithms, memory of the environment behavior, and action consequences using the models of networks while filtering unnecessary data. For learning about the environment, the agent requires a feedback loop based on its actions, thus learning about the environment workings, processes, and causalities.

for desired and undesired behavior. The RL framework incorporates a variety of essential features in artificial intelligence and is applicable to many tasks, which require notions of cause and effect, as well as uncertainty and non-determinism. The RL framework has tight connections to game theory, economics, planning, and fuzzy logic control processes which are characterized by similar requirements.

RL and Bayesian Decision Theory: This section introduces fundamental mathematical and probabilistic formulations in order to establish the simple RL framework in Figure 4.14. At the root, every rigorous RL framework leverages the basic mathematical formulations from probability theory, Bayesian Decision Theory and MDPs, which essentially deal with discrete time, stochastic control processes, and sequential decision-making [Bellman, 1957, pp. 1–3]. As Figure 4.14 illustrates, the agent requires an abstracted version of the current environment state in order to learn a belief or notion of the real-world state. In order to achieve this, fundamental methods from graph and probability theory can be used to represent the environment and the agents' belief of the current environment state [Murphy, 2012, p. 176]. The same framework applies to strategic games such as chess where two players have to develop a strategy to win the game within

its environment (chessboard). Further, the same characteristics apply to many other real-world problems in game theory, economics, and especially logistic planning, where the processes always come with a notion of uncertainty and the main hypothesis is to increase the expected utility of the controllable system (e.g. for instance the fleet operation in a city) with respect to future decisions and uncertain outcomes. Here it is important to increase the probability of a desired future outcome, which has the same statistical and probabilistic characteristics as aiming to win a gamble while playing k-armed bandits.

In this thesis, however, the focus is set on planning and logistic control processes, therefore this work will omit all other application domains. Generally, an underlying real-world environment (e.g. a city) changes its state dynamically. In almost all RL frameworks the environment state is formulated based on MDPs which defines transition probabilities in order to model how the environment state changes over time (i.e. called Markov Chain). At each time step, when the environment state can be observed, the representation or a mapping of the environment state, that is used as input information for the agent in the form of data. In Bayesian Decision Theory this is referred to as an (environment state) observation $\boldsymbol{s} \in \mathbb{S}$. In the context of a fleet operation, the environment state and fleet state would be sub-states of the whole dynamic environment state, which again would be collectively observed by an agent. Based on the previously made environment observations, an agent (e.g. a fleet manager) may plan actions for the future in order to reach certain goals [Lin et al., 2018, p. 176]. In order to achieve certain goals, the agent requires a set or repertoire of possible actions. This is called action space $\mathbb{A}$. In the context of real-world fleet management, the agent would be a fleet manager which plans the routes of taxis or trucks to their destinations. Here the action space would be the order of processed customer (demand) requests and the selection of adequate vehicles (fleet supply). In order to assess the value of the planned and executed actions, a feedback loop in the form of a reward, cost (penalty), or error (loss) function is required. Hereby, the executed or planned actions $\boldsymbol{a}$ naturally have to be compatible with the desired environment state. If a sequence of actions (decision procedure) can be learned by the agent, to reach a certain goal, the agent has learned a strategy to maximize the expected utility. This learned decision procedure or agent strategy is called policy $\boldsymbol{\pi} : \mathbb{S} \rightarrow \mathbb{A}$ which is defined to specify the optimal action sequence for each possible environment state observation. The term optimality is meant in the sense that the learned action sequence should minimize the expected (future) loss or maximize the expected (future) achievable reward. In the example of a cost or loss function $L(\boldsymbol{s}, \boldsymbol{a})$ this can be formulated as:

$$\boldsymbol{\pi}(\boldsymbol{s}) = \underset{\boldsymbol{a} \in \mathbb{A}}{\operatorname{argmin}} \, \mathbb{E}[L(\boldsymbol{s}, \boldsymbol{a})] \tag{4.16}$$

which is known as the *maximum expected utility principle* [Miller et al., 2015] that models the rational behavior of an agent, or with respect to this work's application, the rational decisions of a fleet operation planning system. The word *expected* can be found with two meanings, first in Bayesian versions it is referred to the expected value of a given data distribution $P(\boldsymbol{s})$. In the frequentist version, the expected value is referred as the expected value of a posterior probability distribution $\rho(\boldsymbol{s})$ which takes the expected value at some time-step in the future. According to

Bayesian Decision Theory, an optimal action at the current time step can be computed when having observed the current environment state $\boldsymbol{s}$ and an action $\boldsymbol{a}$ are selected that minimize the posterior expected loss (cost) (or maximizes the reward function). This probabilistic decision rule is called the *Bayes estimator* and is denoted by

$$\rho(\boldsymbol{a}|\mathbf{s}) \triangleq \mathbb{E}_{p(\boldsymbol{a}|\mathbf{s})}[L(\boldsymbol{s},\boldsymbol{a})] = \sum_{\boldsymbol{s}} L(\boldsymbol{s},\boldsymbol{a})p(\boldsymbol{a}|\mathbf{s})\,. \tag{4.17}$$

Markov Decision Processes and Markov (State) Chain: In continuous time versions the sum is replaced by an integral, whereas the sum illustrates the discrete formulation setting. Generally, the MDP formulation refers to fully observable environments, where all environmental state changes can be observed instantly and therefore completely characterize the workings of the environment. In real-world systems, most systems however are characterized by the term partially observable, which refers to the issue that some changes in the environment state may be delayed or may not be observed directly and therefore can not be directly associated with the agent's made decisions and actions. Nevertheless, partially observable environments can be converted into MDPs [Martin, 2004, p. 14]. The formulation via MDP moreover comes with certain properties which are important and briefly introduced in the following paragraph.

Definition 19: A Markov Process (or Markov Chain) is a tuple $\langle \mathbb{S}, \boldsymbol{t} \rangle$, where $\mathbb{S}$ is a (finite) set of states, $\boldsymbol{t}$ is a state transition probability matrix, $\boldsymbol{t}_{ss'} = [S_{t+1} = s' \mid S_t = s]$.

The RL framework therefore can be fully formulated using the MDP formulation. The formulation defines a discrete time and stochastic control process where the agent observes the environment state $\boldsymbol{s}_t \in \mathbb{S}$ (abstraction of city data and fleet data in Figure 5.1). The agent starts to gather an initial observation $\boldsymbol{s}_0$ and continues to observe the environment for each consecutive time step t. For each time step, the agent has to select a sequence of actions $\boldsymbol{a} \in \mathbb{A}$, where good actions result in a corresponding reward $r_t \in R$. Finally, the control loop encloses where the agent obtains the new environment state observation $\boldsymbol{s}_{t+1}$, which is affected and has changed based on the previously chosen action. In Figure 4.15, the interaction between an agent and the environment is shown. The agent observes the environment state s_{t+1} and select action a_{t+1} accordingly. While observing the next environment state s_{t+2} the agent receives the reward for selecting action a_{t+1} from the previous time step. Furthermore, the typical RL framework is

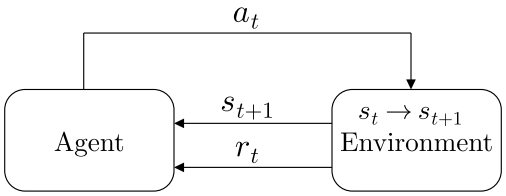


Figure 4.15: RL agent and environment.

Legend:	
a_t	action
s_t	state
s_{t+1}	next state
r_t	reward

Table 4.1: Legend of Figure 4.15.

defined by a Markovian stochastic control process, that again comes along with some definitions:

Definition 20: An stochastic control process with discrete time is Markovian only if it satisfies the Markov property:

$$P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) = P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_0, \boldsymbol{a}_0, ..., \boldsymbol{s}_T, \boldsymbol{a}_T) \ .$$

The Markov property states that all future actions and states of the environment must be causally associative with the current observed environment state. In other words, the learning agent must be able to reason in an MDP without requiring the full history of all passed states. Thus, the agent's learning and decision-making must be possible within a fixed time horizon. The stochastic and discrete-time MDP is further defined as follows:

Definition 21: A MDP is a 5-tuple of parameters $\langle \mathbb{S}, \mathbb{A}, \boldsymbol{t}, R(\cdot), \gamma \rangle$ where

1. $\mathbb{S}$ is a finite set of environment states,
2. $\mathbb{A}$ is a finite set of (agent) actions,
3. $\boldsymbol{t} : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \rightarrow [0, 1]$ is a probability transition function matrix with conditional probabilities for describing the probability of changing from state to next state,
4. $R(\cdot) : \mathbb{S} \times \mathbb{A} \times \mathbb{S}$ is a continuous reward function, which defines a set of achievable rewards or cost (penalties) in a range $r_{max} \in \mathbb{R}^+$ (e.g., $[0, r_{max}]$) and $c_{max} \in \mathbb{R}^+$ (e.g., $[0, r_{max}]$),
5. $\gamma \in [0, 1)$ which is the discount factor.

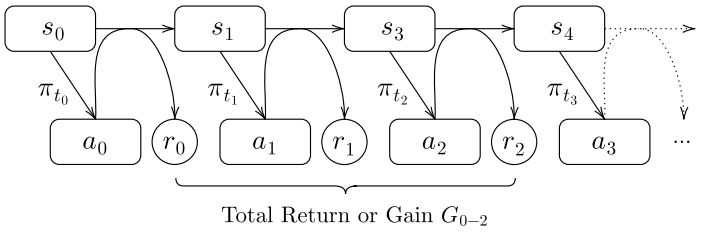


Figure 4.16: A visualized Markov Decision Process (MDP) or Markov Chain with state transitions, selected actions based on the agent's policy and received rewards based on the current policy and chosen actions.

Legend:	
$\boldsymbol{s}$	state sequence
$\boldsymbol{a}$	action sequence
$\boldsymbol{r}$	reward sequence
T	until terminal state

Table 4.2: Legend of Figure 4.16.

Figure 4.16 moreover shows the visualization of the discrete time and stochastic MDP based on Markov state transitions, which are defined as follows.

Definition 22: Each Markov state $\boldsymbol{s}_t$ has a successor state $\boldsymbol{s}_{t+1}$, where the state transition matrix defines the state transition probabilities by

$$\boldsymbol{t}_{s_t, s_{t+1}} = P[s_{t+1}|s_t] = \text{from} \begin{bmatrix} & \overset{\text{to}}{} & \\ p_{1,1} & \cdots & p_{1,n} \\ \vdots & & \vdots \\ p_{n,1} & \cdots & p_{n,n} \end{bmatrix}$$

where each row of the matrix defines in what state the agent is possibly in. Further, they must sum up to 1, which corresponds to a normalization to 100% probability.

Definition 23: A sample of episodes from a MDP is a random state sequence, that is sampled from the probability distribution of sequences of states.

Markov Reward Process: In order to create a value judgment of good or bad agent behavior, there must be a functional definition for a total gain depending on previously selected actions. Generally, this tells the agent that if initially being in state $\boldsymbol{s}_0$ how the magnitude of the reward is to transition to another more desired state $\boldsymbol{s}_1$. So based on the reached state sequences the agent also receives a sequence of positive total rewards or penalty costs. This is called the total return or gain $\boldsymbol{G}_t$.

Definition 24: The total return or gain $\boldsymbol{G}_t$ is the overall sequence of discounted rewards from time-step t and is denoted by

$$\boldsymbol{G}_t = \boldsymbol{R}_{t+1} + \gamma \boldsymbol{R}_{t+2} + ... = \sum_{t=0,k=0}^{\infty} \gamma \, \boldsymbol{R}_{t+1} \, . \tag{4.18}$$

The selectable weight parameter $\gamma \in [0, ..., 1]$ is called the discount factor which represents the fact that future rewards are less certain than rewards closer to the actual agent state. If γ is close to 0 the behavior of the agent leads to "*myopic*" evaluation. Otherwise, if γ is closer to 1, this will lead to a rather "*far-sighted*" agent behavior, which increases the priority of future rewards. This formulation of the total gain is required if the environment of the agent does not provide perfect knowledge about the evolution of its processes. In other words, if the environment model is not perfectly known in its behavior, this formulation helps the agent to focus on reachable rewards in its developed plan and strategy.

Agent Policy: The agent's policy defines the strategy of how an agent selects the sequence of actions and is based on the observed states, e.g. see Figure 4.16. In the mathematical formalism, the policy further is described as a mapping from states to actions, as follows:

1. $\boldsymbol{\pi}(\boldsymbol{s}) : \mathbb{S} \rightarrow \mathbb{A} \, ,$
2. $\boldsymbol{\pi}(\boldsymbol{s}, \boldsymbol{a}) : \mathbb{S} \rightarrow [0, 1] \, .$

Policies thereby can be categorized using two criteria. The first criterion refers to being stationary and non-stationary. Non-stationary refers to the definition of being time-step dependent and stationary as being time-step independent. The second criterion is of being either deterministic or stochastic. In the deterministic case, the policy is formulated by equation (1), whereas the stochastic version is formulated in Equation (2), where $\boldsymbol{\pi}(\boldsymbol{s}, \boldsymbol{a})$ defines the action probability of selecting $\boldsymbol{a}$ among the state $\boldsymbol{s}$. In the next sections, the different ways of learning an RL policy (agent strategy) will be covered. The following sections refer to the different possibilities of learning an RL policy.

4.10 Reinforcement Learning Variants

This section describes the inner methods and components of an RL agent. The agent may include one or more of the following components:

1. a representation of the agent state,
2. a representation of utility, in the sense from the agent perspective, how desired each state is for reaching a goal,
3. a direct representation of the policy $\boldsymbol{\pi}(\boldsymbol{s}), \boldsymbol{\pi}(\boldsymbol{s}, \boldsymbol{a})$ and the executable actions,
4. a planning algorithm for planning the next sequence of actions,
5. and finally a model of the environment (environment state).

The second component can be distinguished in *model-free* and *model-based* methods that aim to represent the value function of an agent in a indirectly or directly way, see Figure 4.17. The

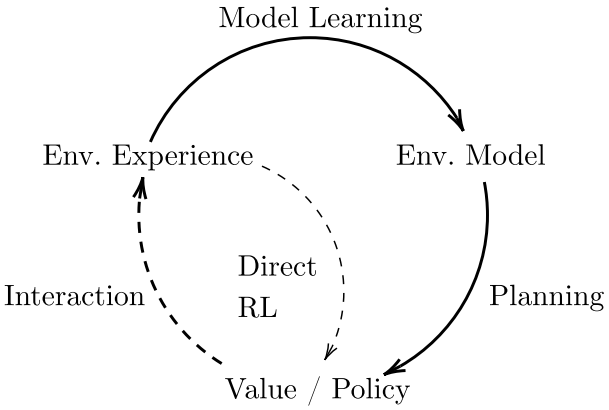


Figure 4.17: The dependencies of model-free and model-based RL methods [Sutton, Barto, et al., 1998].

difference hereby between *model-free* and *model-based* is that model-free methods do not require any model that defines the state transitions $\boldsymbol{t}$. Instead, model-free methods solely aim to learn from the action and reward feedback, while learning a representation of the environment model indirectly (as an approximation). This has a big advantage, especially if the environment model is very complex and hard to capture in its fullest complexity. A drawback of model-free methods usually comes with increased learning times, since the agent needs to build up experience first during a convergence process, in order to learn the workings of the environment processes and dynamics. Further, this section briefly introduces and differs between the common RL methods of *Value-based Methods* and *Policy-based Methods* in more detail.

4.10.1 Value-based Methods

Let us consider an RL framework with MDP $\langle \mathbb{S}, \mathbb{A}, \boldsymbol{t}, R(\cdot), \gamma \rangle$ where the objective of the agent is to find a deterministic policy (strategy) $\boldsymbol{\pi}(\boldsymbol{s}) \in \Pi$. The agent must learn a policy that optimizes the agent's belief in good behavior (i.e. the "state-value function"). This is achievable by finding the policy that yields the optimal and maximum expected return. The maximum expected return

$V_{\boldsymbol{\pi}}(\boldsymbol{s})$ that provides the value of being in state $\boldsymbol{s}$ further is formulated in the following.

Value Function: The value function $V_{\boldsymbol{\pi}}(\boldsymbol{s})$ gives the long-term value of state $\boldsymbol{s}$, where

Definition 25: The state-value function $V(\boldsymbol{s})$ of a Markov Reward Process (MRP) is the expected return starting in state s and is defined by

$$\boldsymbol{V}_{\boldsymbol{\pi}}(\boldsymbol{s}) = \mathbb{E}\left[\boldsymbol{G}_t \mid \boldsymbol{s}_t \in \mathbb{S}\right] = \max_{\boldsymbol{\pi} \in \Pi} V_{\boldsymbol{\pi}}(\boldsymbol{s}) \ . \tag{4.19}$$

The agent state-value function represents the total amount of achievable reward from the start of the agent's plan until the agent's terminal state. The state-value function in Equation (4.19) thereby is characterized by the (i) immediate reward R_{t+1} and the discounted future value of state $\gamma \boldsymbol{V}_{\boldsymbol{\pi}}(\boldsymbol{s}_{t+1})$. Inserting Equation (4.18) in the state-value function in Equation (4.19), the function becomes the well-known Bellman Equation

$$\begin{aligned}
\boldsymbol{V}_{\boldsymbol{\pi}}(\boldsymbol{s}) \leftarrow & \ \mathbb{E}\left[\boldsymbol{G}_t \mid \mathbb{S}_t = \boldsymbol{s}\right] , && (4.20)\\
& \ \mathbb{E}\left[\boldsymbol{R}_{t+1} + \gamma\, \boldsymbol{R}_{t+2} + \gamma^2\, \boldsymbol{R}_{t+3} + \ldots \ \mid \ \mathbb{S}_t = \boldsymbol{s}\right] , && (4.21)\\
& \ \mathbb{E}\left[\boldsymbol{R}_{t+1} + \gamma\, (\boldsymbol{R}_{t+2} + \gamma\, \boldsymbol{R}_{t+3} + \ldots) \mid \ \mathbb{S}_t = \boldsymbol{s}\right] , && (4.22)\\
& \ \mathbb{E}\left[\boldsymbol{R}_{t+1} + \gamma\, \boldsymbol{G}_{t+1} \mid \mathbb{S}_t = \boldsymbol{s}\right] , && (4.23)\\
& \ \mathbb{E}\left[\boldsymbol{R}_{t+1} + \gamma\, V\left(\mathbb{S}_{t+1}\right) \mid \mathbb{S}_t = \boldsymbol{s}\right] , && (4.24)
\end{aligned}$$

which is fundamental in dynamic programming [Bellman, 1966, pp. 1–5]. Complementary, the RL interaction between states, actions, and rewards is particularly visualized in Figure 4.15. Using the Bellman Equation, the state-value function and action-value function can be recursively formulated as

$$\boldsymbol{V}_{\boldsymbol{\pi}}(\boldsymbol{s}_t) \leftarrow \max_{\boldsymbol{a}_t} (\boldsymbol{R}_{\boldsymbol{s}_t}^{\boldsymbol{a}_t} + \gamma \sum_{\boldsymbol{s}_{t+1} \in \mathbb{S}} \boldsymbol{P}_{\boldsymbol{s}_t, \boldsymbol{s}_{t+1}}^{\boldsymbol{a}_t} \boldsymbol{V}_{\boldsymbol{\pi}}(\boldsymbol{s}_{t+1})) , \tag{4.25}$$

$$\boldsymbol{Q}_{\boldsymbol{\pi}}(\boldsymbol{a}_t | \boldsymbol{s}_t) \leftarrow \boldsymbol{R}_{\boldsymbol{s}_t}^{\boldsymbol{a}_t} + \gamma \sum_{\boldsymbol{s}_{t+1} \in \mathbb{S}} \boldsymbol{P}_{\boldsymbol{s}, \boldsymbol{s}_{t+1}}^{\boldsymbol{a}_t} \max_{\boldsymbol{a}_t} (\boldsymbol{Q}_{\boldsymbol{\pi}}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})) , \tag{4.26}$$

$$\boldsymbol{V}_{\boldsymbol{\pi}}(\boldsymbol{s}) \leftarrow \sum_{\boldsymbol{a} \in \mathbb{A}} \boldsymbol{\pi}(\boldsymbol{a}_t | \boldsymbol{s}_t) \boldsymbol{Q}_{\boldsymbol{\pi}}(\boldsymbol{a}_t | \boldsymbol{s}_t) , \tag{4.27}$$

$$\boldsymbol{Q}_{\pi}(\boldsymbol{a}_t | \boldsymbol{s}_t) \leftarrow \boldsymbol{R}_{\boldsymbol{s}}^{\boldsymbol{a}} + \gamma \sum_{\boldsymbol{s}_{t+1} \in \mathbb{S}} \boldsymbol{P}_{\boldsymbol{s}, \boldsymbol{s}_{t+1}}^{\boldsymbol{a}} \boldsymbol{V}_{\boldsymbol{\pi}}(\boldsymbol{s}_{t+1}) . \tag{4.28}$$

Similarly to the Value Function, some classes of algorithms such as Q-Learning and Sarsa aim to build the value function indirectly. The algorithms do not work directly with the $\boldsymbol{V}$-value function but instead make use of the $\boldsymbol{Q}$-value function $\boldsymbol{Q}^{\boldsymbol{\pi}}(\boldsymbol{s}_t, \boldsymbol{a}_t) : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{R}$ which can be

defined as follows:

$$\boldsymbol{Q}^{\boldsymbol{\pi}}(\boldsymbol{s}_t, \boldsymbol{a}_t) \leftarrow \mathbb{E}\left(\sum_{\boldsymbol{s}_{t+1}\in\mathbb{S}} \gamma \boldsymbol{R}_t \,|\, \boldsymbol{s}_t \in \mathbb{S}, \boldsymbol{a}_t \in \mathbb{A}, \boldsymbol{\pi}\right) , \tag{4.29}$$

where the optimal $\boldsymbol{Q}_*(\boldsymbol{s}_t, \boldsymbol{a}_t)$-value function can be defined as

$$\boldsymbol{Q}_*(\boldsymbol{s}_t, \boldsymbol{a}_t) \leftarrow \max_{\boldsymbol{\pi}\in\Pi} \boldsymbol{Q}_{\boldsymbol{\pi}}(\boldsymbol{s}_t, \boldsymbol{a}_t) , \tag{4.30}$$

$$\begin{aligned} \boldsymbol{Q}_{\boldsymbol{\pi}}(\boldsymbol{s}_t, \boldsymbol{a}_t) &\leftarrow \sum_{\boldsymbol{s}_t\in\mathbb{S}} \boldsymbol{t}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \left[\boldsymbol{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) + \gamma\, \boldsymbol{Q}_{\boldsymbol{\pi}}(\boldsymbol{s}_{t+1}, \boldsymbol{a} = \boldsymbol{\pi}(\boldsymbol{s}_{t+1}))\right] , \\ \boldsymbol{Q}_{\boldsymbol{\pi}}^{k+1}(\boldsymbol{s}_t, \boldsymbol{a}_t) &\leftarrow (1-\alpha^k)\boldsymbol{Q}^k(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha^k \boldsymbol{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) , \\ &\quad + \gamma^k \left\{\left[\boldsymbol{Q}_{\boldsymbol{\pi}}^k(\boldsymbol{s}_{t+1}, \boldsymbol{a} = \boldsymbol{\pi}(\boldsymbol{s}_{t+1})) - \boldsymbol{Q}_{\boldsymbol{\pi}}^k(\boldsymbol{s}_t, \boldsymbol{a} = \boldsymbol{\pi}(\boldsymbol{s}_t))\right]\right\} . \end{aligned} \tag{4.31}$$

which again holds if the environment system fulfills the MDP requirements. The difference between the $\boldsymbol{V}$-function and $\boldsymbol{Q}$-function is that the $\boldsymbol{Q}$-function bootstraps the state-action dependency and further aims to compute the optimal policy in a *model-free* way, i.e. without the necessity of any transition matrix $\boldsymbol{t}$. This is known as the basic version of the *Q-learning algorithm* which makes use of the fixed-point iteration of the Bellman equation and is defined for discrete state-action pairs. The Q-learning algorithm further is defined to converge under the assumption that all actions are repeatedly sampled within all accessible agent states [Watkins and Dayan, 1992, pp. 57–68].

Solving the Bellman Equation: Inserting Equation (4.28) in Equation (4.26) results in the recursive formulation of the Bellman expectation equation as follows:

$$\boldsymbol{V}_{\boldsymbol{\pi}}(\boldsymbol{s}_t) = \sum_{\boldsymbol{a}_t\in\mathbb{A}} \boldsymbol{\pi}(\boldsymbol{a}_t|\boldsymbol{s}_t) \left(\boldsymbol{R}_{\boldsymbol{s}_t}^{\boldsymbol{a}_t} + \gamma \sum_{\boldsymbol{s}_{t+1}\in\mathbb{S}} \boldsymbol{P}_{\boldsymbol{s}_t,\boldsymbol{s}_{t+1}}^{\boldsymbol{a}_t} \boldsymbol{V}_{\boldsymbol{\pi}}(\boldsymbol{s}_{t+1})\right) . \tag{4.32}$$

Equation (4.32) can be solved directly and optimally by calculating the inverse as follows:

$$\boldsymbol{V}_{\boldsymbol{\pi}}(\boldsymbol{s}_t) = (\boldsymbol{I} - \gamma\, \boldsymbol{P}_{\boldsymbol{\pi}})^{-1} \boldsymbol{R}_{\boldsymbol{\pi}} . \tag{4.33}$$

However, as Equation (4.33) shows, the Bellman Optimality Equation is non-linear and in general there is no efficient closed-form solution. Hence, it is necessary to use iterative solution methods e.g. *Value Iteration, and Policy Iteration* which are iterative dynamic programming methods. Such methods normally start with an initial estimate and compute iterative solutions that are hopefully closer to the optimum by updating themselves recursively. There are also iterative steps regarding matrix inversion.

4.10.2 Policy-based Methods

Policy-Evaluation, -Improvement and -Iteration: Computing the state-value function $\boldsymbol{V_\pi}$ for establishing a policy $\boldsymbol{\pi}$ is referred to as *policy evaluation* in DP literature and is commonly seen as prediction problem [Sutton and Barto, 1998, p. 60]. At each iteration of an iterative policy evaluation update, the value of every state is updated in order to yield a new approximation value function $\boldsymbol{V}_\pi^{k+1}$ for the next iteration $k+1$. The reason for computing the state-value or the action-value function is to find better policies. This depends on the observation of the current state $\boldsymbol{s}$, where we would like to know if the agent has to change its policy deterministically in order to achieve a better selection of actions, which may not be possible using the current policy $\boldsymbol{a}_t \neq \boldsymbol{\pi}(s)$. The key criterion for improving a deterministic policy is if a greater utility (expected return) value $\boldsymbol{V_\pi}$ or $\boldsymbol{Q_\pi} i$ than the current value can be achieved. In other words, the new policy must obtain a greater or equal expected return from all states $\boldsymbol{s} \in \mathbb{S}$ than the current expected return value. This is called the policy improvement theorem. In the presents of a stochastically generated policy $\boldsymbol{\pi}(\boldsymbol{a}_t|\boldsymbol{s}_t)$ the same idea applies if there are several actions at which the maximum expected return can be achieved. In the stochastic case, it is not required to select a single action among all possible actions as long as infeasible actions are given zero probability, and sub-optimal actions are given relatively less sampling probability [Sutton and Barto, 1998, p. 64]. If the policy $\boldsymbol{\pi}$ could be improved once, it might be possible to improve the policy $\boldsymbol{\pi}_*$ multiple times repeatedly to yield even better $\boldsymbol{\pi}_{**}$. Thus it is possible to formulate a policy trajectory planning problem, where a sequence of actions can be monotonically improved using the current state-value functions via:

$$\boldsymbol{\pi}_{k=0} \overset{E}{\rightarrow} \boldsymbol{V}_{\boldsymbol{\pi}_{k=0}} \overset{I}{\rightarrow} \boldsymbol{\pi}_{k=1} \overset{E}{\rightarrow} \boldsymbol{V}_{\boldsymbol{\pi}_{k=2}} \overset{I}{\rightarrow} \boldsymbol{\pi}_{k=2} \overset{E}{\rightarrow}, ..., \overset{I}{\rightarrow} \boldsymbol{\pi}_* \overset{E}{\rightarrow} \boldsymbol{V}_* \; . \tag{4.34}$$

Here, $\overset{E}{\rightarrow}$ and $\overset{I}{\rightarrow}$ denote the alternating principle of evaluation and policy improvement, where the policy is guaranteed to be updated only if a strict improvement could be reached until the policy is optimal reaching $\boldsymbol{\pi}_*$ and $\boldsymbol{V}_*$. This scheme of improving the policy until it is optimal is called *policy iteration* [Sutton and Barto, 1998, p. 64] which is the functional basis for the *policy gradient* algorithm which will be explained next.

Monte-Carlo Sampling Methods: The previous methods such as *Policy-Evaluation, -Improvement, and -Iteration* require the complete knowledge of the environment in a MDP. Alternatively, *Monte-Carlo Methods* aim to estimate the optimal value functions and policies only by interaction and *experience*. This is done by sampling states, selecting actions, and observing the outcomes from the simulated environment. Learning from actual experience is very advantageous because it does not require any prior knowledge of the environment dynamics, yet still, optimal behavior can be attained and learned [Sutton and Barto, 1998, p. 75]. Therefore, Monte-Carlo methods are based on the *Policy Iteration* scheme where the goal is to learn an approximate value function and the resulting approximate optimal policy. The respective value function and current policy thereby are repeatedly altered to more closely approximate the value function and are repeatedly

improved regarding the current value function. This is illustrated as follows:

$$\boldsymbol{\pi}_{k=0} \overset{E}{\rightarrow} \boldsymbol{Q}_{\boldsymbol{\pi}_{k=0}} \overset{I}{\rightarrow} \boldsymbol{\pi}_{k=1} \overset{E}{\rightarrow} \boldsymbol{Q}_{\boldsymbol{\pi}_{k=2}} \overset{I}{\rightarrow} \boldsymbol{\pi}_{k=2} \overset{E}{\rightarrow}, ..., \overset{I}{\rightarrow} \boldsymbol{\pi}_{*} \overset{E}{\rightarrow} \boldsymbol{Q}_{*} \ , \tag{4.35}$$

where $\overset{E}{\rightarrow}$ is a complete policy evaluation and $\overset{I}{\rightarrow}$ is the whole policy improvement. Over many learning and sampling epochs, the approximate action-value function approaches the real function in an asymptotic convergence process. At the beginning of an epoch, new variations of the policy are *explored*, whereas at the end of an epoch, the best policy choices are selected *greedily*. The policy *exploration-exploitation* principle generally is crucial to find superior valued policies. By observing the state outcomes based on the changes in the policy, Monte Carlo methods can be used to find the optimal policy and value function without any knowledge of the environment dynamics, given that the reward function can be rigorously defined.

Policy Gradient with Monte-Carlo Sampling (Rollouts): In Policy Gradient Methods combined with Monte-Carlo Search Methods, continuous Monte-Carlo sampling, and approximation over epochs is the key to establishing the agent's memory for the environment state and dynamics. The agent therefore trains the policy on a simulative environment in order to alter or improve its actions based on the causal state observations. Generally, the agent aims to alter the policy in a way that yields highly rewarding actions with high probability. It is assumed that future actions do not change past decisions and therefore the selected actions only have an impact on the future. Figure 4.18 illustrates the Policy Gradient Algorithm where the agent observes

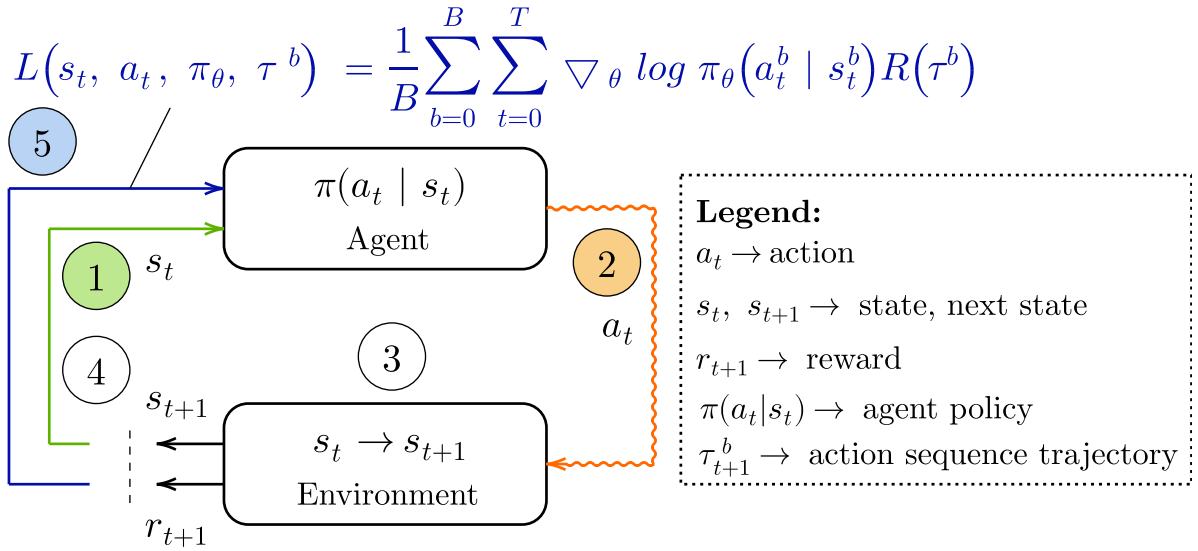


Figure 4.18: The dependencies of the Policy Gradient Algorithm [Levine, 2017].

1. the environment state $\boldsymbol{s}_t$,
2. selects an action sequence $\boldsymbol{a}_t$ based on its belief, i.e. the policy $\boldsymbol{\pi}(\boldsymbol{a}_t|\boldsymbol{s}_t)$ based on state $\boldsymbol{s}_t$,
3. the action changes the environment state, which results in a new environment state,
4. the agent takes new actions based on the new observable environment state, and finally,

5. the agent may adjust its belief, i.e. the policy $\boldsymbol{\pi}(\boldsymbol{a}_t|\boldsymbol{s}_t)$ based on the achieved total rewards $\boldsymbol{R}(\tau_t)$.

Hereby, $\boldsymbol{\tau}_t$ denotes the state-action sequences or also called the Monte-Carlo sampling of different trajectories $\boldsymbol{\tau}_t = \{s_0, a_0, ..., s_B, a_B\}$, where B denotes the batch size. The agent's objective can be formulated via

$$\boldsymbol{J}(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=0}^{T} \boldsymbol{R}(\boldsymbol{s}_t, \boldsymbol{a}_t)\right] = \sum_{b} \boldsymbol{P}(\boldsymbol{\tau}^b, \boldsymbol{\theta})\boldsymbol{R}(\boldsymbol{\tau}^b) \tag{4.36}$$

which is to maximize its expected rewards. Equation (4.36) formulates this as a sum over the product of the probabilities for taking a trajectory of actions $P(\boldsymbol{\tau}^b, \boldsymbol{\theta})$, times the reward for each individual trajectory $\boldsymbol{R}(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The individual policy parameters are stored in the so-called Jacobian matrix $\boldsymbol{J}(\boldsymbol{\theta})$. Later the derivations of the Jacobian provide are calculated using auto-differentiation algorithms such as stochastic gradient descent. Therefore the agent's objective is reformulated in an objective loss function that describes the performance of the agent, and further describes the changes in the policy in order to yield higher probabilities for high rewards as follows:

$$L(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\pi}_{\boldsymbol{\theta}}, \boldsymbol{\tau}^b) = \frac{1}{B}\sum_{b=0}^{B}\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log\left[\boldsymbol{\pi}_\theta(\boldsymbol{a}_t^b|\boldsymbol{s}_t^b)\right] R(\boldsymbol{\tau}^b) \tag{4.37}$$

where T is the time horizon that is bounding the planning time. This is required when using a NN for approximating the state or action spaces [Levine, 2017, pp. 682–697]. The gradient (partial derivation) of the Jacobian furthermore can be calculated with auto-differentiation via

$$\nabla_\theta \boldsymbol{J}(\boldsymbol{\theta}) \approx \frac{1}{B}\sum_{b=0}^{B}\left\{\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log\left[\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{a}_t^b|\boldsymbol{s}_t^b)\right]\right\}\left\{\sum_{t=0}^{T} R(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b)\right\} \tag{4.38}$$

which is the $\sum_{t=0}^{T} \nabla_\theta \log\left[\boldsymbol{\pi}_\theta(\boldsymbol{a}_t^b|\boldsymbol{s}_t^b)\right]$ is called the maximum log-likelihood. This term measures the likelihood of the observed environment states, and therefore states how likely it is to compute an action sequence trajectory under the current policy $\boldsymbol{\pi}_\theta(\boldsymbol{a}_t^b|\boldsymbol{s}_t^b)$. The product with the achieved action sequence rewards, the objective tends to increase the likelihood of actions under the current policy for high-rewarding action sequences. On the contrary, negative action sequences are reduced in probability in the same manner. This formalizes the agent's notion principle of "trial and error" which enables the agent to find highly rewarding actions with high probability [Levine, 2017, p. 13].

Policy Gradients Issues and Improvements: A significant aspect of the Policy Gradient algorithm is that the probabilities of the action sequences are characterized by long-term dependencies. In other words, long action sequences strongly correlate and therefore may vary strongly in structure and order. This is because the probabilities for selecting an action sequences

are calculated via multiplication:

$$\boldsymbol{\pi_\theta}(\boldsymbol{\tau}) = \boldsymbol{P}(\boldsymbol{s}_1) \prod_{t=1} \boldsymbol{\pi_\theta}(\boldsymbol{a}_t|\boldsymbol{s}_t)\boldsymbol{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) \tag{4.39}$$

which is stated to trigger vanishing and exploding gradient issues easily when combined with NN function approximation. However, this issue can be solved by only summing up the gradient instead, which avoids the necessity of computing the product of long probability sequences respectively. The trick works as follows:

$$\boldsymbol{\pi_\theta}(\boldsymbol{\tau}) \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi_\theta}(\boldsymbol{\tau}) = \nabla_{\boldsymbol{\theta}} \boldsymbol{\pi_\theta}(\boldsymbol{\tau}) \tag{4.40}$$

where using the logarithm operator enables to avoid the computation of the product, but instead can be reduced to a sum over all probabilities instead. This being said the basic variant of the REINFORCE algorithm [Williams, 1992, pp. 319–350] using Monte-Carlo Tree Search (MCTS) Rollouts can be formulated as follows: The MCTS sampling method further generates a whole

Algorithm 1: REINFORCE Algorithm with MCTS Rollouts.

1 **Sample** *action sequence trajectory* τ^b *from* $\boldsymbol{\pi}_\theta(\boldsymbol{a}_t^b|\boldsymbol{s}_t^b)$,
evaluate the policy from the actual observed environment state
2 **Update** $\nabla_\theta J(\theta) \approx \frac{1}{B} \sum_{b=0}^{B} \left[\sum_{t=0}^{T} \nabla_\theta log\, \boldsymbol{\pi}_\theta(\boldsymbol{a}_t^b|\boldsymbol{s}_t^b)\right] \left[\sum_{t=0}^{T} R(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b)\right]$
3 **Update** $\theta \leftarrow \theta + \alpha \nabla_\theta \boldsymbol{J}(\theta)$

action sequence trajectory and memorized the exact rewards of the action trajectory. Moreover, the MC sampling method is required to approximate the gradient of the Jacobian Matrix $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. A particular characteristic of the Policy Gradient Method is, that the stochastic policy can vary concisely for different episodes and small differences in the state observations can completely adjust the policy. This is called high variance with respect to the used network optimization algorithm, where volatile gradient updates (gradient descent directions) detain the machine learning model from converging and therefore from learning. In detail, each action sequence can have large differences in the achieved reward, which leads to large gradient fluctuations and therefore deteriorates the convergence process when using a NN for function approximation. There are several options to mitigate high learning variants. The first is to increase the used batch size B, however, if B is chosen too high the sample efficiency is reduced and therefore the machine learning model requires a long time to learn. Thus, the batch size cannot be increased too far, and additional measures are required for reducing the learning variance. Another way is reducing the magnitude of sampled rewards, which is straight-forward. The high variance generally originates from high sensitivity of the model towards minor changes in the training data and state observations. This results in an overestimation (over-adjustment) of the computed gradients. The gradients, therefore, may fluctuate to a degree, where the gradient updates fail to minimize the objective loss function and the stochastic gradient descent algorithm fails to converge. This ultimately reduces the learning capability of the learning model. Conversely, a learning model also may witness high bias, which is the opposite bookend. The gradients are not updated enough (under-adjustment) so the model fails to learn the mapping from states to

action sequences. Another way of detaining the gradient from large magnitude updates is to introduce a so-called baseline, which is the basis for Actor-Critic Methods in the next subsection.

4.10.3 Actor-Critic Methods

The major difference between the Advantage Actor-Critic Algorithm and the REINFORCE policy gradient algorithm is the components 3 and 5 in Figure 4.19. As the Figure shows Actor-Critic Methods introduce a second module called Critic with a so-called baseline function in order to mitigate large gradient update effects, i.e. so-called high learning variance. In order to reduce

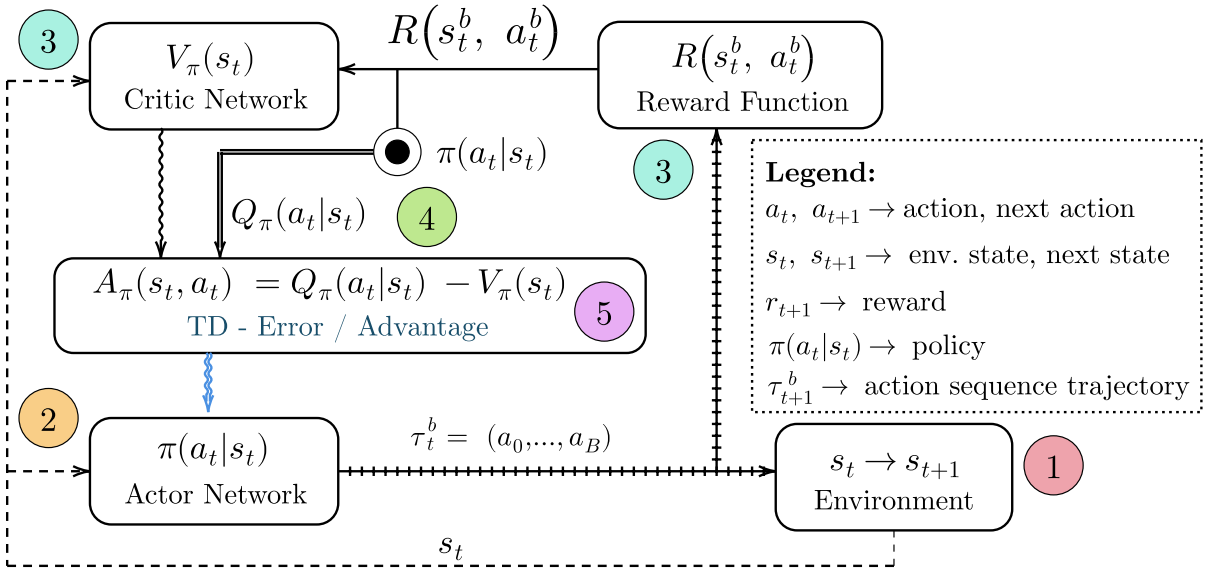


Figure 4.19: The dependencies of the Advantage Actor-Critic Algorithm [Levine, 2017].

the high variability of computed gradients an auxiliary network, called *Critic*, can be used. The Critic further evaluates the expected results of a change in the Actor's gradient, and estimates or predicts if the solution is becoming better according to the changed Actor policy and action selection. In order to do so, the Critic observes the environment state and stores a smoothed version of the achieved reward, which originates from the previous action of the Actor Network. The major difference can be seen in Figure 4.19 and Component 5. Here the so-called advantage function is processed, which essentially is the difference between the total reward of the actor and the smoothed state-value reward from the Critic Network component. The original baseline function is mathematically formulated as follows:

$$\nabla_\theta J(\theta) \approx \frac{1}{B} \sum_{b=0}^{B} \left[\sum_{t=0}^{T} \nabla_\theta \log \boldsymbol{\pi}_\theta(\boldsymbol{a}_t^b | \boldsymbol{s}_t^b) \right] \underbrace{\left[\sum_{t=0}^{T} R(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b) \right]}_{\boldsymbol{Q}(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b)}, \tag{4.41}$$

where the total actor reward $\boldsymbol{Q}(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b) = \sum_{t=0}^{T} R(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b)$ is extended with the state-reward value function $\boldsymbol{V}(\boldsymbol{s}_t^b)$ from the Critic Network as follows:

$$\nabla_\theta J(\theta) \approx \frac{1}{B} \sum_{b=0}^{B} \left[\sum_{t=0}^{T} \nabla_\theta \log \boldsymbol{\pi}_\theta(\boldsymbol{a}_t^b | \boldsymbol{s}_t^b) \right] \underbrace{\left[\boldsymbol{Q}(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b) - \boldsymbol{V}(\boldsymbol{s}_t^b) \right]}_{\boldsymbol{a}(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b)} . \tag{4.42}$$

Advantage Actor Critic: Furthermore, the blue underline in Equation (4.42) defines the advantage function $\boldsymbol{a}(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b)$ which is used to limit the fluctuation of gradient updates and is defined by

$$\boldsymbol{a}(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b) = \boldsymbol{Q}(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b) - \boldsymbol{V}(\boldsymbol{s}_t^b) \tag{4.43}$$

where the final gradient update equation can be reformulated by inserting the advantage function as follows:

$$\nabla_\theta J(\theta) \approx \frac{1}{B} \sum_{b=0}^{B} \left[\sum_{t=0}^{T} \nabla_\theta \log \boldsymbol{\pi}_\theta(\boldsymbol{a}_t^b | \boldsymbol{s}_t^b) \right] \left[\boldsymbol{a}(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b) \right] . \tag{4.44}$$

In Deep Learning, the input features should be zero-centered. This means that RL generally is concerned about whether an action is performed better than the average. If rewards are defined to be always positive $R(\boldsymbol{s}_t^b, \boldsymbol{a}_t^b) > 0$, the Policy Gradient algorithm tries to increase the action sequence or trajectory probability of getting high rewards, even if the algorithm receives much smaller reward values than for previous learning epochs. Let's consider two scenarios where

- (i) the action sequence A receives a reward value of +100 and the action sequence B only receives a reward value of -10,
- (ii) the action sequence A receives +100 and action sequence B achieves a reward value of +5.

In the first scenario (i), the Policy Gradient algorithm increases the probability of sampling the action sequence A for achieving the reward value of +100, while it decreases the probability of sampling the action sequence B which results in a negative reward -10. In the second scenario (ii), the Policy Gradient algorithm will increase both relatively of their reward, since both rewards are positive. However, if a human would decide which trajectory to increase in probability, it would be very likely to decrease the sampling of the action sequence B in both scenarios (i) and (ii). This is the objective of the introduced critic baseline such as $\boldsymbol{V}(\boldsymbol{s}_t^b)$ which adjusts the action sequence selection to achieve rewards that are superior to average action sequences. More details about the development and implementation of a DRL architecture with Encoder-Decoder, PGN, Advantage Actor-Critic training methods, and Policy Gradient Optimization will be presented in the next chapter. The choice of this architecture is based on the remarkable results this architecture has achieved for the optimization of TSPs [Bello et al., 2016].

5. Algorithm and Model Implementations

The previously introduced algorithms for combinatorial optimization are the current state-of-the-art approaches and will be important for comparing algorithms further on course. The current state-of-the-art approaches do not provide scalable and comparably flexible mechanisms to generalize (transferability) towards changes in requirements, i.e. changing the operation environment, problem constraints, and other problem formalism characteristics. This however, is essential for algorithm applications that are inherently changing, such as for fleet planning and operation, which will be investigated later. First, this chapter aims to develop and provide implementation details with respect to a DRL model that has the potential to solve the aforementioned issues. In DL and RL terminology, a learning component is, in general briefly referred to as an "Agent". In this thesis, the Agent comprises the "Actor" and "Critic" networks, which are required for learning to assign near-optimal combinations of vehicles, requests, and service stations for a given vehicle fleet (see Figure 5.1). In more detail, the required components comprise

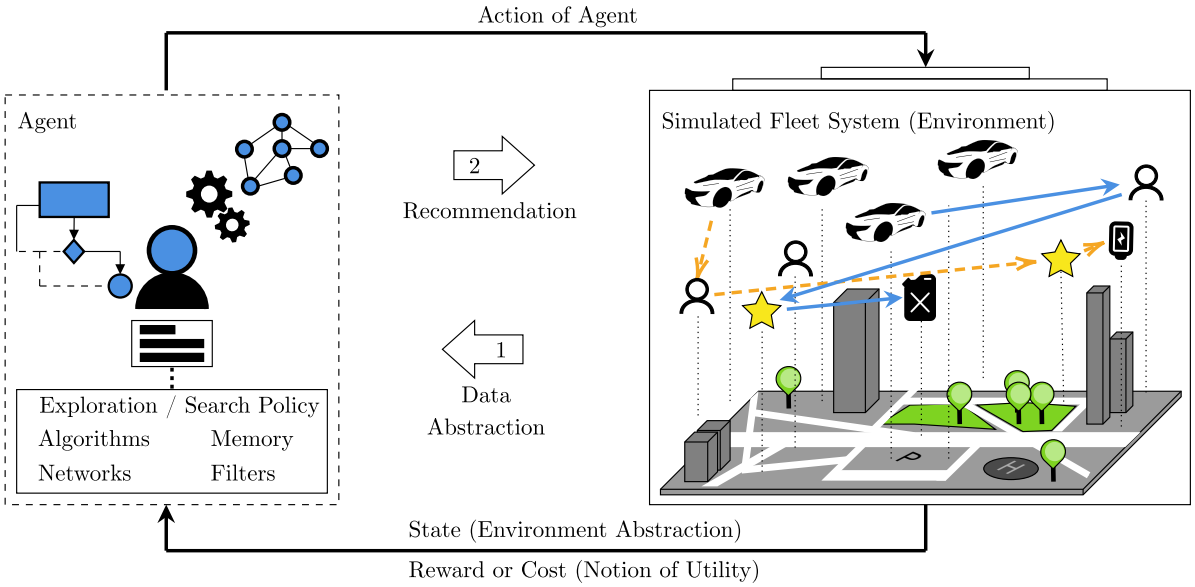


Figure 5.1: A basic Agent that learns good assignments for a given fleet simulation.

- a basic reinforcement learning framework, where the Agent receives environment data (fleet and traffic) as abstracted state, computes a strategy to assign good requests, and sends the

strategy in the form of a sequence of actions to the customer and fleet.
- The customer receives a recommendation as to which vehicle will pick up and drop-off the customer.
- Finally, the vehicle receives a recommendation of a suited service/charging station based on its State-Of-Charge (SOC).

Further, the whole implementation contains a large list of components and modules, which are explained in more detail in this chapter. For first glance, a holistic architectural overview of the implemented modules can be accessed in Section 5.1.

5.1 Architectural Overview

The following architecture essentially has two separate networks that are used as memory structures (see Figure 5.2). Here, the Actor-Critic architecture is used to explicitly model the

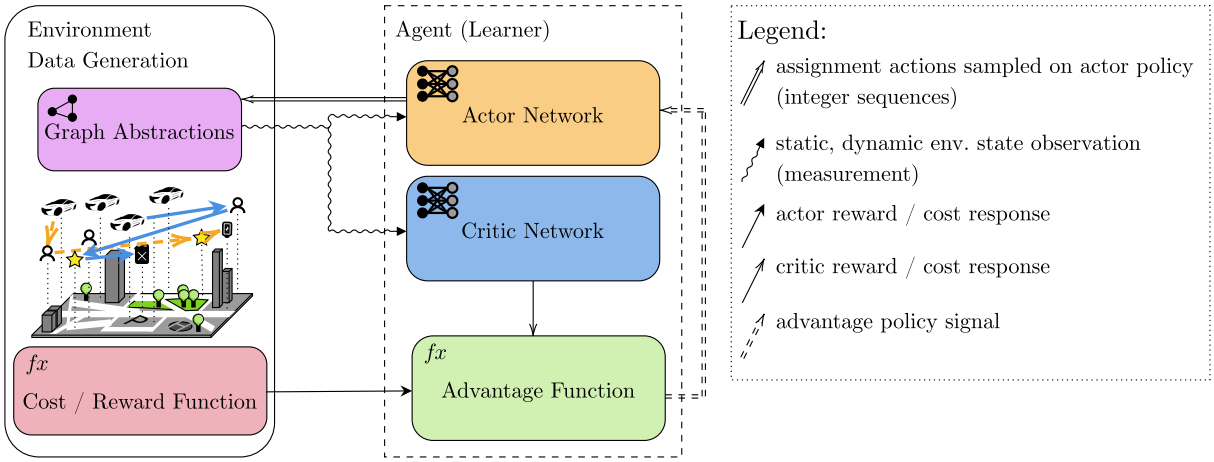
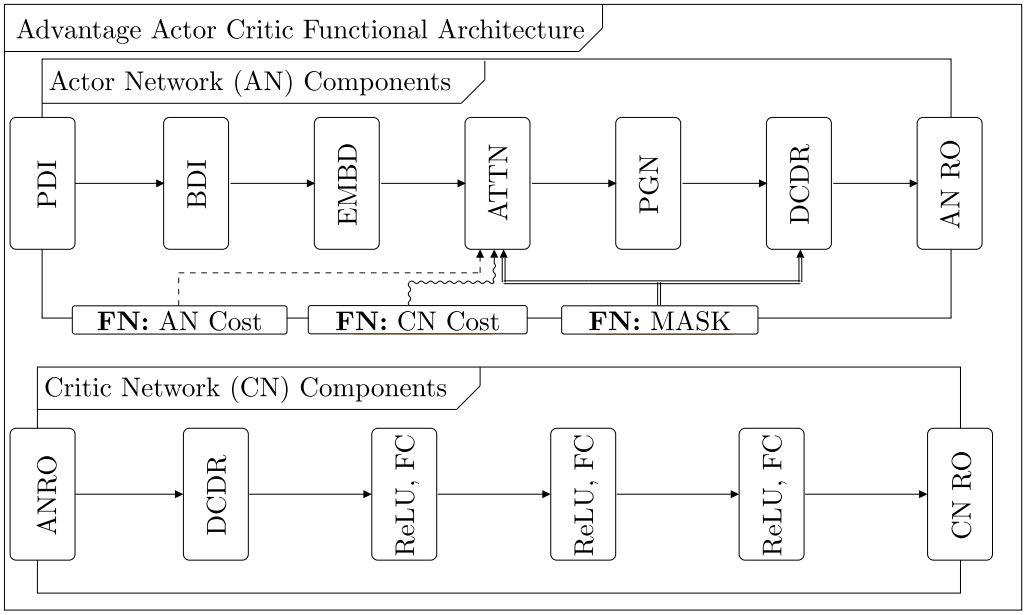


Figure 5.2: The Actor-Critic networks and their training architecture.

Actor network, which aims to learn a policy (that is, for example, a set of heuristic rules or strategies resulting in a good combinatorial solution), while the Critic network mitigates an over- or underestimated policy and thus mitigates greedy actions by the Actor network (see Figure 5.3 and Figure 5.4). The Critic network (also Figure 5.3) should, furthermore learn about the current policy (i.e. a subjective evaluation of the current solution) and also provides a Critic response signal that can be expressed as a scalar error value (also sometimes referred to as the loss value). The Critic response scalar signal is further processed by the advantage function and later provides information about the learning progress of both Actor and Critic network. Note, that more detailed explanations and fundamentals about Actor-Critic methods can be accessed here [Sutton and Barto, 1998, p. 257]. As well, a more detailed list of the components used is shown in Figure 5.5. The most important architectural components are

- the Data Generation module (see Section 5.2) which is responsible for generating arbitrary datasets of TSPs and assignment problems such as MWM problem instances,



Legend:
AN: Actor Network, **CN:** Critic Network, **FN:** Function,
PDI: Processed Dataset Input, **BDI:** Batched Dataset Input,
EMBD: Batch Embedding/Encoding Module, **ATTN:** Attention Mechanism,
PGN: Pointer Generation Network, **DCDR:** Decoder Module,
MASK: Neuron Mask Function, **RO:** Response Output

Figure 5.3: The Actor and Critic network modules and their architecture.

- the Routing API, which provides required data features such as travel times and travel distances,
- the DRL model implementation with respective submodules, and
- the Mathematical MIP and Heuristic Solver implementations for evaluating the solution quality and efficiency.

The extended DRL model developments are based on the previous work of [Nazari et al., 2018] which contains a set of sub-modules such as

- the Encoder Module, see Section 5.3,
- the Attention Module, see Section 5.4,
- the Pointer Network Module, see Section 5.5,
- the Decoder Module, see Section 5.6, and
- the AACMCPG Training Algorithm 5.10 for training the NN architecture.

The following sections will provide further details about the data generation module and how data abstractions are processed in order to serve as input for the Actor and Critic network modules.

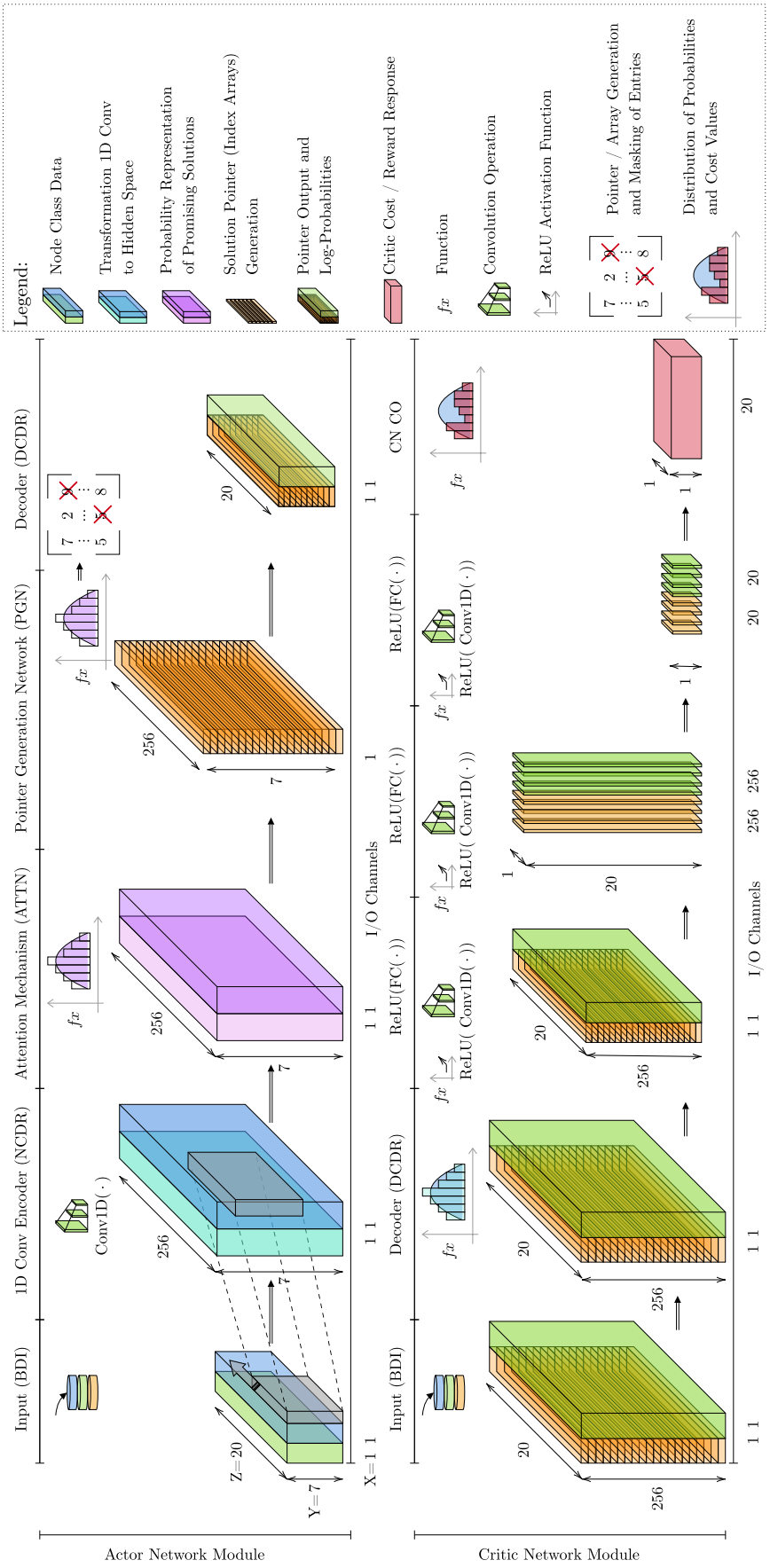


Figure 5.4: The Actor and Critic network modules and individual dimensions.

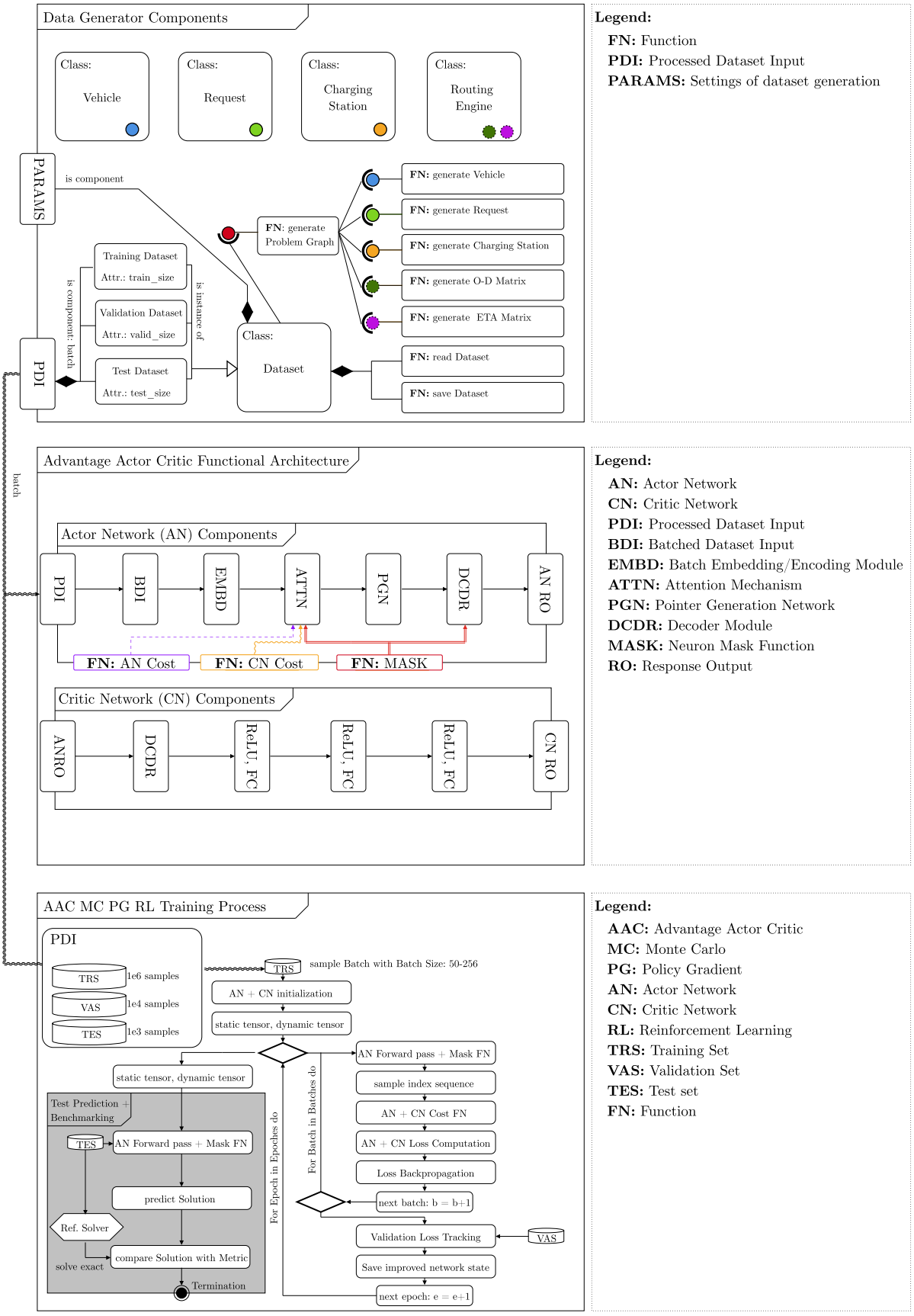


Figure 5.5: The Data Generation Class with data generator modules and the Advantage Actor-Critic Architecture components.

5.2 Data Generation and Processing

The "Data Generation Class" is comprised of data generation components (see Figure 5.5), where its residual subclasses and subfunctions can be used to efficiently generate arbitrary amounts of symmetric and asymmetric TSPs (see Subsection 5.2.1), as well as balanced, unbalanced, and constraint MWBMs graphs in Subsection 5.2.2. Technically, the generation of one million MWBM graphs requires to querying the Data Generation Module 1 million times as well. Hence, the efficiency of the Data Generator has a very significant and direct proportional effect on the complete generation time per training, validation, and test dataset.

5.2.1 Generation of TSPs

Symmetric Euclidean TSP: In more detail, artificially generated TSPs are defined in unit Cartesian $\mathbb{R}^2$ space, which enables to conduct of comparable, reproducible datasets and experiments (see Figure 5.6). Such datasets are later used to generate training, validation, and tests with up to $4M$ = 4e+06, 1e+04, and 1e+03 TSP samples. Thereby, the x and y coordinates are random-uniformly sampled within floating point ranges of $]0, 1[$ so that the coordinates never reach the value of 0.0 and 1.0. The module further enables to generate arbitrary TSP node sets and therefore arbitrary complex TSP problem datasets.

Asymmetric Euclidean TSP: The difference of asymmetric TSPs to symmetric TSPs (see Figures 5.6 and 5.7) is that bidirectional link assignments result in different costs, e.g. node cost 0 to 1 is different from 1 to 0. In order to generate asymmetric TSPs in unit Cartesian $\mathbb{R}^2$ space the individual bidirectional edges are characterized by different bidirectional distances. In order

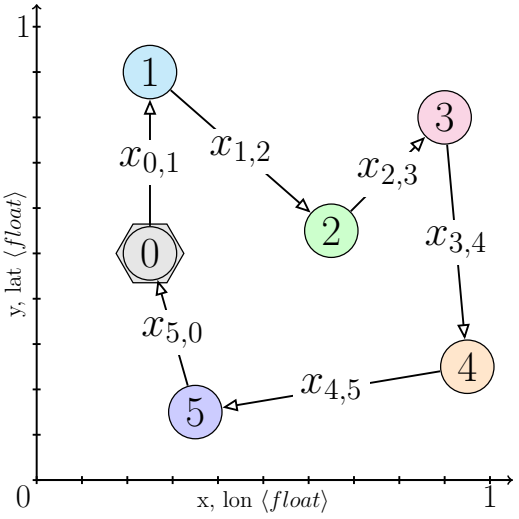


Figure 5.6: Artificial symmetric TSP in unit Cartesian $\mathbb{R}^2$ space.

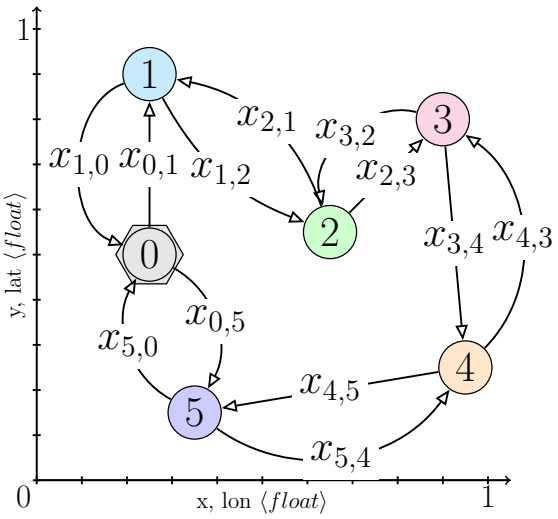


Figure 5.7: Artificial asymmetric TSP graph with different bidirectional costs.

to generate such a TSP graph, the distances within the lower triangle of an O-D matrix need to

be altered by small proportions of the original edge length. For this reason, the Data Generator Class also contains a function that artificially alters the bidirectional edge length, so that the direction of TSP graph traversal results in different distance values. As for the symmetric TSP graph setting, the function enables to sample arbitrary TSPs with random uniformly distributed xy-coordinates within floating point ranges of $]0, 1[$ in unit $\mathbb{R}^2$ space.

Solution Structure of TSPs: Generally, all TSPs, i.e. symmetric and asymmetric TSPs are solved by finding the order of node assignments that yield the lowest cost of the Hamiltonian Path (i.e. one round-trip traversing all nodes once). Hereby, the nodes that are visited are added in sequences (see Figure 5.8). The difference between symmetric and asymmetric TSPs solutions is that fewer solution sequences will result in the same minimal cost. This characteristic, however, is ubiquitous in real road networks and real route planning. Hence, it is interesting to model such problem characteristics for future learning and testing processes.

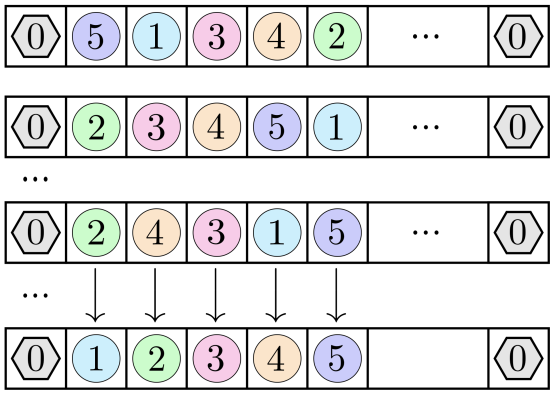
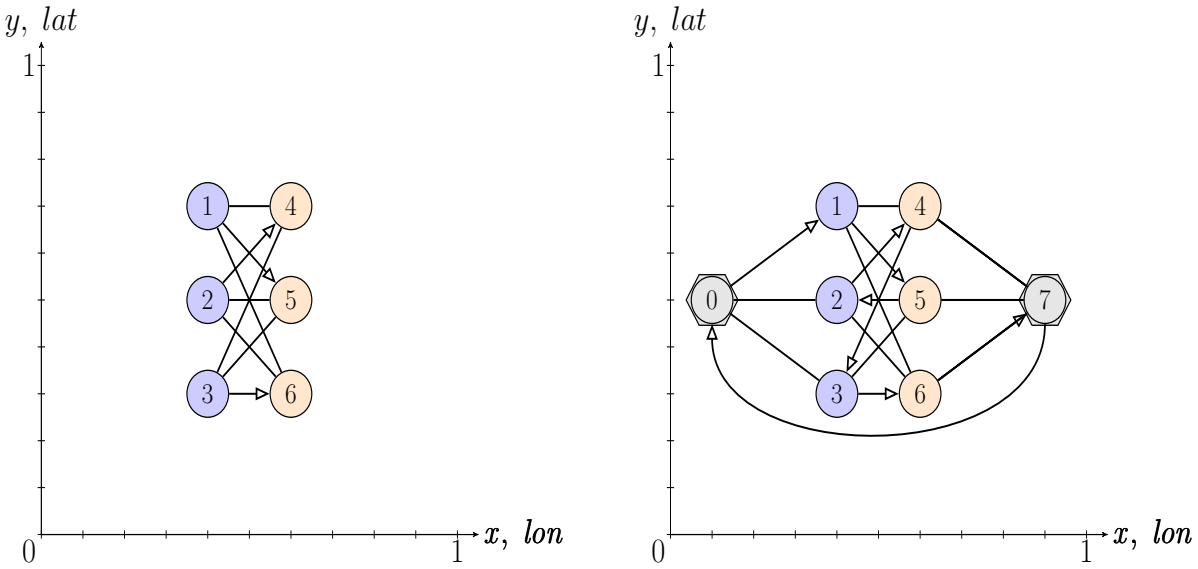


Figure 5.8: Learning of solution permutations as valid TSP solutions.

5.2.2 Generation of MWMs and PDPs

MWMs and PDPs: The generation of MWBMs graphs differs from the TSP by the selection of graph links and thus assignment actions that can be learned and selected by the Actor network. Instead of the Hamiltonian Path, the goal is to find the maximum flow with least cost via the shortest augmenting path (see Figures 9a and 9b). The structural relationship of TSP and MWBM allows for representing an MWBM solution in the same solution structure as for the TSP, with the only difference being that some links are omitted as shown in Figure 9a. General MWBM graphs can have multiple characteristics, i.e. symmetry, asymmetry, balanced and unbalanced, or even constraint instances, as previously shown in Figure 2.2 in Chapter 2, Section 2.5. The characteristics of symmetry have already been explained in Section 5.2.1. Additionally, there is to be mentioned that MWBM graphs can vary in the number of nodes for *from_nodes* (vehicles) and *to_nodes* (requests). Additionally, the links between *from_nodes* and *to_nodes* may be represented as a constraint where a cost value may result in an invalid assignment action, of which an example can be seen in Figure 5.10. There are multiple methods to model constraint



(a) Solution of Vehicle-Request MWBM Graph.

(b) Vehicle-Request MWBM Graph as TSP.

Figure 5.9: Conversion of Vehicle-Request MWBM graph to a TSP tour.

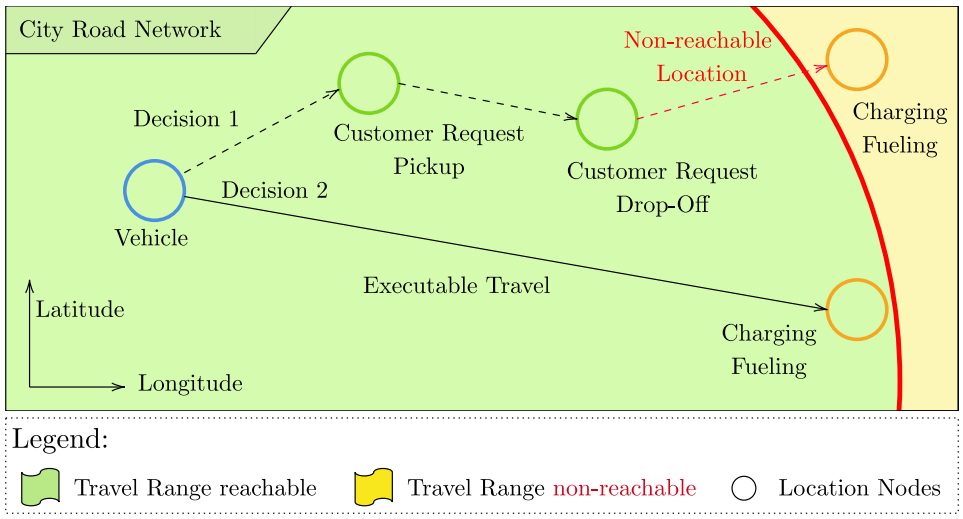


Figure 5.10: Valid and invalid assignment actions for one vehicle node.

MWBM graphs. First, the link cost can be represented as an infinite high-cost value, second invalid assignments can be assigned to dummy nodes. And finally, the link may be filtered and omitted using a mask function. For instance, Figure 5.11 shows such typical unbalanced MWBM graphs, where blue nodes represent vehicles, and green nodes represent occurring client requests. The Figure also shows that the number of vehicle and request nodes may vary, which results in the difficulty that some vehicles or requests must be omitted from the decision problem. By using a directed graph formulation the cost matrix for a MIP formulation typically results in a rectangular cost matrix for an unbalanced number of nodes. Generally, most graph problems

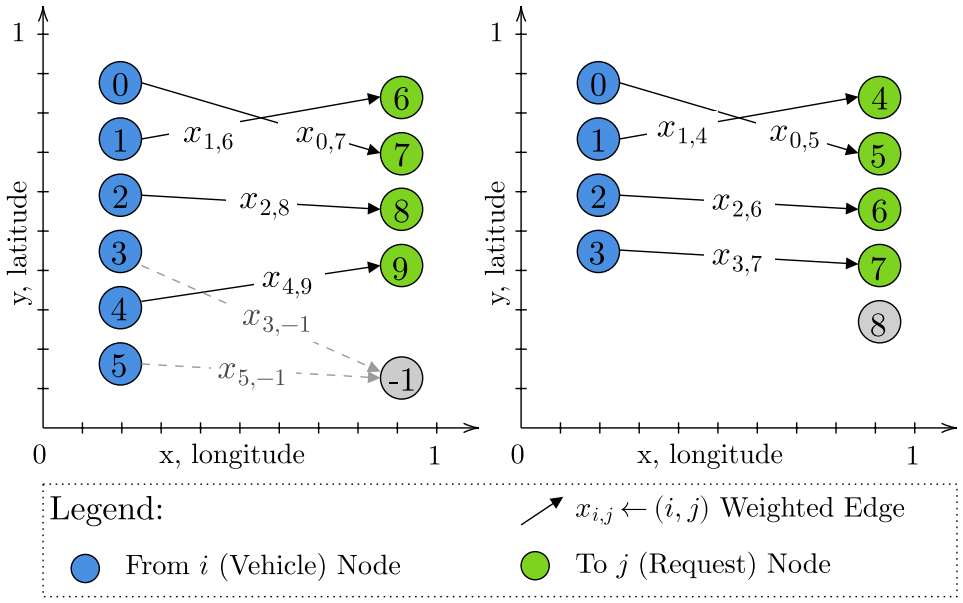


Figure 5.11: Unbalanced MWBM graph for vehicle-request assignments.

occurring in a practical application are typically unbalanced in the sense that the available number of vehicles does not equal the number of requests and charging stations. This can be artificially handled by inserting the number of differing numbers of nodes as dummy nodes for balancing the graph problem. Depending on the MWBM graph problem size, the use of dummy nodes, however, may be costly. This is due to the fact that adding dummy nodes also increases the memory and computation time required for algorithms to solve large individual graph problems. Clearly, defining directed graphs, which avoids empty entries in cost matrices, helps to reduce the graph complexity. This furthermore saves memory and computational resources, especially when being stored in Python dictionaries. However, the MWBM graph can be additionally represented in a MIP formulation as a linear assignment cost sum problem. In other words, here the goal is to calculate the sum of assigned edges that result in the minimization of the global cost. The selection of edges is modeled by selecting the correct combination of decision variables $x_{i,j}$ to minimize the sum of all assignments *from node i to node j*. Next, the generation of real data MWBM graph samples is explained in more detail.

5.2.3 Generation of MWMs with Real Data

The generator modules for generating MWM graph instances are depicted in the component diagram in Figure 5.12. Generally, the modules are necessary to generate the required (real data) datasets for PN training, validation, and testing. In order to be able to generate rather complex graph problem instances the implementation contains a list of modules as follows. First, Figure 5.12 shows the generator classes for generating individual vehicle nodes, request nodes, and charging state nodes for each matching graph. Depending on the settings captured in the *settings.py* component (see Figure 5.12), it is possible to generate arbitrary numbers of arbitrary complex graphs for assignment, planning and matching applications (see Figure 5.13). For

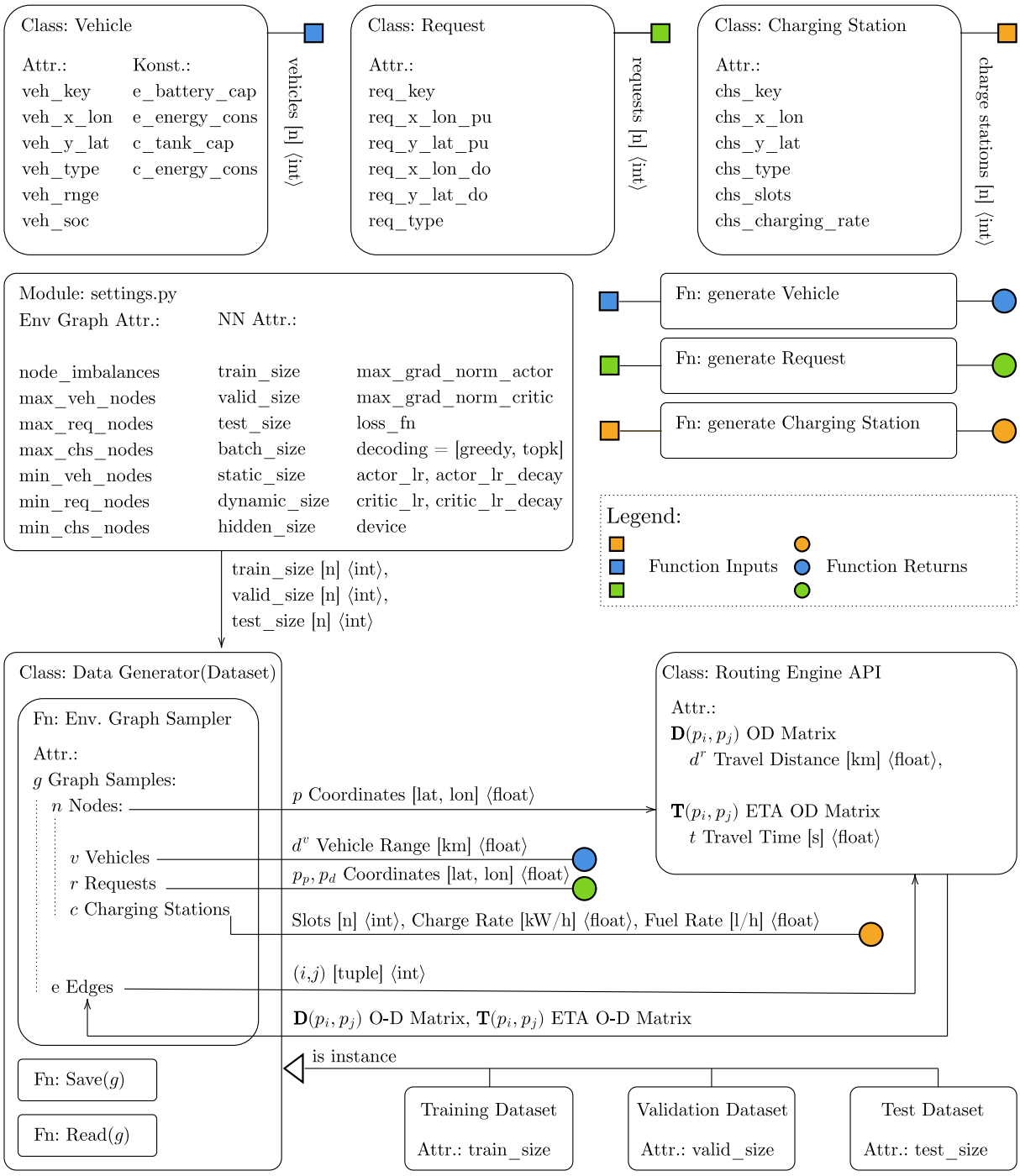


Figure 5.12: A component and data flow chart of the data generator module.

the training process with MWBM graphs the most important parameters are the *train_size*, *valid_size*, *test_size*, and the individual numbers of nodes for generating sets of vehicles, requests, and charging stations. The relevant data features are extracted from the New York Taxi data set offered by the [NYC Taxi and Limousine Commission (TLC), 2020] and aggregated to yield different graph problems for training the PN. The aggregated MWBM graphs and their

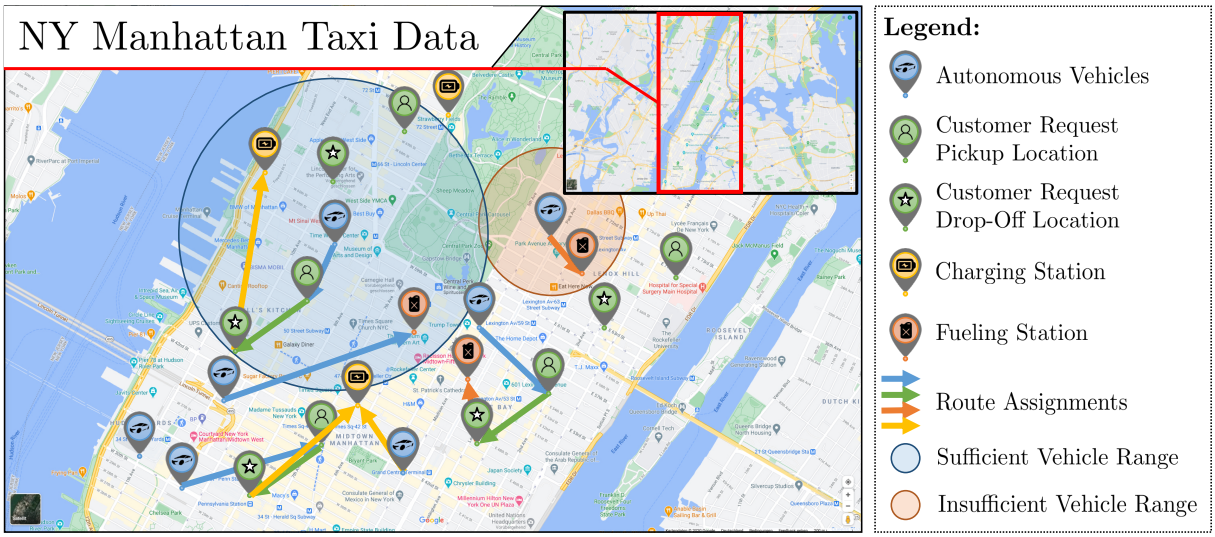


Figure 5.13: The generation of different assignment-, planning-, and matching-graphs.

contextual features must be represented as tensors in order to serve as network input. This is further explained as follows.

Data Input Features: The aggregated TSP, MWBM graphs and their contextual features can be stored in static tensors with static graph features, such as indices and the coordinates of nodes (i.e. vehicles, requests, service or charging stations, the quantities and types of nodes, etc.). On the other hand, dynamic TSP and MWBM graph features such as distances, ETAs, vehicle ranges, capacities, demands, and supplies can be stored in dynamic tensors. Both static and dynamic tensors serve as containers for the Actor networks' input. Next, the individual node classes and their features will be explained as follows.

Vehicle Class: The vehicle class is a programmed object that models a vehicle with a vehicle node $\mathcal{V}$ with its attributes such as the vehicle key *veh_key* as ID, longitude and latitude coordinates *veh_x_lon* and *veh_y_lat*, and the vehicle type *veh_type* with electric and combustion engine settings. Furthermore, each vehicle node instance can have dynamic features such as the individual vehicle driving range *veh_range* based on the vehicle SOC *veh_soc*, which is calculated from the energy/fuel capacity of the vehicle type and their individual energy consumption. Obviously, the calculation of the electric vehicle range is different for electric and combustion engine-driven vehicles. The calculation of the residual energy capacity *e_battery_cap* differs from a combustion-powered vehicle capacity *c_energy_cap*. Furthermore, different consumption parameters *e_energy_cons*, *c_energy_cons* are required to estimate realistic but approximate driving ranges for electric and combustion-driven vehicles. Importantly, when calculating driving ranges the resulting values can have a lot of variance and uncertainty. This requires building additional buffering means which will not be elaborated on in this thesis.

Since the NY Taxi dataset does not provide any information about the used vehicle characteristics, this thesis uses a generation process via sampling different vehicle characteristics. This

is explained as follows, where different vehicle configurations are generated with individually different attributes. The electrical SOC v_s^e and fuel SOC v_s^c is an integer value sampled within a percentage in the range of $[0, 100]$, which is the basis for estimating the *electrical* approximate driving range $\rho_v^e[km]$ via

$$\rho_v^e = \frac{v_s^e \cdot c_v^e}{100} \cdot \frac{1}{v_c^e}\,, \tag{5.1}$$

and the *fuel* driving range $\rho_v^f[km]$ via

$$\rho_v^f = \frac{v_s^c \cdot c_v^c}{100} \cdot \frac{1}{v_c^c}\,, \tag{5.2}$$

for combustion engine-driven vehicles. Furthermore, $c_v^e = 50[kWh]$ and $c_v^c = 97[l]$ are electrical and fuel-based total and nominal battery/fuel tank capacities of individual vehicle types. The parameters $v_c^e = 17[kWh]/100[km]$ and $v_c^c = 14[l]/100[km]$ denote individual vehicle consumption parameters for either electrical or combustion-driven vehicle types. However, also additional or other vehicle-based data features such as capacities (seat occupancy or cargo) can be modeled in a similar way. Next, the generation of requests via the request class is explained.

Request Class: The request class is required for modeling a request node $\mathcal{R}$ and its associated attributes, such as the request key *req_key*, *req_x_lon_pu*, *req_y_lat_pu*, *req_x_lon_do*, *req_y_lat_do*, and *req_type*. The attributes again define the request object with an ID key, the pickup coordinates (*req_y_lat_pu*, *req_x_lon_pu*), the drop-off coordinates (*req_y_lat_do*, *req_x_lon_do*), and the individual request type *req_type*. As the final node class the generation of service stations and charging stations will now be explained in the following.

Charging Station Class: The last of the three node classes is the class "Charging Station" for modeling the present city charging stations as the object with individual attributes. Each object instance of the class Charging Station is equipped with the individual attributes *chs_key*, *chs_y_lat* *chs_x_lon*, *chs_type*, *chs_slots*, *chs_charging_rate* for modeling a charging station and its capacities. More specifically, the attribute *chs_slots* can be used for modeling individual vehicle capacities of the charging station, whereas the *chs_type* and *chs_charging_rate* can be used to mode electrical and gasoline charging stations with individual electrical and fuel charging rates. Since the generation of node classes and assignments comes with many features and complex dependencies, it is important to reduce the graph and contextual complexity when training the Actor and Critic network. A detailed explanation of the process of reducing the complexity and finding similarities between individual TSP and MWBM graph problems is explained in the following section.

5.3 Graph-Contextual Embeddings and Encoding

In general, Graph-Contextual Embeddings, or in short Embeddings, as visualized in Figures 5.14 and 5.15, are used for mapping complex data features or non-numerical data to numerical space within the Encoder module. This is required in order to make the Actor and Critic networks

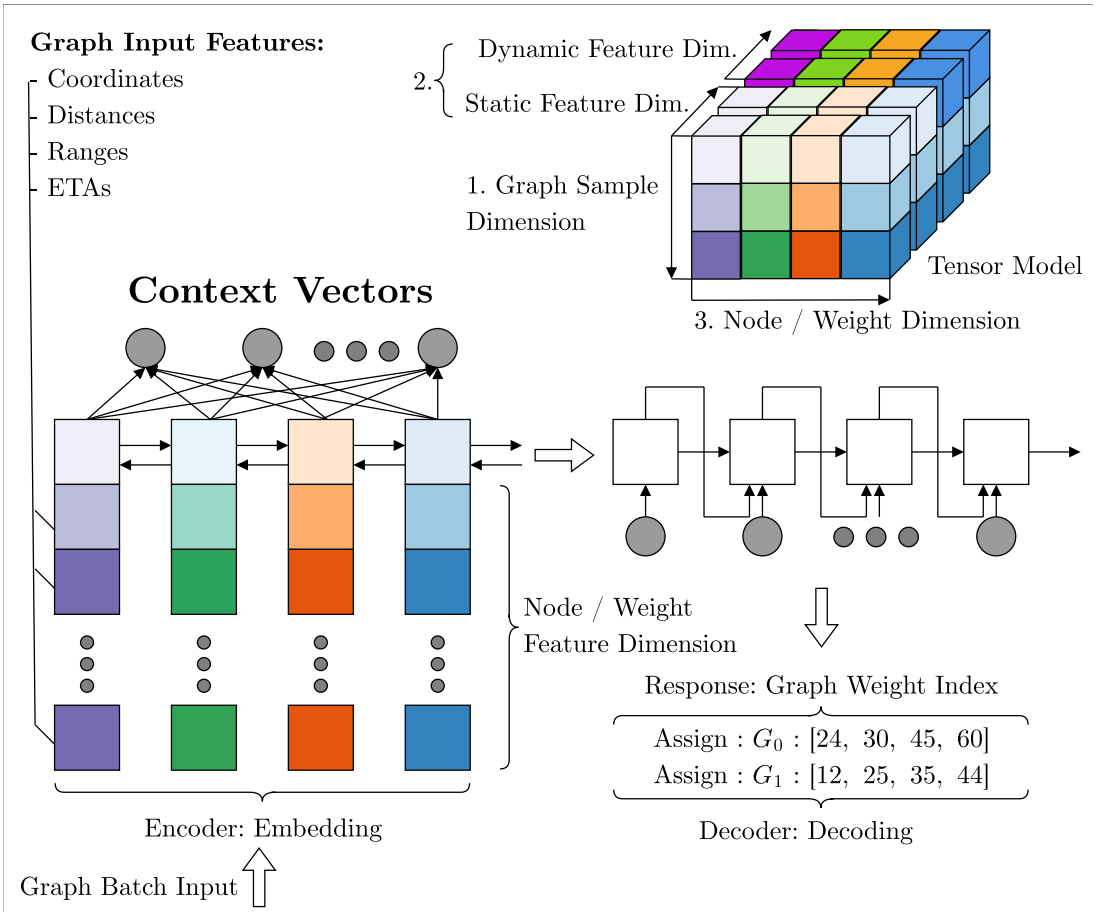


Figure 5.14: The extraction of graph features and the following feature Embedding. The Encoder module is used to learn the contextual and structural components and inter-dependencies of the given Minimum Weighted Bipartite Matching (MWBM) graphs samples. Here, the Encoder is used to learn the contextual information from the static and dynamic tensors which contain the actual graph information. In contrast, the Decoder is used to retrieve the learned assignment indices based on the learned assignment probabilities by the Attention Mechanism.

understand how intrinsic graph features differ by structure (cohesion) or costs (weighted link distributions), even if there is no programming notion of comparing the training graphs yet. For this purpose, Embeddings are used for extracting relevant and useful data in order to transform useful graph features into a vector space. Subsequently, its structures and contextual properties may be learned by a network. Thus, Embeddings are also an efficient method of reducing the input dimensionality of the training graphs, while preserving essential information required for

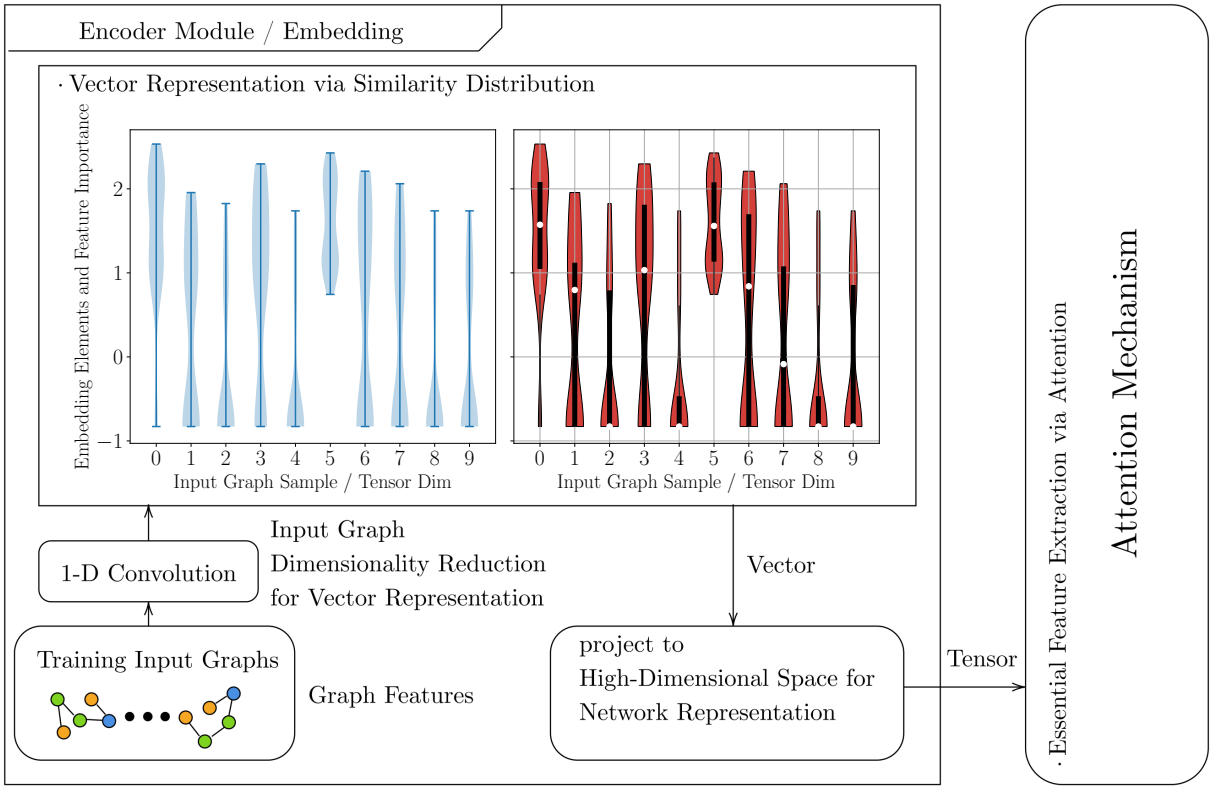


Figure 5.15: The extraction of graph features and the subsequent feature Embedding and mapping to high dimensional space.

computation [Sumanshu Arora, 2020].

A couple of different methods for Embedding data structures exist, such as similarity metrics for comparing words using distance matrices, Principal Component Analysis (PCA), and Kernel PCA [Wold et al., 1987], Word2Vec [Goldberg and Levy, 2014, pp. 1–5] for text-based data or Fourier descriptors [Zahn and Roskies, 1972, pp. 1–13] applied to shape image data. Another way to conceptualize Embeddings is that each Embedding emits a certain measure of similarity between generated training graphs. Thus, when training the network later, the network learns to know the causes of good or bad assignments during the training process and hopefully tries to avoid them on the further training course. Embeddings are a subpart of the Encoder module which works by computing a set of similarity representations for each batch of input graphs. By training a single-layer neural network with 1-D Convolution significant and important static graph features such as node positions and weight costs as well as dynamic features such as long distances or short traveling duration times can be learned and encoded to a vector representation. This is similar to a Word2Vec encoding [Mikolov et al., 2013; Goldberg and Levy, 2014, pp. 1–12, 1–5], where graphs having similar node counts and dynamic features will point to similar vector directions in high dimensional space. This also reduces the required features of the input graphs, which is advantageous for saving computational resources. In the case of structural representation, a Convolutional Kernel is used a bit differently to the known application in image processing. The kernel still applies a sliding window where the enclosed values are multiplied

and summed up. However, the kernel no longer is square, but instead, a wide rectangle with dimensions 2×10 and 3×100 is applied (the dimensions in this example are chosen arbitrarily). The dimension with size 100 is often called "Embedding Size" E (sometimes also referred to as "Dictionary Length" in NLP). The height of the kernel is furthermore the number of graph input features or number of Embeddings that are seen at once while applying the convolution operation. In this case, the width of the kernel should span the length of multiple input graphs within the provided data. The convolutional neural network layer further will apply many such kernels where each of the kernel weights is learned. Each kernel furthermore looks at a graph and its surrounding features in a sequential window, where the resulting output value captures something about each input graph. In this way, the convolution operation can be viewed as a window-based feature extraction method that captures essential features and patterns of the grouped input graphs indicating different structural and contextual patterns. In that way, the convolutional operation is used in order to obtain similar Embeddings for similar graph structures and patterns. In other words, if such Convolutional Kernel is applied to similar or different sets of graphs, it will produce similar or different sets of output vectors. Also note, that the number of input channels, if a 1-D Convolutional Layer is used, has a dimensionality of 2. Hereby, each graph within both static and dynamic features is simultaneously encoded as one context vector as visualized in Figure 5.16. The 1-D Convolution refers to the process that the kernel is applied

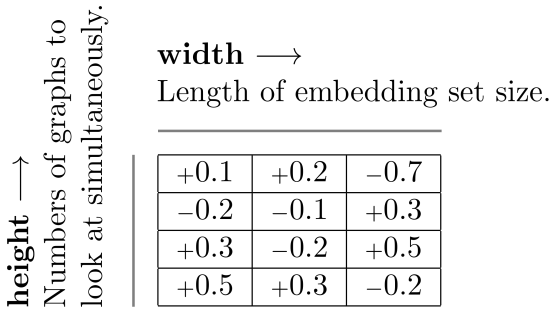


Figure 5.16: The implicated dimensions when transferring the input graphs to vector representations.

sequentially, moving from the input graph to the next stored input graph and its associated features. In order to embed unbalanced graph structures with unbalanced nodes and weights, it is necessary to use "Paddings" in order to keep the dimensions at a constant size. This way, the size of the input graph structures does not matter. This is also required in order for the Convolutional Kernel to sweep over an input sequence correctly while applying the convolution operation.

Application of the Encoder Module: The implemented "Embedding" method takes the static and dynamic tensors with TSP or MWBM graphs as input. In the example with MWBM graphs, the Embedding method represents a sequence of $N_g = v_n \cdot (r_m + s_o)$ assignments among v_n vehicle r_m request and s_o service/charging station nodes in two-dimensional space $s = \{q_i\}_{i=1}^{N}$ where each coordinate is in Euclidian 2-D space $q_i \in \mathbb{R}^2$. Embeddings are used to

track the sampled permutation of the assignments between the points q which are $\mathcal{N}$ origin and $\mathcal{M}$ destination points. Such assignments can be expressed via the assignment sequence $\mathcal{Y}$ which is a sequence list of integers. For MWBM graph features, the implemented Embedding module is used to embed tensors of complete square O-D matrices $\mathbf{D} : \mathbb{R}^{n \times n}$ and ETA matrices $\mathbf{E} : \mathbb{R}^{n \times n}$ with zero-diagonal entries $\text{diag}(\mathbf{D}) : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^n, \quad \text{diag}(\mathbf{D}) = [d_{0,0}, ..., d_{i,i}] := 0$ and $\text{diag}(\mathbf{E}) : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^n, \quad \text{diag}(\mathbf{E}) = [e_{0,0}, ..., e_{i,i}] := 0$. The dimensions of tensors and matrices depend on the numbers of nodes generated for vehicles, $|\mathcal{N}|$ MWBM nodes where $1 \leq i \leq |\mathcal{N}|$ and requests and service/charging station nodes $1 \leq i \leq |\mathcal{M}|$ for $\mathcal{N}$ and $\mathcal{M}$. Finally, individual driving distances are computed from the $\mathcal{N}$ and $\mathcal{M}$ coordinate xy-tuples where $n_i = \langle lon_i, lat_i \rangle$ and $m_j = \langle lon_j, lat_j \rangle$. Generally, there are some advantages and disadvantages of using such Embedding methods: The advantages are that i) these methods allow one to work with complex and unstructured data, as long as one can define a measure of similarity between graphs as a distance metric, ii) the output of Embedding layers' is low-dimensional numerical data which can be further processed with other modules, algorithms, or as in this case a neural network. The disadvantages are that this approach neglects some information present in the training graphs since each individual training graph is replaced with a numerical similarity value. Depending on the graph and Embedding method, the computation of such similarity value can be computationally expensive and thus time-consuming. This also depends on the actual Embedding layer complexity, the activation functions used, and the code/platform efficiency. Some Embedding methods may also be very sensitive to noise in the input data and therefore training, validation, and test set graphs. Hence, the application of Embeddings may require additional data cleaning before such a module may provide qualitative results. It is also possible that some Embedding methods are rather prone to be sensitive to the choice of working hyper-parameters. A rule of thumb is, therefore, that more complex Embedding methods are prone to be more sensitive towards influences of the aspects as aforementioned. As Figure 5.15 shows, the Encoder module is a functional module that is applied before the Attention Mechanism. Thus, the inner processes of the Attention Mechanism are explained in the next Section 5.4.

The specific Batched Data Input (BDI) data structure and implemented 1D Convolutional Embedding method is visualized in Figure 5.17. There exist two cases that are important for implementing the MWBM graph structures with static and dynamic features within two tensors for static and dynamic data. In the first case, a surplus of client nodes requires repeating vehicle indices within the tensors to model all possible assignment possibilities. In the second case, a surplus of vehicles requires to sample new or repeat old service trip nodes of services station locations to balance the assignment possibilities due to fixed tensor dimensions. Thereby, the X-Axis (static and dynamic tensor) contains the sample index for each MWBM graph, whereas the Y-Axis of the static tensor contains static MWBM graph features such as the vehicle ID which is the origin of a trip, and the two indices of request nodes and service station nodes. Furthermore, the axis contains the node types for the vehicle, request pickup, request drop-off and service station node classes. The Y-Axis of the dynamic tensor contains dynamic MWBM graph features such as the driving distances between any node connection, the vehicle ranges (capable driving range), and optionally the estimated times of arrival. Finally, the Z-axis contains an index

range of valid assignment indices for each MWBM graph index. This models the PGN search space and allows the network to limit the search space while searching for good assignments among all assignment possibilities. This way of modeling the search space furthermore allows representing invalid connections in a numeric way, which furthermore is the basis to enable masking individual connections from the search space in the later constraint satisfaction process.

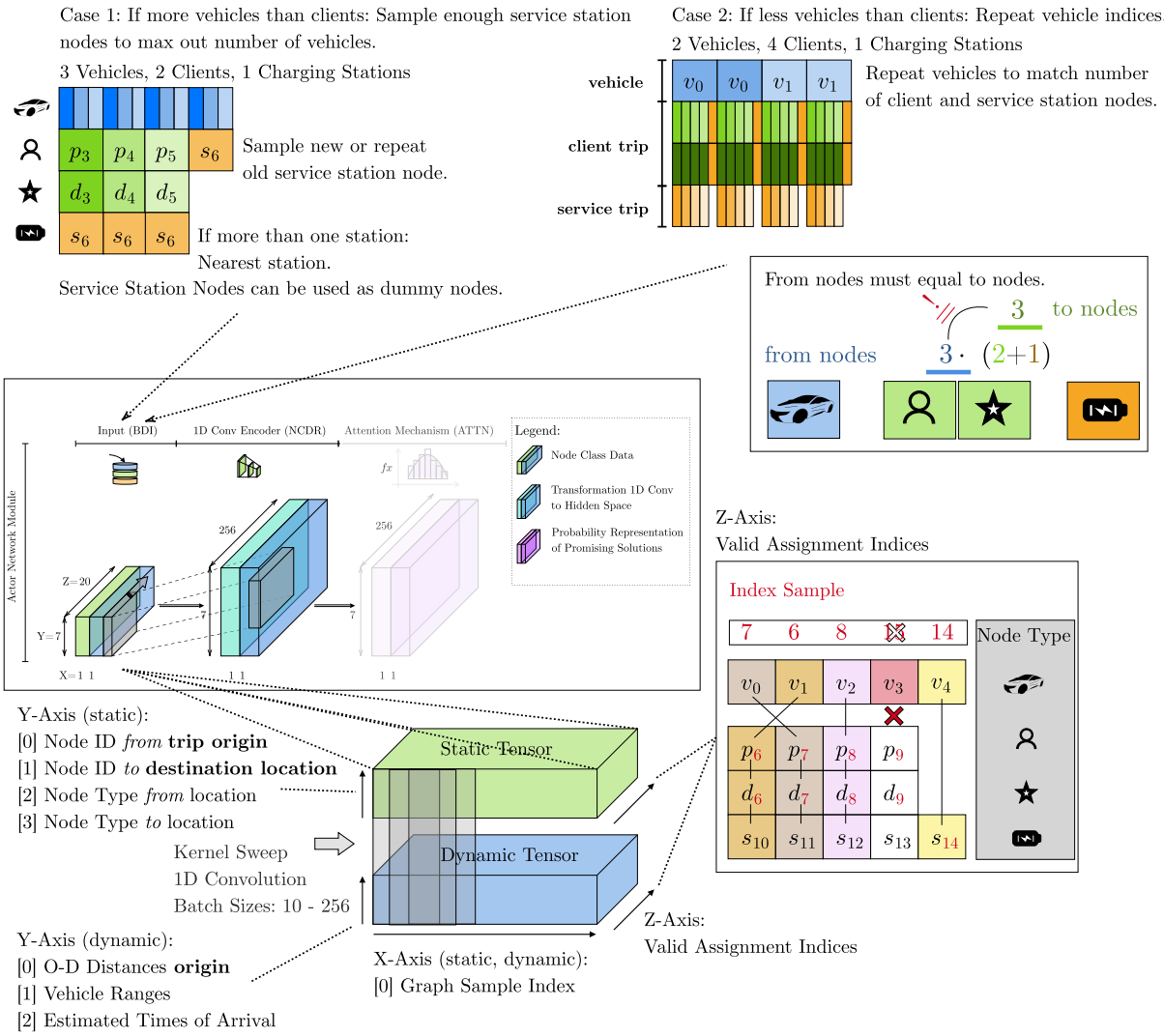


Figure 5.17: The BDI data structure and implemented 1D Convolutional Embedding. This Figure contains a fraction of the whole DRL architecture in Figure 5.3.

5.4 Attention Mechanism and Context Modeling

Generally, an "Attention Mechanism" (see Figure 5.18) can be used to store and adapt probabilities in order to model the learning experience of an Agent (i.e. the Actor network). The Attention

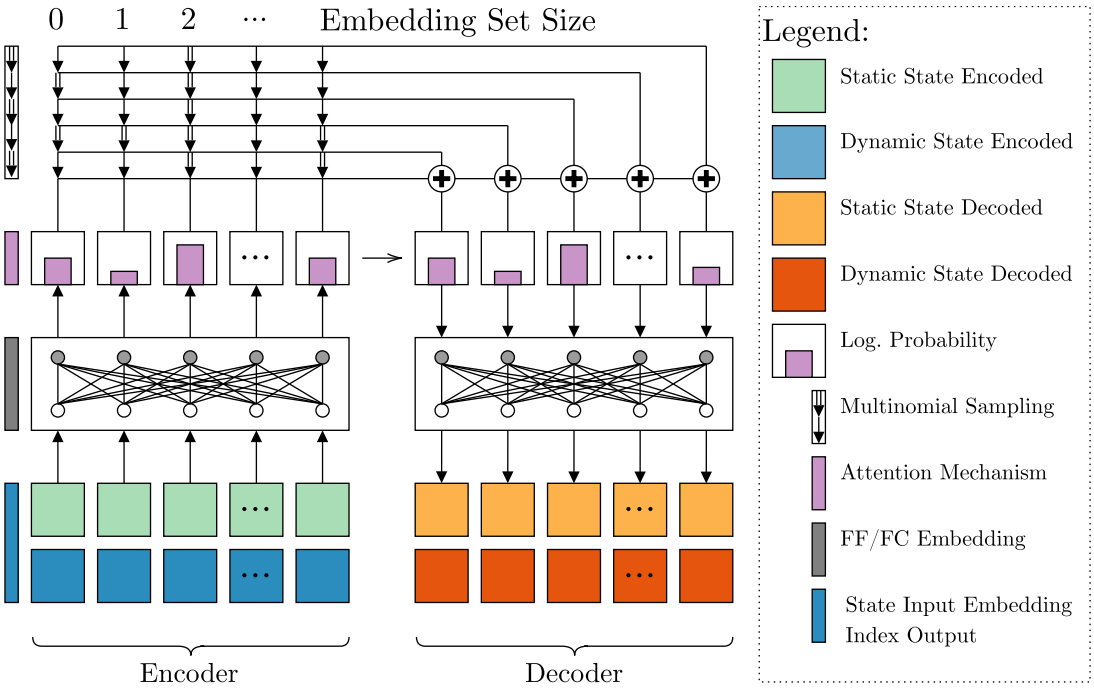


Figure 5.18: A functional view on the Attention Mechanism (ATTN).

Mechanism then learns probability values that are based on the input of similarity sequences (vectors) from the Embedding module. The probabilistic process is known as negative *maximum log-likelihood estimation* from probability theory. The maximum log-likelihood estimate is explained as an estimate of a parameter as a value whose probability is the most likely to have generated the given data or outcome [Sutton and Barto, 1998, p. 153] and is well-known in ML, NLP, and machine translation [Bahdanau et al., 2016, p. 2]. Moreover, Attention probabilities can be updated via different functions, where in this thesis the definition of probabilities is done via negative probability values called *negative logits*, where the range of the updated values is defined by the logit function (see Figure 5.19). The Attention Mechanism is used to learn approximate probabilities of near-optimal assignments of nodes for TSP and links for MWBM graphs. Thereby, the Attention model can be leveraged to pick out the most important assignments even for large and complex graph problems. Hereby, the Attention Mechanism learns a graph summary of the most important graph problem assignments. Additionally, the Attention Mechanism is a context-sensitive structure that extracts relevant information from the data input and Embedding layer. The relevance of the input data is decided via an alignment vector that processes $\mathbf{a}_t$ per decoding step t, and is computed from the embedded inputs $\mathbf{e}_t^i = (\mathbf{s}^i, \mathbf{x}_s^i, \mathbf{x}_d^i)_t$ for each embedded input i, memory state cell $\mathbf{h}_t \in \mathbb{R}^D$ and decoding step t. The Attention Mechanism controls the exploration-exploitation balance and is used to extract relevant information from the graph input data while applying MCTS. Technically, this is a categorical sampling based on the probability distribution of the input sequence. Similar to a probabilistic filter, a variable-length alignment vector $\mathbf{a}_t$ is used to specify the input data sequence index for the next decoding step t and thus controls, which relevant information is extracted from the input data. For an embedded input i,

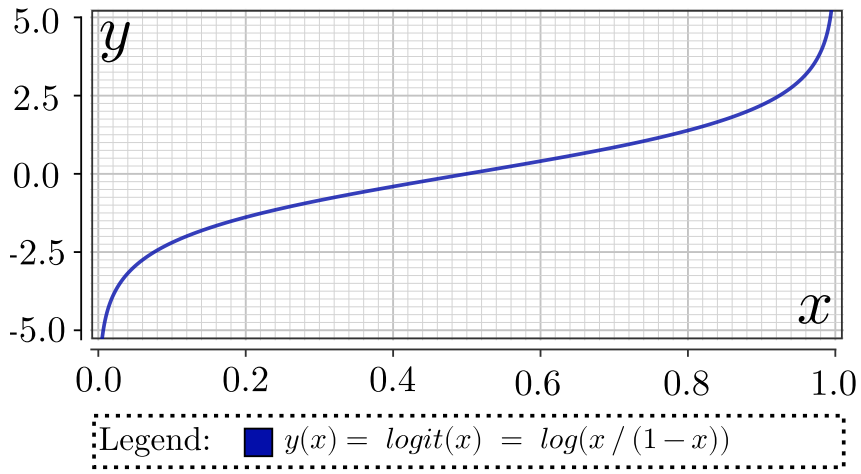


Figure 5.19: A logit function - modeling the negative maximum likelihood probability of the Attention Mechanism (ATTN) that models the Actors' policy $\boldsymbol{\pi}$.

the memory state of the RNN cell is $\bar{\mathbf{x}}_t^s = (\bar{\mathbf{s}}^i, \bar{\mathbf{d}}_t^i)$ and the alignment vector $\mathbf{a}_t$ is computed as

$$\mathbf{a}_t = \mathbf{a}_t(\bar{\mathbf{x}}_t, \mathbf{h}_t) = \text{softmax}(\mathbf{u}_t), \tag{5.3}$$

$$\mathbf{u}_t^i = \mathbf{v}_a^T \tanh(\mathbf{W}_a[\bar{\mathbf{x}}_t^i, \mathbf{h}_t]), \tag{5.4}$$

where "$[\cdot, \cdot]$" is the concatenation of both vectors. The conditional probabilities are calculated by computing the context vector $\mathbf{c}_t$ via

$$\mathbf{c}_t = \sum_{i=0}^{\mathcal{I}-1} \mathbf{a}_t^i \, \bar{\mathbf{x}}_t^i, \tag{5.5}$$

where the embedded input values are normalized with the Softmax function as follows:

$$\boldsymbol{\pi}(\cdot|\mathcal{Y}_t, \mathcal{X}_t) = \text{softmax}(\bar{\mathbf{u}}_t), \tag{5.6}$$

$$\bar{\mathbf{u}}_t^i = \mathbf{v}_c^T \tanh(\mathbf{W}_c[\bar{\mathbf{x}}_t^i, \mathbf{c}_t]). \tag{5.7}$$

Within Equations (5.4-5.7), the learned parameters are the vectors $\mathbf{v}_a, \mathbf{v}_c$ and the weighted matrices $\mathbf{W}_a, \mathbf{W}_c$. After modeling individual assignment qualities with Attention probabilities, the Actor network requires a mechanism to retrieve individual node assignments for TSPs and link assignments for MWBM graphs. Such assignments can be represented as integer sequences, whose generation processes are explained in the following.

5.5 Pointer Generation Process (PGP)

The generation of assignment indices composed as index-integer solution sequences $\mathcal{I} = \{I_0, ..., I_n\}$ requires the Pointer Generation Process (PGP) (see Figure 5.20). Each index I defines an assignment of a link for MWBMs graphs and the traversal of a node for TSPs graphs. Furthermore, based on the current Attention probabilities (i.e. the Actor policy), a categorical (multinomial) sampling

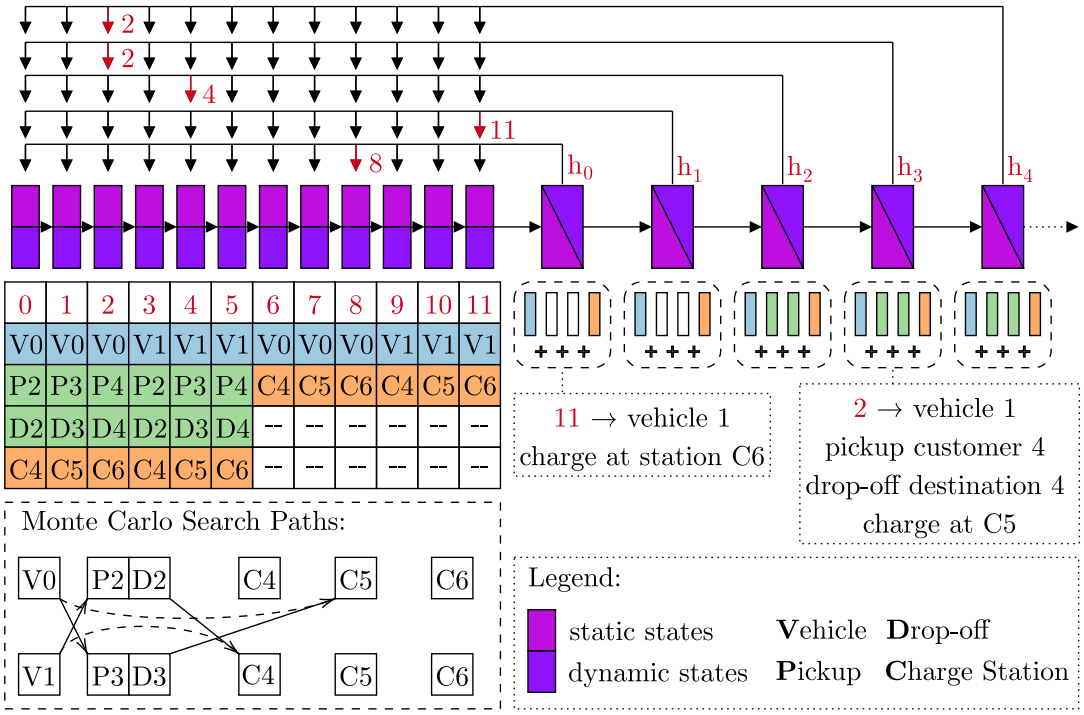


Figure 5.20: RNN PGP with Multinomial Sampling resulting in Monte Carlo Tree Search (MCTS).

process $\mathcal{I} \sim \text{Mult}(n, \boldsymbol{\pi})$ is used to retrieve assignment indices from Attention probabilities that model the Actors' policy π (see Figure 5.20). Note, that a multinomial distribution is generally a multi-binomial distribution [Bishop, 2006, p. 70] $\mathcal{I} = \{I_0, ..., I_n\} = \{\text{Bin}(n_0, p_0), ..., \text{Bin}(n_\mathcal{E}, p_\mathcal{E})\}$. The process involves a sampling of n times from the probabilities with $\boldsymbol{\pi} = [p_0, p_\mathcal{E}]$. In combination with the Attention Mechanism, the addition of the PGP results in a weighted Monte-Carlo Tree Search (MCTS) to generate near-optimal candidate sequences for the solution of combinatorial problems like TSP and MWBM graphs. The mechanisms of the PGP are depicted in Figure 5.20, where individual combinatorial solutions are generated from the Attention Mechanism's probabilities

$$p(\mathcal{Y} = y_0, ..., y_N | \mathcal{X} = x_0, ..., x_N) \tag{5.8}$$

$$= \prod_{i=1}^{N} p(y_{n+1} |\ \mathcal{X}_N, y_0, ..., y_N)\ . \tag{5.9}$$

For multiple node classes in MWBM graphs, the PGP learns to assign the graph links which have the highest Attention probabilities. Hence, the output of the PGP is used to efficiently search for good solutions for multiple mini-batches within training, validation, and test MWBM graphs. In more detail, each pointer vector $\boldsymbol{i} = \{i_t, j_t\}\ \forall \boldsymbol{i} \in \mathcal{I}$ determines which routes are assigned among the observed vehicles and pickup-drop-off and service station nodes.

In the case of TSP graphs, each probability models the assignment of a node within a Hamiltonian tour. Hence, instead of links, the integer sequences represent the IDs of nodes, which is the only major difference. The output of the PGP furthermore serves as input for the

cost or reward function to assess the Actor networks' current policy (i.e. to what degree the Attention probabilities have converged to good assignments).

Now, the complex part about later RL training comes down to formulating a suitable objective function in order to reach a near-optimal Actor policy π^*. Ideally, the learned Actor policy should minimize the cost or maximize the reward function depending on the defined formulation. As aforementioned, during the learning phase, the Actor network yields graph approximations via similarities, and thus by default the Actor policy generates stochastic and approximated solutions $\widetilde{\pi}^*$. Later, the learning phase requires an automated tuning of multiple sets of parameters $\boldsymbol{\theta}$ that parameterize (adapt) the approximated solution $\widetilde{\pi}^*$. Note, that the PGP is active during the learning phase. If the learning phase is terminated (converged) the indices are generated via the decoder module, which is explained next.

5.6 Decoder Module (DCDR)

After each training epoch, a validation step is required in order to assess if the Actor network can generalize on unseen test data graphs. For this step, the assignment indices are not generated via categorical sampling, instead, there are two different methods that are used for generating the final predictions of the Actor network. The final predictions are based on the converged Attention Mechanism probabilities (current Actor policy) after each learning epoch and during the final test predictions of the Actor network. Hereby, the decoder module is required for the reconversion of the Attention probabilities to solution indices for validation and test graph problems of MWBM graphs and TSPs. The decoder module used in this thesis computes the Attention probabilities to an output, i.e. a sequence list of integer indices $\boldsymbol{i} = (i_0, .., i_E)$ given the input sequence state vector $\boldsymbol{x} = (x_1, ..., x_E)$ from the Attention Mechanism. The length of the integer lists depends on the individual MWBM graph sizes and dummy nodes (Paddings) in order to yield the same Embedding size E overall generated unbalanced MWBM graph problems. In this thesis the decoder uses the output gate of the GRUs, as depicted in Figures 5.21 and 5.22, to select solution indices among the highest probabilities given the probability state vector from the Attention Mechanism. In more detail, the decoder module leverages two different methods of retrieving the assignments for MWBM graphs and TSPs, which are explained in the following.

5.6.1 Greedy Decoding

In order to assign the best assignment indices for TSPs this thesis uses greedy decoding that computes the most probable node assignments in order to optimize a complete TSP or Hamiltonian tour. This is well-known from previous work, where also Beam-Search also has been used to compute the final assignment indices from the Attention probabilities [Bello et al., 2016; Nazari et al., 2018]. More precisely, at every time step (decoding step), the greedy decoder computes the pointer vector (integer sequences) from the maximal log-likelihood values until every node has been visited. However, in order to make the decoder module work with MWBM graphs, the decoder module must be changed as follows.

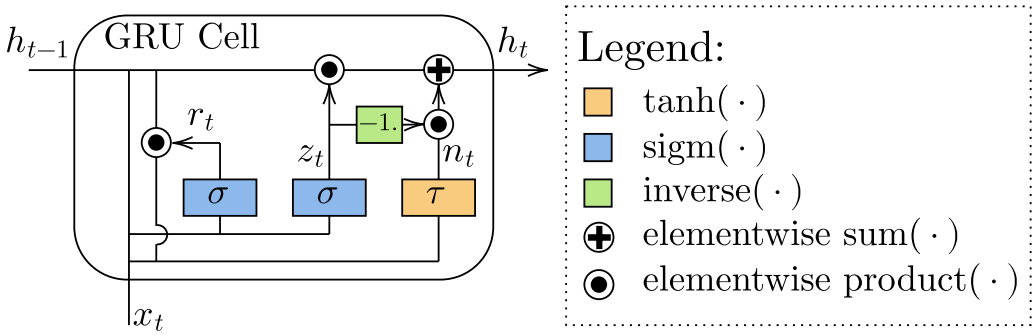


Figure 5.21: A typical Recurrent Neural Network (RNN) Gated Recurrent Unit (GRU) cell with input (last hidden state), reset, update, new information, and output (next hidden state) gate [Dey and Salemt, 2017].

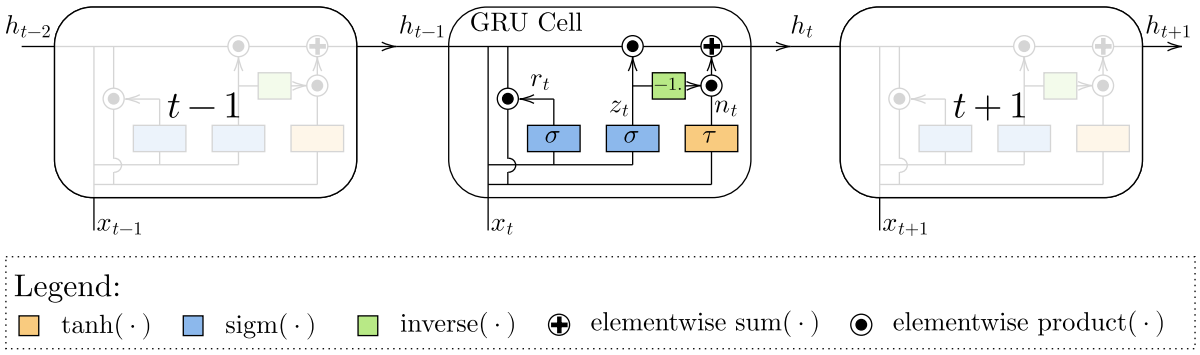


Figure 5.22: Sequentially updated Recurrent Neural Network (RNN) with Gated Recurrent Units (GRUs) cell with input (last hidden state), reset, update, new information and output (next hidden state) gate [Dey and Salemt, 2017].

5.6.2 Top-k Decoding

In order to decode MWBM graphs the greedy decoder is changed to a parallel top-k decoder module. The decoder selects the top-k assignments per MWBM graph, which are computed from the top-k Attention Mechanism probabilities (i.e. the top-k negative log-likelihood values). In other words, the Top-k Decoder Module tries to assign the given number of vehicles all at once, in order to occupy all vehicles. The module further starts with the first input state vector sequence $\boldsymbol{X}_0$ and generates a pointers vector $\boldsymbol{y}_0$ which incorporates the same number of indices as required for assigning all vehicle nodes at once. Instead of stopping the assignments if all nodes are assigned, the termination condition is set if all vehicles are assigned to a feasible request or charging station. This is required in order to prevent the Actor from selecting infeasible assignments which may still happen by small risk. In other words, the decoder does filter infeasible assignments to ensure the emission of feasible assignments only. Finally, the Top-k Decoder Module enables to speed up the learning process but also achieves better learning results, which is explained later. In order to explain how constraints can be incorporated in the learning phase, the next section explains how mask functions can be used during the training and learning phase.

5.7 Mask Functions (MASK) for Constraint Optimization

In this thesis, mask functions are used to represent constraints during training processes of the Actor network. For instance, when aiming to model constraints for MWBM graphs, i.e. limited vehicle ranges and other characteristics, constraints have to be used to represent infeasible assignments (see Figure 5.23). Technically, a mask tensor that is updated via the mask function

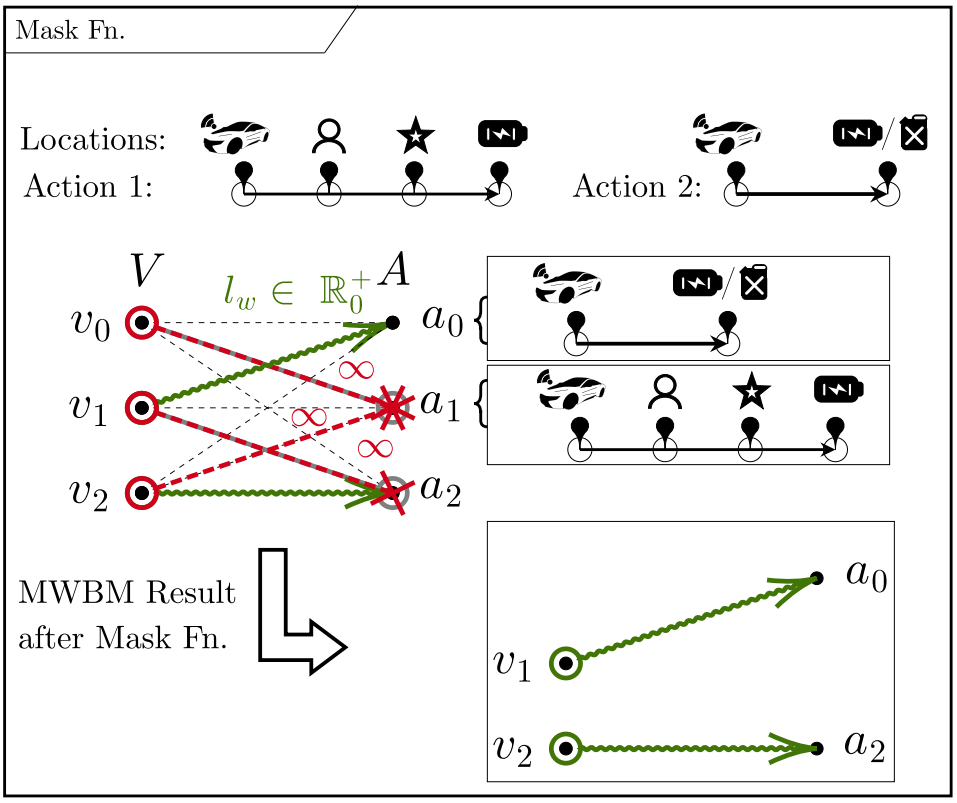


Figure 5.23: A Mask Function for masking individual links of the Maximum Weighted Matching Bipartite (MWBM) graphs.

is used to eliminate undesirable assignments made by the Actor network. In more detail, the mask is used to ensure that individual assignment probabilities are less likely than the residual assignment probabilities. The mask tensor is a binary tensor that is used to bias individual conditional log-probabilities before the softmax(·) operation in the Actor network's forward pass (i.e. before computing the attention probabilities). After representing constraints via the masking of probabilities, the Actor network may learn which assignments may be infeasible. However, the Actor network still does not have any information about which assignments are preferred over all assignments. This is represented by the cost or reward function as follows.

5.8 Cost-Reward Functions

In this thesis, the presented cost and reward functions are required to represent which assignments of links for MWBM graphs or TSPs might be preferred for minimizing the overall combinatorial problem. Furthermore, such functions provide algorithmic and iterative assessment procedures for monitoring the Actor network's learning progress during the training phase. In this thesis, the cost function used for training the Actor-Critic networks consists of two main terms as follows:

$$C_a(\mathbb{S} = (\mathcal{V}, \mathcal{R}, \mathcal{S})|\ \mathbb{A}) = \min \sum_{b=0}^{B} \sum_{\pi=0}^{\Pi-1} w_1 \cdot \boldsymbol{\eta}_{b,\pi} + w_2 \cdot \boldsymbol{\zeta}_{b,\pi}\,. \tag{5.10}$$

The first term describes the similarity of the assignments made with respect to node balance, whereas the second term is used to penalize assignments that are less desirable. In this thesis, the second term is used to minimize the driving distance to clients while trying to assign vehicles with the correct vehicle range. However, other criteria can also be modeled, such as minimizing ETAs, maximizing fleet utilities, profits, and more. In more detail, w_1 and w_2 are scalar factors, where $\boldsymbol{\eta}_{b,\pi}$ is the first term that provides an angular Boolean vector similarity value. This vector is particularly important for learning with unbalanced MWBM graphs. The second term $\boldsymbol{\zeta}_{b,\pi}$ is crucial to minimize the second objective, described by the difference between required route distances and individual vehicle ranges. In more detail, the two objective terms are defined as follows:

$$\boldsymbol{\eta} = |\,(\boldsymbol{v}_{f,\pi} - \boldsymbol{r})\,| + |\,(\boldsymbol{v}_{e,\pi} - \mathbf{s})\,|\,, \tag{5.11}$$

$$\boldsymbol{\zeta} = \left[\left(\psi\ \boldsymbol{d}'_{b,\pi}\right) - \boldsymbol{r}'_{b,\pi}\right]\,. \tag{5.12}$$

Note, that due to the prime operator $\square'$ individual terms such as the trip distances $\boldsymbol{d}_{b,\pi}$, and current vehicle ranges $\boldsymbol{d}_{b,\pi}$ are min-max scaled via normalization. The prime operator is further used to free individual terms from their Système International (d'unités)) (SI)-unit and value dependency. All distance and vehicle range values should be normalized to a value range of $[0, 1]$, which is similar to using a percentage for costs and rewards. If individual rewards or costs may additionally be negative, it is necessary to standardize the features used for costs and rewards around a zero mean. Crucially, this further stabilizes the learning procedure and mitigates over-fitting/exploding gradient phenomena.

5.9 Critic Network Architecture

The Critic network architecture has already been shown previously in Section 5.1, Figure 5.3. The Critic network furthermore maps the input graph samples and features in the static and dynamic input state tensors $\mathcal{X} = \{\boldsymbol{x}_t, \boldsymbol{x}_d\}$ to an Actor prediction as to whether, changing the Actor policy would result in a decrease or increase of the Cost Function $C_a(\mathbb{S} = (\mathcal{V}, \mathcal{R}, \mathcal{S})|\ \mathbb{A})$ in Equation 5.10. In this case, the Critic network serves as a "teacher" or supervision mechanism to intervene in case the Actor's knowledge and predictions become worse or stagnate. Specifically,

the Critic network achieves this by testing the changes in the Actor policy and providing a more conservative smoothed reward mechanism - preventing the Actor policy from large simultaneous changes (changes in the gradient). Technically, this is achieved via the advantage function, where the conservative Critic values ensure that the Actor gradients are kept from fluctuating between large magnitudes (i.e. during back-propagation). With respect to its architecture, the Critic Module consists of i) three NN modules using 1-D forwarding convolutional process blocks and ii) 2-layer ReLU NN output, as visualized in Figure 5.21. The operation can be explained in more detail by the following sequentially applied processing equations:

$$\boldsymbol{y}_1 = \text{ReLU}(\text{Conv1D}(C, H, K)) \,, \tag{5.13}$$

$$\boldsymbol{y}_2 = \text{ReLU}(\text{Conv1D}(\boldsymbol{y}_1, H, O)) \,, \tag{5.14}$$

$$\boldsymbol{y}_3 = \text{ReLU}(\text{Conv1D}(\boldsymbol{y}_2, O, K, K)) \,. \tag{5.15}$$

In Equation (5.13) $C = 1$ is a singular input channel, and $H = 128$ is the output channel of the first processing block with 128 hidden units. A kernel window of $K = 1$ is used to sweep and convolute over the currently given mini-batch rows with residual graph sequences for each sample graph. The second layer in Equation (5.14) is used as an intermediary, to map the Critic values to higher dimension $H = 128$, i.e. 128 hidden neurons. Finally, in Equation (5.15) is the output channel with dimension $O = 20$ which outputs the Critic Modules response tensor $\boldsymbol{x_c}^{[B,F,E]}$ with initial dimensions of mini-batch size B, feature space size F and Embedding size E.

5.10 Training Process

It has been proposed to train the underlying pointer network architecture via a semi-supervised RL approach [Bello et al., 2016] in order to avoid the generation of millions of solutions to NP-hard graph problems, which would be impractical and tedious. In order to train the Actor module, i.e. also called PGN, it is required to provide a reward/penalty mechanism in order to assess the quality of the networks' predictions during combinatorial optimization. Thus, this avoids the necessity for expensive supervised-labeled data computed by linear programming solvers. The next paragraphs follow this approach using model-free policy-based RL to optimize the pointer network's parameters θ with Monte-Carlo Policy Gradient Optimization [Bello et al., 2016; Nazari et al., 2018] or also called REINFORCE [Williams, 1992]. The networks are trained in an Actor and Critic scheme where the Critic network with the parameters $\theta_{\mathcal{C}}$ serves as an auxiliary network by returning an estimate of the expected reward for every sampled batch. The Critic network reduces the variance of occurring gradients during the training process and is trained with stochastic gradient descent on a mean absolute error objective between the actual reward and the prediction estimate $\theta_{\mathcal{C}}$. This is a slightly different change to [Bello et al., 2016], but has qualitative advantages for the experiments made. The objective is formulated in **Algorithm 2** line 9.

Algorithm 2 and Figure 5.24 formally sample a quantity of B sample graphs $\{s_1, ..., s_B\} \sim \mathbb{S}$ and by sampling a single matching sequence per graph, i.e. $\boldsymbol{\pi}_i \sim P_\theta(\cdot|s_i)$, the gradient in Equation

Algorithm 2: The Advantage Actor-Critic Monte-Carlo Policy Gradient (AACMCPG) Training Process.

1 **Process** *Training set* $\mathtt{X}$, *number of epochs* E, *training steps* T, *batch size* B
2 **Initialize** *Pointer Network Param.* $\boldsymbol{\theta}$, *Critic Network Param.* $\boldsymbol{\theta}^a, \boldsymbol{\theta}^c$
3 **for** *each epoch* e **do**
4 **for** *each batch* $\boldsymbol{B} \in \mathtt{X}$ *and training step* $t \in \{0, ..., |\mathtt{X}| - 1 = T - 1\}$ **do**
5 **PassFW Actor** *static and dynamic tensors from batch* $Actor(\{\boldsymbol{x}^s, \boldsymbol{x}^d, \boldsymbol{x}_0^{sd}\} := \boldsymbol{B})$
6 **Sample** *solution sequence* $\boldsymbol{\pi}_i$ *from* $P_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x}^s_{i,:,:}, \boldsymbol{x}^d_{i,:,:}, \boldsymbol{x}^{sd}_{i,:,0})$ *for* $i \in \{0, ..., B-1\}$
7 **Update** *Actor batch costs or rewards* $\boldsymbol{C}^a_{i,:,:} \leftarrow C(\boldsymbol{\pi}_{i,:,:})$ *for* $i \in \{0, ..., B-1\}$
8 **PassFW Critic** *static and dynamic tensors from batch* $\{\boldsymbol{x}^s, \boldsymbol{x}^d, \boldsymbol{x}_0^{sd}\} := \boldsymbol{B}$
9 **Update** *Critic baseline cost or reward* $\boldsymbol{C}^c_{i,:,:} \leftarrow Critic \leftarrow c(\boldsymbol{x}^s, \boldsymbol{x}^d)$ *for* $i \in \{0, ..., B-1\}$
10 **Update** $l^a_{\boldsymbol{\theta}} \leftarrow \frac{1}{B}\sum_{i=1}^{B}(\boldsymbol{C}^a_i(\pi_i|\mathbb{S}) - \boldsymbol{C}^c_{i\,\mathbb{E}}(\mathbb{S}))\nabla_{\boldsymbol{\theta}} \log P_{\boldsymbol{\theta}}(\boldsymbol{\pi}_{i,:,:}|\mathbb{S})$ *for* $i \in \{0, ..., B-1\}$
11 **Update** $l^c_{\boldsymbol{\theta}} \leftarrow \frac{1}{B}\sum_{i=1}^{B} \parallel (\boldsymbol{C}^a_i(\boldsymbol{\pi}_i|\mathbb{S}) - \boldsymbol{C}^c_{i\,\mathbb{E}}(\mathbb{S})) \parallel_2^2$
12 **Update** $\boldsymbol{\theta}^a \leftarrow ADAM(\boldsymbol{\theta}^a)$
13 **Update** $\boldsymbol{\theta}^c \leftarrow ADAM(\boldsymbol{\theta}^c, \nabla^c, \boldsymbol{\theta}^a, l^a_{\boldsymbol{\theta}})$
14 **end** *for*
15 **Return** $\boldsymbol{\theta}$
16 **end** *training*

(10) is approximated with Monte Carlo sampling in **Algorithm 2** line 10. The Critic baseline $c(\boldsymbol{x}^s, \boldsymbol{x}^d)$ is computed by a second, i.e the Critic network, which returns a second estimate of the Actor reward based on the current static and dynamic state. The policy gradient algorithm is used to approximate the distributions of training data MC sampling (also called Rollouts) by drawing with batch size B independent and identically distributed (i.i.d.) sample MWBM graphs $\mathcal{X}_t = \{\mathcal{G}_0, ..., \mathcal{G}_B\}$ and sample unique assignments per graph i.e. $\boldsymbol{\pi} \sim P(\cdot|\boldsymbol{x}_b)$ where the gradient is computed via the mean squared error objective

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \leftarrow \sum_{b}^{B}(\boldsymbol{C}^a_b(\boldsymbol{\pi}|\boldsymbol{x}_t, \boldsymbol{x}_d) - \boldsymbol{C}^c_{\mathbb{E}_b}(\boldsymbol{x}_t, \boldsymbol{x}_d))\, \nabla_{\boldsymbol{\theta}} \log \boldsymbol{p}_{\boldsymbol{\theta}}(\boldsymbol{\pi}|\boldsymbol{x}_t, \boldsymbol{x}_d) \tag{5.16}$$

which defines the Actor loss value $l_{\boldsymbol{A}^c_b, \boldsymbol{C}^c_b}$. For adjusting the Critic network gradients via stochastic gradient descent a Critic loss function is used and is defined by the mean squared error objective

$$l_{\boldsymbol{C}} \leftarrow \frac{1}{B}\sum_{i=1}^{B} \parallel (\boldsymbol{C}^a_b(\boldsymbol{\pi}|\boldsymbol{x}_t, \boldsymbol{x}_d) - \boldsymbol{C}^c_{\mathbb{E}_b}(\boldsymbol{x}_t, \boldsymbol{x}_d)) \parallel_2^2 \ . \tag{5.17}$$

In the end, this enables the critic network to learn the outcome of the changing Actor policy and resulting Actor assignments. Technically, the Critic learns the well-known Mean Squared Error (MSE) between the Critic's predictions and the current Actor networks' policy, i.e. the current negative log-likelihood probability distribution. Ideally, the AACMCPG algorithm iteratively progresses in teaching the Actor network to learn a good policy. The current policy of the Actor network hereby is optimized via stochastic gradient descent during back-propagation. Technically, this is achieved by employing the sophisticated ADAM optimizer [Kingma and Ba, 2015].

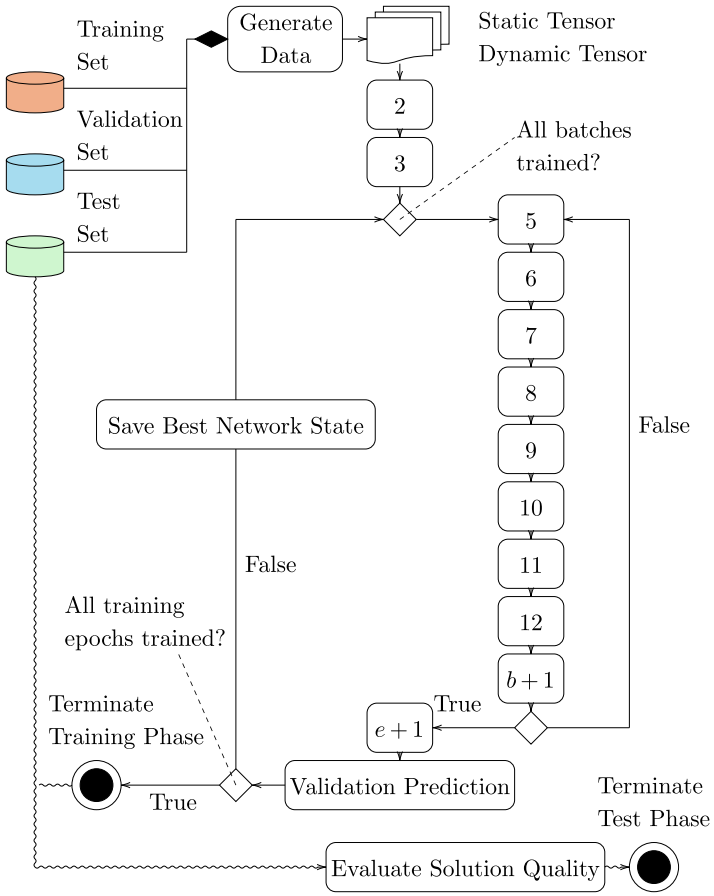


Figure 5.24: Advantage Actor-Critic Monte-Carlo Policy Gradient (AACMCPG) training process with line references to Algorithm 2.

5.11 Activation Functions

Different activation functions can have beneficial and adverse effects on learning efficiency and quality. In this section, some activation functions are briefly introduced. Since the Encoder Module in Section 5.3 requires at least a 1-D Convolutional Layer with activation functions, it is also possible to experiment with different activation functions for increasing or decreasing the Embedding complexity and quality, as shown in Figure 5.25. In this thesis, the focus is on hyperbolic tangent (Tanh) activations used in the Attention Mechanism, whereas LeakyReLU activations are used in the Encoder and Decoder modules. Later experiments with TSP include comparisons with a relatively new activation function called MISH [Misra, 2019]. Essentially, the different curvatures of the activation functions shown, influence the calculation of gradients for currently activated value magnitudes before the neuron activation. For large value magnitudes, i.e. $x > 5$ modern activation functions such as LeakyReLU suffer less from the vanishing gradient phenomenon, compared to Tanh and Softmax activations. One benefit of LeakyReLU activations is that they enable efficient learning (due to the function's simplicity) compared to more complex activation functions. On the other hand, it is well-known that monotonic (continuously differentiable) activation functions provide benefits for learning more qualitative

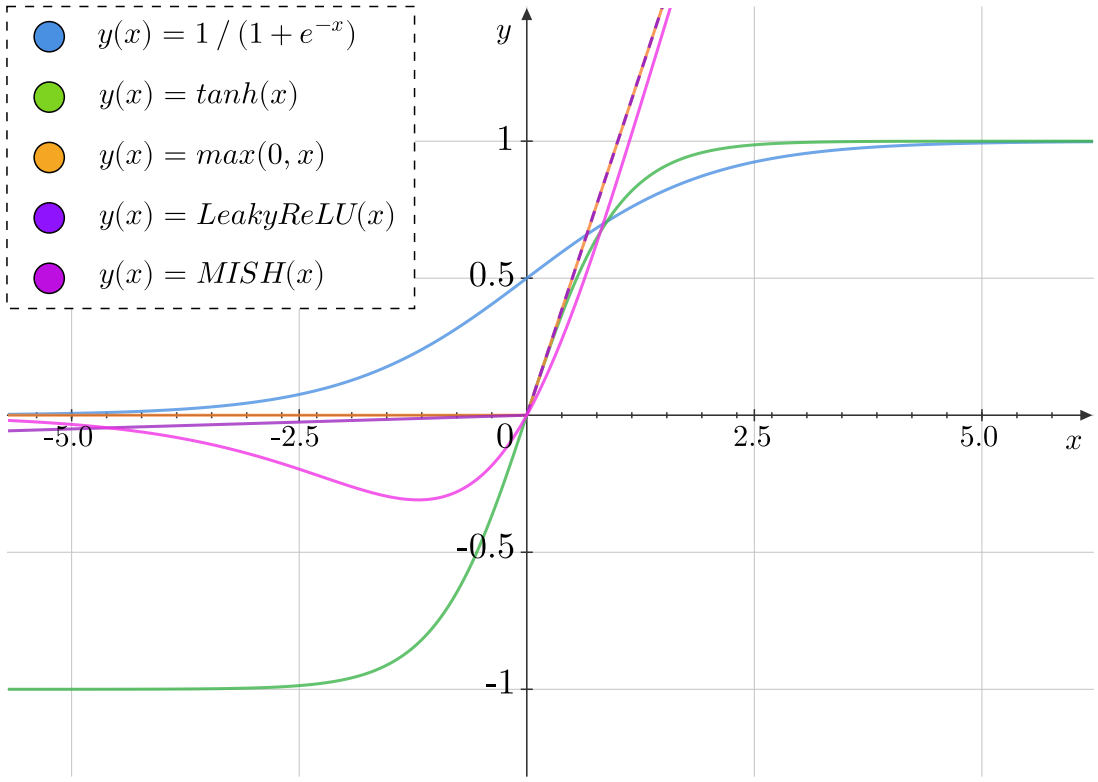


Figure 5.25: Enhanced application of activation functions (i, purple) LeakyReLU and (ii, violet) MISH [Misra, 2019].

solutions. Hence, this will be investigated in the later testing phase within this thesis.

5.12 Avoidance of Over-fitting and Exploding Gradients

Generally, two training issues are very likely to occur during the training phase. For instance, there is the i) exploding gradient and the ii) vanishing gradient phenomena, which are explained in Chapter 4, Section 4.7 in a more fundamental way. Briefly, the AACMCPG training algorithm is known to suffer more with respect to the exploding gradient phenomena, where gradient adaptions start to fluctuate during learning process. Those result in an undesirable rapid change in the Actors' policy during learning, which also mitigates the learning process and learned qualitative solutions. This can be explained by analyzing the definition of the algorithm. Since multiple index sequences (assignment indices) can be updated at once, multiple changes of indices may result in high changes of the cost/reward feedback signal that is computed using the cost function. In other words, if extremely high cost or reward values are used in the cost function, it is very likely that the AACMCPG training process deteriorates by way of the exploding gradient phenomenon. In order to mitigate both phenomena, the well-known mechanisms of maximum gradient clipping between gradient magnitudes of $[0, 1]$ [Graves, 2013], and dropout with probability of 0.2 [Gal and Ghahramani, 2016] are used within this thesis. Furthermore, within this thesis two decays of Actor and Critic learning rates via the factors $\gamma_\alpha = 0.99$ and

$\gamma_\beta = 0.96$ are used. This finally helps to stabilize the learning phase in the majority of cases.

5.13 Real Data and Feature Scaling

The use of real data and physical metrics (i.e. distances, ranges, etc.) with different units, such as SI- or United States Customary System (USCS)-units [Newell and Tiesinga, 2019; Judson, 1976], requires additional means to reduce undesirable learning phenomena (see Section 5.12) caused by large values with different value and unit ranges. For the most part, real datasets usually come with units such as SI- or USCS-units [Newell and Tiesinga, 2019; Judson, 1976], however, different value ranges and units can have negative impacts on the AACMCPG learning progress. Some changes within the cost function, see Section 5.8, are required as follows. For instance, any used term that relates to a SI- or USCS-unit must be at least min-max scaled (normalized) and sometimes standardized, if negative value ranges are allowed to appear within the definition of the cost function or reward function. Standardization is typically understood as re-scaling data distributions to zero-mean and standard deviation of 1 (with unit variance). As an example, the cost function

$$C(\boldsymbol{Y}|\boldsymbol{X}) = \min \sum_{i=0}^{\mathcal{P}-1} \sum_{j=0}^{\mathcal{D}-1} \left(\boldsymbol{d}'_{i,j} - \boldsymbol{r}_i^{v,\prime} \right) \cdot \boldsymbol{x}_{i,j} \tag{5.18}$$

minimizes the deviation of route distances for vehicle routes to client trips and the currently available vehicle range. This leads to the assignment behavior that vehicles with suited ranges are selected for individual lengths of client trips. Since distances and ranges are affiliated with SI- or USCS-units, they have to be min-max scaled, where the prime operator $\square'$ denotes the so-called feature min-max scaling. The operation is defined in

$$\boldsymbol{x}' = \frac{\boldsymbol{x} - \min(\boldsymbol{x})}{\max(\boldsymbol{x}) - \min(\boldsymbol{x})} \,, \tag{5.19}$$

where min-max scaling is ideally applied for distances and ranges that provide positively defined value ranges. In the case of real data rewards and penalties the values should be standardized instead. Both operations allow increasing the learning efficiency and quality during the AACMCPG learning process.

5.14 Algorithm Baselines for Benchmarking

This section contains a short description of the algorithms involved for benchmarking the different Actor-Critic PGN solutions. Basically, the PGN can be well compared against other scalable heuristics such as the Greedy heuristic as follows.

5.14.1 Greedy Heuristic (GRH)

The Greedy Heuristic (GRH) is a very simple algorithm that computes the next best MWBM assignment using an available cost matrix $C(\boldsymbol{x})$ row i by row $i+1$, until a minimal solution

is found. The Greedy heuristic pseudo-code in Algorithm 3, is explained in more detail in the

Algorithm 3: Greedy Search Heuristic Method.

```
Result: Greedy Heuristic Assignment Method
Repeat until all tasks are assigned to vehicles;
Minimize cost matrix C(x), maximum cost value m_c;
while min C(x) ≤ m_c do
    find next min value;
    (i_l, j_l), (i_min, j_min) = min C(x);
    add value to objective value;
    o_c^k = o_c^{k-1}(i_min, j_min);
    for j = 0, ..., m do
        C(i_min, j_l) = M_c;
        for i = 0, ..., n do
            C(i_l, j_min) = M_c;
        end
    end
end
```

following:

1. Input a random cost matrix $C(\cdot)$, and select the maximum cost value m_c;
2. While a minimum value can be found that is lower m_c;
3. Locate the minimum value over rows i and columns j;
4. Update the objective value o_c^k, of the current iteration k by adding to the previous iterative objective value o_c^{k-1};
5. For all columns j, locate the minimum value row and column position $\langle i_{min}, j_{min} \rangle$, assign the task to the assignment list, and set the minimum value to maximum cost m_c for all residual row entries;
6. For all rows i, locate the minimum value, assign the task and set the minimum value to the maximum cost m_c for all residual column entries;
7. until no minimum values are left being lower than the maximum value m_c;

Hereby, it may happen that the Greedy algorithm does not find the optimal nor the complete solution of a combinatorial optimization problem.

5.14.2 k-Regret Heuristic (KRH)

The next heuristic called k-Regret Heuristic (KRH) is an insertion-based algorithm that attempts to insert (or add) a link from vehicle nodes $\mathcal{V}$ to the next node $\mathcal{A}$ in order to build a rapid solution of a superior assignment Graph $\mathcal{G}(\mathcal{V}, \mathcal{A})$. The algorithm checks if adding a link to the overall solution would cause a violation of the previously found valid solution and if an improvement would be the cause of adding the currently selected edge. During the insertion process, and as more edges are added to the graphs, more and more violations occur. In order to mitigate the

computation times the measure k is set to define an upper bound to the amount of violations tolerated. A higher value selection for k thus means longer computation times, but also superior solution qualities achieved in many cases generally.

5.14.3 Hungarian-Munkres-Kuhn Algorithm (HMK)

The Hungarian-Munkres-Kuhn (HMK) algorithm [Munkres, 1957; Kuhn, 1956; Kuhn, 1955, pp. 32–38, 253–258, 83–97] implemented[1] is a general algorithm to solve maximum flow or MWBM graphs optimally. One MWBM problem graph hereby can be solved via MIP formulation with a cost matrix $\boldsymbol{C}$. Thereby, $\boldsymbol{C}_{i,j}$ is a cost for assigning a vehicle $v^{[e,g]} \in \mathcal{V}$ to a client request r and or service station s. In the optimization community, the vehicle node is termed as (a "worker") and the link to the next node is termed (a "job" or "task") of a second node set. Generally, the algorithm solves the equation

$$\min \sum_{i} \sum_{j} \boldsymbol{C}_{i,j} \boldsymbol{X}_{i,j} \,, \tag{5.20}$$

where each row represents an assignment to at most a column, and vice versa. If the MWBM graph is unbalanced the cost matrix will have either more rows than columns or vice versa. In this case, not every row must be assigned to a column and not every column must be assigned to a row (vice versa).

5.14.4 Jonker Volgenant Castanon (JVC)

The MIP formulation in Equation (5.20) equally holds for the Jonker-Volgenant-Castanon (JVC) algorithm. In comparison to the HMK algorithm, the JVC uses an n-Dijkstra method to find the shortest-augmenting path among all augmenting paths within an MWBM graph [Jonker and Volgenant, 1987, pp. 325–340]. Where the Hungarian-Munkres-Kuhn finds any augmenting path out of the feasible ones, the JVC method finds the shortest augmenting path among all constructed augmenting paths. The JVC algorithm is based on the HMK Method [Munkres, 1957], however, the additional pre-processing steps are explained in the following. Again the MIP Formulation minimizes a balanced or unbalanced cost matrix

$$\min \sum_{i \in V} \sum_{j \in A} \boldsymbol{C}_{i,j} \cdot \boldsymbol{X}_{i,j} \,, \tag{5.21}$$

from vehicle nodes to action nodes (client request pickup, to drop-off, and service station nodes). Hereby, only one vehicle is allowed to be assigned to one client or service station, as defined in Equations (5.22)-(5.23):

$$\sum_{i \in \mathcal{V}} c_{i,j} = 1 \qquad \forall i \in \mathcal{V} \,, \tag{5.22}$$

[1] accessed 07/30/2020: https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.linear_sum_assignment.html

$$\sum_{j \in \mathcal{A}} c_{i,j} \leq 1 \qquad \forall j \in \mathcal{A}\,. \tag{5.23}$$

Here, $\boldsymbol{X}_{i,j}$ is again a Boolean matrix, see Equation (5.24), that defines if an assignment is made (True=1) or not (False=0):

$$x_{i,j} \in \{0,1\} \qquad \forall j \in \mathcal{V}, \forall j \in \mathcal{A}\,. \tag{5.24}$$

The pseudo-code of the JVC algorithm is presented in Algorithms 4 and 5.

The algorithm uses a couple of processing steps, that are explained in the following. First the algorithm uses a couple of steps such as the

1. *initialization step* with sub-steps such as a) column reduction, b) reduction transfer, c) reduction of unassigned rows, similar to auction algorithms,

2. *termination step*, if the row assignment is complete,

3. *augmentation step*, where an auxiliary network is generated to determine which generates an alternating path with minimal total reduced objective cost from unassigned row i to unassigned column j,

4. *update step of the dual solution*, where the complementary slackness variables are updated, and finally, the

5. *return step to the termination Step 2) [Selmair et al., 2021, pp. 1–8],*

where the JVC algorithm computes the optimal integer sequence to assign each row entry to a minimal column entry $v_j = \min_i(c_{i,j})$. In the existence of unbalanced graphs and non-square matrices, the algorithm does not assign all rows or columns, which is particularly important for unbalanced graphs with non-square matrices. Moreover, the JVC algorithm minimizes the primal objective

$$\min \sum_i \sum_j c_{i,j} x_{i,j} \text{ s.t. } \sum_i x_{i,j} = 1,\ \sum_j x_{i,j} = 1\,, \tag{5.25}$$

$$x_{i,j} \leq 0,\ \forall (i,j) \in E\,, \tag{5.26}$$

and maximizes the dual objective

$$\max\Big\{\sum_i u_i + \sum_j v_j\Big\} \text{ s.t. } c_{i,j} - u_i - v_j \leq 0\,, \tag{5.27}$$

where the algorithm uses a reduction transfer to reduce the number of columns or rows. This is completed from unassigned to assigned rows where for each row i:

$$j_1 = x_i, \tag{5.28}$$
$$\mu = \min\{c_{i,j} - v_j : j = [1, ..., n], \forall j \neq j - 1\}\,, \tag{5.29}$$
$$v_{j1} = v_{j1} - (\mu - u_i)\ , u_i = \mu\,. \tag{5.30}$$

After augmentation Step 3), the partial integer solution assignments are updated. Additionally, the dual values in Equation (5.27) are updated to restore the complementary slackness conditions:

$$c_{ik} - u_i - v_k = 0, \text{if } x_i = k\ \forall \text{ assigned columns and} \tag{5.31}$$
$$k, i = 1, .., n \text{ and} \tag{5.32}$$
$$c_{ik} - u_i - v_k \leq 0\ . \tag{5.33}$$

Overall, the time-space complexity of the first two initialization procedures is stated to be $\mathcal{O}(n^2)$, while the augmenting reduction procedure has a complexity of $\mathcal{O}(Rn^2)$. Here, R is the real-valued range of the optimization problem-specific cost coefficients [Jonker and Volgenant, 1987].

5.14.5 Simplex Algorithm (Cplex)

Introduced for the first time in 1965, the ILP algorithm that is important for leveraging Cplex is called the well-known simplex algorithm [Dantzig, 1965]. In general, it is known that the simplex method requires stochastic (expected) polynomial complexity by finding the global optimum for a variety of combinatorial optimization problems [Spielman and Teng, 2004, pp. 385–463]. In order to apply Cplex [Nelder and Mead, 1965, pp. 308–313], a MIP representation of the respective combinatorial optimization problem is required. With respect to the methodological principle, the algorithm aims to find the minimum convex polytopes while constructing increasing path lengths within a vertex (graph network), that models the overall combinatorial optimization problem graph and its cost parameters [Klee and Minty, 1972, pp. 159–175].

Algorithm 4: JVC Algorithm 1a)-1c).

Result: Optimal assignment, Shortest Augmenting Path $v_j = min_i(c_{i,j})$ of Equation (5.21) w. r. t. constraints (5.22-5.24).

Step 1) Initialization;

Step 1a) column reduction;

for $j = n, ..., 1$ **do**
  $c_j = j; h = c_{1j}; i_1 = 1;$
  for $i = 2, ..., n$ **do**
    if $c_{i,j} < h$ **then**
      $h = c_{i,j};$
      $i_1 = i;$
      $v_j = h;$
    else
      if $x_{i1} = 0$ **then**
        $x_{i_1} = j;$
        $y_j = i_1;$
      else
        $x_i = -x_i;$
        $y_j = 0;$
      end
    end
  end
end

Result: Step 1c): Pre-Processing augmenting reduction of unassigned rows.

for *Unassigned row list* L, $\forall i \in L$ **do**
  repeat;
  $k_1 = \min\{c_{i,j} - v_j : j = 1, .., n\};$
  select j_1 with $c_{i,j_1} - v_{j_1} = k_1;$
  $k_2 = \min\{c_{i,j} - v_j : j = 1, ..., n; j \neq j_1\};$
  select j_2 with $w = c_{i,j_2} - v_{j_2} = k_2; j_2 \neq j_1;$
  $u_i = k_2;$
  if $k_1 < k_2$ **then**
    $v_{j_1} = v_{j_1} - (k_2 - k_1)$;
  else
    if j_1 *is assigned* **then**
      $j_1 = j_2;$
      $r = y_{j_1};$
      if $r \leq 0$ **then**
        $x_r = 0;$
        $x_0 = j_1;$
        $y_{j_1} = i;$
        $i = r;$
      end
    end
  end
  until $k_1 = k_2$ or $r = 0$
end

Algorithm 5: JVC Algorithm Steps 3)-4).

Result: Optimal assignment, Shortest Augmenting Path $v_j = \min_i(c_{i,j})$ of Equation (5.21) w. r. t. constraints (5.22-5.24).

Steps 3) and 4): Augmentation and Update;

Modified version of Dijkstra's shortest path;

```
for Unassigned row i and ∀i* do
    for Unassigned row j = 1, .., n do
        L_to_scan = {1, ..., n};
        d_j = ∞;
    end
    i = i*;
    d_i* = 0;
    μ = 0;
    repeat;
    for ∀j ∈ (L_out(i), L_to_scan) do
        if μ + c_i,j − u_i − v_j ≤ d_j then
            d_j = μ + c_i,j − u_i − v_j;
            pred[j] = i;
        end
        μ = ∞;
        for ∀j ∈ L_to_scan do
            if d_j ≤ μ then
                μ = d_j;
                μ_j = j;
            end
            i = y_μj;
            L_to_scan = L_to_scan − {μ_j};
        end
        until y_μj = 0;
    end
end
```

6. Benchmarking and Results

The previous Chapter 5 Algorithm and Model Implementations, presented the software components developed, as well as functions and algorithms that are required to learn artificial and real data-based fleet planning and assignment problems. In order to provide enough computational power to train the presented Pointer Generation Network (PGN) approach, sophisticated and suitable computation resources are required as presented in this chapter. Additionally, this chapter presents state-of-the-art algorithms that serve to enable qualitative and quantitative efficiency comparisons for the results presented in Section 6.2. Finally, Section 6.2 presents all results within this thesis and serves as a successive basis for the next Chapter 7 Conclusions & Future Research.

6.1 Benchmark Setup

In order to conduct experiments for training and testing PGN network solutions, comparably high computational resources are required for generating millions of training, ten thousands of validation, and thousands of test graphs, especially when training the Encoder-Decoder PGN multiple times. In this thesis, a local computer with multiple GPUs is used to develop and test solutions. After successful development, the training is ideally done in the cloud (GPU cluster) to reach the full potential and best qualitative solution behavior. The full training process flow is shown in Figure 6.2, where additional monitoring means are crucial to survey the current learning progress of the PGN. For each training run, the Data Generator module accesses a database that includes large sets of client, vehicle, and service station data. The generator further generates millions of training samples for training, tens of thousands of validation samples for validation (generalization) monitoring, and thousands of test samples for final test, quality, efficiency, and reliability benchmarking. After each training run has been completed, the PGN network solves a large set of test graphs, which subsequently are solved by state-of-the-art exact solvers and approximate heuristics to perform comparisons. The PGN code is deployed on two systems in this thesis. The code runs on a development computer with Ubuntu 18.04.5 LTS 64 bit, 15.6 Gigabyte (GiB) Random-Access Memory (RAM), Intel Core i7-6700K CPU at 8 $\times$ 4.00 Gigahertz (GHz), mainly using two Nvidia GeForce GTX 1080 GPUs. Second, in the final sophisticated training, the code is run in the cloud (Microsoft (MS) Azure, Amazon Web Service (AWS)), where it is trained with NC-Series Virtual Machines (VMs) using the Nvidia Tesla K80 GPUs with 12 GiBs and an Intel Xeon E-52690 v3 (Haswell) processor. The PGN leverages the Compute Unified Device Architecture (CUDA) 10.1 which makes training 10-20

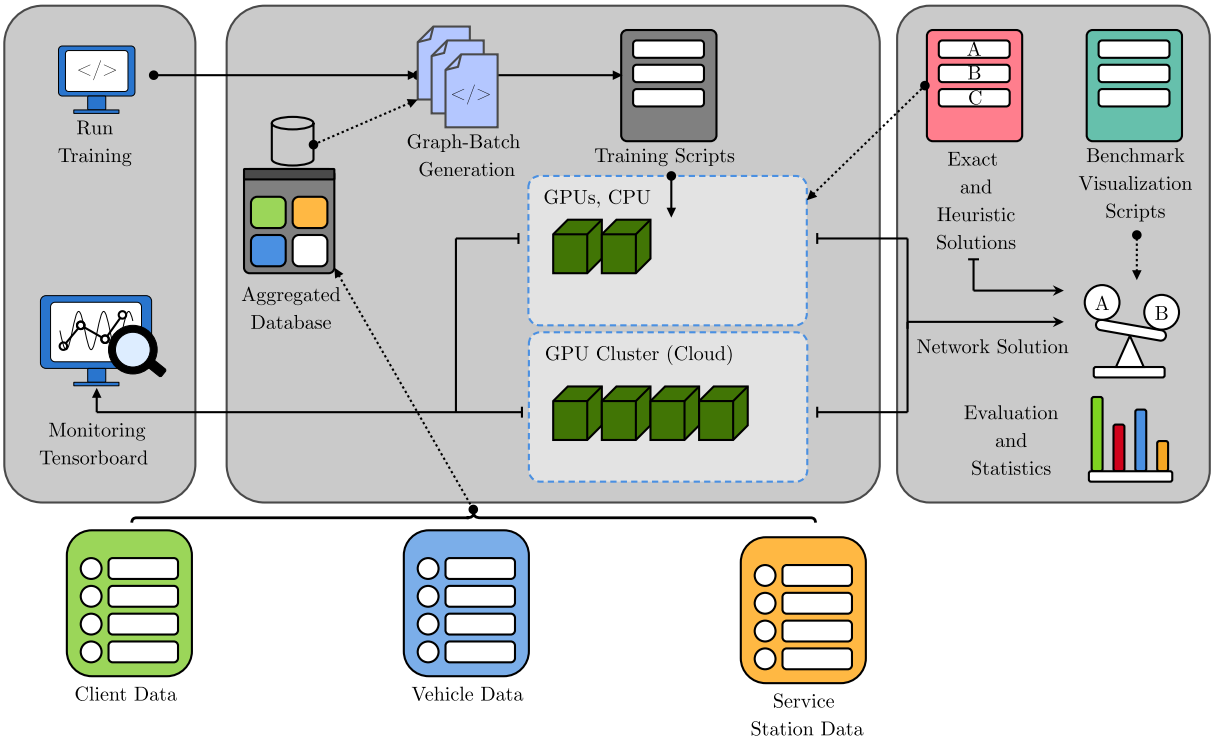


Figure 6.1: The basic benchmark setup.

times faster compared to CPU-based training. The required deployment memory is around 1.8 GiB in total, while a single GPU Utilization is approximately 80% during training. In more detail, the data generation process for training, validating, and testing the PGN is explained next.

6.1.1 Data and Network Parameters

Data Processing: Initially, during each training run, the data generator in Figure 6.2 generates a training set with one to four million graph samples individually. Each graph sample consists of predefined amounts of vehicle, client, and service station nodes. After the training set generation, the validation set with 10 thousands of validation graph samples is generated. This is required to monitor if the PGN network can learn to solve graph samples that have not been seen before. Furthermore, this allows one to monitor if the PGN network can actually progress to learn better solutions for the final test set samples. After the validation set generation, the final training set with 1000 training graph samples is generated. Importantly, the test set represents the problem graph samples that are solved in reality. In this thesis, the test set hereby is generated for benchmarking the PGN network and other algorithms in the final benchmarking stage. Thus, the test set is not available to the PGN network during the training phase. The individual data generation setups for successive benchmarks are shown in Table 6.1 as follows. As mentioned in Chapter 4 **Deep Reinforcement Learning**, the training set, validation set, and test set are divided into mini-batches and thus are not taken into consideration all at once. Here, the Batch Size B (e.g. see Table 6.1) defines how many problem graphs per mini-batch are shown to

No. Benchmarks	Dataset Sizes [$\|\mathcal{G}\|$ Graphs]				Learn. Epochs
	X	V	T	B	L
1. TSP (artificial)	$1 - 4e6$	10.000	1.000	256	10-50
2. PDP/MWBM (artificial)	$1 - 2e6$	10.000	1.000	15	5,10,15,20
3. PDP/MWBM (real data)	1e5 - 1e6	10.000	1.000	50	5,10,15,20

Table 6.1: Important benchmark and PGN network parameters.

the PGN network simultaneously for learning. For Benchmark 3 each mini-batch consists of 50 problem graphs, that are sampled from the training set during training, and sampled from the validation set during the validation phase at the end of each learning epoch L. As illustrated in Table 6.2, the learning progress of the actor and critic sub-network modules and their decays can be defined as follows. In this thesis, a designated actor learning rate α and its learning rate decay γ_α is defined. Both parameters define the initial learning rate and its decay throughout the learning process. Similarly, the critic learning rate β and the decay γ_β define to which degree the critic module intervenes and stabilizes the actor learning policy if the actor network does not succeed to learn improvements during training. The exact parameter values are shown in Table 6.2 as follows.

No. Benchmarks	Learning Rates (LR)		LR-Decays	
	α Actor LR	β Critic LR	γ_α	γ_β
1. TSP (artificial)	1e−3	1e−3	0.99	0.96
2. PDP/MWBM (artificial)	5e−4	5e−4	0.99	0.96
3. PDP/MWBM (real data)	5e−4	5e−4	0.99	0.96

Table 6.2: Important actor and critic module parameters.

Network Weight Initialization: During the network weight initialization the PGN network weights are initialized via Xavier Uniform Distribution. This method fills the input tensors with uniformly distributed values, and the resulting tensor is uniformly sampled from $\mathcal{U}(-\boldsymbol{a}, \boldsymbol{a})$ with

$$\boldsymbol{a} = G \times \sqrt{\frac{6}{\boldsymbol{f}_i + \boldsymbol{f}_o}}. \tag{6.1}$$

This is also known as Glorot initialization, where in this thesis the gain parameter is set to $G = 1$ [Glorot and Bengio, 2010; Pytorch Xavier Uniform Sampling, 2021, p. 253].

6.1.2 Training Monitoring

During the network's training process, there are a couple of signals that need to be monitored. First, there is the actor loss (error) signal, which indicates the learning and training progress of the actor if the loss is decreased. Second, there is the critic loss, which successively rises in magnitude, if the actor loss increases and decreases if the actor loss decreases. Here, the critic loss signal is used to stabilize the actor learning process. Next, there is the moving average cost signal, where a decrease signalizes if the actor network module is capable of learning how to reduce the

Objective Cost function for each new training batch with batch size B. In this thesis a tool called Tensorboard is used to monitor the training processes implemented in Pytorch (see Figure 6.2). Tensorboard enables to monitor individual PGN network and training parameters by tracking

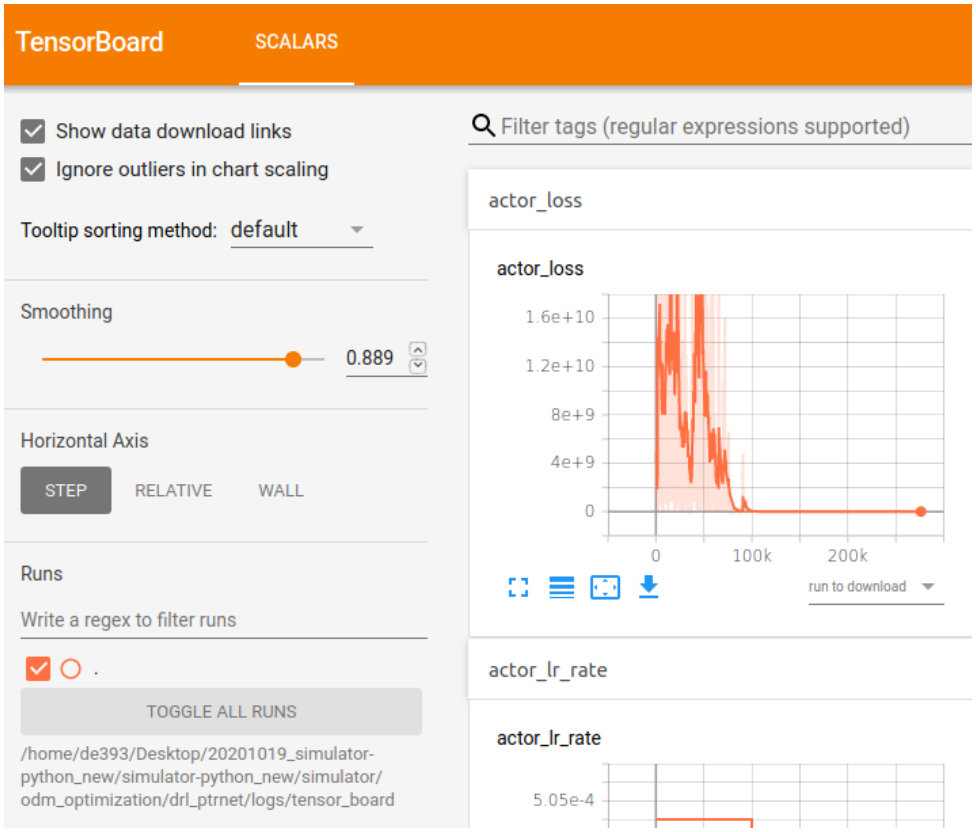


Figure 6.2: Monitoring the training process via Tensorboard (Screenshot).

the parameters over training iterations. Furthermore, the plot of each individual parameter can be dynamically inspected, and different axis formats can be altered to show linear as well as logarithmic axes. Finally, the slider smoothing controls how smooth the moving average of the parameters being tracked is computed, and thus enables to show long-term trends of parameter changes to be displayed. Finally, the next section contains a summary of all benchmarking tests and results within this thesis.

6.2 Benchmark Results

In general, the result section is characterized by a listed and ordered structure, where the made benchmarks increase in problem size and complexity. In order to do so, the overall problem complexity has been divided into groups and features. The first benchmark test, is whether the implemented PGN network can handle link weight symmetries and asymmetries as mentioned and introduced in Chapter 2 Fleet Dispatching and Combinatorial Optimization, Section 2.5, Figure 2.2.

6.2.1 Benchmark 1: Symmetry and Asymmetric Graphs

In this benchmark, one to four million TSP graph samples for the training set X are generated, as shown in Table 6.1. Additionally, a validation set of 10000 TSP graphs is generated to validate the learning progress of the PGN. For the final benchmarking, a test set of 1000 TSP graphs is generated by the Data Generator module. More precisely, each TSP graph contains $N = 20$ nodes, where each node is defined by coordinates that are uniformly drawn from unit 2-D square $[0, 1]^2$. During training a mini-batch size Batch Size (BS) with 256 TSP graphs is used to train the Actor module. The code was tested on the Microsoft Azure platform using NC-Series VMs powered by Nvidia Tesla K80 GPUs with 12 GiB RAM and an Intel Xeon E5-2690 v3 (Haswell) processor. Furthermore, CUDA 10.1 is used, where during testing the maximum GPU usage reaches 80%. The individual baselines are compared with the MAPE quality metric computed via MAPE($\boldsymbol{d}_0, \boldsymbol{d}_x$). Hereby, the MAPE value scores are defined by

$$\text{MAPE}(\boldsymbol{d}_0, \boldsymbol{d}_x) = \frac{1}{B} \sum_{b=0}^{\mathbf{B}-1} \frac{|\boldsymbol{d}_x - \boldsymbol{d}_o|}{\boldsymbol{d}_o}, \tag{6.2}$$

where d_o is the total TSP tour length computed by the OR solver using the local search setting. The OR tour lengths are compared to the calculated tour lengths computed from the PGN. The MAPE finally calculates the mean of all summed tour length deviations divided by d_o. This finally provides an evaluation value as a percentage that is independent of tour length units. The MAPE is calculated on the basis of the 1000 TSP graphs generated for the test set. Compared to previous work, the implemented PGN model is extended to work with asymmetric distances, which is called a Distance-Based Pointer Generation Network (DBPN) in this benchmark. Compared to the previous model Coordinate-Based Pointer Generation Network (CBPN), the new model thus can solve asymmetric TSP instances, which is not possible for the CBPN model. Hence, generating symmetric TSPs, both models can be trained and compared with respect to their solution quality or accuracy. As a reference baseline, the OR TSP solver is used to generate qualitative TSP tour results. During the benchmarks, different loss and activation functions were tested, which can be seen in Chapter 5, Section 5.11, Figure 5.25.

Results for Symmetric TSPs: Figure 6.3 shows the proportions of TSP graphs tested on the x-axis as a percentage, while on the y-axis the individual MAPE values achieved are shown as a quality performance measure. In Figure 5 different lines are shown, where the purple circled line is the first implementation of the DBPN without any modifications on loss and activation functions. The light blue crossed line shows the improved performance for the DBPN using modifications on loss and activation functions. The orange circled line and green crossed line are the performance achieved by the CBPN approaches. The CBPN approaches achieve better solution quality. However, they cannot capture the notion of TSP asymmetry. All network baselines are compared against the OR-Tools state-of-the-art solver, represented by the red colored dotted line at a MAPE of 0.00. The results shown in Figure 6.3 illustrate that the extended DBPN method with linear 1-D Convolutional Embeddings achieves a MAPE of 0.091,

which is a relative deviation of 9.1% compared to the OR solver. Here, the CBPN achieves a slightly better solution quality indicated by a MAPE of 4.6%. By using a different loss function Accumulated Actor-Critic Cross-Entropy Loss (AACEL) and the activation function called Mish the solution quality of the DBPN can be improved from a MAPE of $0.1289 \approx 13\%$ to $0.0592 \approx 6\%$. From a reliability perspective, Figure 6.3 shows that the initial implementation of the DBPN (including ReLU activation functions, mean actor loss) only achieved a solution quality of $0.10 \leq 10\%$ for 44% of 1000 TSP test graphs, i.e. 440 TSP graph problems. Here, the reliability could be improved using the actor critic cross entropy loss and Mish activation functions instead. For the aforementioned test set size of 1000 TSP graph problems, the DBPN requires 0.55 seconds, where OR by reference requires 15.03 seconds to finish all 1000 TSP graphs. In other words, the DBPN is $15.03[s]/0.55[s] \approx 27$ faster than the OR reference solver.

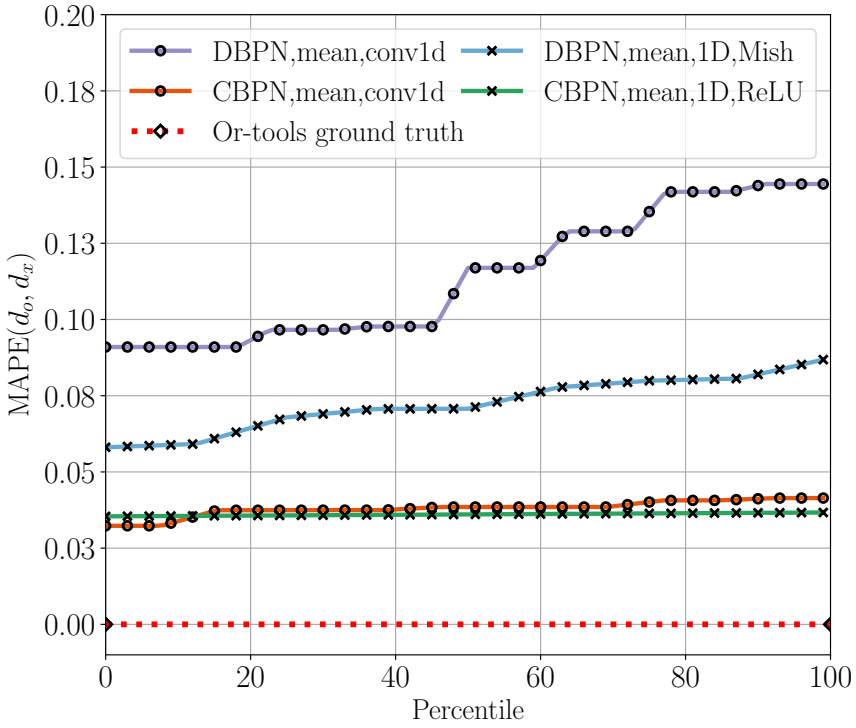


Figure 6.3: The benchmark results for symmetric TSPs with percentual coverage plot with MAPE quality assessment [Hamzehi et al., 2021b].

Results for Asymmetric TSPs: As previously mentioned, the CBPN approach by definition cannot capture the notion of TSP graph asymmetry. Thus the extended DBPN approach can only be compared with the OR reference solver in this second sub-benchmark. While training and aiming to solve asymmetric TSP graphs, network hyper-parameters are used that are almost exactly the same. The difference is that a small notion of random biases is exerted on a proportion of TSP graph links, which results in asymmetric TSP graphs for training and testing the DBPN. Again 1000 TSP test graphs are used, but this time a random uniformly distributed

bias $d^b_{i,j} = \mathcal{U}(0,1)$ is exerted on 1-22% of the existing TSP links per graph via $d^a_{i,j} = d^b_{i,j} + d^s_{i,j}$ or $d^a_{ji} = d^b_{ji} + d^s_{ji}$. Here, d^s_{ji} is the original symmetric distance value between any node pair. The random bias value is added on one of the link directions i,j or j,i via $d^s_{i,j}$, d^s_{ji}. By summing both distances the new biased and now asymmetric distance value $d^a_{i,j}$ is calculated. More precisely, the individual benchmarks are repeated 10 times to provide test reliability. Thus, there are 10 entries marked in Figure 6.4, illustrating the results with respect to asymmetric TSPs graphs. In contrast to the previous benchmark with symmetric TSPs, the solution quality of the DBPN (including the actor critic entropy loss and the Mish activation function) tested decreases by 2.3%. This can be inferred by a MAPE increase from 6.1% to 8.3%. Thus, the results of Figure 6.4 lead to the conclusion that the extended and improved DBPN approach is comparably robust when solving asymmetric TSP graphs. For this, the DBPN requires $0.51[s]$ seconds for 1000 asymmetric TSP graph instances, whereas OR requires 14.49 seconds for the same task. Thus, this illustrates that the DBPN method developed is more efficient than OR in general, and the tests indicate that the DBPN developed is currently approximately $14.49[s]/0.51[s] \approx 27$ times more efficient.

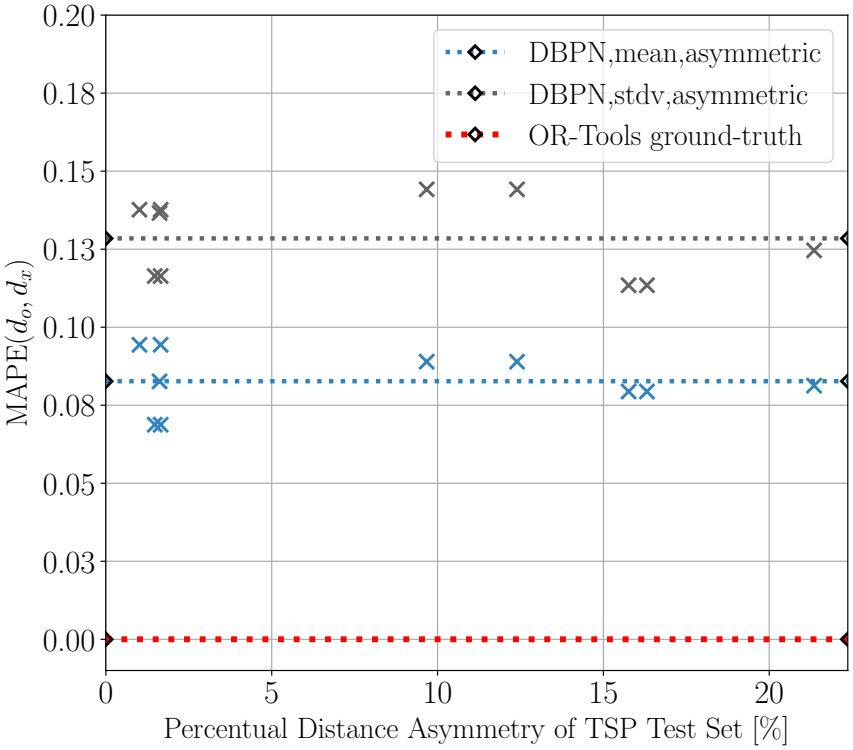


Figure 6.4: The benchmark results for asymmetric TSPs, asymmetry, and MAPE quality assessment [Hamzehi et al., 2021b].

6.2.2 Benchmark 2: Balanced and Unbalanced Graphs

In general, the results from Benchmark 1 are important for Benchmark 2. Here, the overall motivation is to reformulate the TSP formulation to work for multiple and unbalanced node classes such as vehicles, client-request pickup nodes, delivery or drop-off nodes, and service (charging or fueling) nodes (see Figure 6.5). This can be achieved by reformulating the TSP shortest-

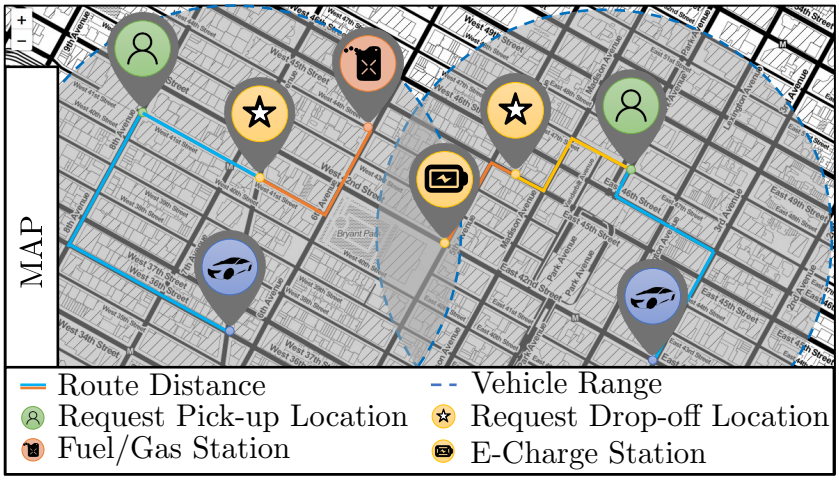


Figure 6.5: Motivation: Collaborative Routing, Service, and Maintenance Application [Hamzehi et al., 2021a].

path problem to an augmenting shortest-path problem for MWBM graphs. In comparison to Benchmark 1, where it is shown that the shortest-path for symmetric and asymmetric TSP distances can be learned for fixed node amounts by the PGN. The second Benchmark aims to test unbalanced MWBM problems with multiple node classes, asymmetric costs for each route, and increasing node complexity. Overall, the generic MWBM problem is a special case of the maximum flow or minimum cut problem. Here, the mathematical goal is to minimize the overall Objective Cost Function, which also can be expressed as MIP formulation. The multi-vehicle assignment problem is defined for a set of graphs $\mathcal{G}(\mathcal{N}, \mathcal{L})$ that contain random uniformly sampled bipartite graphs $\mathcal{U}(\mathcal{G}(\mathcal{N}, \mathcal{L}))$, where each graph again contains unbalanced sets of vehicle nodes $\mathcal{V} = \{v_0, ..., v_{n-1}\}$, request nodes $\mathcal{R} = \{r_0, ..., r_{m-1}\}$ and service or charging station nodes $\mathcal{S} = \{s_0, ..., s_{o-1}\}$ summarized by $\mathcal{N} = \mathcal{V} \cup \mathcal{R} \cup \mathcal{S}$. Each random vehicle assignment graph is characterized by a set of links $\mathcal{L}$ which represent all possible vehicle assignments or combinations among client requests and service station nodes. The complexity $|\mathcal{L}| = |\mathcal{V}| \cdot (|\mathcal{R}| + |\mathcal{S}|)$ can be represented by the amount of assignment links that increase by adding vehicle, request or service station nodes. Each graph problem can be expressed via a MIP formulation as follows

$$C(\boldsymbol{Y}|\boldsymbol{X}) = min \sum_{i=0}^{\mathcal{R}-1} \sum_{j=0}^{\mathcal{V}-1} [d_{i,j} - r_i] \cdot x_{i,j} \tag{6.3}$$

subject to the constraints

$$x_{i,j} \in [0,1],\ d_{i,j} \in [0,...,\sqrt{2}],\ r_i \in [0,...,1], \tag{6.4}$$

$$\sum_{i=0}^{\mathcal{V}-1} x_{i,j} \leq 1,\ \sum_{j=0}^{\mathcal{R}-1} x_{i,j} \leq 1,\ \sum_{k=0}^{\mathcal{S}-1} x_{i,j} \leq 1, \tag{6.5}$$

$$\sum_{i=0}^{\nu-1}\sum_{j=0}^{\zeta-1}[-d_{i,j} + r_i] \cdot x_{i,j} \geq \varkappa \cdot max\, r_i, \tag{6.6}$$

$$\sum_{i=0}^{\mathcal{R}-1}\sum_{j=0}^{\mathcal{V}-1} x_{i,j} = min\left[|\mathcal{V}|, |\mathcal{R}+\mathcal{S}|\right]. \tag{6.7}$$

Here, $\nu \in \mathcal{V}$ is the set of origin nodes, i.e. artificially sampled vehicle nodes, and $\zeta \in \{\mathcal{R}, \mathcal{S}\}$ is the set of destination nodes, i.e. the sampled pickup/drop-off locations and service stations. The decision variables are represented by assignment sequences that contain integers for the assignment of individual graph links $\mathcal{X} = \{x_0, ..., x_{\mathcal{L}}\}$. Additionally, $\mathcal{Y} = \{y_0, ..., y_{\mathcal{L}}\}$ are the resulting objective cost values for each route link assignment. Furthermore, the constraints in equation (6.4) limit the range of distance, and range values between $[0, 1]$, and importantly x is a binary matrix $\mathbb{1}(x; cond) = \{0, 1\}$ where the values mean

$$\mathbb{1}(x; cond) = \begin{cases} 1 & \textbf{if } \text{vehicle assignment}, \\ 0 & \textbf{otherwise}. \end{cases} \tag{6.8}$$

The constraint in equation (6.6) represents the condition that no vehicle should exceed its own vehicle range for any client or service station route. Within the constraint, the distance values $d_{i,j,k}$ and range values r_i are represented via the negative Euclidean distance based on the random uniformly sampled coordinates within unit 2-D square $[0, 1]^2$. This allows problems to be designed where the longest possible driving distance and vehicle range is $\sqrt{2}$. Here, the possibility exists that the PGN draws assignments that exceed the current vehicle range, which, however, is not desired and should be avoided during learning and testing. Further, the term $\varkappa \cdot max\, r_i$ describes that individual range values are normalized based on the vehicle with maximum driving range. This allows to represent individual vehicle range values as relative percentage within float value ranges of $[0, 1]$, see Equation (6.4). Finally, constraint (6.7) ensures that vehicle or request/service station unbalances are considered and handled. In other words, the solution length depends on the minimal amount of nodes between the vehicle node set and the aggregated request and service station node set within each MWBM graph.

Specific Benchmark Details: Benchmark 2 again requires the generation of training, learning, and test data sets. The graph complexity hereby increases by adding more nodes for each sample graph within training, validation, and test data. In more detail, the node amounts of vehicles, client requests, and charging stations are randomly sampled within ranges of $[3, 6]$ in order to

get different complex MWBM graphs. As shown in Table 6.1, the training data consists of artificially generated 1-2 Million unbalanced MWBM graphs and the validation set again consists of 10.000 unbalanced MWBM graphs. For the final benchmark and quality evaluation, the Data Generator module is used to generate 1000 random artificial MWBM test graphs. The PGN parameters are again Xavier random-uniformly initialized, and maximum gradient clipping of 1.0 using the ℓ_2 norm and Dropout of 0.2 is used to avoid the exploding gradient phenomenon and possible over-fitting. The code runs on a PC with Intel Core i7-6700K, 8 Core Processors, running on 4 GHz, and 2 GeForce GTX 1080 GPUs for training and solving the problems with the other algorithm baselines. The Operating System (OS) used is Ubuntu 18.04.3 LTS with 16 GiB RAM memory. Across the benchmarks, the tested PGN leverages GRUs with 128 hidden units and a mini-batch size of 15. This means, that 15 MWBM Graphs are simultaneously processed until no batches are left within the training data. Subsequently, the next learning epoch begins starting with the same 15 batches in the beginning. The learning epochs progress until a predefined maximum learning epoch parameter L is reached. Here the network is trained for $L = [5, 10, 15, 20]$ learning epochs, as shown in Table 6.1.

Evaluation and Solution Scoring: Due to the existence of unbalanced MWBM graphs, the length of the calculated solution sequences may vary depending on the minimum number of nodes of vehicles or client request, and service station nodes. Hence, a simple definition of an accumulated objective cost (sum) or a mean-based definition would not work for comparisons. Since the prior objective is to find all unique assignments before minimizing driving distances and maximizing the fleet range, it is necessary for later benchmarking to define a function model as Objective Cost as well as Evaluation Function. The developed Objective Function and evaluation function is as follows:

$$s_x(x_{i,j}) = \left[\sum_{i=0}^{\mathcal{R}-1} \sum_{j=0}^{\mathcal{V}-1} x_{i,j} \right]^2 + \sum_{i=0}^{\mathcal{R}-1} \sum_{j=0}^{\mathcal{V}-1} [-d_{i,j} + r_i] \cdot x_{i,j} \ . \tag{6.9}$$

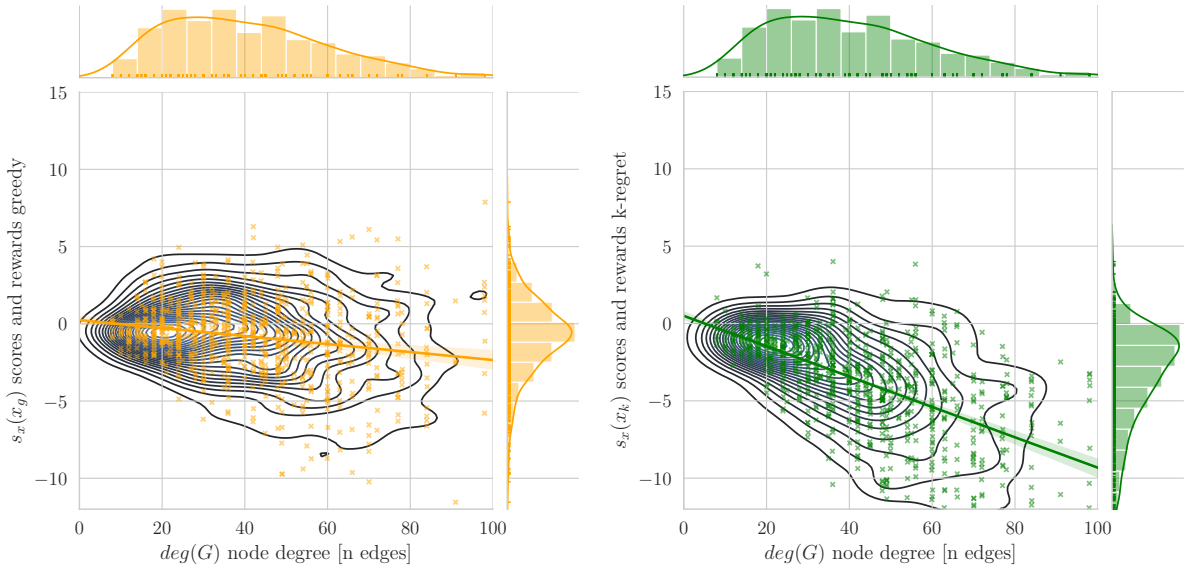
Thereby, all compared algorithms solve the 1000 unbalanced MWBM graphs according to the aforementioned cost function and are benchmarked against the optimal Cplex solver baseline as follows. In the next step, the Optimality Gap Score (OGS) is calculated in Equation (6.10)

$$S = \frac{\mathbf{s}_{x^*} - \mathbf{s}_x}{\mathbf{s}_{x^*}}, \tag{6.10}$$

which calculates the difference between the near-optimal cost values s_x calculated by the individual algorithms and the optimal cost values $\mathbf{s}_{x^*}$ provided by Cplex. Finally, the OGS score is represented by the scalar value S, where a higher score illustrates a closer solution toward the optimal solution of Cplex.

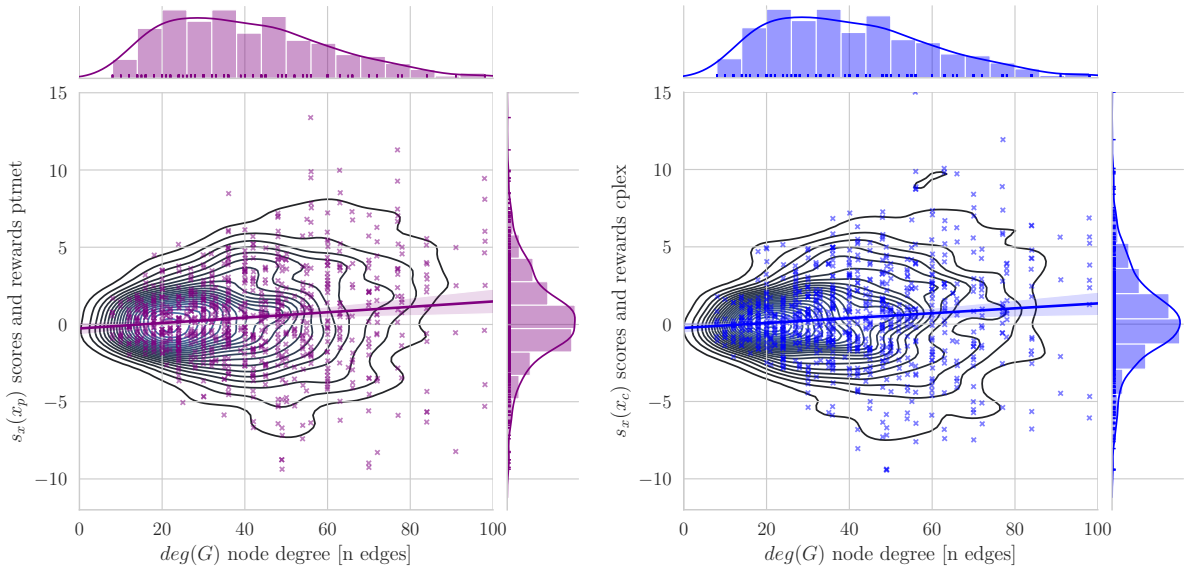
Results for unbalanced MWBMs: In this paragraph, the results of the PGN compared to the three other baselines with increasing complexity are presented. The baselines include a simple naive Greedy Heuristic, a k-Regret Heuristic, the self-developed PGN, and the optimal

solver Cplex. In order to provide more details about the stochasticity within the test data, the individually achieved Optimality Gap Scores (OGS) (y-axis) are visualized over the node degree $deg(G)$ (x-axis) using Kernel Density Estimation (KDE) plots (see Figures 6.6a, 6.6b, 6.6c and 6.6d). The Figures 6.6a, 6.6b, 6.6c, and 6.6d also include a linear regression plot which are



(a) The Greedy KDE.

(b) The k-Regret KDE.



(c) The PGN KDE.

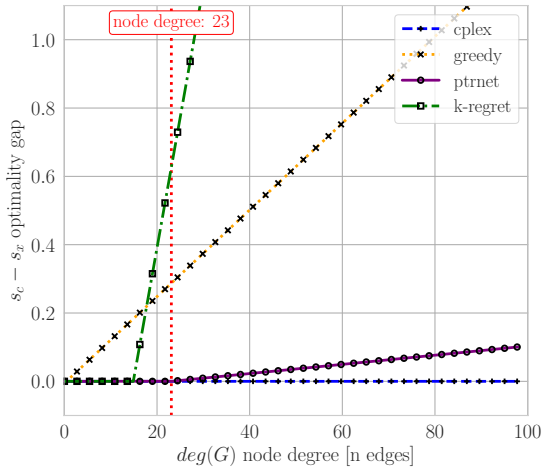
(d) The Cplex KDE.

Figure 6.6: The Kernel Density Estimates (KDEs) of individual algorithm OGS distributions and Linear Regression [Hamzehi et al., 2021b].

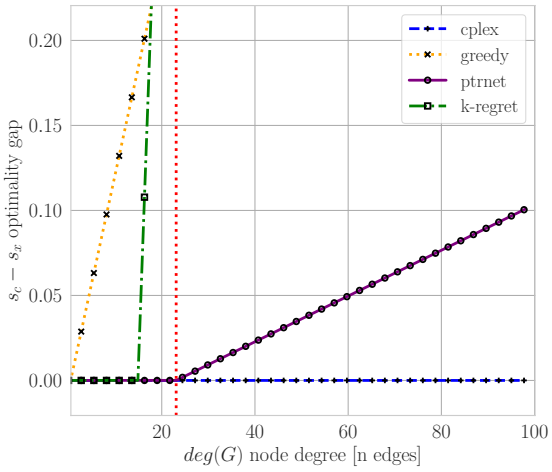
used to visualize trends of the achieved solution quality (scores) depending on different graph complexities. For instance, the figures show that the highest sampling density of MWBM graphs exists for a node degree of 20-50. In this node degree interval, the presented results can be interpreted with the highest stochastic confidence. Furthermore, the figures show that with increasing node degree (i.e. the measure of graph complexity) Cplex and the PGN find the highest scores, which indicates very closely optimal solutions (positive slope of linear regression). In contrast, the Greedy and k-Regret Heuristics do not find optimal solutions, which is indicated by the negative slope of the linear regression line. The k-Regret Heuristic requires tuning the parameter k, which, however, does not always can be optimized for all given MWBM graphs. Sometimes, the k-Regret Heuristic can outperform the Greedy Heuristic, however, this occurs seldom for fixed parameters k. Moreover, the results of the PGN seem to be promising in comparison with the optimal Cplex solutions. In order to provide more detailed insights, a more suited visualization is established subsequently. The regressive score lines of Figures 6.6a, 6.6b, 6.6c, and 6.6d are relatively compared with the most optimal Cplex baseline as follows. By calculating the relative score delta via Equation (6.10), the next Figures 6.7a and 6.7b can be generated.

The figures show all four baselines where the best scores achieved by Cplex are set on the zero axis. All slightly deteriorated solutions are indicated by a positive deviation from 0. Complementary, Figure 6.7b provides a closer view of 6.7a. Both Figures show that the PGN learns optimal solutions for MWBM graphs with node complexity below 23. Above a node degree of 23 $deg(G)$, the PGN solution quality degrades by $10[\%]/77[n] = 0.14\%$ per node degree. Relatively, the solution quality of the Greedy Heuristic deteriorates by $100[\%]/80[n] = 1.3\%$, while the k-Regret heuristic solution quality even decreases by $100[\%]/7[n] = 11\%$ per node degree for $deg(G) > 16[n]$. Interestingly, the k-Regret Heuristic reaches optimal results for comparably simple problems below a node degree of 16. Overall, Cplex always calculates the optimal solution, which is also illustrated in the next Figure 6.7c. Here, Figure 6.7c shows the coverage of all algorithms, which serves the analysis as a generic indicator for reliability. Cplex again provides the optimal baseline, however, more interestingly, the PGN matches the reliability of Cplex by 60% (PGN baseline matches Cplex baseline until the percentile of 60). Above a percentile of 60%, the plot illustrates that in total 40% percent of the remaining solutions are solved sub-optimally, but still outperform the Greedy and k-Regret heuristics. Clearly, the analytic results from Figure 6.7c also illustrate that the Greedy heuristic never finds the global optimum, but only provides approximated solutions. In contrast, the k-Regret heuristic solves very simple MWBM graphs optimally but degrades increasingly as aforementioned. Particularly, the next Figure 6.7d shows the potential of the PGN, which clearly outperforms Cplex with respect to solution time. The most efficient heuristic is the simple Greedy Heuristic, while the k-Regret Heuristic is also outperformed by the implemented PGN. For rather complex MWBM graphs, i.e. node degrees above 60 the PGN outperforms all other baselines with respect to the achieved solution times.

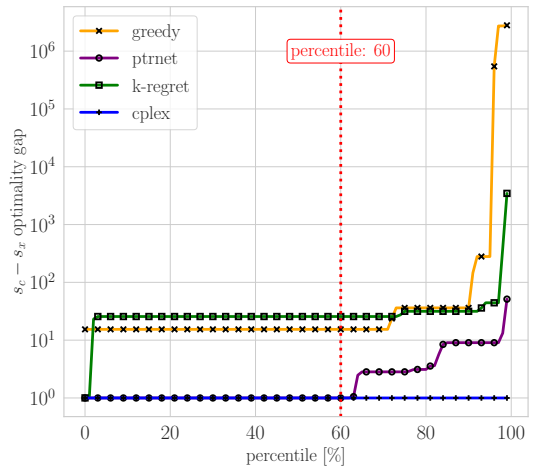
By introducing the Score Optimality-Time (SOT) function 6.8, a composed score-optimality-time (SOT) function of the achieved solution qualities and times can be created. The score



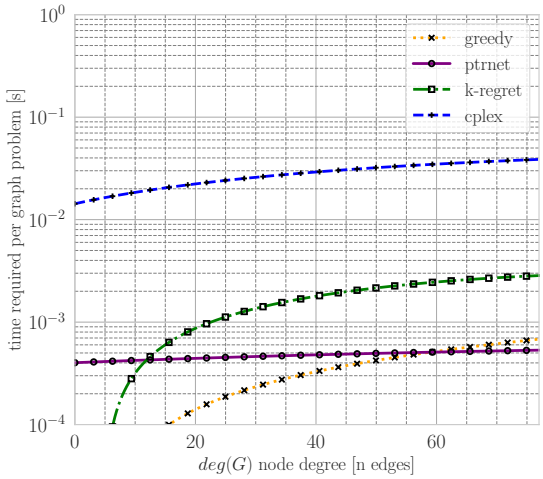
(a) Optimality gaps w.r.t. the test graph complexity.



(b) Enlarged version of Figure 6.7a.



(c) OGS with coverage (reliability).



(d) Solution times required for test data.

Figure 6.7: Algorithm solution time w.r.t. graph complexity [Hamzehi et al., 2021a].

metric (SOT) incorporates the solution optimality and time efficiency via

$$SOT(s_x, t_x) = s_x \cdot \log\left(\frac{1}{t_x}\right) , \tag{6.11}$$

where additionally the individual algorithmic results are depicted in Figure 6.8. Since the algorithmic solution times achieved by the individual algorithms are at the range/scale of $10^{-5} - 10^{-2}$ the values are artificially transferred to the same value range of the Optimality Gap scores. This is achieved by calculating the so-called inverse log-times via $\log 1/t_x$. The resulting values of the combined indicator function in Equation (6.11) are shown in Figure 6.8. Here,

the figure shows a significant peak at a node degree $deg(G)$ of 5. At this point, the heuristic algorithms, i.e. the Greedy and k-Regret heuristics, provide very good and efficient results. Thus, both heuristics initially achieve a higher SOT score compared to all other algorithms. However, with increasing MWBM graph complexity (i.e. increasing node degree), the solution quality degrades faster than for all other baselines.

Alongside the Greedy and k-Regret heuristic, Cplex achieves efficient and optimal solutions for comparably small MWBM graphs. Here, the solution quality by Cplex also outperforms the implemented PGN. However, for relatively complex problems with node degree $deg(G) > 50$, Figure 6.8 shows that the PGN outperforms even Cplex with respect to the combined indicators of solution quality and solution efficiency (time). Complementary, Table 6.3 shows multiple delta

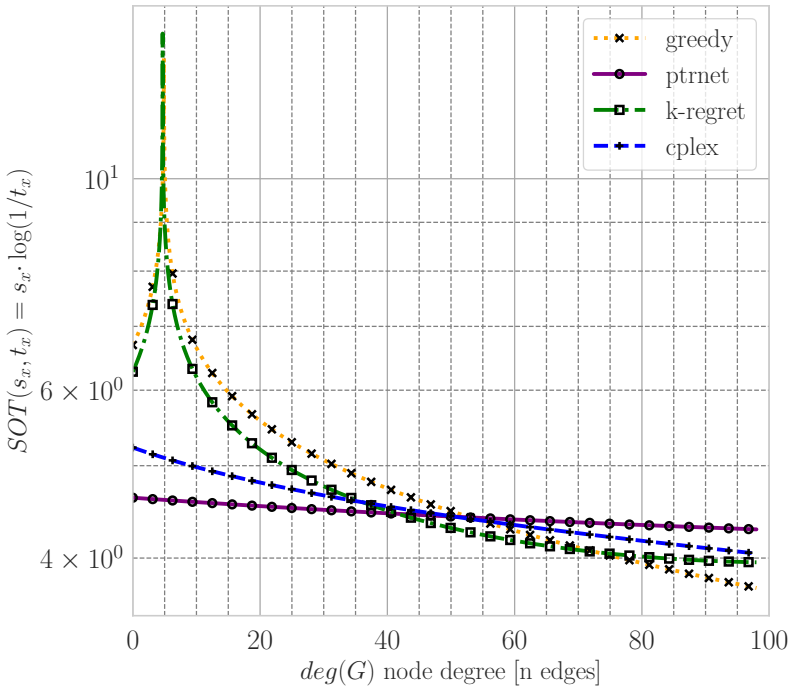


Figure 6.8: Combined time and solution quality comparison [Hamzehi et al., 2021a].

scores such as the delta of the Optimality Gap Score (OGS), the Assignment Gap Score (AGS), and the achieved augmented path tour length (ATL) relative to Cplex. Here, the AGS delta score basically shows if all assignments have been found by the algorithms and the ATL delta score essentially is the sum of all costs for each algorithm and subsequently subtracted from the optimal and minimal Cplex tour length. Note, that the scores of Table 6.3 are based on the optimal solution of Cplex, hence the column of Cplex has only zero entries. More importantly, the table shows that the PGN finds all assignments (solution completeness) equal to Cplex and the k-Regret heuristic. Here, the Greedy heuristic does not find all assignments. The PGN furthermore achieves the minimal mean of the achieved augmented path tour length deltas. In other words, this means that the PGN additionally is the best algorithm for minimizing the

objective MWBM costs compared to the other heuristics. Thus, the PGN achieves the best (minimal) delta OGS score of Equation (6.10). With respect to the achieved regressive solution

Metric	Greedy	k-Regret	Ptr-Net	Cplex
$\mu(\nabla S_{OGS})$	-1.2e+05	-1.0e+02	-0.4e+00	+0.0e+00
$\sigma^2(\nabla S_{OGS})$	+3.3e+11	+2.6e+05	+8.2e+01	+0.0e+00
$\mu(\nabla S_{AGS})$	-0.1e−01	+0.0e+00	+0.0e+00	+0.0e+00
$\sigma^2(\nabla S_{AGS})$	+1.5e+00	+0.0e+00	+0.0e+00	+0.0e+00
$\mu(\nabla S_{ATL})$	+0.3e−02	+0.1e−01	+0.2e−03	+0.0e+00
$\sigma^2(\nabla S_{ATL})$	+1.3e+01	+1.4e+02	+0.4e−01	+0.0e+00

Table 6.3: Delta Optimality Gap Score (OGS), Delta Assignment Gap Score (AGS), and Delta Augmented Path Tour Length (ATL) for a test dataset size of 1e+3 MWBM graphs [Hamzehi et al., 2021a].

times in Figure 6.7d, the next Table 6.4 shows the actual average solution time values $\mu(t_x)$ for all four algorithm baselines Greedy, k-Regret, PGN, and Cplex, i.e. $\mu(t_g)[s]$, $\mu(t_k)[s]$, t$\mu(t_p)[s]$, and $\mu(t_c)[s]$. For statistical completeness, the next Table 6.5 also provides the achieved solution

$deg(\mathbb{G})$	$\mu(t_g)$[s]	$\mu(t_k)$[s]	$\mu(t_p)$[s]	$\mu(t_c)$[s]
18	5.8e−02	1.5e−03	5.8e−04	5.8e−02
28	2.6e−02	1.1e−03	5.4e−04	2.6e−02
32	2.6e−02	1.2e−03	5.2e−04	2.6e−02
36	2.7e−02	1.9e−03	5.6e−04	2.7e−02
40	2.7e−02	2.3e−01	5.5e−04	2.7e−02
44	2.8e−02	2.1e−01	5.4e−04	2.8e−02
49	2.4e−02	8.7e−04	5.2e−04	2.4e−02
60	3.2e−02	2.5e−03	5.7e−04	3.2e−02
77	2.6e−02	1.7e−03	5.4e−04	2.6e−02

Table 6.4: Mean values of algorithm solution times in seconds [s] [Hamzehi et al., 2021a].

time variances for all four algorithm baselines $\sigma(t_g)[s], \sigma(t_k)[s], \sigma(t_p)[s], \sigma(t_c)[s]$ respectively. In combination with both Tables, Figure 6.7d shows that the PGN solution time is stable when increasing the MWBM graph complexity, i.e. an increase of the node degree $deg(G)$. Thus, for large problem instances, the PGN provides the second lowest variance in column $\sigma^2(t_p)$[s]. Here, the Greedy heuristic is the most stable (lowest variance). In contrast, Cplex ends up having the fourth highest variance, which shows that the solver does not provide real-time results. Interestingly, even the k-Regret heuristic outperforms Cplex with respect to the achieved solution time variance.

6.2.3 Benchmark 3: Constrained and Real Data Graphs

The third and final benchmark includes the results for constrained unbalanced MWBM test graphs. In comparison to the benchmarks made before, more algorithms and baselines for further comparisons are implemented. For qualitative comparisons, some well-performing baselines such

$deg(\mathbb{G})$	$\sigma^2(t_g)$[s]	$\sigma^2(t_k)$[s]	$\sigma^2(t_p)$[s]	$\sigma^2(t_c)$[s]
18	2.5e−09	5.4e−09	1.2e−09	1.1e−03
28	3.6e−08	5.5e−07	1.4e−09	3.4e−05
32	0.0e+00	0.0e+00	0.0e+00	0.0e+00
36	0.0e+00	0.0e+00	0.0e+00	0.0e+00
40	4.3e−08	1.1e+00	2.2e−09	3.5e−05
44	4.4e−08	1.0e+00	1.5e−09	1.0e−04
49	5.6e−08	9.4e−07	2.1e−09	6.2e−05
60	3.4e−08	6.9e−07	1.6e−09	3.0e−05
77	6.9e−09	4.0e−06	2.6e−10	4.3e−06

Table 6.5: Variance values of algorithm solution times in seconds [s] [Hamzehi et al., 2021a].

as OR-Tools local search, the Greedy Heuristic (GHR), Cplex, Random Iterative Search, the Hungarian-Munkres-Kuhn (HMK) algorithm, the Jonker-Volgenant-Castanon (JVC) [Jonker and Volgenant, 1987, pp. 325–340], the CBC-coin Branch and Cut ILP solver, [Vigerske, 2017, p. 1] and Vogel's approximation method [Selmair et al., 2019, pp. 261–266] are included. The qualitative results for each of the 2000 test graphs (2 test datasets) can be seen in Figure 10.

The figure shows that for unbalanced and constrained MWBM graphs the PGN solution is close to the optimal solution provided by Cplex (blue line). Here all objective costs achieved by the PGN are below the costs achieved by the Vogel's heuristic (VGL) and the Greedy heuristic (GRH). The figure also shows the algorithmic median and mean objective cost values for all test graphs which show the same fact. The next Figure 6.9 provides a closer zoom on the first 50 subgraphs within the first 1000 MWBM graphs. The plot below the objective values shows each individual node configuration for each MWBM test graph. Further, the plot shows that the test graphs are unbalanced with respect to the amounts of vehicle nodes, client request, and service station nodes solved by the PGN network. Interestingly, the network objective values have a higher variance than the other heuristic baselines, which is a similar variant to the Cplex optimal baseline. This indicates that the PGN learns to solve individual MWBM graphs in an intelligent manner, which comes close to the solutions provided by Cplex. The figure also shows that the PGN does not provide optimal solutions equal to Cplex.

In order to look for a behavioral pattern, the next plot aims to cluster all solutions depending on all unique node configurations available in the test data. From the data generation process where 3-6 nodes are sampled for each class of vehicles, requests, and service stations, it follows that there must be 64 individually different classes of node configurations (or graph variants) in total. This can be seen in the next Figure 6.10. The upper plot of Figure 6.10 shows the objective values, and more specifically the medians and means of all algorithm baselines such as the Greedy (GRH), Cplex (CPL), Hungarian-Munkres-Kuhn (HMK), Jonker-Volgenant-Castanon (JVC), Cbc-coin (CBC), random iterative tabu search (RTS), and the final Pointer Generation Network (PGN) objective values. The values are clustered with respect to their individual node configuration occurrence in the test data sets. Figure 6.10 shows that there is no general conspicuous pattern, however, there is one peak around the unique node configuration index of 50. The test graph is relatively big compared to other test graphs with 6 nodes for all vehicle-, request- and service station sets. This means, that the individual node configuration does not

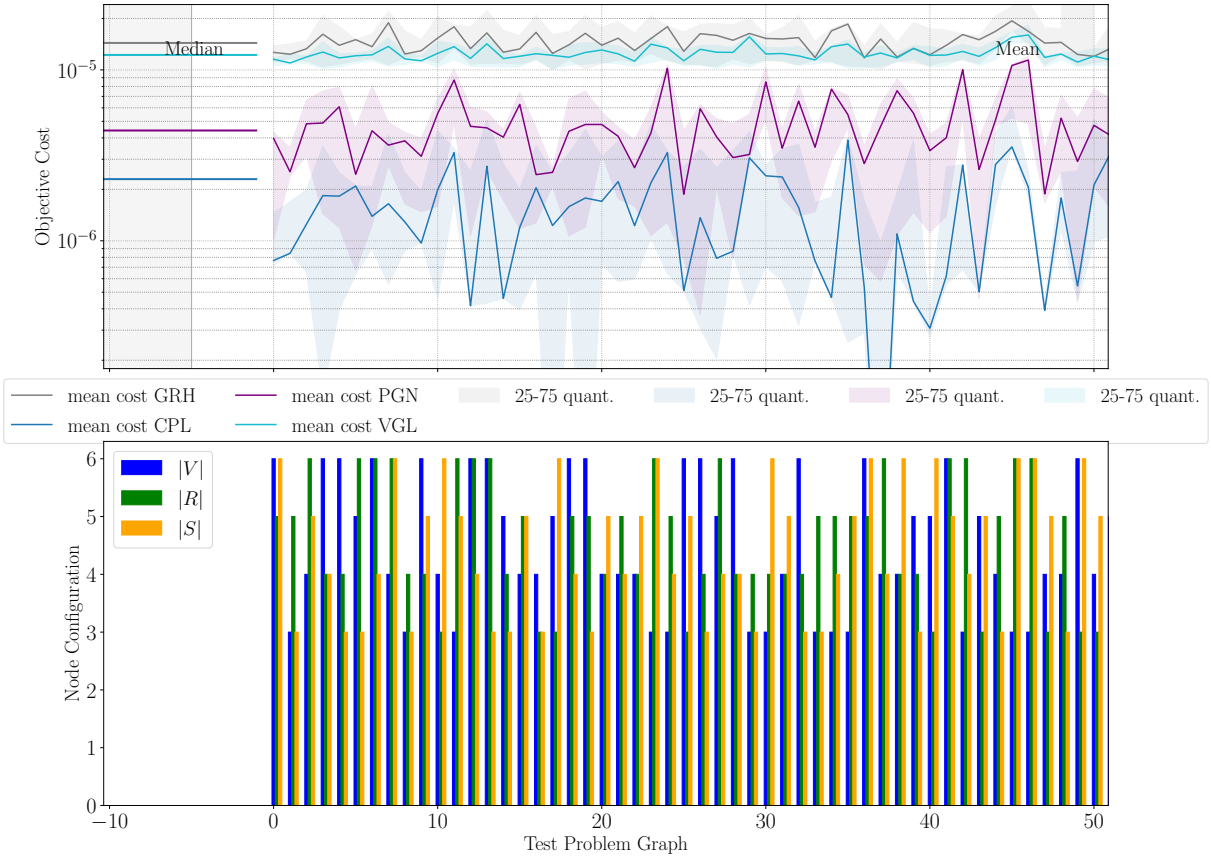


Figure 6.9: Enlarged version of Figure 10.

have an impact on the solution quality, however, the size of each individual graph surely does, specifically on the heuristic-based solution methods. Supplementary, the next Figure 6.11 shows the solution times achieved by the individual baselines. The figure in general shows the medians, 25% and 75% quantiles of all achieved solution time measurements for each algorithm baseline. The plot below further shows the individual MWBM test graph node configurations among 1000 test graph problems. A closer look at Figure 6.11 reveals that all algorithm solution times fluctuate depending on the size of the individual test graph. The individual test graphs further are shown in the plot below in the same figure. Interestingly, the local search heuristic from OR-Tools and the optimal Cplex solver have noticeable fluctuations, whereas in the case of Cplex, this can be explained by its stochastic run-times which are already known in the literature. Moreover, the plot shows that simple heuristics are more stable with respect to their solution time, while more sophisticated algorithms tend to be characterized with more solution time variance. In general, the plot also shows that the solution times do mainly vary in the cases of larger MWBM test graphs. It can be observed that the node configuration only has an indirect influence. In other words, it does not matter which class of nodes has larger node sets than any other node class. Finally, the next Figure 6.12 provides a qualitative summary of the solution qualities achieved among all algorithm baselines.

In Figure 6.12, the x-axis shows all algorithms and their average objective costs on the y-axis.

Metric	LCS	GRH	CPL	HMK	JVC	CBC	RTS	**PGN**	VGL
$\sum C(x)$	1.1148	1.1474	0.00486	0.00897	0.008974	0.0090	0.01065	**0.0093**	0.0247
$\mu(x)$	4.3e-04	3.0e-03	2.4e-06	4.9e-06	4.5e-06	4.5e-06	5.16e-06	**4.7e-06**	1.2e-05
$\tilde{\mu}(x)$	4.90e-06	1.43e-05	2.31e-06	4.22e-06	4.22e-06	4.22e-06	4.92e-06	**4.41e-06**	1.2e-05
MAE	43.0e-03	0.003e-03	0.0	2.06e-06	2.06e-06	2.07e-06	2.68e-06	**2.23e-06**	9.94e-06
MSE	4.30e-02	3.01e-03	0.0	2.06e-06	2.06e-06	2.07e-06	2.68e-06	**2.24e-06**	9.93e-06
MAPE[%]	9.2	12.1	00.0	2.7	2.7	2.8	3.2	**3.2**	10.6
RMSE	0.0015	0.0034	0.0	0.0014	0.0014	0.0014	0.0016	**0.0014**	0.0031
RMSPE[%]	1.5	3.4	0.0	1.4	1.4	1.4	1.6	**1.4**	3.1
MEDE	2.70e-06	1.20e-05	0.0	1.94e-06	1.94e-06	1.95e-06	2.57e-06	**2.12e-06**	9.94e-06
MEDPE[%]	22	100	0	16	16	16	21	**18**	82

Table 6.6: Baseline solution quality results normed to Cplex optimal solutions.

Generally, Figure 6.12 reveals that the PGN achieves mean objective costs close to the JVC, HMK, Coin Branch and Cut/Bound (CBC) algorithms. In the ranking, those algorithms share the second best qualitative results behind the optimal solutions by the Cplex solver. The PGN has less variance in the found qualitative solutions than all other heuristics such as the local search (LCS) by the OR-Tools solver, the greedy heuristic (GRH), and Vogel's approximation method (VGL). Clearly, the figure also shows that the PGN can find better solutions than the simple naive Greedy heuristic GRH, which is a positive key indicator that the combinatorial solutions learned by the network are better or more intelligent than simply finding the next best MWBM edge. The potential of the approach can again be seen in Figure 6.13 and Table 6.7, which reveals that the PGN requires the least median time for each among the tested MWBM graphs. Here, the PGN requires $0.0003[s]$ seconds (median) for each MWBM graph, whereas the optimal MIP Solver CPLEX, requires $0.02[s]$ seconds, being much less efficient for the same task. In numbers, the efficiency gain factor between the PGN approach and CPLEX is $0.02[s]/0.0003[s] = 66.67$. Compared to the most efficient heuristics such as the Local Search (LCS) (LCS OR-Tools, requires $0.0091[s]$) and Vogel's approximation (requires $0.0470[s]$) the efficiency gain using the PGN approach would be factors of $0.0091[s]/0.0003[s] = 30.33$ and $0.0470[s]/0.0003[s] = 156.67$. Hence, the results of the PGN approach show that factors of 10 to 200 in efficiency could be gained using the PGN for static batching and large-scale real fleet dispatching scenarios.

6.3 Executive Summary

This chapter has shown three different benchmarks, where our proposed DRL method called Pointer Generation Network (PGN) shows efficient capabilities for solving balanced, unbalanced, symmetric, asymmetric, and even constraint combinatorial optimization problems with artificial TSPs and MWBM graphs with respect to real-world vehicle-to-request-to-services station settings.

The first among three benchmarks shows that for the first time, a PGN-based approach is used to solve symmetric but also asymmetric TSPs. Using different forms of feature Embeddings (coordinates and distances), two different methods are used and compared to OR-Tools LCS (ground truth). By achieving MAPE values of 9.1% for the DBPN and 4.1% for the original CBPN method, the results show that embedding with distances compared to coordinates results

Metric [s]	LCS	GRH	CPL	HMK	JVC	CBC	RTS	**PGN**	VGL
$\max(t)$	0.0500	0.1370	0.0408	0.1412	0.1490	0.2891	1.2897	**0.0004**	0.0758
quant. 75%	0.0137	0.0817	0.0270	0.0799	0.0816	0.2187	0.9160	**0.0004**	0.0564
$\tilde{\mu}(t)$	0.0091	0.0807	0.0257	0.0793	0.0805	0.1958	0.7641	**0.0003**	0.0470
quant. 25%	0.0054	0.0803	0.0230	0.0789	0.0797	0.1812	0.7181	**0.0003**	0.0440
$\min(t)$	0.0040	0.0791	0.0193	0.0782	0.0790	0.1592	0.6675	**0.0003**	0.0420

Table 6.7: Baseline solution times with minimum, median, maximum, and 25-75% quantiles.

in a noteworthy deterioration of solution quality performance. However, this is a necessity, if a Neural Network based solution method should be enabled to learn real-world graph instances with asymmetric graphs. When comparing the solution times of both methods the solution times required for the DBPN and CBPN approximately doubles, when using distances instead of coordinates as Embeddings. Nevertheless, the DBPN method is approximately 27 times, and the CBPN method is approximately 53 times more efficient than OR-Tools LCS. As a final note, the CBPN method is generally unable to learn asymmetric graph instances by design, and thus, cannot be used for solving real-world graph examples without any adaptations.

The second benchmark is extended towards testing balanced and unbalanced MWBM graphs, which often occur in realistic vehicle-to-request fleet assignment or matching settings. The key essence of the proposed approach is that it learns to solve balanced and unbalanced MWBM graphs for an upper bound, i.e. a node degree of 23, where a synthetic dataset is used. Within this benchmark, the PGN provides solution qualities that match the solution quality of the MIP solver CPLEX up to the aforementioned node degree bound. However, the PGN solution quality degrades by 0.14% per node degree which is less than all other tested heuristics. In other words, when testing larger and more complex graphs, the PGN method outperforms the tested heuristics Greedy and k-Regret with respect to solution quality. Here, CPLEX is used as a ground-truth mechanism and thus always provides optimal results. Important to note is also that the PGN provides the least solution times overall tested methods, Greedy, k-Regret, and even CPLEX. This property is highly desired for efficient solution methods and thus leaves the conclusion that the PGN provides an efficient solution method, working as a trainable efficient and near-optimal heuristic for balanced and unbalanced methods.

The third and final benchmark data set is extended to match the complexity and characteristics of real-world vehicle-to-request matching/assignment graphs. Here, the dataset includes balanced, unbalanced, symmetric, asymmetric, and also constraint MWBM graphs, which have been stated as a target from the beginning. Multiple solver baselines and algorithms such as LCS (heuristic), GRH (heuristic), CPLEX-IL (ground-truth solver), HMK (exact), JVC (exact), CBC (exact), Random Tree Search (RTS) (dynamic programming), Vogel's Heuristic (VGL) (heuristic) are compared to the proposed PGN with algorithmic (mask reuse in reward, updated reward function) and Embedding adaptions compared to both previous benchmarks. The key outcome of the

third benchmark is that again the PGN shows the best solution times when it comes to solution efficiency. Additionally, the PGN outperforms all other heuristics when turning the subject to overall solution quality. Thereby, the PGN nearly reaches (overall 3.2% MAPE to CPLEX) the solution quality of open-source exact solvers such as HMK, JVC, and CBC. Only CPLEX, which is used as a ground-truth solver, provides exact and optimal results. As a final note, the PGN provides a very efficient (approx. 66 times faster than CPLEX) solution method, by also providing near-optimal results, shown as 3.2% MAPE. The major drawbacks should be pointed out as follows. Generally, the PGN method becomes increasingly unstable if more variables and constraints are added to the objective function and overall optimization problem. This requires extra means (monitoring and algorithmic changes) to cope with such challenges. If enough countermeasures are set and a high learning stability is provided, the proposed PGN method is a very interesting/efficient choice for solving batches of small to medium-sized MWBM graphs in a realistic vehicle-to-request-to-service station setting; and can be adapted/generalized to other settings with some efforts as well.

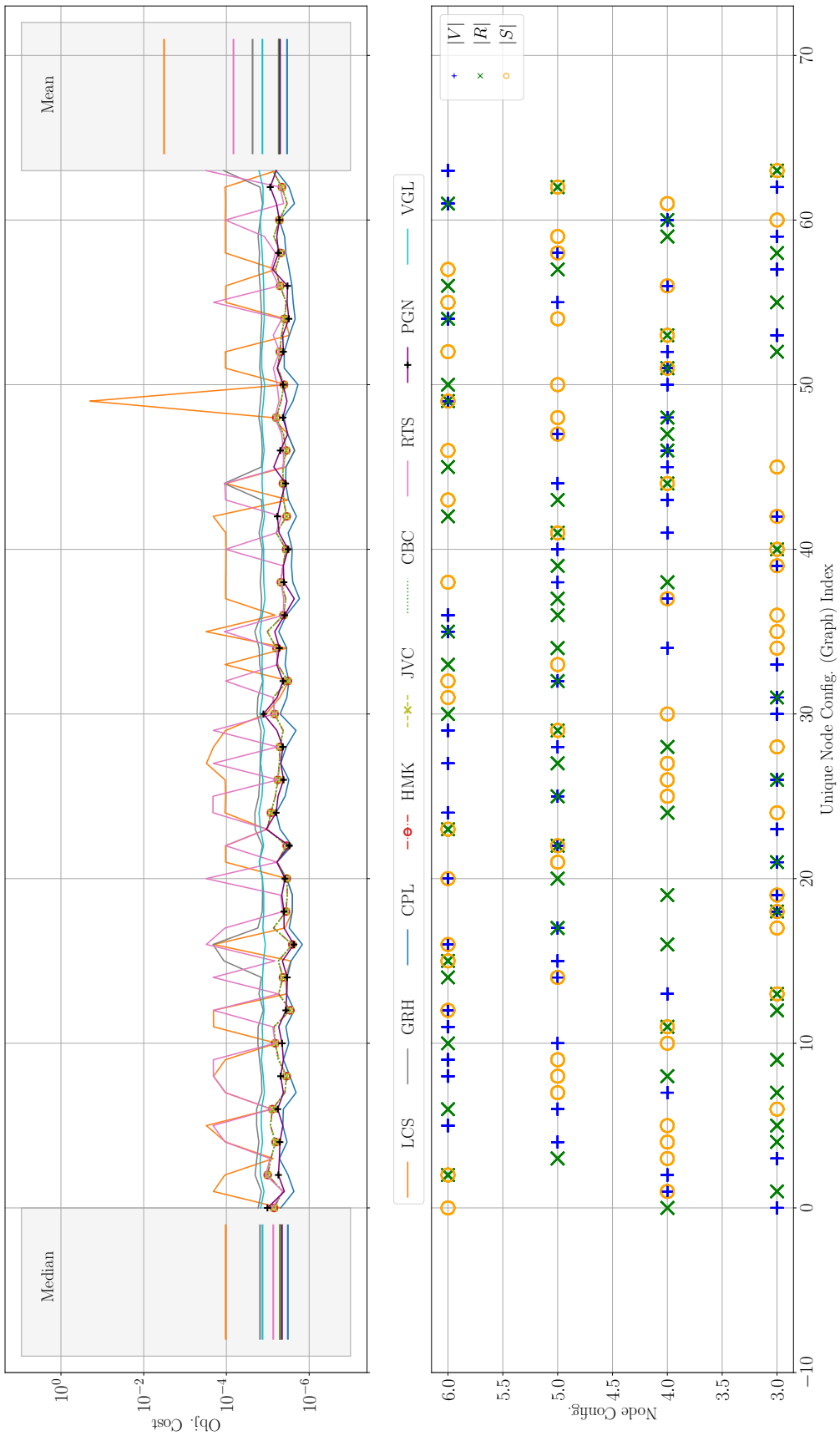


Figure 6.10: Solution quality with clustered objective values with respect to unique node configurations.

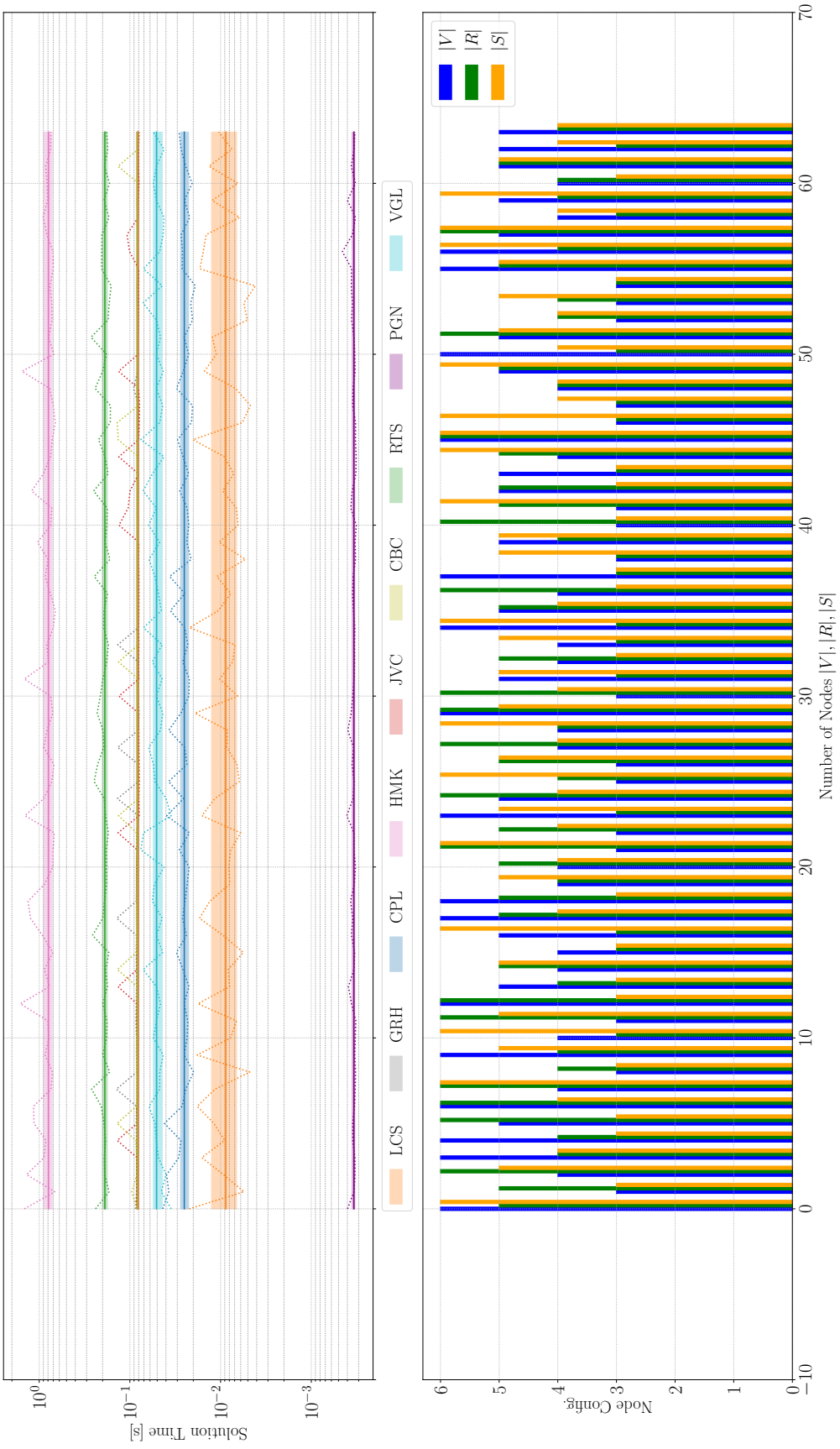


Figure 6.11: Solution times of the benchmarked algorithms.

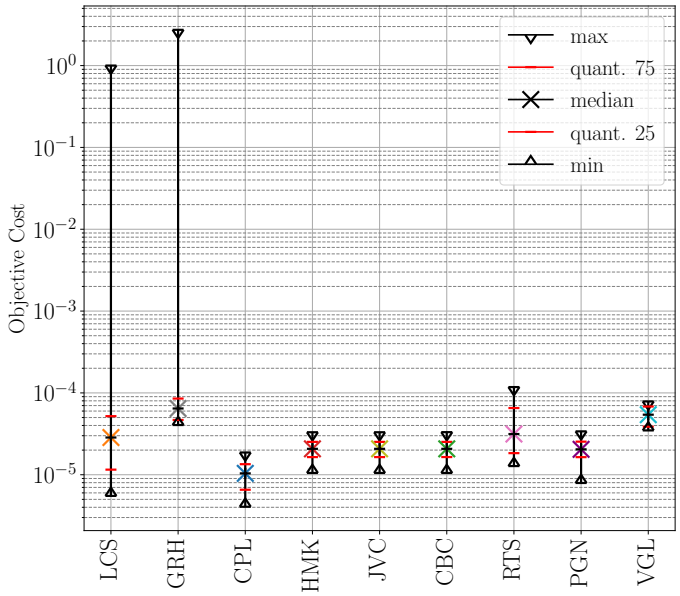


Figure 6.12: Benchmarked algorithms with resulting solution errors (higher mean objective cost means worse solution quality).

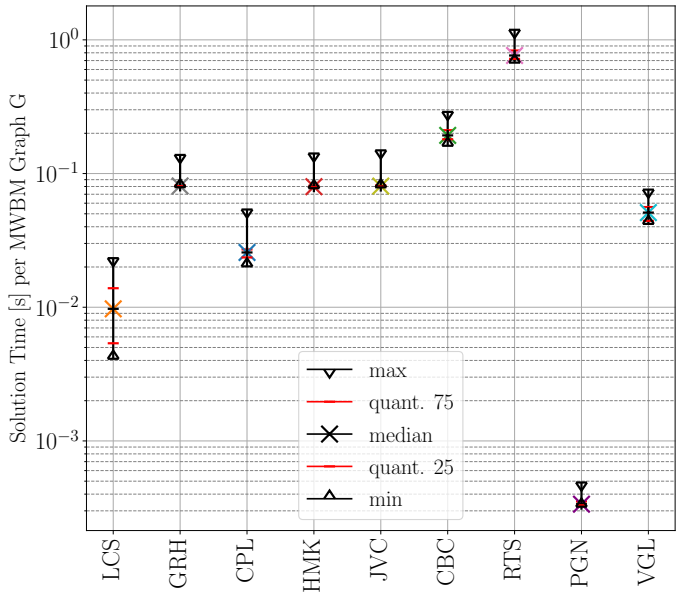


Figure 6.13: Benchmarked algorithms with resulting solution times per MWBM graph (higher solution time means less efficient).

7. Conclusions & Future Research

This thesis developed and presented a novel deep reinforcement learning method that can be used to efficiently solve vehicle-to-client request assignments and higher abstractions of combinatorial fleet planning problems. This chapter summarizes the conclusions made and the further research directions possible within this context of deep reinforcement learning and the solution of combinatorial optimization problems.

7.1 Answers to Research Questions

Initially, some research questions were proposed, which are answered in the following section.

- *Research Question 1: Can a Deep Reinforcement Learning approach learn near-optimal fleet routing and assignment controls for generally occurring graph structures in collaborative fleet routing and dispatching?* The general answer to this question is yes, with the limitations that are explained in Section 7.2 in more detail. Particularly, with increasing size and graph complexity, the final solution quality achieved deteriorates. The network size should be increased, which also results in increased requirements such as requiring more computational resources, memory, and training time.
- *Research Question 2: If Research Question (RQ) 1 is yes, does this assumption hold for symmetric and asymmetric graphs?* The results of Chapter 6 and Benchmark 2 in Section 6.2.2 show that the asymmetry has a noticeable impact on the resulting solution quality. More specifically, the MAPE degrades from 6.1% to 8.3% when learning symmetric and asymmetric graphs. Also, the model requires double the amount of memory to learn asymmetric graphs.
- *Research Question 3: If RQ 1 and RQ 2 are yes, does this assumption hold for balanced and unbalanced graphs?* Yes, balanced and unbalanced graphs can be learned by the presented PGN approach, which is shown in Chapter 6, Section 6.2.2. The results show that the network can interpolate, which means that learning from larger complex graphs and solving smaller and less complex graphs can be achieved with near-optimal results. With increasing graph complexity, the PGN approach solution quality deteriorates for node degrees above 23 when solving for 1000 unbalanced MWBM graphs.
- *Research Question 4: If RQ 1, RQ 2, and RQ 3 are yes, does this assumption hold for mathematical problem constraints?* At first, this question had to be answered as no. In the presented benchmark in Chapter 6, Section 6.2.2 it was not possible to model hard constraints with a mask function only. The reason for this is that, the more constraints

and more neurons are masked, the learning capabilities of the PGN are increasingly limited (due to the loss of learnable information). It is possible to learn soft constraints by using the cost function and modeling soft constraints via a soft mask function. Hard constraints, however, could only be satisfied with a 100% guarantee by applying a hard mask function in the Decoder Module. This is a novelty that is shown in Chapter 6, Section 6.2.3 in more detail. In particular, this approach has shown to be successful and did not have any major losses with respect to the solution quality. Given these facts, this question is now answered as yes.

- *Research Question 5: If RQ 1, RQ 2, RQ 3, and RQ 4 are yes, can a DRL-based approach be applied to real fleet data?* In Chapter 6, Section 6.2.3, real New York Taxi Data is used to generate the distributions of training, validation, and test data. Here, realistic distances and vehicle ranges are generated to generate the MWBM problem graphs with 3-6 nodes of vehicle, client requests, and service stations. The results of Chapter 6, Section 6.2.3, show that input data normalization between value ranges of $[0, 1]$ is recommended to improve the learning efficiency and thus the solution quality. In general, large value ranges deteriorate the learning capabilities of the PGN network, since very small cost values or large rewards can lead to greedy and stagnating learning behaviors.

7.2 Limitations of the PGN

Next, this section will outline important limitations found while training and benchmarking the presented Actor Critic PGN model. First, there is the aspect that

- the presented Actor Critic PGN learns a heuristic and does not work like a basic mathematical solver. In other words, there is no guarantee of optimality, and sophisticated evaluation and monitoring means are required to ensure aspects as such (MIP competitive solution quality). In other words, a limitation of this approach is that permanent human/machine supervision is required to monitor correct training and solution behavior.
- With respect to the PGN solution performances achieved, it can be observed that the solution quality deteriorates increasingly if the test graph structures diverge increasingly from the training graph samples. In other words, if different graph variants are trained, the training graph distribution has an influence on the final prediction/solution quality of the PGN. Generally, small graphs are learned easier than large and complex graph samples.
- Another important aspect during training is learning efficiency and stability. In general, a lot of computational resources are required for training large Actor-Critic PGN models, whereas later on during operation, a lot of computational resources can be saved. In more detail, the computational resources required for computation do not ultimately decrease compared to traditional solutions with exact solvers but rather are shifted to an earlier time of the data processing chain. Note, that efficient data generation and processing methods increasingly matter for the overall algorithm operation process. In this work, one or two weeks are required to generate 1 Million that is 1e06 MWBM graphs to complete the training dataset automatically.

- With respect to very important learning stability, the generation of artificial data usually results in a stable and efficient learning process. On the contrary, using real data slows down the data generation process, hence mitigating re-learning capabilities, deteriorates the learning stability, and therefore also indirectly decreases the final achievable solution quality compared to the generation of artificial data. The learning stability is largely influenced by the choice of the DRL model and its sub-components. Generally, the presented policy gradient procedure is considered a method of high variance, hence the model is prone to exploding gradient phenomena, which can be observed via oscillating learning or achieved cost signals within monitoring tools, e.g. Tensorboard and similar. Additionally, learning phases with many graph features or large and very dissimilar imbalanced graphs are very likely to result in unstable learning. This is also a strong limitation, which shows that the application of such an ML model is currently technically limited on the side of learnable graph complexities. However, the application scenarios of such a DRL model are different from the ones of mathematical MIP solvers. In this case, traditional mathematical MIP solvers and MWBM algorithms are superior to DRL methods. Another limitation is that the stability of the presented PGN model is influenced by the design of the used cost or reward function. In practice, summed equations usually result in comparably stable learning, whereas on the other hand, product or differences result in high cost or reward fluctuations, which should be avoided for stable learning behaviors.
- Generally, in order to achieve excellent learning performances during training, the PGN requires a rigorous optimization of hyper-parameters (network parameter settings), thus a lot of testing is required beforehand. Not only, the behaviors of the PGN learning quality, efficiency, and stability can change drastically, but also only one hyper-parameter can be changed at a time in order to observe the behavioral outcome. Consequently, this further makes the hyper-parameter optimization process a time-consuming effort.
- Compared to other models implemented, the Actor Critic PGN is relatively memory expensive (compared to “Convolutional Architectures”) due to the many required sub-components within the architecture. Additionally, enabling the model to learn and solve asymmetric cost problems approximately doubles the required memory for symmetric cost or distance scenarios. Here, a minimum of 800 MB of memory is required while training symmetric TSPs with 20 nodes, whereas 1.7 GiB is required to solve the same setup for asymmetric TSPs (due to the inclusion of bidirectional edges).
- Overall, the learning, training, and validation process requires quite large amounts of data, processing power, and memory requirements. As a rule of thumb, it is not recommended to train the network below 500,000 training samples, if a final solution quality above 80% should be achieved. Since graph samples with 20-50 nodes per graph sample are comparably small graphs, the training for larger graphs would require even more data and more processing power. For practical applications, training with small node counts still may be very useful for different applications, however, from a scientific perspective, the capabilities of such models are still quite limited. In other words, at this point in time, training such a model still does not provide acceptable solution qualities for very large

graphs, i.e. containing more than 50 nodes per graph problem.

- Another limitation is that constraints are comparably hard to realize practicably in the learning process, whereas they can be easily implemented in the decoding process of the Actor Critic PGN. Generally, adding too many constraint formulations can lead to unstable learning and decoding (solution) behaviors, which limits the model to some use cases with complex constraint formulation scenarios.
- The learning of asymmetric graphs requires more memory than symmetric graphs. The benchmarks showed that learning asymmetric graphs requires approximately twice the amount of computational memory during the training and learning phases on GPUs. Besides, the first benchmark shows that the learning of asymmetric MWBM graphs with 20 Nodes doubles the required memory consumption from 0.8 GiB to 1.7 GiB in the case batch sizes of 256 MWBM graphs are used.

7.3 Conclusions

This section summarizes the results from Chapter 6, which includes benchmarks and tests of the Actor Critic PGN for 1) symmetric and asymmetric TSPs in Section 6.2.1, 2) balanced and unbalanced MWBM in Section 6.2.2, and 3) symmetric, asymmetric, balanced, unbalanced and constrained MWBM in Section 6.2.3. Generally, the presented PGN approach can be used to find very good (superior to state-of-the-art heuristics) and even near-optimal solutions for NP-hard problems (like the symmetric TSP). However, it is important to note, that the approach does not solve the TSP, but rather finds a good heuristic (i.e. good solution strategies or rules) automatically. In other words, there is no guarantee that such an approach finds always the optimal solution, which can be explained by the limited memory capacity of every neural network and thus the DRL approach. In reality, the most occurring combinatorial problems like the directed MWBM problem are not NP-hard. Although there are such heavy limitations, DRL approaches like the presented PGN can be game-changing in practice and operation. Besides, a scientifically interesting thing is that the presented PGN method may be capable of showing if a given combinatorial real world problem is NP-hard or not.

The first benchmark in Section 6.2.1 shows two different methods called CBPN and DBPN, which are part of the benchmarks for symmetric and asymmetric TSPs. The benchmarks showed that both methods can be used to learn efficient heuristics to solve both symmetric and asymmetric planar 2-D Euclidean TSP problems respectively. More specifically, the DBPN presents an extended and novel approach to tackle the asymmetric TSP setting and thus provides crucial advantages for the application of optimizing collaborative fleets within real road networks with one-way roads or tidal lanes, or other asymmetric travel cost settings. Moreover, the DBPN provides a scalable method for combinatorial optimization problems by achieving near-optimal combinatorial solutions. The solution quality largely depends on the training quality, where conversely, the benchmarks showed that a MAPE of 6% compared to state-of-the-art solvers such as OR can be learned with high reliability. The potential of the presented PGN method can be seen when looking at the solution times. The solving for 1000 TSP instances is 27 times

faster than with the OR solver. Compared to the CBPN method the DBPN variant achieves solution qualities by only 3% worse, which enables the DBPN to model more complex use cases with respect to real world application.

The second benchmark in Section 6.2.2 focuses on the learning of balanced and unbalanced MWBM graphs in a vehicle to client request or service station assignment setup. The focus lies on finding complete solutions for unbalanced problems which are a major challenge for the presented approach. In the experiments, three different baselines (algorithm comparisons) are shown, while optimal solutions are provided by Cplex. The results show that the PGN method learns globally optimal solutions to an upper bound depending on the number of nodes, or more specifically the actual node degree of 23 (i.e. the graph complexity expressed as graph coherence or number assignment possibilities). But for more complex MWBM graphs, the solutions found by the PGN deteriorate at a rate of 10% per node degree.

The third benchmark in Section 6.2.1 focuses on the learning of relatively complex (mix of balanced, unbalanced, and constrained) MWBM graphs. Again, multiple algorithm and solver baselines (such as OR-Tools local search, the Greedy Heuristic (GRH), the simplex algorithm via Cplex, Random Iterative Search, the Hungarian-Munkres-Kuhn (HMK) algorithm, the Jonker-Volgenant-Castanon (JVC), the CBC-Coin Branch and Cut ILP solver and Vogel's approximation method) are used for qualitative and quantitative comparisons and other analyses.

Based on the changes made to the PGN architecture in 6.2.3, the results show that the PGN is capable of finding near-optimal solutions even for constrained MWBM graphs. In the case of the best training possible, the PGN outperforms all other heuristics and can even equal the solution quality of the JVC, HMK, and CBC baselines, showing the advantage of much faster computation times. Only Cplex still provides better results with respect to solution optimality. Generally, the presented PGN network architecture has significant advantages when solving small to medium-sized (10-50 nodes) MWBM graph problems. In this case, the PGN approach is comparably flexible when compared to state-of-the-art heuristics (Greedy, k-Regret, Vogel's Approximation, etc.). Moreover, the PGN can be trained on different TSP or MWBM graphs based on different city data and still can find very good solutions. The approach is furthermore very efficient when thousands of MWBM graphs should be processed via batches (multiple graphs in one group). By using a mask function for constraint modeling during the learning and decoding phase, even constrained MWBM graphs can be learned and constraint satisfaction guarantees can be ensured, i.e. required when the route length of the vehicle-to-client assignments could exceed the current vehicle range.

Further, the results from Benchmark 3 show that the learning stability of the PGN deteriorates for increasingly complex MWBM graphs. However, if the learning phase is completed successfully (actor training and validation cost signals are minimized as much as possible), the PGN learns competitive solutions with state-of-the-art solvers such as Cplex and OR. In general, the PGN hereby outperforms state-of-the-art heuristics in solution quality and provides competitive solution efficiencies. The next section will summarize possible future research on this topic. Also recommendations are provided in order to lead future research toward promising directions.

7.4 Future Research

Overall, the results of this thesis have shown that the use of the Embedding and Attention architecture has a large influence on the solution quality, learning efficiency, and precious memory-computational resources. From an architectural perspective, it would be very interesting to explore how the solution quality could be improved when introducing and implementing more sophisticated Embedding methods and Multi-Head Attention modules. The work of [Kool et al., 2018] shows that even for complex combinatorial and dynamic graphs, optimal results can be expected. The drawback of this extension, however, may be that the training time is further extended, while memory requirements may also increase. For this reason, more sophisticated Embedding methods should be introduced and tested, which could decrease the memory required for Multi-Head attention modules. Another way of improving the solution quality is to implement a complementary decoding method such as Beam Search, which can be seen in the work of [Bello et al., 2016]. However, the already known method would require an extension to unbalanced and constrained MWBM graphs, which is not straightforward. Since the presented PGN approach does not match the solution quality of Cplex for unbalanced and constrained MWBM graphs, it would be scientifically interesting to extend the already existing results of Benchmark 3 for fixed node sets. Generally, it is advised to train the presented PGN approach with fixed node sizes for training, validation, and test graphs. This is because, the training runs on variable graphs have not shown that the PGN performance could be improved, and also the training has shown to deteriorate concerning stability. Additionally, an interesting idea would be to train multiple PGN networks for different MWBM graph structures, but this requires large computational resources which may only be available in the cloud.

From a testing perspective, it would be scientifically interesting to include tests with fixed solution time for optimal and heuristic algorithms. Since not all algorithms could be set up with respect to this aspect, this would only be possible with Cplex, OR-Tools algorithms, and the CBC solver. All other optimal transport algorithms would only output a solution if the algorithm has reached its termination condition. On one side, the termination condition could be relaxed, however, this makes it more difficult to compare to fixed solution time settings. Another scientifically interesting approach using neural Sinkhorn Layers is shown in [Emami and Ranka, 2018], which could yield advantages with respect to the learning time required. Since this approach is novel, only very few authors have tried to leverage it. The presented approach sometimes requires 1 week on the GPU setup shown in Chapter 6, Section 6.1, and 1 day in the cloud. In other words, the learning time is another important perspective to consider, where other aforementioned approaches may be very useful. Finally, it would be scientifically interesting to see how this approach performs under dynamic fleet routing and approximation conditions. The capabilities of retraining the PGN from an offline to an online approach are important. Hence, this thesis may serve to provide creative hints for future research that is still to come.

Acronyms

AACMCPG	Advantage Actor-Critic Monte-Carlo Policy Gradient
AACEL	Accumulated Actor-Critic Cross-Entropy Loss
AC	Actor-Critic
ADP	Approximate Dynamic Programming
AI	Artificial Intelligence
ALNS	Adaptive Large Neighborhood Search
AM	Attention Mechanism
AP	Assignment Problem
API	Application Programming Interface
ATTN	Attention
AWS	Amazon Web Service
BDI	Batched Data Input
BS	Batch Size
CBC	Coin Branch and Cut/Bound
CBPN	Coordinate-Based Pointer Generation Network
CNN	Convolutional Neural Network
CO	Combinatorial Optimization
COP	Combinatorial Optimization Problem
CUDA	Compute Unified Device Architecture
CVRP	Capacitated Vehicle Routing Problem
DBPN	Distance-Based Pointer Generation Network
DCO	Discrete Combinatorial Optimization
DL	Deep Learning
DNN	Deep Neural Network
DP	Dynamic Programming
DQN	Deep Q-learning Network
DRL	Deep Reinforcement Learning
ETA	Estimated Time of Arrival
ELU	Exponential Linear Unit
FCFS	First-Come-First-Serve
GDP	Gross Domestic Product
GCAN	Graph Convolutional Attention Network
GCN	Graph Convolutional Network
GiB	Gigabyte
GHz	Gigahertz
GMHAN	Graph Multi-Head Attention Network
GNN	Graph Neural Network
GPU	Graphics Processing Unit
GPS	Global Positioning System
GRH	Greedy Heuristic
GRU	Gated Recurrent Unit
HMK	Hungarian-Munkres-Kuhn
HR	Heuristic
ID	Identifier
ILP	Integer Linear Program
ITS	Intelligent Transportation Systems
i.i.d.	independent and identically distributed
JVC	Jonker-Volgenant-Castanon
KDE	Kernel Density Estimation
KDEs	Kernel Density Estimates

KRH	k-Regret Heuristic
LCS	Local Search
LKH	Lin-Kernighan Heuristic
LSAP	Linear Sum Assignment Program
LSTM	Long-Term Short-Term Memory
MAPE	Mean Absolute Percentage Error
MC	Monte-Carlo
MCTS	Monte-Carlo Tree Search
MDP	Markov Decision Process
MHR	Meta-Heuristic
MIP	Mixed Integer Programming
ML	Machine Learning
MRP	Markov Reward Process
MWM	Minimum Weighted Matching
MWBM	Minimum Weighted Bipartite Matching
MWBPM	Minimum Weighted Bipartite Perfect Matching
MWMPs	Minimum Weighted Matching Problems
MS	Microsoft
MSE	Mean Squared Error
mTSP	Multi-Travelling Salesman Problem
mVRP	Multi-Vehicle Routing Problem
NET	Networks
NLP	Natural Language Processing
NN	Neural Network
NTM	Neural Turing Machine
O-D	Origin-Destination
ODM	On-Demand Mobility
OGS	Optimality Gap Score
OP	Orienteering Problem
OR	Operations Research
OS	Operating System
PCA	Principal Component Analysis
PCTSP	Price Collecting TSP
PDP	Pickup-and-Delivery Problems
PGN	Pointer Generation Network
PGP	Pointer Generation Process
PN	Pointer Network
PSO	Particle Swarm Optimization
QAP	Quadratic Assignment Problems
QL	Q-Learning
RAM	Random-Access Memory
ReLU	Rectified Linear Unit
RH	Ride Hailing
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RQ	Research Question
RS	Ride Sharing
RTS	Random Tree Search
Seq2Seq	Sequence-to-Sequence
Struc2Vec	Structure-to-Vector
SA	Simulated Annealing
SDVRP	Split Delivery Vehicle Routing Problem
SI	Système International (d'unités))
Sigm	Sigmoid
SL	Scheduling
SPG	Sinkhorn Policy Gradient
SOC	State-Of-Charge
SOF	State-Of-Fuel
SOT	Score Optimality-Time
SPCTSP	Stochastic Prize Collecting TSP
TanH	Tangens Hyperbolicus
TSP	Traveling Salesman Problem
TSPs	Traveling Salesman Problems
USCS	United States Customary System
VGL	Vogel's Heuristic
VM	Virtual Machine
VNS	Variable Neighborhood Search
VRP	Vehicle Routing Problem
VRPs	Vehicle Routing Problems

Notation

Conventions

This work uses different fonts for different types of symbols and mathematical entities.

- Sets containing integer numbers and real-values are denoted by *uppercase blackboard bold* letters.
- Tensors are denoted by *uppercase bold sans-serif* letters.
- Matrices are denoted by *uppercase bold serif* letters.
- Scalars and vectors are denoted by *lowercase serif letters*.
- Random variables are denoted by *uppercase standard calligraphic letters*.
- Functions are generally denoted by *standard calligraphic non-italic letters*.
- Algebraic and Numerical Spaces are denoted by *capital double-struck letters* (e.g. $\mathbb{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$).
- Graph Theory sets are denoted by *uppercase calligraphic letters* (e.g. $\mathcal{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$) and Machine Learning sets are denoted by *Typewriter letters* (e.g. `ABCDEFGHIJKLMNOPQRSTUVWXYZ`).
- Distance Matrices e.g. the origin-destination planar 2-D Euclidean $\mathbb{R}^2$ Distance Matrices, computed from vectors are denoted by *calligraphic italic letters with function parentheses "$D(\cdot)$"*.

The elements of a vector $\boldsymbol{a}$ are denoted by $\boldsymbol{a}_i$, the elements of a matrix $\boldsymbol{A}$ as $\boldsymbol{A}_{i,j}$, and the elements of a 3-D tensor (i.e. a multi-dimensional matrix) by $\boldsymbol{A}^{[A,B,C]}$, where the tensor is indicated by multiple superscripts $[A, B, C]$, that are separated by commas. In some equations, a dot is used to increase the readability of a product (e.g. $\mathbf{a} \cdot \mathbf{b}$), but this is not the same as a scalar-vector product of two vectors $\mathbf{c}^T\mathbf{x}$. Further, this should not be confused with the Hadamard-Schur element-wise product of matrices $\mathbf{C} \odot \mathbf{X}$.

Numbers & Arrays

a	An integer or real scalar variable.
$\langle a, b \rangle$	A tuple of scalar variables.
$\{(a, b)\} := \{(a, b)(b, a)\}$	A permutation as Cauchy notation.

A	A scalar constant.
$\boldsymbol{A}$	A matrix.
$\boldsymbol{a}$	A vector.
$\boldsymbol{A}^{[A,B,C,D]}$	A tensor with dimensions $ABCD$.
$\boldsymbol{W}$	A weight matrix.
$\boldsymbol{I}^{N\times M}$	The $N \times N$ identity matrix.
$\boldsymbol{X}^{N\times M}$	Dataset inputs (matrix with N rows, M columns, one for each graph sample).
$\boldsymbol{Y}^{N\times M}$	Dataset outputs (matrix with N rows, M columns, one for each graph sample).
$\boldsymbol{x}^{N}$	Input data sample vector with size N.
$\boldsymbol{y}^{M}$	Output data sample vector with size M.
a	A scalar random variable.
$\mathbf{a}$	A vector-valued random variable.
$\mathbf{A}$	A matrix-valued random variable.

Table 7.1: The Numbers & Arrays Notation.

Scalars

I	Length of input dimension.
E	Embedding Set Size or dictionary length.
L	Number of Learning Epochs.

Table 7.2: The Scalar Notation.

Spaces

$\mathbb{A}$	A numerical space.
$[a, b]$	The real-valued interval including a and b.
$]a, b]$	The real-valued interval excluding a and b.
$\mathbb{A}/\mathbb{B}$	A numerical space subtraction, i.e. the space containing elements of numerical space $\mathbb{A}$ that are not in space $\mathbb{B}$.
$\mathbb{N}$	The numerical space of natural numbers.
$\mathbb{N}^{+}$	The numerical space of positive natural numbers.

$\mathbb{Q}^+$	The numerical space of rational numbers.
$\mathbb{Q}^+$	The numerical space of positive rational numbers.
$\mathbb{R}$	The numerical space of real-valued numbers.
$\mathbb{R}^+$	The numerical space of positive real-valued numbers.
$\mathbb{R}^2$	The numerical space of the 2-Dimensional Euclidean Space.

Table 7.3: The Space Notation.

Nodes and Locations

$v_n^{[e,g]}$	A e electrical or g gas combustion engine vehicle node v with ID n.
r_m^p	A client request pickup node r with ID m.
r_m^d	A client request drop-off node r with ID m corresponds to a pickup node.
$s_o^{[e,g]}$	A service station node s with e electrical charging or g gasoline fueling possibility and ID o.

Table 7.4: The Node Notation.

Sets

$\{0, 1\}$	A set containing 0 and 1.
$\{0, 1, ..., n\}$	A set of all integers between 0 and N.
$\mathcal{H}$	The hypotheses set capacity, denoting the complexity of a machine learning model (e.g. NN).
$\mathcal{G}$	A graph in which nodes n denote a random variable $\mathrm{n}^{(i)}$ and each edge denotes a conditioned dependency which is *directed* or a correlated dependency which is *undirected*.
$\mathcal{N}$	The set of nodes.
$\mathcal{L}$	The set of links w.r.t. numbers of nodes and their adjacency.
$\mathcal{V}$	The set of vehicle nodes.
$\mathcal{R}$	The set of request nodes.
$\mathcal{C}$	The set of charging or maintenance nodes.
$\mathcal{Q}$	The set of coordinate tuples $q_i = \langle x_i, y_i \rangle = \{q_i\}_{i=0}^{\mathcal{N}-1}$ set.
$\mathtt{G}$	The set of graph samples.

$\mathtt{X}$	The set of training problem distributions containing the training examples.
$\mathtt{V}$	The set of validation problem distributions containing the training examples.
$\mathtt{T}$	The set of testing problem distributions containing the training examples.
$\mathcal{B}$	The set of batches that is a subset of training, validation or test set examples from $\mathcal{X}$.
$\mathcal{P}^*$	The set of solution paths (node index sequences) that minimize or maximize a combinatorial problem.

Table 7.5: The Set Notation.

Indexing

$\boldsymbol{a}_i$	Select element i of vector $\boldsymbol{a}$, with the index starting at 0.
$\boldsymbol{a}_{-1i}$	All elements of a vector $\boldsymbol{a}$ except for element i.
$\boldsymbol{A}_{i,j}$	Element i, j of matrix $\boldsymbol{A}$.
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$.
$\boldsymbol{A}_{:,j}$	Column j of matrix $\boldsymbol{A}$.
$\boldsymbol{A}_{i,j,k}$	Element i, j, k of a 3-D tensor $\boldsymbol{A}$.
$\boldsymbol{A}_{:,:,k}$	2-D slice of a 3-D tensor $\boldsymbol{A}$.
k	Number of available indices for action selection.
t	Discrete time steps denote the time steps $\{0, ..., n\}$.
s'	s' is the next iterative successor state of s.
π'	π' is the next iterative successor policy over π.

Table 7.6: The Index Notation.

Operators

$\lvert\mathcal{A}\rvert$	The cardinality length or size of set $\mathcal{A}$.
$\lvert\lvert\mathbf{a}\rvert\rvert_2$	The ℓ^2 norm of vector $\mathbf{a}$.
$\lvert\lvert\mathbf{a}\rvert\rvert_p$	The ℓ^p norm of vector $\mathbf{a}$.

$\square'$	Returns a rescaled vector with values ranging between $[0, 1]$ given a vector $\mathbf{a}$ or matrix $\mathbf{A}$ via $\mathbf{a}' = \frac{\mathbf{a}-\min(\mathbf{a})}{\max(\mathbf{a})-\min(\mathbf{a})}$.
$\max_a(\mathbf{a})$	Returns the maximum value given a vector $\mathbf{a}$ or matrix $\mathbf{A}$.
$\text{argmax}_a(\mathbf{a})$	Returns an index value of a at the maximum value given a vector $\mathbf{a}$ or matrix $\mathbf{A}$.
$\min_a(\mathbf{a})$	Returns the maximum value given a vector $\mathbf{a}$ or matrix $\mathbf{A}$.
$\text{argmin}_a(\mathbf{a})$	Returns an index value of a at the minimum value given a vector $\mathbf{a}$ or matrix $\mathbf{A}$.
$\text{softmax}(a)$	Transforms input values of vector or matrix a in value range $[0, 1]$ via $[e^{a_1}, e^{a_2}, ..., e^{a_l}]^T / \sum_{i=1}^{l} e^{a_i}$.
$\det(\mathbf{A})$	Determinant of matrix $\mathbf{A}$.
$\text{diag}(\mathbf{A})$	Returns the diagonal vector given of matrix $\mathbf{A}$.
$\ln(a)$	Natural logarithm of a as an inverse function of the exponential function $\exp(a)$.
$\exp(a)$	e^a, where $e \approx 2.71828$ is the base of the natural logarithm.
$\text{sign}(a)$	Returns the sign of a variable a.
$\mu(\mathbf{a})$	Returns the mean of a 1-D vector $\mathbf{a}$.
$\tilde{\mu}(\mathbf{a})$	Returns the median of a 1-D vector $\mathbf{a}$.
$\sigma^2(\mathbf{a})$	Returns the variance of a 1-D vector $\mathbf{a}$.
$\sigma(\mathbf{a})$	Returns the standard deviation of a 1-D vector $\mathbf{a}$.
$\doteq$	Equality relationship (true by definition).
$\approxeq$	approximately equal.
$\in$	scalar is an element of a set, e.g. $a \in \mathcal{A}$.
$\subset$	subset of a set, e.g. $a \subset \mathcal{R}$.

Table 7.7: The Operator Notation.

Functions

$d(\mathbf{a}, \mathbf{b}) = \|\|\mathbf{a} - \mathbf{b}\|\|_2$	The Euclidean distance of vector a.
$f : \mathbb{A} \to \mathbb{B}$	A function f with numerical space $\mathbb{A}$ and the range of numerical space $\mathbb{B}$.
$f(\mathbf{x}, \theta)$	A function of vector $\mathbf{x}$ parameterized by θ.
$\mathbb{1}(x; cond)$	The conditional indicator function.

Table 7.8: The Function Notation.

Calculus

$f'(a)$, $\frac{df}{dx}(a)$	Derivative of $f : \mathbb{R} \to \mathbb{R}$ at input point a.
$\frac{\delta f}{\delta x}(\boldsymbol{a})$	The partial derivative of $f : \mathbb{R}^n \to \mathbb{R}$ at input point a.
$\nabla f'(a) \in \mathbb{R}^n$	Gradient of $f : \mathbb{R}^n \to \mathbb{R}$ at input $\mathbf{a}$.
$\nabla f'(\boldsymbol{A}) \in \mathbb{R}^{m \times n}$	Matrix derivatives of $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ with input matrix $\mathbf{A}$.
$\nabla f(\boldsymbol{A})$	Tensor derivatives f with input tensor A.

Table 7.9: The Sub- and Superscript Notation.

Linear Algebra

$\mathbf{A}^\top$	Transpose of matrix $\mathbf{A}$.
$\det(\mathbf{A})$	Determinant of matrix $\mathbf{A}$.
$\mathbf{A} \odot \mathbf{B}$	Element-Wise (Hadamard-Schur) Product of Matrices $\mathbf{A}$ and $\mathbf{B}$.
$[\mathbf{A}, \mathbf{B}]$	Concatenation of matrices $\mathbf{A}$ and $\mathbf{B}$ does depend on dimensions and concatenation axis.
$\mathrm{tr}(\mathbf{A})$	Trace of matrix $\mathbf{A}$.
$\mathbf{e}^{(i)}$	The i-th standard basis vector (a one-hot vector).

Table 7.10: The Linear Algebra Operational Notation.

Probability Theory

$P[X = x]$	A probability that a random variable X takes on the value x.
$X \sim p(x)$	A random variable X is chosen with the distribution $p(x) \doteq P[X = x]$.
$\mathbb{E}[X]$	The expectation value of a random variable X, i.e. $\mathbb{E}[X] = \sum_x p(x)\, x$.
$\bar{\sigma} = Std[f(x)]$	The standard deviation of a random variable $\frac{1}{N}\sum_{i=0}^{N-1}(x_i - \mu)^2$.
$\bar{\sigma}^2 = Var[f(x)]$	The variance of a random variable $\sqrt{(\frac{1}{N}\sum_{i=0}^{N-1}(x_i - \mu)^2)}$.
$\Pi = \{\pi_0, ..., \pi_i\}$	A policy which short-term for a probability distribution $\{P(y_0\|x_0), ..., P(y_i\|x_i)\}$ over an action index sequence.
$\boldsymbol{Y} = \{y_0, ..., y_i\}$	A learned or calculated decision sequence output cf. "policy" $\Pi = \{\pi_0, ..., \pi_i\}$.
$p(\boldsymbol{Y}\|\boldsymbol{X})$	A probability distribution defining the mapping of a specific input sequence $\boldsymbol{X}$ to an output sequence $\boldsymbol{Y}$ of decision variables.
$p(s', r\|s, a)$	The probability of transition to state $s\prime$ with reward r, from state s and action a.
$p(s'\|s, a)$	The probability of transition to state $s\prime$ from state s and action a.
$r(s, a, s')$	The expected immediate reward on transition from s to $s\prime$ under action a.

Table 7.11: The Probability & Combinatorics & Information Theory Notation.

Machine Learning

$\mathbb{A}$	An action space.
$\mathbb{S}$	A state space.
$P(\mathbf{x}, \mathbf{y})$	A data-generating distribution function.
$U(\mathbf{x}, \mathbf{y})$	A uniformly distributed data generating function.
$\mathbb{F}$	The hypothesis search space.
$C(\theta, \mathbf{x}, \Pi)$	A cost function depending on learned parameters θ, selected decision matrix $\mathbf{x}$ or policy Π .
$P_{u(0,1)}(\mathcal{G})$ = $P_{u(0,1)}(\{(\mathbf{x},\mathbf{y})\})$	The set of graph samples with uniformly distributed coordinates between (0,1).
$\mathcal{X}$	The set of training problem distributions containing the training examples.

$\mathcal{V}$	The set of validation problem distributions containing the training examples.
$\mathcal{T}$	The set of testing problem distributions containing the training examples.
$\mathcal{B}$	The set of batches that is a subset of training, validation or test set examples from $\mathcal{X}$.
$\mathcal{P}^*$	The set of solution paths (node index sequences) that minimize or maximize a combinatorial problem.
α, β	The learning step-size parameters.
γ	A discount-rate parameter.
δ_t	The temporal-difference error at t (a random variable).
ϵ	The probability of taking a random action in an ϵ-greedy policy.
λ	A decay-rate factor.
$\mu(s)$	The on-policy distribution over states.
μ	The $\|S\|$-vector of the $\mu(s)$.

Table 7.12: The Machine Learning Notation.

Routing Specific Metrics

$\mathbf{D}$	A O-D square $\mathbb{R}^{n \times n}$ matrix.
$\mathbf{E}$	A ETA square $\mathbb{R}^{n \times n}$ matrix.

Table 7.13: The Routing Specific Notation.

Appendix

A.1 Publications

Generally, this thesis summarizes the work done in the following publications:

- [Hamzehi et al., 2019], "Combinatorial Reinforcement Learning of Linear Assignment Problems,"
- [Hamzehi et al., 2021b], "Distance-Based Neural Combinatorial Optimization for Context-based Route Planning,"
- [Hamzehi et al., 2021a], "Approximate Collaborative Fleet Routing with a Pointer Generation Neural Network Approach,"
- [Selmair et al., 2021], "Evaluation of Algorithm Performance for Simulated Square and Non-Square Logistic Assignment Problems," and
- the Master Thesis of David Gackstetter with the Title: "Anticipative Fleet Reallocation using Multi-Agent Deep Reinforcement Learning".

A summary of the results can be found in Chapter 7, "Conclusions & Future Research".

A.2 Stochastic Error Metrics

$$\mathrm{MAE}(\boldsymbol{s}_{x^*}, \boldsymbol{s}_x) = \frac{1}{N} \cdot \sum_{i=0}^{N-1} |\boldsymbol{s}_{x^*} - \boldsymbol{s}_x| \tag{A.1}$$

$$\mathrm{MSE}(\boldsymbol{s}_{x^*}, \boldsymbol{s}_x) = \frac{1}{N} \cdot \sum_{i=0}^{N-1} (\boldsymbol{s}_x - \boldsymbol{s}_{x^*})^2 \tag{A.2}$$

$$\mathrm{MAPE}[\%](\boldsymbol{s}_{x^*}, \boldsymbol{s}_x) = \frac{1}{N} \cdot \frac{|\boldsymbol{s}_x - \boldsymbol{s}_{x^*}|}{\boldsymbol{s}_{x^*}} \tag{A.3}$$

$$\mathrm{RMSE}(\boldsymbol{s}_{x^*}, \boldsymbol{s}_x) = \sqrt{\frac{1}{N} \cdot \sum_{i=0}^{N-1} (\boldsymbol{s}_x - \boldsymbol{s}_{x^*})^2} \tag{A.4}$$

$$\text{RMSPE}[\%](\boldsymbol{s}_{x^*}, \boldsymbol{s}_x) = \sqrt{\frac{1}{N} \cdot \sum_{i=0}^{N-1} (\boldsymbol{s}_x - \boldsymbol{s}_{x^*})^2} \tag{A.5}$$

$$\text{MEDE}[\%](\boldsymbol{s}_{x^*}, \boldsymbol{s}_x) = \tilde{\mu}(\boldsymbol{s}_x) - \tilde{\mu}(\boldsymbol{s}_{x^*}) \tag{A.6}$$

$$\text{MEDPE}[\%](\boldsymbol{s}_{x^*}, \boldsymbol{s}_x) = \frac{\tilde{\mu}(\boldsymbol{s}_x) - \tilde{\mu}(\boldsymbol{s}_{x^*})}{\max\left[\tilde{\mu}(\boldsymbol{s}_x) - \tilde{\mu}(\boldsymbol{s}_{x^*})\right]} \tag{A.7}$$

A.3 Symmetric and Asymmetric Origin Demand Distance Matrices

$$\boldsymbol{D}_S = \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} \\ d_{21} & d_{22} & d_{23} & d_{24} & d_{25} \\ d_{31} & d_{32} & d_{33} & d_{34} & d_{35} \\ d_{41} & d_{42} & d_{43} & d_{44} & d_{45} \\ d_{51} & d_{52} & d_{53} & d_{53} & d_{55} \end{bmatrix}$$

$$\boldsymbol{D}_A = \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} \\ d_{21} & d_{22} & d_{23} & d_{24} & d_{25} \\ d_{31} & d_{32} & d_{33} & d_{34} & d_{35} \\ d_{41} & d_{42} & d_{43} & d_{44} & d_{45} \\ d_{51} & d_{52} & d_{53} & d_{53} & d_{55} \end{bmatrix}$$

A.4 Combinatorial Complexity

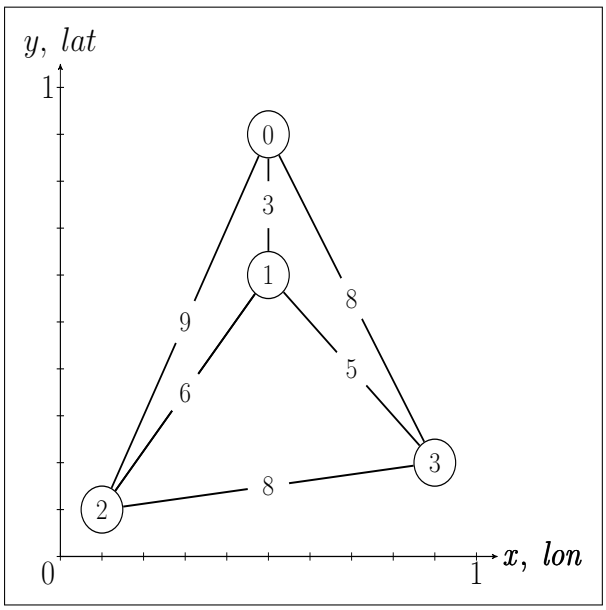
Many challenges that appear in modern ODMs originate from multi-objective decision making problems which are traditionally known as shortest-path and maximum flow or transport problems. Well-known examples here are euclidean-shortest path problems such as TSPs [Applegate et al., 2006b; Held and Karp, 1970; Shaw, 1998; Ropke and Pisinger, 2006] and knapsack [Pisinger, 1999], VRPs and assignment problems such as MWMs and dynamic (time-based) scheduling [Erdmann et al., 2019]. In detail, within mobility services TSPs can be classified as single vehicle shortest-path problems and the extended versions of multi-vehicle route optimization such as VRPs and MWMs. Traditionally, such problems can be grouped to the term combinatorial optimization, which are extensively researched within the OR domain and logistic management. Combinatorial optimization problems as such, typically are characterized by the presents of features such as Geo-locations, driving directions, and target route locations within a road network. The mathematical problem formulations have in common to be based on a graph representation for modeling road segments, junctions and the present vehicles which is unified by the term environment state in this thesis. The same problem structures also apply to robotics and automation logistics within a production plant, where the difference mostly are fixed and

less complex environment structures, however, with more strict time constraints for optimization. Generally, the domain of combinatorial planning problems has been proven to be solvable in polynomial computation time and therefore is called NP-complete/hard [Papadimitriou, 1977; Karp, 1975, pp. 237–244, 45–68]. As such, the problem combinatorial and feature complexity seems to be unsolvable for short computation times or rather termed non-scalable for large problem sizes as present in large-scale fleet management. This also applies to the VRP which is a problem where multiple TSPs must be solved to route each individual vehicle through a couple of locations and back to the depot. By extending the VRP with time windows, multiple drivers, resources and range capacities the problem can be characterized increasingly complex. Generally, the family of NP-hard problems also contains scheduling problems where time windows must be optimized in the correct combinatorial order. And additionally, there are other combinatorial problems such as called knapsack problems, where a combination sequence of different items of different sizes must be optimized to fit in a container of fixed size, which is again NP-hard [Pisinger, 2005, p. 1]. This means that depending of the number of input items, the time for calculating a solution takes polynomial time in the number of inputs.

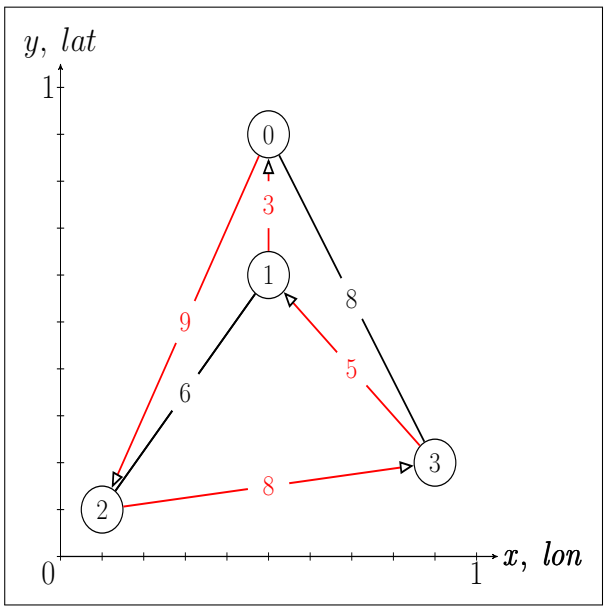
As a signature routing problem example, Figure 1a shows a *symmetric* TSP, where each indexed circle represents a location with xy-coordinates. The xy-coordinate values are sampled within an identity square $[0, 1]$. Further, the values that adhere at the graph edges or links are the cost of traversing the link individually. The node indexed with the number 0 is called the *depot node* which is equally the origin and terminal node of a valid route. By calculating the distances for all route possibilities, one can observe, that the shortest-path is the node sequence $\langle 02310 \rangle$ with a total distance (summed link distances) of $9 + 8 + 5 + 3 = 25$ as depicted in Figure 1b. The problem complexity however increases when more location nodes are added to the problem formulation. The example in Figure 1a contains six possible routes. However, for ten locations the number of routes increases to $(9 - 1)! = 362880$ without the depot node. For 20 locations, the number of possible routes increases to $(20 - 1)! = 1.2\text{e}^{17}$. Finding the solution therefore is computationally intractable and would take an enormous amount of for exhaustive search algorithms.

Second, Figures 2a shows the same graph, however this time as *directed* with *symmetric node degrees* and *symmetric adjacency* [Erdős and Rényi, 1963]. For the *directed* graph the distances for a node tuple such as node 0 to node 1 do not equal bidirectionally. This further increases the graph complexity for finding the solution as shortest-path node sequence $\langle 03120 \rangle$, that is depicted in Figure 2b. This shows that the global optimal solution might be different when considering different directions.

Note. In real routing problem scenarios, there is a distance or cost matrix that contains the distances or travel times for all travel connections. Usually, travel times, vehicle speeds and costs are functions of distances between all connections and therefore location tuples. However, for real application scenarios there is a need for realistic distance matrices for TSPs, VRPs and other variants [Flood, 1956; Clarke and Wright, 1964]. Thus, the issue of modeling asymmetry in distance and time matrices must be included as follows. From literature it is known that the solution of *symmetric* problems are not comparable of those for more realistic *asymmetric*

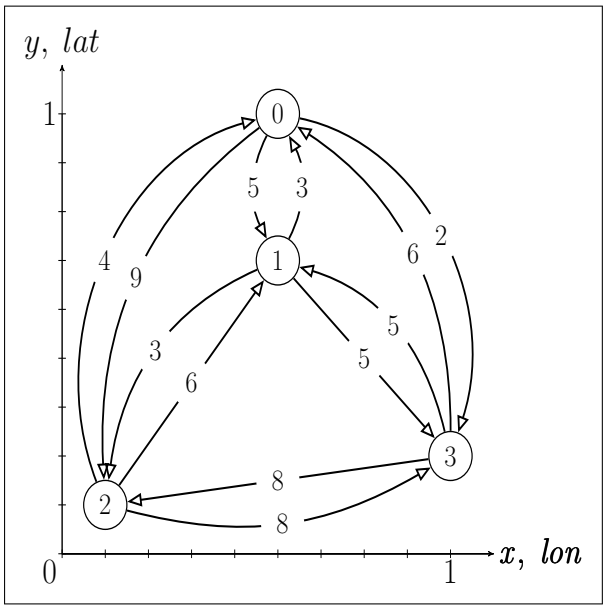


(a) Problem Undirected Symmetric TSP with Symmetric Adjacency.

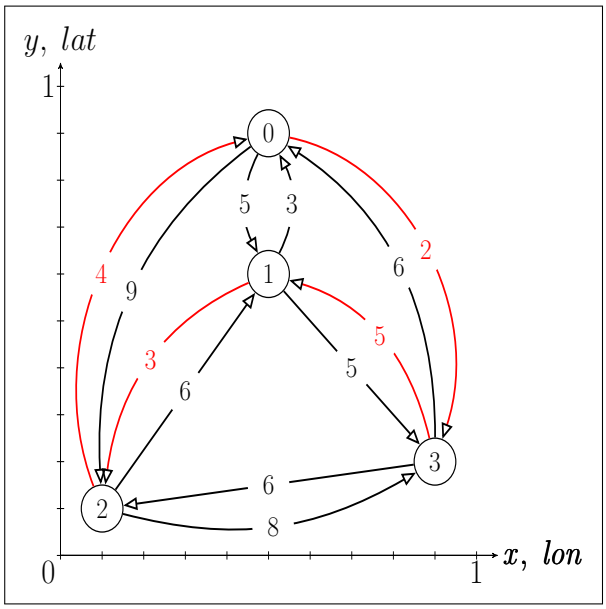


(b) Solution Undirected Symmetric TSP with Symmetric Adjacency.

Figure 1: Symmetric TSP Graphs.

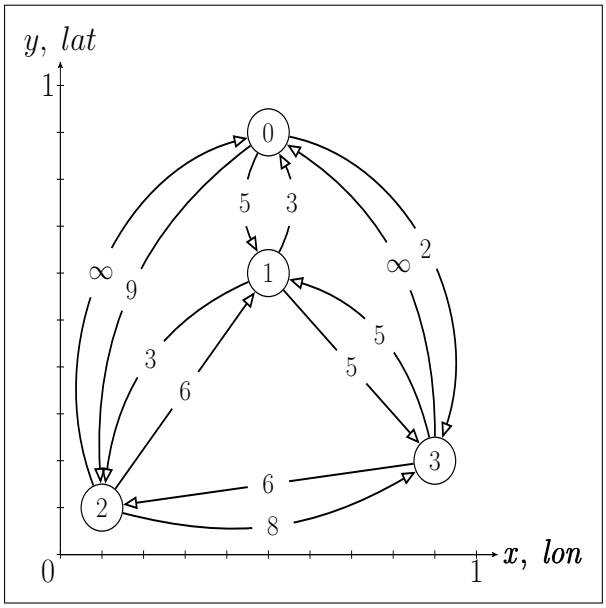


(a) Problem Directed Asymmetric TSP with Symmetric Adjacency.

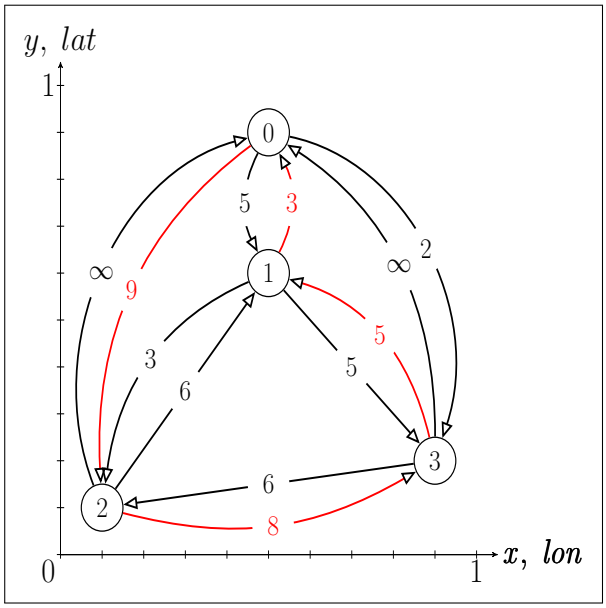


(b) Solution Directed Asymmetric TSP with Symmetric Adjacency.

Figure 2: Asymmetric TSP Graphs.



(a) A problem visualization of a TSP with asymmetric adjacency.



(b) A solution visualization of a TSP with asymmetric adjacency.

Figure 3: Symmetric/Undirected and Asymmetric/Directed Planar 2D Euclidean TSP Graphs.

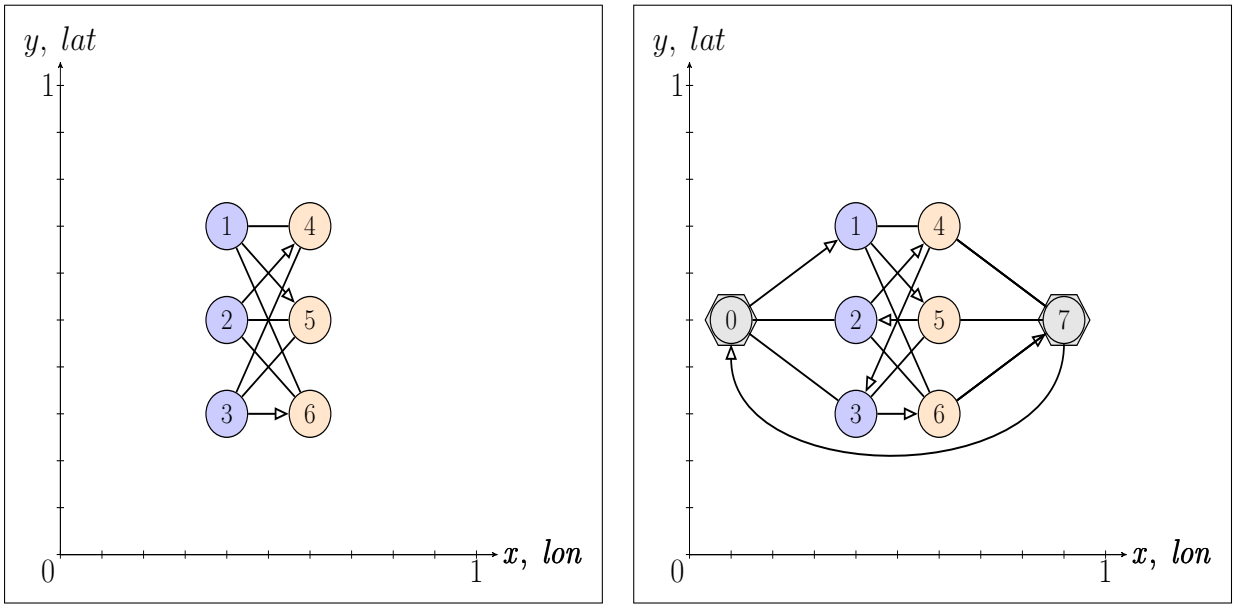
graph problems. Furthermore, it is known that methods that normally are used for calculating *symmetric* graph solutions usually deteriorate or even stop working when given an *asymmetric* problem statement [Rodríguez and Ruiz, 2010].

Third, Figures 3a shows the graph as *directed asymmetric* graph with *asymmetric adjacency* [Erdős and Rényi, 1963]. Here, some *directions* are missing or infeasible which is denoted by ∞ costs adhering to the links.

Figures 4a shows a symmetric bipartite graph with set of three vehicle nodes $\mathscr{V} = \{v_0, ..., v_{i-1}\}$, colored in blue and with denoted by the index $[1, 2, 3]$. Besides the vehicle nodes Figure 4a depicts a set of three request nodes $\mathscr{R} = \{r_0, ..., r_{j-1}\}$, colored in orange with index $[4, 5, 6]$. The solution of the *maximum cardinality matching problem* where the solution is the tuple sequence $(1, 5), (2, 4), (3, 6)$. Furthermore, Figure 4b shows how the symmetric *maximum cardinality matching problem* with the bipartite graph in Figure 4a can be converted to a TSP shortest-path tour. The target here is to find the shortest weighted route that contains all locations visits. In other words, the longest sequence (with *maximum cardinality*) that minimizes the route cost is desired. The solution sequence further can be modeled by incorporating a source and sink node tuple $(0, 7)$, colored in gray hexagons, where the solution sequence is $\langle 015243670 \rangle$.

Generally, the matching or TSP COP becomes more complex if vehicle and request node imbalances can occur as visualized in Figures 5a and 5b. This increased variability of appearing graph structures therefore requires for constraint modeling. For instance, Figure 5a shows an matching graph with *imbalanced* node distribution. Furthermore, the graph in Figure 5a shows an graph with imbalanced vehicle and request nodes.

In order to find the *maximal set* that contains the *maximum amount* of *valid* edges $\mathcal{E}$ in the



(a) Vehicle-Request Matching.

(b) Max Flow and Optimal Transport.

Figure 4: Converting a vehicle-request graph to a TSP tour. In general, the MWM can be illustrated as special problem of the max flow problem family.

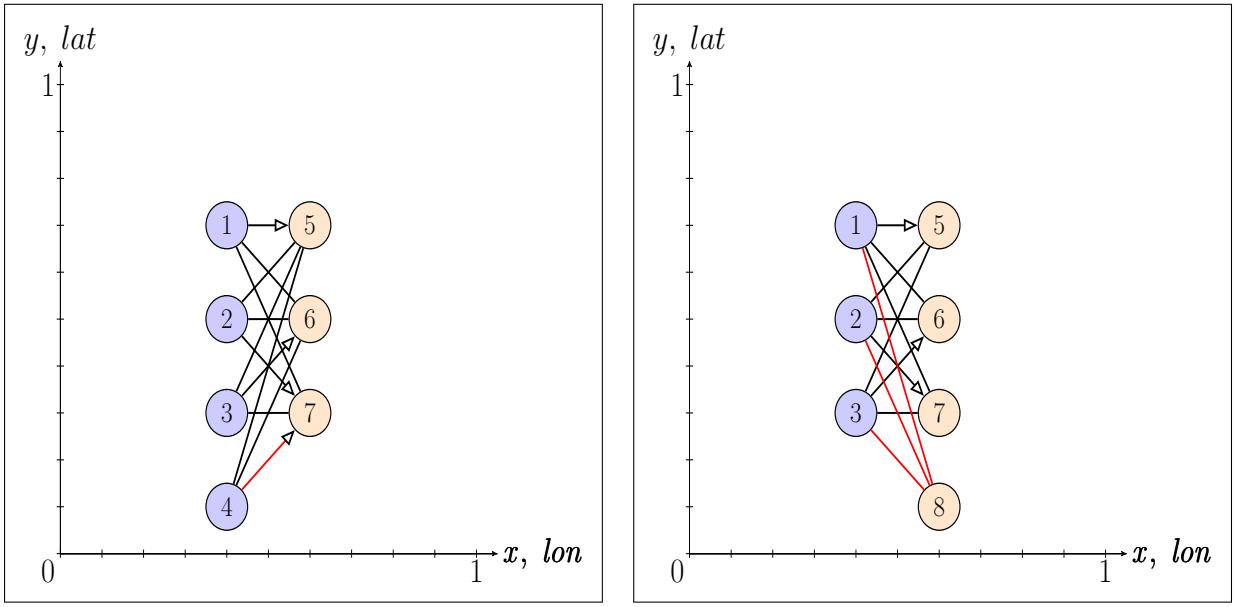
matching or assignment set $\mathcal{M}$ denoted by $\mathcal{M} \subseteq \mathcal{E}$, it is required to model some additional constraints to prevent duplicate matchings/assignments. This is called *valid* and *perfect cardinality* matching. If there are additional edge weight, the graph problem becomes a *maximum* or *minimum cardinality weighted matching* problem, where the task is to find the maximum amount of edges with *least (minimal) costs* or *highest (maximal) costs*.

For instance, there is a matching conflict for the nodes with indices 4 and 7 in Figure 5a and for the nodes with indices 1, 2, 3, and 8 in Figure 5b.

Generally, Figure 6 shows simple way of solving a *weighted cardinality matching* problem with a *Greedy* heuristic approach. The Greedy heuristic always matches the nearest neighboring node as shown in Figure 6. Obviously, this approach does not guarantee to find the global optimum and therefore it does not find the *maximum and perfect* matching. The reason is that the greedy solution of some special instances for *balanced* graphs, result in duplicate matchings when assigning the nearest neighbors, as shown in Figure 6. If a resolving constraint is modeled that solves this conflict, the result is still not perfect since one node has to take a suboptimal assignment. Generally, this constraint modeling problem is depending on the individual edge cost distribution.

One of the rawest forms of the euclidean-planar TSP within unit square (i.e. x, y-coordinates are within the value range of $[0, 1]^2$ and symmetric edges can be seen in Figure 1a. Here, the goal is to find the minimal-weighted that is the shortest route or path with least cost, called Hamiltonian Cycle which visits every location once, e.g. see for the visualization in Figure 1b.

Definition 26 (*Hamiltonian Cycle (Round Trip)*)**:** A Hamiltonian Cycle is a cycle route in an undirected graph that passes though each node exactly once [Plotkin, 2010, p. 9].



(a) Demand supply imbalance $|\mathcal{V}| > |\mathcal{R}|$.

(b) Demand supply imbalance $|\mathcal{R}| > |\mathcal{V}|$.

Figure 5: Demand and supply occurrence imbalances with vehicle-request matching.

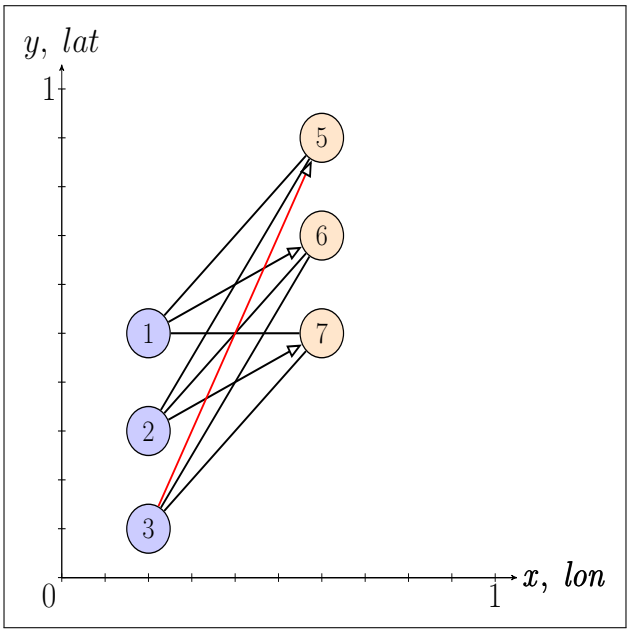


Figure 6: Sub-optimality of Greedy Heuristic Assignments.

Definition 27 (*Symmetric Traveling Salesman Problem*)**:** Given an undirected complete, weighted, euclidean-planar and symmetric graph, the TSP is the problem of finding a minimum-weighted or minimum-cost Hamiltonian Cycle [Plotkin, 2010, p. 9].

A.5 Symmetric Euclidean-Planar TSP

Figure 1a further shows an example TSP, where each indexed circle represents a location with uniformly distributed xy-coordinates within unit identity square $[0, 1]$. Further, the values that adhere at the graph edges or links are the cost of traversing the link individually. The node indexed with the number 0 is called the *depot node* which is equally the origin and terminal node of a valid route.

Given a sample graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ with a set of nodes and edges, term *symmetric* thereby refers to the definition of the triangle inequality.

Definition 28 (*Triangle Inequality (Route Distance Symmetry)*)**:** The triangle inequality states that for any triplet of nodes i, j and k the edge weights or costs between satisfy

$$w_{i,j} \leq w_{i,k} + w_{k,j} \quad \forall i, j, k \in \mathcal{N} \,. \tag{A.8}$$

Hereby, a weighted edge refers to the cost of traveling between pair of nodes. In other words, equation (A.8) defines that there is no faster detour route than selecting the direct way from i to j. Generally, one cannot expect to find good approximation and solution algorithms for general TSP graphs which is stated by the well-known theorem as follows:

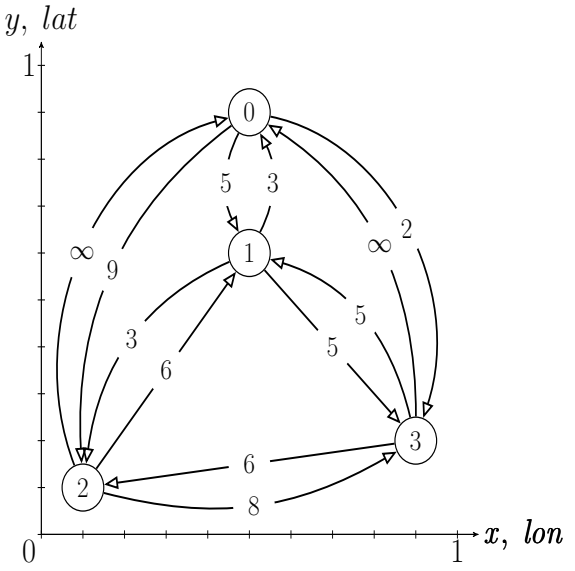
Theorem 1. *No polynomial time algorithm exists that approximates and solves a TSP within a constant factor unless $P = NP$[Plotkin, 2010, p. 10].*

In reality when dealing with real road networks, the triangle inequality is generally violated when facing one-way streets and other directional limitations. Thus, the TSP problem formulation can become even more complex as visualized in Figures 7a and 7b.

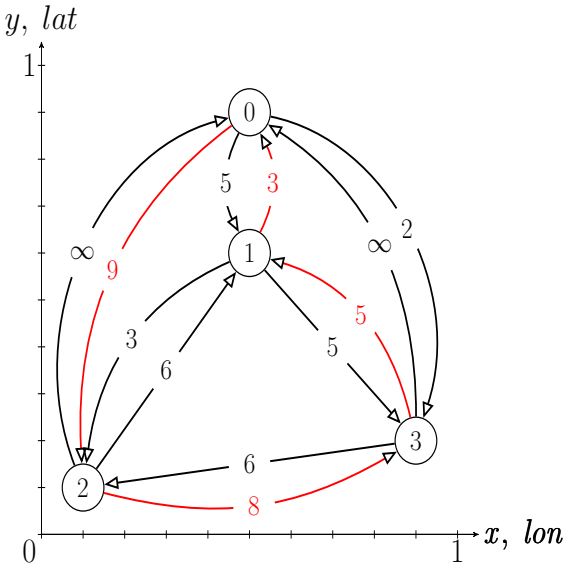
A.6 MWBM and Pickup Drop-Off / Delivery Problems (PDP)

Alongside the TSP there is another combinatorial optimization problem called VRP where multiple TSP are solved dependently. Initially appeared in a publication by [Dantzig and Ramser, 1959, pp. 80–91] the goal is to find the optimal set of routes for a fleet of vehicles which traverse a graph in order to visit or deliver a given set of different customers, as visualized in Figure 8b. Again, since finding the solution to one TSP already is NP-hard/complete, finding multiple-optimal solutions for the VRP is again NP-hard/complete [Toth and Vigo, 2002, pp. 1–26].

Both TSP and VRP can be unified as shortest path problems, which in general can be seen as finding the minimal weighted tree among a network. Thus the formulation of both can be abstracted to a so-called optimal transport or maximum cardinality/closure optimization which is characterized by the max-flow min-cut theorem. In the further sections this will be called MWM problem, which will be introduced next.

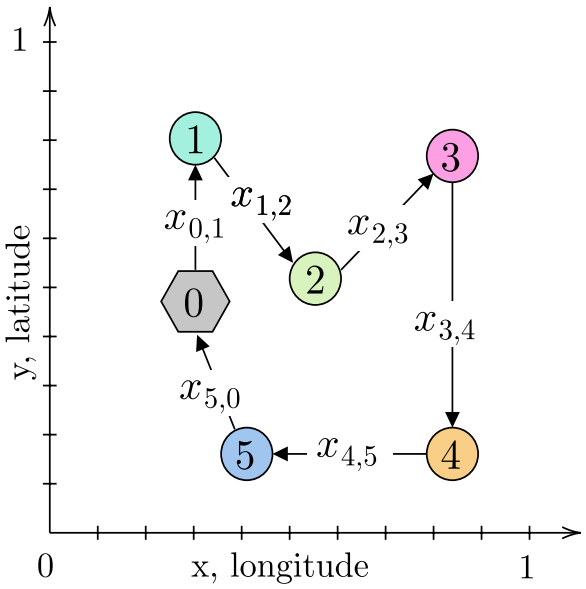


(a) A problem visualization of a TSP with asymmetric adjacency.

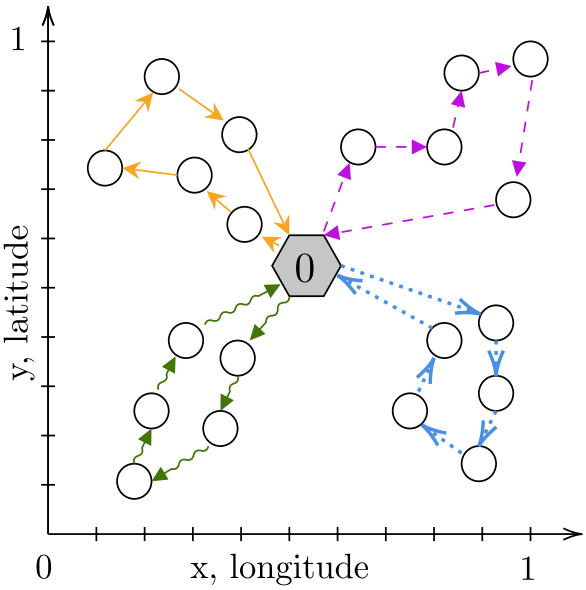


(b) A solution visualization of a TSP with asymmetric adjacency.

Figure 7: Symmetric undirected and asymmetric directed planar 2D Euclidean TSP Graphs.



(a) A general visualization of a TSP with depot node.

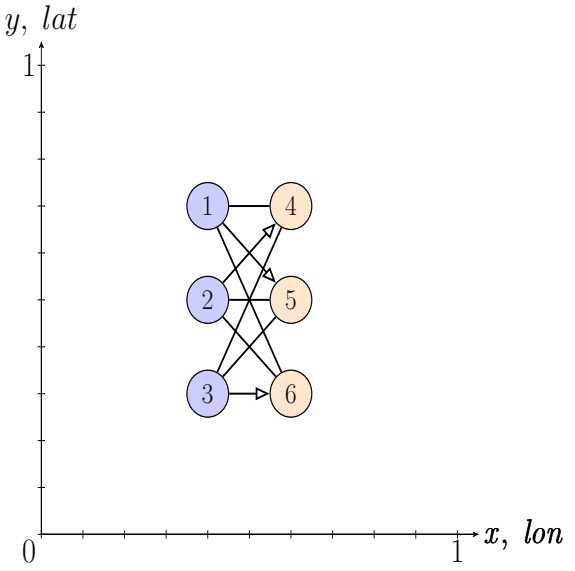


(b) A general visualization of a VRP with depot node.

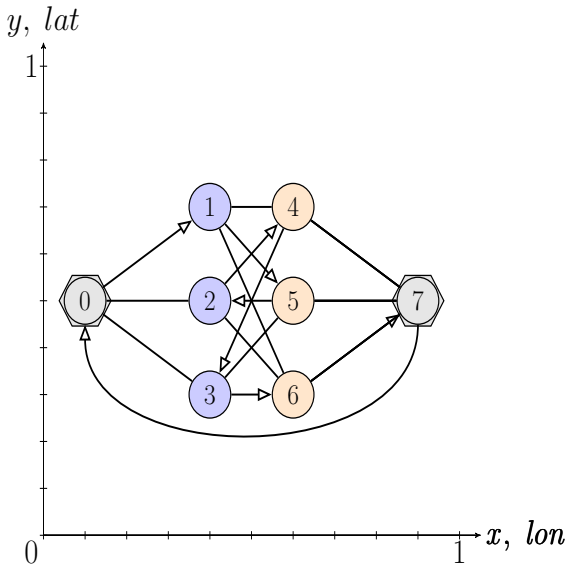
Figure 8: Extension visualization from TSP graphs to VRP graphs.

A.7 Maximum Weighted Matching

Another more general formulation of TSPs and VRPs is the MWM Problem, which involves finding the maximum and feasible flow through a network. Based on the TSP formulation, the formulation can be transferred to the MWM formulation by introducing a sink (0) and source node (7) to the overall node distribution, visualized in Figures 9a and 9b. Since the



(a) Vehicle-request matching problem with sub-optimal solution.



(b) Max Flow and Optimal Transport.

Figure 9: Converting a vehicle-request graph to a TSP tour.

MWM formulation can be defined in a linear programming scheme, it is possible to apply linear programming solvers to solve TSPs, VRPs and MWMs. However, the exponentially increasing combinatorial complexity during combinatorial optimization does not allow for optimal scalability, especially when solving many large-scale graph instances with many nodes and edges in repetitive manner. In more detail, this characteristic of combinatorial optimization problems will be addressed in the next paragraph.

A.8 Supplementary Figures

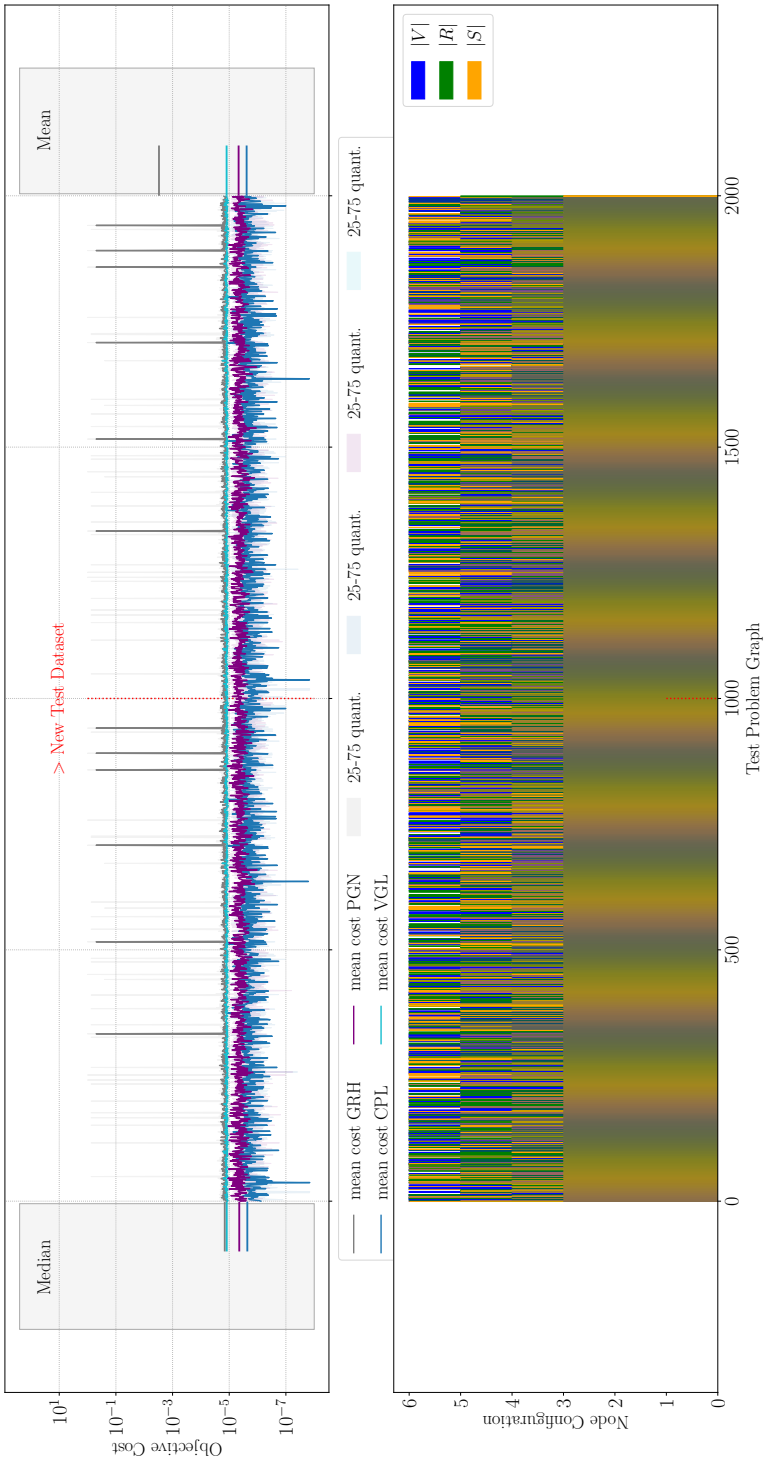


Figure 10: Benchmarked algorithms the resulting solution errors, Overview of Figure 10 (higher mean objective cost means worse solution quality) Part 1.

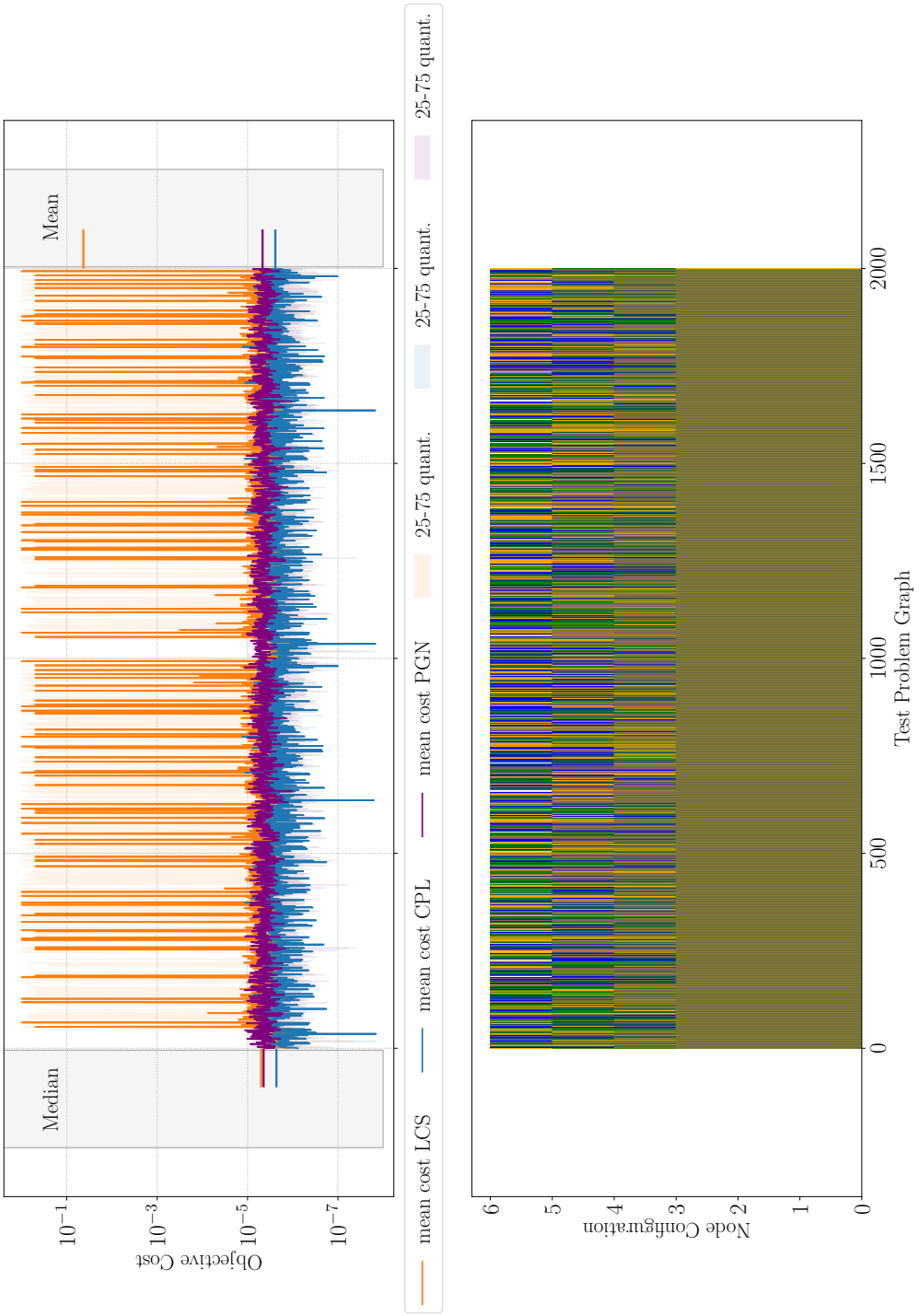


Figure 11: Benchmarked algorithms the resulting solution errors (higher mean objective cost means worse solution quality) Part 2.

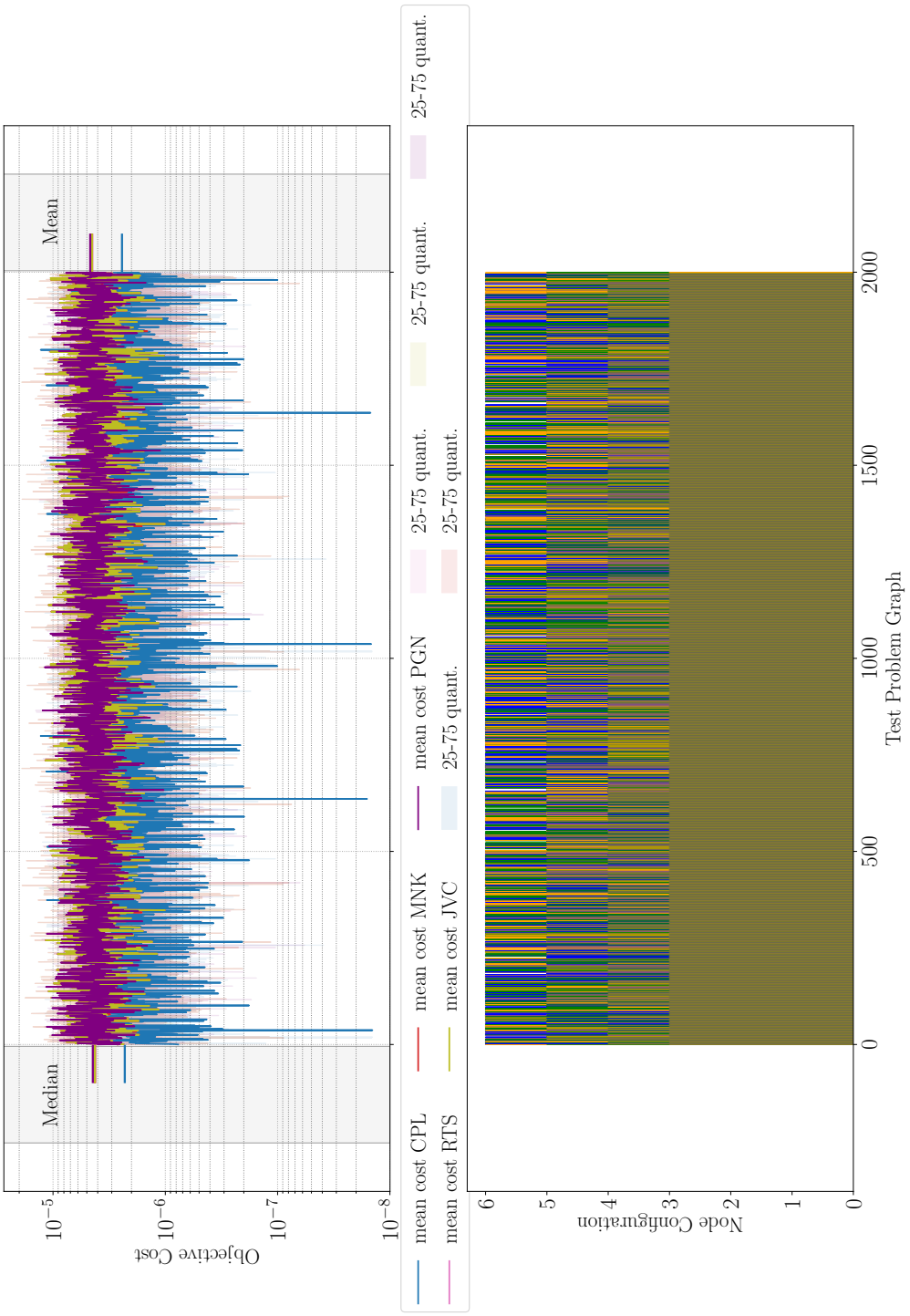


Figure 12: Benchmarked algorithms the resulting solution errors (higher mean objective cost means worse solution quality) Part 3.

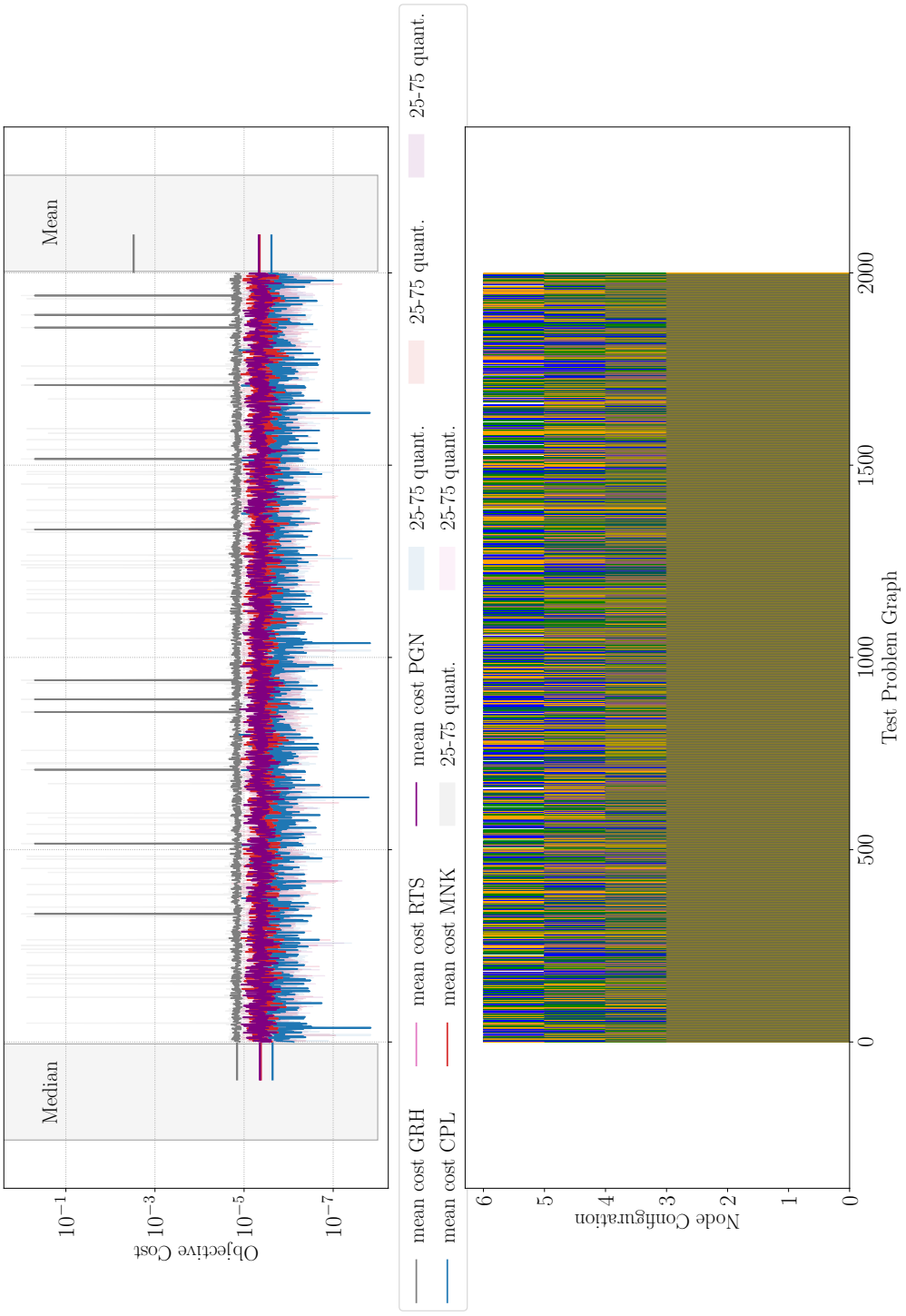


Figure 13: Benchmarked algorithms the resulting solution errors (higher mean objective cost means worse solution quality) Part 4.

List of Figures

List of Tables

Bibliography

Abadi, Martín et al. (2016). "Tensorflow: A system for large-scale machine learning." In: *12th USENIX Symposium on operating systems design and implementation)*, pp. 265–283.

Aljaafreh, Ahmad et al. (2011). "Vehicular data acquisition system for fleet management automation." In: *Proceedings of 2011 IEEE International Conference on Vehicular Electronics and Safety*. IEEE, pp. 130–133.

Alonso-Mora, Javier et al. (2017). "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment." In: *Proceedings of the National Academy of Sciences* 114.3, pp. 462–467.

Andersson, Arne et al. (1998). "Sorting in linear time?" In: *Journal of Computer and System Sciences* 57.1, pp. 74–93.

Applegate, David et al. (2006a). *Concorde TSP solver*.

Applegate, David L et al. (2006b). *The traveling salesman problem: a computational study*. Princeton university press.

Bahdanau, Dzmitry; Kyunghyun Cho; Yoshua Bengio (2014). "Neural machine translation by jointly learning to align and translate." In: *arXiv preprint arXiv:1409.0473*.

Bahdanau, Dzmitry et al. (2016). "An actor-critic algorithm for sequence prediction." In: *arXiv preprint arXiv:1607.07086*.

Balinski, Michel L (1967). "Labelling to obtain a maximum matching." In: *Combinatorial Mathematics and Its Applications (Proceedings Conference Chapel Hill, North Carolina*, pp. 585–602.

Belkin, Mikhail et al. (2019). "Reconciling modern machine-learning practice and the classical bias–variance trade-off." In: *Proceedings of the National Academy of Sciences* 116.32, pp. 15849–15854.

Bellman, Richard (1957). "A Markovian decision process." In: *Journal of mathematics and mechanics*, pp. 679–684.

Bellman, Richard (1966). "Dynamic programming." In: *Science* 153.3731, pp. 34–37.

Bellman, Richard E (2015). *Adaptive control processes: a guided tour*. Vol. 2045. Princeton university press.

Bello, Irwan et al. (2016). "Neural combinatorial optimization with reinforcement learning." In: *arXiv preprint arXiv:1611.09940*.

Bengio, Yoshua et al. (2015). "Towards biologically plausible deep learning." In: *arXiv preprint arXiv:1502.04156*.

Bilali, Aledia et al. (2019a). "An Analytical Model for On-Demand Ride Sharing to Evaluate the Impact of Reservation, Detour and Maximum Waiting Time." In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, pp. 1715–1720.

Bilali, Aledia et al. (2019b). "Analyzing the Impact of Anticipatory Vehicle Routing on the Network Performance." In: *Transportation Research Procedia* 41, pp. 494–506.

Bilali, Aledia et al. (2019c). "Impact of service quality factors on ride sharing in urban areas." In: *2019 6th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*. IEEE, pp. 1–8.

Bilali, Aledia et al. (2020). "Analytical and Agent-Based Model to Evaluate Ride-Pooling Impact Factors." In: *Transportation Research Record*, p. 0361198120917666.

Bishop, Christopher M (2006). *Pattern recognition and machine learning*. springer.

Bishop, Christopher M et al. (1995). *Neural networks for pattern recognition*. Oxford university press.

Boyd, Stephen; Stephen P Boyd; Lieven Vandenberghe (2004). *Convex optimization*. Cambridge university press.

Brown, George W; John Von Neumann (1950). *Solutions of games by differential equations*. Tech. rep. Rand Corp Santa Monica, CA.

Campbell, Murray; A Joseph Hoane Jr; Feng-hsiung Hsu (2002). "Deep blue." In: *Artificial intelligence* 134.1-2, pp. 57–83.

Caruana, Rich; Steve Lawrence; C Lee Giles (2001). "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping." In: *Advances in neural information processing systems*, pp. 402–408.

Chen, Zhao et al. (2018). "Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks." In: *International Conference on Machine Learning*, pp. 794–803.

Cheriyan, Joseph; Torben Hagerup; Kurt Mehlhorn (1996). "An o(n^3)-Time Maximum-Flow Algorithm." In: *SIAM Journal on Computing* 25.6, pp. 1144–1170.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." In: *CoRR* abs/1406.1078. arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

Christofides, Nicos (1976). *Worst-case analysis of a new heuristic for the travelling salesman problem*. Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group.

Chung, Junyoung et al. (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling." In: *arXiv preprint arXiv:1412.3555*.

Chuxing, Didi (2019). *Didi Chuxing Machine Learning Mobility Blog, Neural Network Application to Demand Driven Mobility Systems*. URL: https://www.futurecar.com/3676/Chinas-Didi-Chuxing-Teams-Up-with-NVIDIA-for-Autonomous-Driving-and-Cloud-Computing (visited on 02/02/2020).

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

Clevert, Djork-Arné; Thomas Unterthiner; Sepp Hochreiter (2015). "Fast and accurate deep network learning by exponential linear units (elus)." In: *arXiv preprint arXiv:1511.07289*.

Cook, William; Andre Rohe (1999). "Computing minimum-weight perfect matchings." In: *INFORMS journal on computing* 11.2, pp. 138–148.

Cunningham, William H; AB Marsh (1978). "A primal algorithm for optimum matching." In: *Polyhedral Combinatorics*. Springer, pp. 50–72.

Cuturi, Marco (2013). "Sinkhorn distances: Lightspeed computation of optimal transport." In: *Advances in neural information processing systems*, pp. 2292–2300.

Cybenko, George (1989). "Approximation by superpositions of a sigmoidal function." In: *Mathematics of control, signals and systems* 2.4, pp. 303–314.

Cygan, Marek; Harold N Gabow; Piotr Sankowski (2015). "Algorithmic applications of baur-strassen's theorem: Shortest cycles, diameter, and matchings." In: *Journal of the ACM (JACM)* 62.4, pp. 1–30.

Dandl, Florian; Klaus Bogenberger (2018a). "Booking processes in autonomous carsharing and taxi systems." In: *Proceedings of 7th Transport Research Arena, Vienna*.

Dandl, Florian; Klaus Bogenberger (2018b). "Comparing future autonomous electric taxis with an existing free-floating carsharing system." In: *IEEE Transactions on Intelligent Transportation Systems* 20.6, pp. 2037–2047.

Dandl, Florian et al. (2019). "Evaluating the impact of spatio-temporal demand forecast aggregation on the operational performance of shared autonomous mobility fleets." In: *Transportation* 46.6, pp. 1975–1996.

Dandl, Florian et al. (2020). "Dual-Horizon Forecasts and Repositioning Strategies for Operating Shared Autonomous Mobility Fleets." In: *99th Annual Meeting of the Transportation Research Board (TRB 2020)*.

Dantzig, George B (1955). "Linear programming under uncertainty." In: *Management science* 1.3-4, pp. 197–206.

Dantzig, George B; John H Ramser (1959). "The truck dispatching problem." In: *Management science* 6.1, pp. 80–91.

Dantzig, George Bernard (1965). *Linear programming and extensions*. Vol. 48. Princeton university press.

De Myttenaere, Arnaud et al. (2016). "Mean absolute percentage error for regression models." In: *Neurocomputing* 192, pp. 38–48.

Demyanov, Sergey (2015). "Regularization methods for neural networks and related models." PhD thesis.

Desler, JF; S Louis Hakimi (1969). "A graph-theoretic approach to a class of integer-programming problems." In: *Operations Research* 17.6, pp. 1017–1033.

Deudon, Michel et al. (2018). "Learning heuristics for the tsp by policy gradient." In: *International conference on the integration of constraint programming, artificial intelligence, and operations research*. Springer, pp. 170–181.

Dey, Rahul; Fathi M Salemt (2017). "Gate-variants of gated recurrent unit (GRU) neural networks." In: *2017 IEEE 60th international midwest Symposium on circuits and systems (MWSCAS)*. IEEE, pp. 1597–1600.

Dijkstra, Edsger W et al. (1959). "A note on two problems in connexion with graphs." In: *Numerische mathematik* 1.1, pp. 269–271.

Dinic, EA; MA Kronrod (1969). "An algorithm for the solution of the assignment problem." In: *Soviet Math. Dokl.* Vol. 10. 6, pp. 1324–1326.

Dinic, Efim A (1970). "Algorithm for solution of a problem of maximum flow in networks with power estimation." In: *Soviet Math. Doklady*. Vol. 11, pp. 1277–1280.

Donahue, Jeffrey et al. (2015). "Long-term recurrent convolutional networks for visual recognition and description." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2625–2634.

Dorigo, Marco; Luca Maria Gambardella (1997). "Ant colonies for the travelling salesman problem." In: *biosystems* 43.2, pp. 73–81.

Duan, Ran; Seth Pettie (2014). "Linear-time approximation for maximum weight matching." In: *Journal of the ACM (JACM)* 61.1, pp. 1–23.

Duan, Ran; Seth Pettie; Hsin-Hao Su (2018). "Scaling algorithms for weighted matching in general graphs." In: *ACM Transactions on Algorithms (TALG)* 14.1, pp. 1–35.

Duchi, John; Elad Hazan; Yoram Singer (2011). "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7.

Eberhart, Russell; James Kennedy (1995). "A new optimizer using particle swarm theory." In: *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. Ieee, pp. 39–43.

Edmonds, Jack (1965). "Maximum matching and a polyhedron with 0, 1-vertices." In: *Journal of research of the National Bureau of Standards B* 69.125-130, pp. 55–56.

Edmonds, Jack; Richard M Karp (1972). "Theoretical improvements in algorithmic efficiency for network flow problems." In: *Journal of the ACM (JACM)* 19.2, pp. 248–264.

Emami, Patrick; Sanjay Ranka (2018). "Learning permutations with sinkhorn policy gradient." In: *arXiv preprint arXiv:1805.07010*.

Engelhardt, Roman; Florian Dandl; Klaus Bogenberger (2020). "Speed-up heuristic for an on-demand ride-pooling algorithm." In: *arXiv preprint arXiv:2007.14877*.

Engelhardt, Roman et al. (2019). "Quantifying the benefits of autonomous on-demand ride-pooling: A simulation study for munich, germany." In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, pp. 2992–2997.

Erdmann, Marvin; Florian Dandl; Klaus Bogenberger (2019). "Dynamic Car-Passenger Matching based on Tabu Search using Global Optimization with Time Windows." In: *2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*. IEEE, pp. 1–5.

Erdmann, Marvin et al. (2020). "Dynamic Car-Passenger Matching of Online and Reservation Requests." In: *99th Annual Meeting of the Transportation Research Board (TRB 2020)*.

Erdős, Paul; Alfréd Rényi (1963). "Asymmetric graphs." In: *Acta Mathematica Hungarica* 14.3-4, pp. 295–315.

Feder, Tomás; Rajeev Motwani (1995). "Clique partitions, graph compression and speeding-up algorithms." In: *Journal of Computer and System Sciences* 51.2, pp. 261–272.

Feng, Jiekun; Mark Gluzman; JG Dai (2020). "Scalable Deep Reinforcement Learning for Ride-Hailing." In: *arXiv preprint arXiv:2009.14679*.

Flood, Merrill M (1956). "The traveling-salesman problem." In: *Operations research* 4.1, pp. 61–75.

Ford Jr, Lester Randolph; Delbert Ray Fulkerson (1962). *Flows in networks*. Princeton university press.

François-Lavet, Vincent et al. (2016). "Deep reinforcement learning solutions for energy microgrids management." In: *European Workshop on Reinforcement Learning (EWRL 2016)*.

Fu, Zhang-Hua et al. (2019). "Targeted sampling of enlarged neighborhood via Monte Carlo tree search for TSP." In.

Gabow, Harold N (1976). "An efficient implementation of Edmonds' algorithm for maximum matching on graphs." In: *Journal of the ACM (JACM)* 23.2, pp. 221–234.

Gabow, Harold N (1990). "Data structures for weighted matching and nearest common ancestors with linking." In: *Proceedings of the first annual ACM-SIAM Symposium on Discrete algorithms*, pp. 434–443.

Gabow, Harold N; Robert E Tarjan (1991). "Faster scaling algorithms for general graph matching problems." In: *Journal of the ACM (JACM)* 38.4, pp. 815–853.

Gabow, Harold N; Robert Endre Tarjan (1985). "A linear-time algorithm for a special case of disjoint set union." In: *Journal of computer and system sciences* 30.2, pp. 209–221.

Gabow, Harold N et al. (1986). "Efficient algorithms for finding minimum spanning trees in undirected and directed graphs." In: *Combinatorica* 6.2, pp. 109–122.

Gal, Yarin; Zoubin Ghahramani (2016). "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." In: *international conference on machine learning*, pp. 1050–1059.

Galil, Zvi; Silvio Micali; Harold Gabow (1982). "Priority queues with variable priority and an O (EV log V) algorithm for finding a maximal weighted matching in general graphs." In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. IEEE, pp. 255–261.

Gamache, Michel; Renaud Grimard; Paul Cohen (2005). "A shortest-path algorithm for solving the fleet management problem in underground mines." In: *European journal of operational research* 166.2, pp. 497–506.

Geng, Xu et al. (2019). "Multi-modal graph interaction for multi-graph convolution network in urban spatiotemporal forecasting." In: *arXiv preprint arXiv:1905.11395*.

Github (2020a). *Github Pytorch Implementations*. https://github.com/zhengsr3/Reinforcement_Learning_Pointer_Networks_TSP_Pytorch and https://github.com/Rintarooo/TSP_DRL_PointerNet and https://github.com/pemami4911/neural-combinatorial-rl-pytorch and https://github.com/mveres01/pytorch-drl4vrp. (Visited on 08/10/2020).

Github (2020b). *Github Tensorflow Implementations*. https://github.com/devsisters/neural-combinatorial-rl-tensorflow/blob/master/README.md and https://github.com/aurelienbibaut/Actor_CriticPointer_Network-TSP and https://github.com/devsisters/neural-combinatorial-rl-tensorflow and https://github.com/chaitjo/learning-tsp and https://github.com/MichelDeudon/encode-attend-navigate and https://github.com/MichelDeudon/neural-combinatorial-optimization-rl-tensorflow. (Visited on 08/10/2020).

Gleyzal, A (1955). "An algorithm for solving the transportation problem." In: *Journal of Research of the National Bureau of Standards* 54.4, pp. 213–216.

Glorot, Xavier; Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, pp. 249–256.

Glorot, Xavier; Antoine Bordes; Yoshua Bengio (2011). "Deep sparse rectifier neural networks." In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323.

Glover, Fred (1989). "Tabu search—part I." In: *ORSA Journal on computing* 1.3, pp. 190–206.

Glover, Fred; Manuel Laguna; Rafael Martí (2018). "Principles and Strategies of Tabu Search." In: *Handbook of Approximation Algorithms and Metaheuristics: Methodologies and Traditional Applications* 1.

Goldberg, Andrew V; Alexander V Karzanov (2004). "Maximum skew-symmetric flows and matchings." In: *Mathematical Programming* 100.3, pp. 537–568.

Goldberg, Andrew V; Robert Kennedy (1997). "Global price updates help." In: *SIAM Journal on Discrete Mathematics* 10.4, pp. 551–572.

Goldberg, Yoav; Omer Levy (2014). "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method." In: *arXiv preprint arXiv:1402.3722*.

Google Deep Mind Lecture (2020). *Neural Network Fundamentals*. url: https://storage.googleapis.com/deepmind-media/UCLxDeepMind_2020/L2%20-%20UCLxDeepMind%20DL2020.pdf (visited on 08/10/2020).

Google OR-Tools, (2016). url: https://developers.google.com/optimization/routing.

Graves, Alex (2012). "Long short-term memory." In: *Supervised sequence labelling with recurrent neural networks*. Springer, pp. 37–45.

Graves, Alex (2013). "Generating sequences with recurrent neural networks." In: *arXiv preprint arXiv:1308.0850*.

Graves, Alex; Greg Wayne; Ivo Danihelka (2014). "Neural turing machines." In: *arXiv preprint arXiv:1410.5401*.

Gudmundsson, Joachim et al. (2007). "Small Manhattan networks and algorithmic applications for the earth mover's distance." In: *Proceedings of the 23rd European Workshop on Computational Geometry*, pp. 174–177.

Gurumurthy, Krishna Murthy; Kara M Kockelman; Michele D Simoni (2019). "Benefits and costs of ride-sharing in shared automated vehicles across austin, texas: Opportunities for congestion pricing." In: *Transportation Research Record* 2673.6, pp. 548–556.

Hamzehi, Sascha et al. (2019). "Combinatorial Reinforcement Learning of Linear Assignment Problems." In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, pp. 3314–3321.

Hamzehi, Sascha et al. (2021a). "Approximate Collaborative Fleet Routing with a Pointer Generation Neural Network Approach." In: *2021 International Federation of Automatic Control [Unpublished Paper]*. IFAC, pp. 1–8.

Hamzehi, Sascha et al. (2021b). "Distance-Based Neural Combinatorial Optimization for Context-based Route Planning." In: *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*. IEEE, pp. 1–7.

Han, Yijie (2002). "Deterministic sorting in O (n log log n) time and linear space." In: *Proceedings of the thiry-fourth annual ACM Symposium on Theory of computing*, pp. 602–608.

Hardt, Cornelius; Klaus Bogenberger (2016). "The Price of Shared Vehicles–On current and future Pricing Strategies in Mobility Sharing Systems." In: *Transp. Res. Board 95th Annu. Meet.*

Hardt, Cornelius; Klaus Bogenberger (2018). "Empirical Analysis of Demand Patterns and Availability in Free-Floating Carsharing Systems." In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, pp. 1186–1193.

Harvey, Nicholas JA (2006). "Algebraic structures and algorithms for matching and matroid problems." In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, pp. 531–542.

Hauschild, Anne-Christin et al. (2013). "Peak detection method evaluation for ion mobility spectrometry by using machine learning approaches." In: *Metabolites* 3.2, pp. 277–293.

He, Kaiming et al. (2016). "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Held, Michael; Richard M Karp (1970). "The traveling-salesman problem and minimum spanning trees." In: *Operations Research* 18.6, pp. 1138–1162.

Hochreiter, Sepp (1998). "The vanishing gradient problem during learning recurrent neural nets and problem solutions." In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02, pp. 107–116.

Hochreiter, Sepp; Jürgen Schmidhuber (1997). "Long short-term memory." In: *Neural computation* 9.8, pp. 1735–1780.

Hoffman, AJ; HM Markowitz (1963). "A note on shortest path, assignment, and transportation problems." In: *Naval Research Logistics Quarterly* 10.1, pp. 375–379.

Hopcroft, John E; Richard M Karp (1973). "An n^5/2 algorithm for maximum matchings in bipartite graphs." In: *SIAM Journal on computing* 2.4, pp. 225–231.

Hopfield, John J; David W Tank (1985). "Neural computation of decisions in optimization problems." In: *Biological cybernetics* 52.3, pp. 141–152.

Horn, Mark ET (2002). "Fleet scheduling and dispatching for demand-responsive passenger services." In: *Transportation Research Part C: Emerging Technologies* 10.1, pp. 35–63.

Hornik, Kurt (1991). "Approximation capabilities of multilayer feedforward networks." In: *Neural networks* 4.2, pp. 251–257.

Hornik, Kurt; Maxwell Stinchcombe; Halbert White, et al. (1989). "Multilayer feedforward networks are universal approximators." In: *Neural networks* 2.5, pp. 359–366.

Hottung, André; Kevin Tierney (2019). "Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem." In: *arXiv preprint arXiv:1911.09539*.

Ioffe, Sergey; Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *arXiv preprint arXiv:1502.03167*.

Jaillet, Patrick; Jin Qi; Melvyn Sim (2016). "Routing optimization under uncertainty." In: *Operations research* 64.1, pp. 186–200.

Jain, Anil K; Jianchang Mao; K Moidin Mohiuddin (1996). "Artificial neural networks: A tutorial." In: *Computer* 29.3, pp. 31–44.

James, JQ; Albert YS Lam (2017). "Autonomous vehicle logistic system: Joint routing and charging strategy." In: *IEEE Transactions on Intelligent Transportation Systems* 19.7, pp. 2175–2187.

Johnson, David S (1990). "Local optimization and the traveling salesman problem." In: *International colloquium on automata, languages, and programming*. Springer, pp. 446–461.

Jonker, Roy; Anton Volgenant (1987). "A shortest augmenting path algorithm for dense and sparse linear assignment problems." In: *Computing* 38.4, pp. 325–340.

Joshi, Chaitanya K; Thomas Laurent; Xavier Bresson (2019). "An efficient graph convolutional network technique for the travelling salesman problem." In: *arXiv preprint arXiv:1906.01227*.

Judson, Lewis Van Hagen (1976). *Weights and measures standards of the United States: a brief history*. Vol. 447. Department of Commerce, National Bureau of Standards.

Kaempfer, Yoav; Lior Wolf (2018). "Learning the multiple traveling salesmen problem with permutation invariant pooling networks." In: *arXiv preprint arXiv:1803.09621*.

Kalchbrenner, Nal; Phil Blunsom (2013). "Recurrent continuous translation models." In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1700–1709.

Kanai, Sekitoshi; Yasuhiro Fujiwara; Sotetsu Iwamura (2017). "Preventing gradient explosions in gated recurrent units." In: *Advances in neural information processing systems*, pp. 435–444.

Kanellakis, Paris-C; Christos H Papadimitriou (1980). "Local search for the asymmetric traveling salesman problem." In: *Operations Research* 28.5, pp. 1086–1099.

Kao, Ming-Yang et al. (2001). "A decomposition theorem for maximum weight bipartite matchings." In: *SIAM Journal on Computing* 31.1, pp. 18–26.

Karmarkar, Narendra (1984). "A new polynomial-time algorithm for linear programming." In: *Proceedings of the sixteenth annual ACM Symposium on Theory of computing*, pp. 302–311.

Karp, Richard M (1975). "On the computational complexity of combinatorial problems." In: *Networks* 5.1, pp. 45–68.

Karzanov, Alexander V (1973). "An exact estimate of an algorithm for finding a maximum flow, applied to the problem "on representatives"." In: *Problems in Cybernetics* 5, pp. 66–70.

Karzanov, AV (1976). "Efficient implementations of Edmonds' algorithms for finding matchings with maximum cardinality and maximum weight." In: *Studies in Discrete Optimization*, pp. 306–327.

Ke, Jintao et al. (2017). "Short-term forecasting of passenger demand under on-demand ride services: A spatio-temporal deep learning approach." In: *Transportation Research Part C: Emerging Technologies* 85, pp. 591–608.

Khalil, Elias et al. (2017). "Learning combinatorial optimization algorithms over graphs." In: *Advances in Neural Information Processing Systems*, pp. 6348–6358.

Kingma, Diederik P; Jimmy Ba (2014). "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980*.

Kingma, Diederik P; Jimmy Lei Ba (2015). "Adam: A method for stochastic gradient descent." In: *ICLR: International Conference on Learning Representations*.

Kirkpatrick, Scott; C Daniel Gelatt; Mario P Vecchi (1983). "Optimization by simulated annealing." In: *science* 220.4598, pp. 671–680.

Klambauer, Günter et al. (2017). "Self-normalizing neural networks." In: *Advances in neural information processing systems*, pp. 971–980.

Klee, Victor; George J Minty (1972). "How good is the simplex algorithm." In: *Inequalities* 3.3, pp. 159–175.

Kool, Wouter; Herke van Hoof; Max Welling (2018). "Attention solves your TSP, approximately." In: *stat* 1050, p. 22.

Kuhn, Harold W (1955). "The Hungarian method for the assignment problem." In: *Naval research logistics quarterly* 2.1-2, pp. 83–97.

Kuhn, Harold W (1956). "Variants of the Hungarian method for assignment problems." In: *Naval Research Logistics Quarterly* 3.4, pp. 253–258.

Lawler, Eugene L (2001). *Combinatorial optimization: networks and matroids*. Courier Corporation.

LeCun, Yann; Yoshua Bengio, et al. (1995). "Convolutional networks for images, speech, and time series." In: *The handbook of brain theory and neural networks* 3361.10, p. 1995.

Lee, Chungmok; Kyungsik Lee; Sungsoo Park (2012). "Robust vehicle routing problem with deadlines and travel time/demand uncertainty." In: *Journal of the Operational Research Society* 63.9, pp. 1294–1306.

Levine, Sergey (2017). "Policy Gradients." In: *Lecture*, pp. 679–684. URL: http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_4_policy_gradient.pdf.

Liao, Ziqi (2003). "Real-time taxi dispatching using global positioning systems." In: *Communications of the ACM* 46.5, pp. 81–83.

Lin, Kaixiang et al. (2018). "Efficient large-scale fleet management via multi-agent deep reinforcement learning." In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1774–1783.

Lin, Shen; Brian W Kernighan (1973). "An effective heuristic algorithm for the traveling-salesman problem." In: *Operations research* 21.2, pp. 498–516.

Maas, Andrew L; Awni Y Hannun; Andrew Y Ng (2013). "Rectifier nonlinearities improve neural network acoustic models." In: *Proc. icml*. Vol. 30. 1, p. 3.

Mairal, Julien et al. (2014). "Convolutional Kernel Networks." In: *ArXiv* abs/1406.3332.

Malazgirt, Gorker Alp; Osman S Unsal; Adrian Cristal Kestelman (2019). "TauRieL: Targeting Traveling Salesman Problem with a deep reinforcement learning inspired architecture." In: *arXiv preprint arXiv:1905.05567*.

Marchisio, Alberto et al. (2018). "A Methodology for Automatic Selection of Activation Functions to Design Hybrid Deep Neural Networks." In: *CoRR* abs/1811.03980. arXiv: 1811.03980. URL: http://arxiv.org/abs/1811.03980.

Martin, Mario (2004). *Multi-Agent Systems*. URL: https://www.cs.upc.edu/~mmartin/Doct-part.pdf (visited on 08/10/2020).

Medress, Mark F. et al. (1977). "Speech understanding systems: Report of a steering committee." In: *Artificial Intelligence* 9.3, pp. 307–316.

Micali, Silvio; Vijay V Vazirani (1980). "An O (v| v| c| E|) algoithm for finding maximum matching in general graphs." In: *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*. IEEE, pp. 17–27.

Mikolov, Tomáš et al. (2010). "Recurrent neural network based language model." In: *Eleventh annual conference of the international speech communication association*.

Mikolov, Tomas et al. (2013). "Efficient estimation of word representations in vector space." In: *arXiv preprint arXiv:1301.3781*.

Miller, Jeff et al. (2015). *Earliest known uses of some of the words of mathematics*.

Misra, Diganta (Aug. 2019). "Mish: A Self Regularized Non-Monotonic Neural Activation Function." In: *arXiv preprint arXiv:1908.08681*.

Mitchell, Melanie (1998). *An introduction to genetic algorithms*. MIT press.

Mladenović, Nenad; Pierre Hansen (1997). "Variable neighborhood search." In: *Computers & operations research* 24.11, pp. 1097–1100.

Mnih, Volodymyr et al. (2013). "Playing atari with deep reinforcement learning." In: *arXiv preprint arXiv:1312.5602*.

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning." In: *Nature* 518.7540, p. 529.

Mucha, Marcin; Piotr Sankowski (2004). "Maximum matchings via Gaussian elimination." In: *45th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, pp. 248–255.

Munkres, James (1957). "Algorithms for the assignment and transportation problems." In: *Journal of the society for industrial and applied mathematics* 5.1, pp. 32–38.

Murphy, Kevin P. (2012). *Machine Learning A Probabilistic Perspective*. Vol. 66. Massachusetts Institute of Technology.

Nations, United (n.d.). *Report of the Secretary-General on the 2019 Climate Action Summit and the Way Forward in 2020*. URL: https://www.un.org/en/climatechange/assets/pdf/cas_report_11_dec.pdf (visited on 02/02/2020).

Nazari, Mohammadreza et al. (2018). "Reinforcement learning for solving the vehicle routing problem." In: *Advances in Neural Information Processing Systems*, pp. 9839–9849.

Nelder, John A; Roger Mead (1965). "A simplex method for function minimization." In: *The computer journal* 7.4, pp. 308–313.

Nesterov, Yurii; Arkadii Nemirovskii (1994). *Interior-point polynomial algorithms in convex programming*. SIAM.

Newell, David B; Eite Tiesinga (2019). "The international system of units (SI)." In: *NIST Special Publication* 330, pp. 1–138.

Nowak, Alex et al. (2017a). "A note on learning algorithms for quadratic assignment with graph neural networks." In: *Proceeding of the 34th International Conference on Machine Learning (ICML)*. Vol. 1050, p. 22.

Nowak, Alex et al. (2017b). "Revised Note on Learning Algorithms for Quadratic Assignment with Graph Neural Networks." In: *arXiv preprint arXiv:1706.07450*.

NYC Taxi and Limousine Commission (TLC) (2020). *(NYC Taxi Dataset.* URL: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page (visited on 08/10/2020).

Ollivier, François (2009). "Looking for the order of a system of arbitrary ordinary differential equations." In: *Applicable Algebra in Engineering, Communication and Computing* 20.1, pp. 7–32.

Orlin, James B (2013). "Max flows in O (nm) time, or better." In: *Proceedings of the forty-fifth annual ACM Symposium on Theory of computing*, pp. 765–774.

Padberg, Manfred; Giovanni Rinaldi (1988). "Branch-and-cut approach to a variant of the traveling salesman problem." In: *Journal of Guidance, Control, and Dynamics* 11.5, pp. 436–440.

Papadimitriou, Christos H. (1977). "The Euclidean traveling salesman problem is NP-complete." In.

Pascanu, Razvan; Tomas Mikolov; Yoshua Bengio (2013). "On the difficulty of training recurrent neural networks." In: *International conference on machine learning*, pp. 1310–1318.

Paszke, Adam et al. (2016). *Pytorch: An open source machine learning framework that accelerates the path from research prototyping to production deployment.* URL: https://pytorch.org/ (visited on 08/05/2020).

Pavone, Marco et al. (2020). "Online hypergraph matching with delays." In: *arXiv preprint arXiv:2009.12022*.

Pele, Ofir; Michael Werman (2009). "Fast and robust earth mover's distances." In: *2009 IEEE 12th International Conference on Computer Vision*. IEEE, pp. 460–467.

Pisinger, David (1999). "Linear Time Algorithms for Knapsack Problems with Bounded Weights." In: *Journal of Algorithms* 33.1, pp. 1 –14. ISSN: 0196-6774. DOI: https://doi.org/10.1006/jagm.1999.1034. URL: http://www.sciencedirect.com/science/article/pii/S0196677499910349.

Pisinger, David (2005). "Where are the hard knapsack problems?" In: *Computers & Operations Research* 32.9, pp. 2271–2284.

Plotkin, Serge (2010). *Optimization Paradigms.* URL: http://theory.stanford.edu/~trevisan/cs261/all-notes-2010.pdf (visited on 08/05/2020).

Potts, Chris N; Mikhail Y Kovalyov (2000). "Scheduling with batching: A review." In: *European journal of operational research* 120.2, pp. 228–249.

Powell, Warren B (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality.* Vol. 703. John Wiley & Sons.

Powell, Warren B; Patrick Jaillet; Amedeo Odoni (1995). "Stochastic and dynamic networks and routing." In: *Handbooks in operations research and management science* 8, pp. 141–295.

Powell, Warren B; Yosef Sheffi (1982). "The convergence of equilibrium algorithms with predetermined step sizes." In: *Transportation Science* 16.1, pp. 45–55.

Pytorch Doc. (2020a). *Pytorch Documentation.* URL: https://pytorch.org/docs/master/generated/torch.nn.GRU.html (visited on 08/10/2020).

Pytorch Doc. (2020b). *Pytorch Documentation.* URL: https://pytorch.org/docs/master/generated/torch.nn.utils.clip_grad_norm_.html (visited on 08/10/2020).

Pytorch Xavier Uniform Sampling (2021). *Xavier Uniform Distribution Initialization of Weights.* URL: https://pytorch.org/docs/stable/_modules/torch/nn/init.html#xavier_uniform_ (visited on 01/23/2021).

Ramshaw, Lyle; Robert E Tarjan (2012). "On minimum-cost assignments in unbalanced bipartite graphs." In: *HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2012-40R1.*

Reinelt, Gerhard (1994). *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag.

Rodríguez, Alejandro; Rubén Ruiz (2010). "The effect of asymmetry on traveling salesman problems." In.

Ropke, Stefan; David Pisinger (2006). "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows." In: *Transportation science* 40.4, pp. 455–472.

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

Rosenkrantz, Daniel J; Richard E Stearns; Philip M Lewis II (1977). "An analysis of several heuristics for the traveling salesman problem." In: *SIAM journal on computing* 6.3, pp. 563–581.

Rumelhart, David E; Geoffrey E Hinton; Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science.

Rumelhart, David E; Geoffrey E Hinton; Ronald J Williams (1986). "Learning representations by back-propagating errors." In: *nature* 323.6088, pp. 533–536.

Salimans, Tim et al. (2017). "Evolution strategies as a scalable alternative to reinforcement learning." In: *arXiv preprint arXiv:1703.03864*.

Schiffer, Maximilian; Sebastian Stütz; Grit Walther (2018). "Electric commercial vehicles in mid-haul logistics networks." In: *Behaviour of Lithium-Ion Batteries in Electric Vehicles*. Springer, pp. 153–173.

Schiffer, Maximilian; Grit Walther (2018a). "An adaptive large neighborhood search for the location-routing problem with intra-route facilities." In: *Transportation Science* 52.2, pp. 331–352.

Schiffer, Maximilian; Grit Walther (2018b). "Strategic planning of electric logistics fleet networks: A robust location-routing approach." In: *Omega* 80, pp. 31–42.

Schiffer, Maximilian et al. (2019). "Vehicle routing and location routing with intermediate stops: A review." In: *Transportation Science* 53.2, pp. 319–343.

Schiffer, Maximilian et al. (2021). "Integrated planning for electric commercial vehicle fleets: A case study for retail mid-haul logistics networks." In: *European Journal of Operational Research* 291.3, pp. 944–960.

Schmöller, Stefan et al. (2019). "Journal für Mobilität und Verkehr." In: *Neue Formen der Mobilität*.

Schrank, David; Bill Eisele; Tim Lomax (2012). "TTI's 2012 urban mobility report." In: *Texas A&M Transportation Institute. The Texas A&M University System* 4.

Schwarting, Wilko; Javier Alonso-Mora; Daniela Rus (2018). "Planning and decision-making for autonomous vehicles." In: *Annual Review of Control, Robotics, and Autonomous Systems*.

Selmair, Maximilian; Sascha Hamzehi; Klaus-Jürgen Meier (2021). "Evaluation of Algorithm Performance for Simulated Square and Non-Square Logistic Assignment Problems." In: Peer Review: Transportation Science, manuscript ID TS-2021-0083, pp. 1–8.

Selmair, Maximilian et al. (2019). "Solving Non-Quadratic Matrices In Assignment Problems With An Improved Version Of Vogel's Approximation Method." In: *ECMS*, pp. 261–266.

Shaw, Paul (1998). "Using constraint programming and local search methods to solve vehicle routing problems." In: *International conference on principles and practice of constraint programming*. Springer, pp. 417–431.

Shimomura, Masato; Yasuhiro Takashima (2016). "Application of monte-carlo tree search to travelingsalesman problem." In: *The 20th Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI)*, pp. 352–356.

Silver, David et al. (2016). "Mastering the game of Go with deep neural networks and tree search." In: *nature* 529.7587, p. 484.

Silver, David et al. (2017). "Mastering the game of go without human knowledge." In: *Nature* 550.7676, pp. 354–359. doi: https://doi.org/10.1038/nature24270.

Spielman, Daniel A; Shang-Hua Teng (2004). "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time." In: *Journal of the ACM (JACM)* 51.3, pp. 385–463.

Srivastava, Nitish; Elman Mansimov; Ruslan Salakhudinov (2015). "Unsupervised learning of video representations using lstms." In: *International conference on machine learning*, pp. 843–852.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting." In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Sumanshu Arora (2020). *(Understanding Pytorch 1 dimensional CNN (Conv1d) Shapes For Text Classification.* url: https://medium.com/@sumanshusamarora/understanding-pytorch-conv1d-shapes-for-text-classification-c1e1857f8533 (visited on 12/10/2020).

Sundermeyer, Martin; Ralf Schlüter; Hermann Ney (2012). "LSTM neural networks for language modeling." In: *Thirteenth annual conference of the international speech communication association.*

Sutskever, Ilya; Oriol Vinyals; Quoc V Le (2014). "Sequence to sequence learning with neural networks." In: *Advances in neural information processing systems*, pp. 3104–3112.

Sutton, Richard S; Andrew G Barto (1981). "Toward a modern theory of adaptive networks: expectation and prediction." In: *Psychological review* 88.2, p. 135.

Sutton, Richard S; Andrew G Barto (1998). *Reinforcement learning: An introduction.* Vol. 1. 1. MIT press Cambridge.

Sutton, Richard S; Andrew G Barto, et al. (1998). *Introduction to reinforcement learning.* Vol. 2. 4. MIT press Cambridge.

Sutton, Richard S et al. (2000). "Policy gradient methods for reinforcement learning with function approximation." In: pp. 1057–1063.

Syed, Arslan Ali et al. (2019a). "Asynchronous Adaptive Large Neighborhood Search Algorithm for Dynamic Matching Problem in Ride Hailing Services." In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, pp. 3006–3012.

Syed, Arslan Ali et al. (2019b). "Neural network based large neighborhood search algorithm for ride hailing services." In: *EPIA Conference on Artificial Intelligence*. Springer. Cham: Springer International Publishing, pp. 584–595. isbn: 978-3-030-30241-2.

Szegedy, Christian et al. (2017). "Inception-v4, inception-resnet and the impact of residual connections on learning." In: *Thirty-first AAAI conference on artificial intelligence.*

Tensorflow: A system for large-scale machine learning (2016). url: https://www.tensorflow.org/ (visited on 08/05/2020).

Thorup, Mikkel (2004). "Integer priority queues with decrease key in constant time and the single source shortest paths problem." In: *Journal of Computer and System Sciences* 69.3, pp. 330–353.

Tieleman, Tijmen; Geoffrey Hinton (2012). "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." In: *COURSERA: Neural networks for machine learning* 4.2, pp. 26–31.

Tomizawa, Nobuaki (1971). "On some techniques useful for solution of transportation network problems." In: *Networks* 1.2, pp. 173–194.

Toth, Paolo; Daniele Vigo (2002). *The vehicle routing problem*. SIAM.

Uber (2019). *Uber Machine Learning Mobility Blog, Neural Network Application to Demand Driven Mobility Systems*. URL: https://eng.uber.com/uber-ai-blog-2019 (visited on 02/02/2020).

Vapnik, Vladimir (2013). *The nature of statistical learning theory*. Springer science & business media.

Vaswani, Ashish et al. (2017). "Attention is all you need." In: *Advances in neural information processing systems*, pp. 5998–6008.

Vigerske, Stefan (2017). *COIN-OR*.

Villani, Cédric (2008). *Optimal transport: old and new*. Vol. 338. Springer Science & Business Media.

Vinyals, Oriol; Meire Fortunato; Navdeep Jaitly (2015). "Pointer networks." In: *Advances in Neural Information Processing Systems*, pp. 2692–2700.

Von Neumann, John; Arthur W Burks, et al. (1966). "Theory of self-reproducing automata." In: *IEEE Transactions on Neural Networks* 5.1, pp. 3–14.

Voudouris, Christos; Edward Tsang (1999). "Guided local search and its application to the traveling salesman problem." In: *European journal of operational research* 113.2, pp. 469–499.

Vujanović, Davor et al. (2012). "Evaluation of vehicle fleet maintenance management indicators by application of DEMATEL and ANP." In: *Expert Systems with Applications* 39.12, pp. 10552–10563.

Watkins, Christopher JCH; Peter Dayan (1992). "Q-learning." In: *Machine learning* 8.3-4, pp. 279–292.

Wayne, Kevin Daniel; Eva Tardos (1999). *Generalized maximum flow algorithms*. Cornell University.

Weikl, Simone; Klaus Bogenberger (2013). "Relocation strategies and algorithms for free-floating car sharing systems." In: *IEEE Intelligent Transportation Systems Magazine* 5.4, pp. 100–111.

Werbos, Paul J (1988). "Generalization of backpropagation with application to a recurrent gas market model." In: *Neural networks* 1.4, pp. 339–356.

Williams, Ronald J (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning." In: *Machine learning* 8.3-4, pp. 229–256.

Witzgall, Christoph; Charles T Zahn (1965). "Modification of Edmonds' maximum matching algorithm." In: *J. Res. Nat. Bur. Standards Sect. B*. Citeseer.

Wold, Svante; Kim Esbensen; Paul Geladi (1987). "Principal component analysis." In: *Chemometrics and intelligent laboratory systems* 2.1-3, pp. 37–52.

Yao, Huaxiu et al. (2018). "Deep multi-view spatial-temporal network for taxi demand prediction." In: *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ye, Jieping (2018). "Big Data at Didi Chuxing." In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 1341–1341.

Zahn, Charles T; Ralph Z Roskies (1972). "Fourier descriptors for plane closed curves." In: *IEEE Transactions on computers* 100.3, pp. 269–281.

Zaremba, Wojciech; Ilya Sutskever; Oriol Vinyals (2014). "Recurrent neural network regularization." In: *arXiv preprint arXiv:1409.2329*.

Zeiler, Matthew D (2012). "ADADELTA: an adaptive learning rate method." In: *arXiv preprint arXiv:1212.5701*.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die bei der promotionsführenden Einrichtung TUM School of Engineering and Design der TUM zur Promotionsprüfung vorgelegte Arbeit unter der Anleitung und Betreuung durch Univ.-Prof. Dr.-Ing. Klaus Bogenberger ohne sonstige Hilfe erstellt und bei der Abfassung nur die gemäß § 6 Ab. 6 und 7 Satz 2 angebotenen Hilfsmittel benutzt habe.

Ich habe keine Organisation eingeschaltet, die gegen Entgelt Betreuerinnen und Betreuer für die Anfertigung von Dissertationen sucht, oder die mir obliegenden Pflichten hinsichtlich der Prüfungsleistungen für mich ganz oder teilweise erledigt.

Ich habe die Dissertation in dieser oder ähnlicher Form in keinem anderen Prüfungsverfahren als Prüfungsleistung vorgelegt.

Die vollständige Dissertation wurde in der Schriftenreihe des Lehrstuhls für Verkehrstechnik veröffentlicht. Die promotionsführende Einrichtung TUM School of Engineering and Design hat der Veröffentlichung zugestimmt.

Ich habe den angestrebten Doktorgrad noch nicht erworben und bin nicht in einem früheren Promotionsverfahren für den angestrebten Doktorgrad endgültig gescheitert.

Die öffentlich zugängliche Promotionsordnung der TUM ist mir bekannt, insbesondere habe ich die Bedeutung von § 28 (Nichtigkeit der Promotion) und § 29 (Entzug des Doktorgrades) zur Kenntnis genommen. Ich bin mir der Konsequenzen einer falschen Eidesstattlichen Erklärung bewusst.

Ort, Datum, Unterschrift