Machine and deep learning models for vehicle routing problems: a literature review

by

B. J. Bijvoet

Literature Assignment

in partial fulfillment of the requirements for the degree of

Master of Science
in Mechanical Engineering

at the Department Maritime and Transport Technology of Faculty Mechanical, Maritime and Materials Engineering of Delft University of Technology

Student number:	4456300
MSc track:	Multi-Machine Engineering
Report number:	2021.MME.8591
Supervisors:	Dr. B. Atasoy
	C. Karademir (PhD)
Date:	December 24, 2021

Preface

This report has been written for the literature assignment of the Multi-Machine Engineering track of the Master of Science Mechanical Engineering at the Delft University of Technology. What began as an interest in combinatorial optimization problems eventually started a strong interest in deep learning. My interest in combinatorial optimization problems, which originated from a two-period course in the first year of the master's program, is what brought me to this assignment subject. During the assignment, I became increasingly enthusiastic about deep learning.

At the beginning of this assignment, I started reading some papers on the subject and pretty soon realized that this was going to be a challenging time. Due to little background in machine learning, I was required to spend a lot of time reading books and watching videos to understand the fundamentals of deep learning, reinforcement learning, and a combination of these (deep reinforcement learning). Some challenges that I encountered during the past fourteen weeks were how to deal with a large amount of available literature, how to concisely describe the relevant literature, and how to find a good balance between reviewing the literature while learning about the topic. I would like to thank my supervisors Dr. Bilge Atasoy and Çiğdem Karademir for helping me to get back on track from time to time.

Out of interest and in preparation for a thesis assignment later this academic year, I will shortly start with the Deep Learning Specialization course offered by DeepLearning.AI.

Bas Bijvoet
Delft, December 2021

Abstract

Vehicle routing problems (VRPs), a generalization of the traveling salesman problem, are very extensively studied combinatorial optimization problems for their practical application. Many solution methods, e.g., exact and heuristic algorithms, are proposed in the last few decades but require relatively much computation time due to the NP-hard nature of the VRP. Additionally, to build such algorithms, much expert knowledge is required. The recent developments in a subfield of machine learning, deep learning, make it possible to solve routing problems in a purely data-driven manner or assist heuristic methods. This requires less problem-specific knowledge and can outperform the traditional solution methods in terms of objective value and computation time. In this literature review, introductions for the vehicle routing problem and machine learning are given first. Then, an existing categorization for algorithmic machine learning structures is described. The main body of this report is the review of the recent machine and deep learning methods to solve static and dynamic routing problems. The reviewed literature is summarized in tables that indicate the main characteristics of each work. Moreover, the results of several machine and deep learning based solution methods for the static VRP are compared to each other. Lastly, the challenges and opportunities for future research of deep learning based solution methods for the vehicle routing problem are discussed. The main challenges of these methods are scalability, generalization, and adaptability. Therefore, future research could be focused on addressing these challenges to improve deep learning methods for practical routing applications.

List of abbreviations

A2C	advantage actor-critic
A3C	asynchronous advantage actor-critic
AC	actor-critic
ACER	actor-critic with experience replay
ADI	accumulated driver income
ADP	approximate dynamic programming
ALNS	adaptive large neighborhood search
AS	active search
AVI	approximate value iteration
BPTT	backpropagation through time
BS	beam search
C	costs
CNN	convolutional netural network
COP	combinatorial optimization problem
CS	computer science
CVRP	capacitated VRP
CVRPSD	CVRP with split deliveries
CVRPTW	CVRP with time windows
DD	dynamic and deterministic
DDQN	double deep Q-network
DEVPDPTW	dynamic EV pickup and delivery problem with TW
DNN	deep neural network
DoD	degree of dynamism
DP	dynamic programming
DQN	deep Q-network
DRL	deep reinforcement learning
DS	dynamic and stochastic
DSCCVRP	dynamic and stochastic customer CVRP
E2E	end-to-end
EAS	efficient active search
EV	electric vehicle
FF	feed-forward
GHS	greenhouse gas
GNN	graph neural network
GRU	gated recurrent unit
IL	imitation learning
JSSP	job shop scheduling problem
KM	Khun-Munkres algorithm
L2C	learning to configure
L2S	learning to search
LNS	large neighborhood search
LSH	local search heuristic
LSTM	long short-term memory
MARL	multi-agent RL
MC	Monte Carlo
MCVRP	CVRP with multiple vehicles
MCVRPTW	MCVRP with time windows
MDP	Markov decision process
MFRL	mean-field reinforcement learning
MIP	mixed-integer programming
ML	machine learning
MoD	mobility on demand
NN	neural network
NYC	New York city
OP	orienting problem
OR	operation research
ORR	order response rate
PCTSP	prize collecting TSP
PDP	pickup and delivery problem
PG	policy gradient
PN	pointer network
PPO	proximal policy optimization
QoS	quality of service
REINFORCE w/b	REINFORCE with baseline
ReLu	rectified linear unit
RL	reinforcement learning
RNN	recurrent neural network
SA	simulated annealing
SD	static and deterministic
SGD	stochastic gradient descent
SPCTSP	stochastic PCTSP
SS	static and stochastic
tanh	hyperbolic tangent
TD	temporal-difference
TW	time windows
TSP	traveling saleman problem
VRP	vehicle routing problem

Contents

Preface i

Abstract ii

List of abbreviations iii

1 Introduction 1
- 1.1 Background 1
- 1.2 Objectives 1
- 1.3 Related works 1
- 1.4 Outline 2

2 Vehicle routing problem introduction 2
- 2.1 Mathematical model for the MCVRP 2
- 2.2 Solution methods 3
  - 2.2.1 Exact methods 3
  - 2.2.2 Heuristic methods 4
    - 2.2.2.1 Construction heuristics 4
    - 2.2.2.2 Two-phase heuristics 4
    - 2.2.2.3 Improvement heuristics 4
  - 2.2.3 Metaheuristic methods 5
    - 2.2.3.1 Tabu search 5
    - 2.2.3.2 Simulated annealing 5
    - 2.2.3.3 Large neighbourhood search 5
    - 2.2.3.4 Genetic algorithms 6
- 2.3 VRP categorization 6

3 Machine learning introduction 7
- 3.1 Supervised learning 7
- 3.2 Reinforcement learning 8
- 3.3 Deep neural networks 10
  - 3.3.1 Perceptron architecture 10
  - 3.3.2 Deep neural network architecture 11
  - 3.3.3 Training DNNs 12
- 3.4 Attention mechanisms 13
  - 3.4.1 Recurrent neural networks 13
  - 3.4.2 Encoder-decoder architecture 15
  - 3.4.3 Attention model 16
  - 3.4.4 Pointer Network 16
  - 3.4.5 Transformer model 16
- 3.5 DRL algorithms 17
  - 3.5.1 Value-based methods 17
  - 3.5.2 Policy-based methods 17
  - 3.5.3 Actor-critic methods 18

4 Solving routing problems using machine learning 18
- 4.1 Categorization of algorithmic structures 18
- 4.2 Static routing 19
  - 4.2.1 End-to-end learning (constructive) 19
    - 4.2.1.1 Single-vehicle routing problem 20
    - 4.2.1.2 Multi-vehicle routing problem 22
  - 4.2.2 Learning to configure 23
  - 4.2.3 Learning to search (improvement) 23
- 4.3 Dynamic dispatching 25

4.3.1 Order matching 25
4.3.1.1 Ride-hailing 26
4.3.1.2 Ride-sharing 28
4.3.2 Fleet management 28
4.3.3 Integrated dispatching 29
4.4 Dynamic routing 29
4.4.1 End-to-end learning 30
4.4.2 Learning to configure 31
4.5 Overviews 31
4.5.1 Overview routing 31
4.5.2 Overview dispatching 33
4.5.3 Performance comparison static CVRP 34

5 Challenges and opportunities for future research 35
5.1 Scalability 35
5.2 Generalization 35
5.3 Adaptability 35
5.4 Training and testing time 36
5.5 Inconsistencies 36
5.6 Benchmark datasets and algorithms 36

6 Conclusion 37

References 38

Appendix A LSTM and GRU 46

1 Introduction

The vehicle routing problem (VRP) is one of the most studied problems in transportation and operations research, due to its practical applications. The VRP is a generalization of the traveling salesman problem (TSP), for which the objective is to find the shortest possible tour through a set of cities and return to the starting city. In the VRP, the objective is to find the optimal set of routes for a fleet of vehicles to serve a set of customers. The context of the VRP is often that of delivering goods, which are stored in a depot, to customers with a demand for these goods. The VRP was introduced by Dantzig and Ramser (1959), who tried to find the optimal routes for gasoline delivery trucks. Shortly afterward, Clarke and Wright (1964) proposed a greedy heuristic to solve the VRP, which was more effective than the approach of Dantzig and Ramser (1959). Thereafter, many other VRP variants and solution methods have been proposed. Since the VRP (and its variants) is an NP-hard combinatorial optimization problem (COP), solving VRPs with a large number of customers is challenging. However, with the surge of e-commerce and on-demand transportation, solving larger-sized VRPs in less time becomes increasingly important. Increasing transportation efficiency is important for several reasons. The first and foremost reason is that it is the fastest way to decrease transportation-related greenhouse gas (GHS) emissions. The GHS of road transport, which accounts for approximately 72% of total transportation-related GHS emissions, is still rising but must be reduced by 60% compared to 1990 levels before 2050 according to the European Green Deal (EEA, 2020). To achieve climate neutrality, cleaner and more efficient transportation is needed. Cleaner road transportation can be achieved by, e.g., a shift to electric vehicles (EVs), but will take some time. The faster way is to increase the transportation efficiency, which can be achieved by better VRP solution methods. Secondly, increased transportation efficiency will contribute to a higher user experience of transportation services. Thirdly, more efficient transportation benefits both consumers and businesses by reducing the costs associated with delivering goods or personal transportation services.

1.1 Background

Traditionally, solving VRPs was tied to the operations research (OR) community who proposed exact methods, heuristic methods, metaheuristic methods, and several modeling techniques, e.g., stochastic programming, robust optimization, and change-constrained programming. To assist these solution methods, there is a rich history of applied machine learning techniques to increase the solution quality. Among which, predicting uncertainties in stochastic VRP variants and configuring algorithms, such as hyper-parameter tuning or cluster customers, are the most popular (Bai et al., 2021; Soeffker et al., 2021). Since the recent breakthrough of deep reinforcement learning (DRL) in playing the Atari games (Mnih et al., 2015) and the game of Go (Silver et al., 2016; 2017), the computer science (CS) community has become more interested in solving COP problems in general, but also specifically in the VRP.

1.2 Objectives

The goal of this literature review is to determine to what extent deep learning can contribute to and improve over conventional VRP solution methods. To understand the applied methods, one must have a basic understanding of deep learning, reinforcement learning, and a combination of these (deep reinforcement learning). Therefore, an introduction to these machine learning concepts will be given first. Then, to identify the similarities and differences of applied methods, the literature is categorized based on existing taxonomies. The challenges and advantages of the reviewed deep learning solution methods will be addressed to indicate the potential of deep learning to solve VRPs and point out some opportunities for future research.

1.3 Related works

This review aims to provide an overview of the recently proposed deep learning methods to solve VRPs. Besides the many existing surveys for conventional VRP solution methods, the literature that reviews machine learning methods to solve the VRP is also increasing. The survey of

Mazyavkina et al. (2021) gives an overview of reinforcement learning (RL) frameworks as a sole tool for COPs, whereas Vesselinova et al. (2020) reviewed machine learning methods for COPs formulated on graphs specifically. Bai et al. (2021) reviewed hybrid solution methods that combine analytical with ML techniques for the VRP. Haydari and Yilmaz (2020) and Farazi et al. (2020) surveyed deep reinforcement learning (DRL) methods for transportation systems, where the former specifically focused on autonomous driving and traffic signal control. Hildebrandt et al. (2021) and Soeffker et al. (2021) focused on the stochastic and dynamic VRP, where the former reviewed RL methods and the latter prescriptive analytic methods. RL for ride-sharing systems and DRL methods for demand-driven services are surveyed by Qin et al. (2021) and Zong et al. (2021), respectively. Where most of the aforementioned reviews focused on a specific subfield of vehicle routing, the scope of this literature review is a little broader. This literature review aims to identify the most important deep learning approaches to solve routing problems of different variants. Also, this literature review categorizes the reviewed works differently, gives a comprehensive overview of applied algorithms, and the results of reviewed literature will be compared. Additionally, this work describes the reviewed literature in a little more depth compared to most of the aforementioned reviews.

1.4 Outline

The remainder of this report is organized as follows. In section 2, the vehicle routing problem is briefly introduced, including a mathematical formulation, solution method, and categorization. A machine learning introduction is given in section 3, in which, i.a., reinforcement learning, deep learning, attention mechanisms, and algorithms will be discussed. Then, in section 4 the literature on machine and deep learning for static and dynamic routing problems is reviewed. Finally, in section 5 the challenges and opportunities for future research are discussed and section 6 is the concluding section.

2 Vehicle routing problem introduction

The vehicle routing problem (VRP) requires the determination of a set of routes to be completed by a fleet of vehicles to visit a set of customers (vertices) (Toth and Vigo, 2002). The vehicles start and end in a depot and serve customers by delivering or collecting goods. Often the objective is to minimize the total route length while satisfying operational constraints. The VRP formulation was first introduced by Dantzig and Ramser (1959) who were concerned with finding the optimal routes of gasoline trucks to supply gas stations. Since then, the VRP is one of the most studied combinatorial optimization problems and many exact and approximate solution methods have been proposed. The VRP is a generalization of the traveling salesman problem in which a single vehicle must visit all customers through one tour. The VRP is known as an NP-hard combinatorial optimization problem. In the following subsections, the mathematical model of a VRP variant is introduced, a brief overview of conventional solution methods is described, and finally a categorization for the VRP is discussed.

2.1 Mathematical model for the MCVRP

The VRP is defined on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_0, ..., v_n\}$ is the set of vertices and $\mathcal{E} = \{(i,j) : (i,j) \in \mathcal{V}^2, i < j\}$ is the set of edges. Vertex v_0 corresponds to the depot and vertices $\{v_1, ..., v_n\}$ represent the customers. Each edge $(i,j) \in \mathcal{E}$ is associated with a cost c_{ij}, which represent the cost (e.g., distance or travel time) to travel from vertex i to j. Loop arcs (i,i) are generally not allowed, which can be imposed by setting $c_{ii} = \infty$ for all $\forall i \in \mathcal{V}$. If the cost matrix c is asymmetric $(c_{ij} \neq c_{ji})$, the edges are called directed, otherwise the edges called undirected. Each customer is associated with a demand d_i for $i = \{1, ..., n\}$. In the capacitated VRP with multiple vehicles (MCVRP), the customer demand must be satisfied by a vehicle $k \in \mathcal{K}$ with capacity C, where $\mathcal{K} = \{1, ..., K\}$. The load of the vehicle after visiting customer i is denoted by u_i for each $i \in \mathcal{V} \setminus \{0\}$. In the following formulation, the binary decision variable x_{ijk} is 1 if edge $(i,j) \in \mathcal{E}$ is traversed by vehicle $k \in \mathcal{K}$ and is 0 otherwise. The binary decision variable y_{ik}

for each $i \in \mathcal{V}, k \in \mathcal{K}$ is 1 if vertex i is served by vehicle k and is 0 otherwise. In this example, the objective is to minimize the total cost (Toth and Vigo, 2002):

$$\min \sum_{i\in\mathcal{V}} \sum_{j\in\mathcal{V}} \sum_{k\in\mathcal{K}} c_{ij} x_{ijk} \tag{2.1}$$

subject to

$$\sum_{k\in\mathcal{K}} y_{ik} = 1 \quad \forall i \in \mathcal{V} \setminus \{0\}, \tag{2.2}$$

$$\sum_{k\in\mathcal{K}} y_{0k} = K, \tag{2.3}$$

$$\sum_{j\in\mathcal{V}} x_{ijk} = \sum_{j\in\mathcal{V}} x_{jik} = y_{ik} \quad \forall i \in \mathcal{V}, k \in \mathcal{K}, \tag{2.4}$$

$$u_{ik} - u_{jk} + Cx_{ijk} \leq C - d_j \quad \forall i, j \in \mathcal{V} \setminus \{0\}, k \in \mathcal{K}, i \neq j, \tag{2.5}$$

$$d_i \leq u_{ik} \leq C \quad \forall i \in \mathcal{V} \setminus \{0\}, k \in \mathcal{K}, \tag{2.6}$$

$$y_{ik} \in \{0, 1\} \quad \forall i \in \mathcal{V}, k \in \mathcal{K}, \tag{2.7}$$

$$x_{ijk} \in \{0, 1\} \quad \forall i, j \in \mathcal{V}, k \in \mathcal{K}. \tag{2.8}$$

Constraints (2.2) and (2.3) ensure that each customer is served by exactly one vehicle and that exactly K vehicles are leaving the depot, respectively. Constraints (2.4) impose that when a vehicle enters a vertex, the same vehicle leaves that vertex. Constraints (2.5) and (2.6) are a generalization of the subtour elimination constraints, proposed by Miller et al. (1960) for the TSP and extended by Desrochers and Laporte (1991) for the VRP, and impose both the capacity and connectivity requirements, such that $d_i + d_j \leq C$. If $x_{ijk} = 1$, these constraints ensure that $u_{jk} \geq u_{ij} + d_j$.

2.2 Solution methods

The VRP can be solved by a variety of solution methods and in this section several solution methods will be briefly described. Exact solution methods will result in an optimal solution but come at the cost of computational time. For large problem instances, it is often not possible to determine the optimal solution because of time limitations. Therefore, heuristic solution methods can be used to find a feasible and reasonably good quality solution in less computational time. However, it is not guaranteed to find the optimal solution and most often the final solution is a local-optima. Heuristic methods are often problem-specific, whereas metaheuristic methods are more general solution methods that are applicable to a wider variety of applications. They provide strategy guidelines and a general structure to design a heuristic for a specific problem. The solution quality of metaheuristic methods tends to be higher than with heuristic methods but requires more computational time. For a comprehensive review of solution methods for the VRP see, e.g., Laporte (2009) and Bräysy and Gendreau (2005a) (for heuristic methods), and Bräysy and Gendreau (2005b) (for metaheuristic methods).

2.2.1 Exact methods

Exact methods to solve the VRP are only suitable for small-scale problems since the VRP belongs to the problem difficulty class NP-hard. According to Baldacci et al. (2012), brand-and-cut and reduced set partitioning algorithms are the most successful exact methods for solving the capacitated VRP (CVRP). Exact methods are not of very much use for ML approaches to solve routing variants. Hence, the reader is referred to Baldacci et al. (2012) for details on exact methods.

2.2.2 Heuristic methods

Contrasting to exact methods, heuristic methods do not guarantee finding the optimal solution. However, they typically produce good solutions in shorter computing times. Heuristic methods can be classified into three categories: constructive, two-phase, and improvement heuristics (Laporte and Semet, 2002). Construction heuristics build a solution gradually by inserting customers into a vehicle route or by merging partial routes. The two-phase heuristic decomposes the problem into two components: clustering of vertices and route construction. This can be done in two fashions: cluster-first route-second or route-first cluster-second. In the former case, vertices are feasibly clustered first and then a route is constructed for each of them. In the latter case, a tour is constructed through all vertices and then divided into feasible vehicle routes. Improvement heuristics attempt to iteratively improve a feasible solution (often initially generated by a construction heuristic) by exchanging edges or vertices within or between routes. In case edges or vertices are exchanged within routes, the method is classified as intraroute methods. Exchanging edges or vertices between routes are interroute methods.

2.2.2.1 Construction heuristics

A very well-known heuristic solution method for the VRP is the Clarke and Wright savings algorithm (Clarke and Wright, 1964). The algorithm is based on a savings computation and starts with an individual route for every customer (so initially $n-1$ routes). The savings are computed as $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ for each $i, j \in \mathcal{V}$ and ordered decreasingly. The highest saving of two routes, with one route containing edge $(i, 0)$ and the other route containing $(0, j)$, are merged (if feasible) by deleting edges $(i, 0)$ and $(0, j)$ and introducing (i, j). This is repeated until no further improvement is possible. In case a solution is required with a fixed number of vehicles, routes can be merged even with negative savings until the required number of vehicles is reached. Other examples of construction heuristics are the nearest neighbor algorithm, the nearest insertion algorithm, and the random order algorithm. The nearest neighbor algorithm starts the first route by finding the vertex closest to the depot and then adds subsequent vertices closest to the last added vertex at the end of the route until no vertices can be feasibly added, where after a new route is started. This procedure is repeated until no unrouted vertices are left. The nearest insertion algorithm starts a route with two vertices (which can be selected with different methods) and then adds the vertex closest to any of the vertices in the route in such a way that it minimizes the route length (it may be placed between two vertices). A new route is started once no feasible insertions are possible and repeated until no unrouted vertices are left. The random insertion algorithm is similar to the nearest insertion algorithm but selects random vertices to insert instead of the closest vertex to any of the vertices in the route.

2.2.2.2 Two-phase heuristics

The sweep heuristic, introduced for the VRP by Wren and Holliday (1972) but commonly contributed to Gillett and Miller (1974), is a cluster-first route-second method. It creates feasible clusters by rotating a ray centered at the depot, where after routes are created by solving a TSP for each cluster. Instead of using a Cartesian coordinate system, vertices are represented by polar coordinates (θ_i, ρ_i), where θ_i is the angle (relative to an arbitrary vertex i^* for which $\theta_i^* = 0$) and ρ_i is the ray length. The vertices are ranked in increasing order of θ_i. Then, an unused vehicle is assigned to unrouted vertices having the smallest angle (as long its capacity is not exceeded), which is repeated for multiple vehicles until there are no more unrouted vertices. Thereafter, routes are constructed by solving the TSP for each vehicle (or cluster). Route-first cluster second methods are not competitive with other heuristic methods (Laporte and Semet, 2002) for the VRP and hence are not discussed.

2.2.2.3 Improvement heuristics

Lin (1965) introduced the interroute λ-opt method, where λ edges from a tour are removed, where after the unconnected vertices are connected in all possible ways. The first identified or most profitable reconnection is implemented. This procedure is continued until no further improvements can be made. Thompson and Psaraftis (1993) proposed an intraroute method for exchanging vertices between different routes, where a circular permutation of b routes is considered and k vertices from each route are shifted to the next route.

2.2.3 Metaheuristic methods

Many metaheuristic methods often exploit a local search procedure, which iteratively searches in the neighborhood of the current solution to improve it. Such a local search procedure often converges to a local optimum and cannot escape the local optimum. The local optimum found depends on the starting conditions of the local search. Repeating the local search many times with random starting conditions may result in multiple local optima and increases the chance that one local optimum is the global optimum, but is not successful for large and complicated problems (Hillier and Lieberman, 2014). Therefore, metaheuristic methods allow the local search to escape local optima and guide the local search in the most promising direction. The escape of local optima of different metaheuristics is dealt with in different ways.

2.2.3.1 Tabu search

Tabu search, introduced by Glover (1986), is a deterministic metaheuristic that allows non-improving moves of the local search to escape the local optimum and reapplies the local search in the best neighborhood of the local optimum. To avoid cycling back to the previous local optimum, a tabu list of a certain length with prohibited moves (the inverse moves) is maintained. A stopping criterion is used to stop the iterations. The use of an adaptive memory, introduced by Rochat and Taillard (1995), is a powerful concept to enhance the tabu search algorithm. The tabu search algorithm is used to solve the VRP by, e.g., Osman (1993) and Taillard et al. (1997).

2.2.3.2 Simulated annealing

Simulated annealing, proposed by Kirkpatrick et al. (1983), is a stochastic metaheuristic that escapes local optima by allowing non-improving moves of the local search in accordance with a random process. An improving move (similar to tabu search) is always performed and the probability of performing a non-improving move depends on the solution quality of that move, where moves resulting in a much worse solution quality are less like to be performed. A parameter that influences the probability of performing a non-improving move is changed during the process to guide the local search in the most promising direction. The stopping criterion is determined by a schedule of that parameter, indicating how to update the parameter and the number of iterations per parameter value. The simulated annealing algorithm is used to solve the VRP by, e.g., Osman (1993) and Chiang and Russell (1996).

2.2.3.3 Large neighbourhood search

The large neighborhood search (LNS) heuristic, proposed by Shaw (1997), starts with an initial solution, relaxes some vertices (removes vertices from routes), and then re-optimizes (reconstructs) the routes using a branch and bound procedure. The removal of vertices is stochastic but based on related features of vertices (such as location, time windows, or the tour it currently is in). Only if the new solution is better than the previous solution it will be used in the next iteration. The re-optimization may also be done using other insertion methods. Compared to local search-based metaheuristics (tabu search and simulated annealing), the LNS heuristic can explore the solution space more quickly, since it changes the solution per iteration more. The adaptive large neighborhood search (ALNS), proposed by Ropke and Pisinger (2006), is based on the LNS, but uses several removal and insertion heuristics and uses statistics acquired during the search to choose between them at each iteration. The ALNS of Ropke and Pisinger (2006) is embedded in a simulated annealing metaheuristic. To remove vertices from the solution it uses three removal heuristics: a stochastic removal heuristic based on feature relation, a random removal heuristic, and a worst removal heuristic. To re-optimize (and reconstruct) they use two parallel insertion heuristics (which construct multiple routes simultaneously): a basic greedy insertion heuristic (which tries at most n possibilities per iteration to insert an unrouted vertex resulting in the smallest increase of total route length) and a regret insertion heuristic (introduced by Potvin and Rousseau (1993), which incorporates a sort of look-ahead information by determining the regret value that indicates the difference in total route length between the best and second-best insert location of an unrouted vertex and then inserts the unrouted vertex with the minimum regret value at its best location. The ALNS keeps track of the performance of each combination of removal and insertion heuristic and updates their weight parameter during the search, which affects the probability of selecting a combination of removal and insertion heuristic.

2.2.3.4 Genetic algorithms

Genetic algorithms process a population of trial solutions (rather than a single trial solution as with tabu search and simulated annealing) in each iteration and are based on the theory of evolution (Charles Darwin). The members (different VRP solutions) of the population are measured by their fitness (e.g., total route length), where the members with the highest fitness have a higher chance of becoming parents. The number of parents and how to choose them from the population is a design choice and often in accordance with a stochastic policy. Parents are paired at random and get children who share features of their parents, i.e., a new feasible solution is created from two other solutions. Some children's features are mutated to explore new solution regions. The fitness of each of the children is evaluated and a new population is generated from the children and previous population. A stopping criterion is used to stop the iterations. The genetic algorithm is used to solve the VRP by, e.g., Prins (2004) and Baker and Ayechew (2003).

2.3 VRP categorization

According to Pillac et al. (2013), the vehicle routing problem can be classified into four categories based on the evolution and quality of information (Psaraftis, 1980). Evolution of information refers to whether the information available changes during route execution, so whether the input is all known beforehand or whether the input changes over time (i.e., static or dynamic). Quality of information is related to possible uncertainty of information, i.e., whether all information is deterministic or whether some information is stochastic. The four categories are:

- static and deterministic (SD):
  routes do not change after construction (the problem is solved once) and all information is known beforehand.
- static and stochastic (SS):
  routes do not change after construction (apart from some possible minor changes afterward, such as detour to depot) and some information is only revealed during the execution of the routes.
- dynamic and deterministic (DD):
  some (or all) information changes during the execution of the routes, e.g., dynamic customers.
- dynamic and stochastic (DS):
  some (or all) information is only revealed during the execution of the routes, with stochastic knowledge on the dynamically revealed information.

Psaraftis et al. (2016) build upon this categorization of Pillac et al. (2013) and proposed a taxonomy in which the dynamic VRP is classified into eleven categories. Ojeda Rios et al. (2021) stated that the taxonomy of Psaraftis et al. (2016) contained a rather limited set of solution methods and extended the taxonomy by providing a more comprehensive classification of the solution methods. Another popular taxonomy was provided by Eksioglu et al. (2009), which later was adapted by Braekers et al. (2016). Instead of differentiating between VRP variants with respect to assumptions and characteristics, many surveys focus on reviewing the solution methods for a specific variant of the VRP (Braekers et al., 2016). Popular reviews of specific VRP variants are, e.g., Oyola et al. (2016a; 2016b) and Ritzinger et al. (2016) for the dynamic and stochastic VRP, Bräysy and Gendreau (2005a; 2005b) for the VRP with time windows, Berbeglia et al. (2007) for the static pickup and delivery problem, and Montoya-Torres et al. (2015) for the VRP with multiple depots.

3 Machine learning introduction

A problem may be solved by an algorithm (sequence of instructions), for which the (complex) problem can be divided into smaller less complex problems. A handcrafted rule-based algorithm then tries to solve the sub-problems to solve the overall problem. For complex problems, constructing a rule-based and hierarchical type of algorithm is difficult, but may be able to solve the problem. However, it requires a lot of knowledge on the specific problem at hand. If the algorithm does not account for unexpected phenomena, it is bound to fail. Therefore, algorithms with this chain of rules approach will often not generalize to the complexity of the real world. Generalization, in this case, refers to the ability to perform well on previously unobserved inputs (Goodfellow et al., 2016). Without explicitly programming the rules to solve the problem (which requires problem-specific knowledge and context), one can 'learn' a computer to solve the problem using data to gain knowledge, which is known as machine learning. Besides that machine learning does not require much problem-specific knowledge, it enables to solve problems that are too difficult to solve with algorithms designed by humans.

There are three main types of machine learning: unsupervised, supervised, and reinforcement learning. In unsupervised learning, the goal is to find patterns in data, such as clustering. In supervised learning, the goal is to make predictions using data, for which a labeled training dataset is available to train a model that predicts outcomes compared to a provided target. Examples of supervised learning are regression analysis and classification. In reinforcement learning (RL), the goal is to make predictions using data as well, but the learning algorithm is not given a labeled training dataset. An agent interacts with the environment and must discover the highest rewarding actions. The challenge in reinforcement learning is the trade-off between exploration and exploitation. Actions that are tried before and returned high rewards are preferred by the RL agent (exploit). However, in order to select better actions in the future, the RL agent must try actions that were not selected before (explore) (Bishop, 2006).

In many situations, one wants to minimize (or maximize) an objective function that defines the quality of a solution, for which gradient-based optimization is often used. Suppose the goal is to reduce a function $f(x)$, this can be done by moving x a small step in the opposite direction of the derivative $f'(x)$. However, when the derivative $f'(x) = 0$, it does not provide information about which direction to move in. These stationary points are called local minima (or local maxima or saddle points) and make the optimization more difficult. Most functions have multiple inputs, which all must be considered for gradient-based optimization. Therefore, the derivatives for all inputs are captured by the gradient, i.e., the gradient is a vector containing all partial derivatives, denoted by $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$. If all elements of the gradient are zero, the multidimensional function has reached a stationary point. A method called gradient descent can be used to minimize objective function by moving from point $\boldsymbol{x}$ to $\boldsymbol{x}'$:

$$\boldsymbol{x}' = \boldsymbol{x} - \alpha \cdot \nabla_{\boldsymbol{x}} f(\boldsymbol{x}), \tag{3.1}$$

where $\alpha > 0$ is the learning rate, which determines how much to update the input in the direction of the gradient. If the learning rate is too large, it may overshoot a local or global minimum. On the other hand, with a too small learning rate it may take a very long time to converge to a local or global minimum. The input $\boldsymbol{x}$ is updated most into the direction of the greatest partial derivative. For better and faster convergence, the inputs should be normalized or scaled, such that each input feature is represented within the same range of values.

A (deep) neural network is a very powerful machine learning concept. Before introducing the details of neural networks, an introduction to supervised and reinforcement learning will be given.

3.1 Supervised learning

In supervised learning regression analysis, the goal is to build a model that predicts a scalar value $y \in \mathbb{R}$ as its output given an input vector $\boldsymbol{x} \in \mathbb{R}^n$. In linear regression, the output is a linear function of the input. Let $\hat{y}$ be the output of the model that predicts y and is calculated by:

$$\hat{y} = \boldsymbol{\theta}^T \boldsymbol{x} + \boldsymbol{b}, \tag{3.2}$$

where $\boldsymbol{b}$ is an intercept term (often called bias) and $\boldsymbol{\theta} \in \mathbb{R}^n$ is a vector of weight parameters, which determines how the prediction is affected by the input features. The performance of the prediction can be quantified by, e.g., the mean squared error. Suppose a training dataset consisting of m inputs $\boldsymbol{x}$ and values y. The objective function (also called cost or loss function) can be defined as:

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \Big(\hat{y}_i(\boldsymbol{\theta}) - y_i \Big)^2. \tag{3.3}$$

The weight parameters $\boldsymbol{\theta}$ and bias $\boldsymbol{b}$ must be determined by the machine learning model. A model with only weights can also be used if $\boldsymbol{x}$ is augmented with an extra entry equal to 1, such that the corresponding entry in $\boldsymbol{\theta}$ captures the biases. Then, the weight parameters are iteratively updated to minimize the objective function by:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \cdot \nabla_{\boldsymbol{\theta}_t} J(\boldsymbol{\theta}_t) \tag{3.4}$$

$$= \boldsymbol{\theta}_t - \alpha \cdot \frac{2}{m} \sum_{i=1}^{m} \Big(\hat{y}_i(\boldsymbol{\theta}_t) - y_i \Big) \cdot x_i, \tag{3.5}$$

where t indicates the iteration. The weights are updated until convergence or for a maximum number of iterations, where after a trained model can accurately predict the value y with output $\hat{y}$. The model is trained using a training dataset, hence the model is 'optimized' for the training dataset. The challenge is to generalize to new (unseen) inputs. If the model is trained on a large dataset, it tends to generalize well to new inputs (if the test and training dataset are is identically distributed and if inputs independent from each other).

Stochastic gradient descent
As mentioned earlier, the machine learning model generally generalizes better when trained on more data. However, this requires calculating more partial derivatives. All partial derivatives must be stored, since the average is used to update the model parameters (as in equation (3.5)). As each input may consist of multiple features, for which a partial derivative must be calculated each, a lot of calculations must be performed (and stored). This is computationally costly and time-consuming. The stochastic gradient descent (SGD) method, which is an extension of the gradient descent method, tackles this problem by selecting a random subset of the training dataset to be used in each iteration. The subset is called a batch, which consists of randomly selected instances (data points). At each iteration $t \in \{0, 1, 2, ...\}$, SGD updates the model parameters as in equation (3.4).

3.2 Reinforcement learning

The goal in reinforcement learning is to train an agent that interacts with its environment by taking actions (which change the environment) to maximize the rewards given by the environment (Sutton and Barto, 2018). This is a sequential decision-making process that can be formulated as a Markov decision process (MDP) (Bellman, 1957). The interaction between the agent and the environment is illustrated by Figure 1.

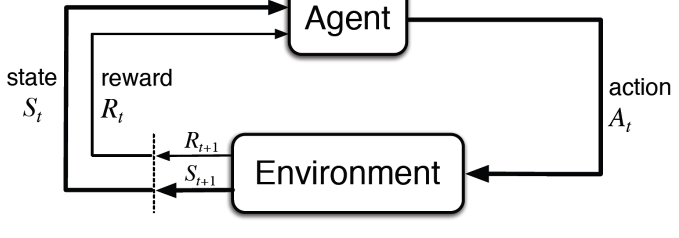


Figure 1: Agent-environment interaction in an MDP. Image retrieved from Sutton and Barto (2018).

At each time step t, the agent observes the environment's state $S_t \in \mathcal{S}$ and selects action $A_t \in \mathcal{A}$. Action A_t causes the state to transition to S_{t+1} and the agent receives a rewards R_{t+1}. The joint probability of getting reward r and reaching state s' given the selected action a in state s is defined by $p(s', r|s, a)$ for all $s', s \in \mathcal{S}, r \in \mathcal{R}$, and $a \in \mathcal{A}$. The sequence of the agent-environment interaction is given by $\{S_0, A_0, R_1, S_1, A_1, R_2, ...\}$. The agent's goal is to maximize the total reward (called return) it receives. To discount for future rewards and ensure a bounded return, a discount rate γ is used to determine the discounted return. More specifically, the agent tries to maximize the expected discounted return, which is defined by:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \\ &= \sum_{k=0}^{T} \gamma^k R_{t+k+1} \\ &= R_{t+1} + \gamma G_{t+1}, \end{aligned} \tag{3.6}$$

where $T \neq \infty$ and $\gamma \in [0, 1]$ for episodic tasks, and $T = \infty$ and $\gamma \in [0, 1)$ for continuing tasks.

The agent's policy defines how the agent should act at any given time. It is a mapping of the perceived environmental state to the probabilities of selecting each action. The policy is denoted by $\pi(a|s)$ for all $a \in \mathcal{A}$ and $s \in \mathcal{S}$. The expected return from a state s is called the state-value. The state-value function $v_\pi(s)$, following a policy π starting in state s is defined by:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k R_{t+k+1}|S_t = s\right] \qquad \forall s \in \mathcal{S}, \end{aligned} \tag{3.7}$$

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value of a variable while following policy π.

Similarly, the action-value function $q_\pi(s, a)$ is the expected return of taking action a from state s, and thereafter following policy π:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a\right] \qquad \forall s \in \mathcal{S}, a \in \mathcal{A}. \end{aligned} \tag{3.8}$$

The goal in RL is to find the optimal policy π_*, where π_* is better than or equal to a policy π' if $v_{\pi_*} \geqslant v_{\pi'}(s)$ for all $s \in \mathcal{S}$. The Bellman optimality equations for the state-value and action-value function are given by:

$$v_*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a)\Big[r + \gamma v_*(s')\Big], \tag{3.9}$$

$$q_*(s, a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a)\Big[r + \gamma \max_{a' \in \mathcal{A}} q_*(s', a')\Big]. \tag{3.10}$$

If $v_*(s)$ is known, the optimal policy can be determined by picking the actions that appear best after a one-step look-ahead search. If q_* is known, it is even easier to determine the optimal policy. A one-step look-ahead search is not needed, since the best actions correspond to the maximum value of $q_*(s, a)$ for all $s \in \mathcal{S}$. Hence, it is not required to know anything about successor states and their values, i.e., it is not required to know the transition function that defines the dynamics of the environment. However, solving the Bellman optimality equations requires an exhaustive search, considering every single possibility, computing their probabilities, and their expected rewards. Therefore, the Bellman optimality equations are often approximated in practice, for which the most popular algorithms are given in section 3.5.

RL categorization

Reinforcement learning algorithms can be categorized based on the way the agent is acting: model-based and model-free. In model-based RL, the agent learns or uses a transition function that predicts the next state and next rewards given a state-action pair. An example of model-based RL is dynamic programming (DP), in which $p(s', r|s, a)$ is known and used by the agent to produce simulated experience (the agent takes actions by taking into account expected future situations before they occur). In model-free RL, the agent does not use a model of the environment and purely learns through trial and error by interaction with the environment. Examples of model-free RL are Monte Carlo (MC) and temporal-difference (TD) methods.

Additionally, there are three main types of model-free RL algorithms:

- value-based methods:
  the agent learns the state- or action-value function and the policy is based on the state- or action-values.
- policy-based methods:
  the agent does not use a value function to determine the optimal actions but learns the policy directly.
- actor-critic methods:
  both the policy and the value function are learned.

Moreover, the type of agent can be categorized as off-policy or on-policy. The policy that determines the actions of the agent is called the behavior policy. The policy that learns about the values of states (and actions) is called the target policy. For an off-policy agent, the target and behavior policy are different. For an on-policy agent, the target and behavior policy are the same.

3.3 Deep neural networks

A deep neural network (DNN) is a very powerful ML concept to approximate a function that maps an input to an output. DNNs are used by many ML approaches for solving routing problems. In this section, the architecture of a single-layer neural network (perceptron) will be introduced first. Thereafter, the architecture of a DNN is explained.

3.3.1 Perceptron architecture

The perceptron is the most simple neural network (NN), containing an input layer and output node. The output of the perceptron can, e.g., be a value that predicts the class of an input. Suppose a training instance with $\boldsymbol{x} = [x_1, ..., x_n]$ and $\boldsymbol{y} \in \{-1, 1\}$, where n indicates the number of features. The architecture of the perceptron is depicted in Figure 2.

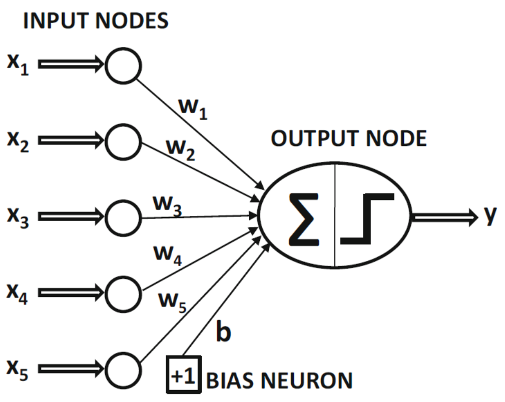


Figure 2: Perceptron architecture. Image retrieved from Aggarwal (2018).

The output of the perceptron can be defined as (with model weights $\boldsymbol{w}$ instead of $\boldsymbol{\theta}$):

$$\hat{y} = \text{sign}\Big\{ \sum_{j=1}^{n} w_j x_j + b \Big\} = \text{sign}\{\boldsymbol{w} \cdot \boldsymbol{x} + b\}, \tag{3.11}$$

where the sign function maps a value to +1 or -1. The weights can be updated with SGD using, e.g., the error as the objective value: $E(\boldsymbol{x}) = y - \hat{y}$. The sign function in this example acts as a so-called activation function (more examples of activation function will be given later). Similar to the input augmentation in the example of linear regression, the bias can be captured by adding an extra input entry to the input. The output of the perceptron then is calculated by:

$$\hat{y} = \text{sign}\Big\{\sum_{j=1}^{n+1} w_j x_j\Big\} = \text{sign}\{\boldsymbol{w} \cdot \boldsymbol{x}\}, \tag{3.12}$$

where this output can for example represent the probability of an input being one class out of two.

3.3.2 Deep neural network architecture

The perceptron is not able to approximate a complex function that maps an input to output, but deep neural networks are. A deep neural network (DNN) contains 2 or more computational layers. In a DNN, the layers between the input layer and output layer are called the hidden layers. The architecture of a deep neural network, with 2 hidden layers, is depicted in Figure 3.

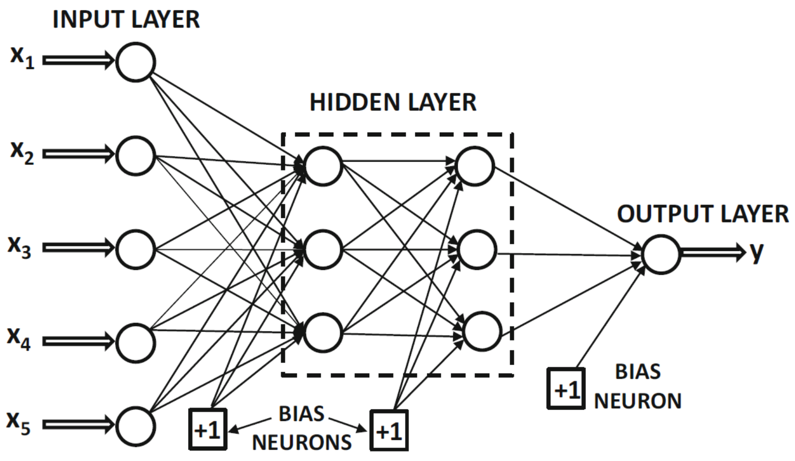


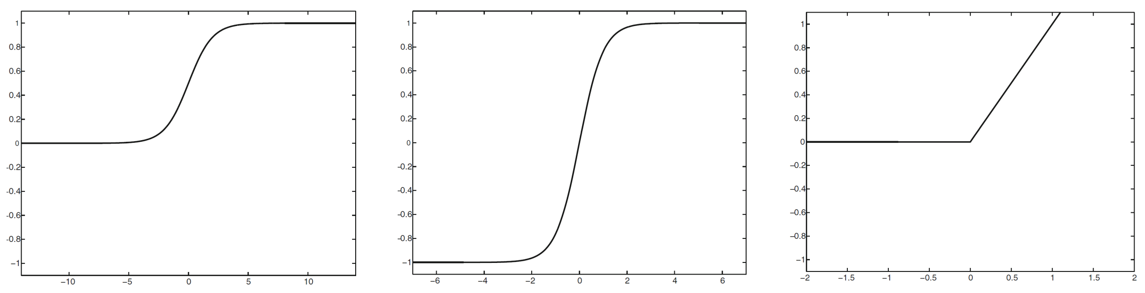
Figure 3: Example of deep neural network architecture. Image retrieved from Aggarwal (2018) (slightly adjusted).

The above-depicted example is also called a feed-forward (FF) network. More specifically, it is a fully connected feed-forward network, because each neuron in a layer is connected to all neurons in the next layer. The connections between neurons represent a scalar weight. The output of a hidden layers is denoted by vector $\boldsymbol{h}^{(i)}$ for $i \in \{1, ..., l\}$, where l is the number of hidden layers. Each neuron in the hidden layers uses an activation function denoted by Φ. The most used activation functions are the sigmoid function (σ), the hyperbolic tangent (tanh), and the rectified linear unit (ReLu), which are depicted in Figure 4 and defined as:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}, \tag{3.13}$$

$$\tanh(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1}, \tag{3.14}$$

$$\text{ReLu}(x) = \max\{0, x\}. \tag{3.15}$$



(a) Sigmoid (b) Tanh (c) ReLu

Figure 4: Common activation functions. Images retrieved from Aggarwal (2018).

In the previous example of the perceptron, the output layer contains one output neuron but the output layer may contain several output neurons. The number of output nodes typically depends on the problem. For example, if the objective is to determine a probability distribution to classify the input in of the k classes, the output layer contains k neurons. The probability distribution is often calculated using the softmax function:

$$\hat{y}_i = \text{softmax}(\hat{\boldsymbol{y}})_i = \frac{\exp(\hat{y}_i)}{\sum_{j=1}^{k} \exp(\hat{y}_j)} \qquad \forall i \in \{1, ..., k\}. \tag{3.16}$$

According to this probability distribution, a decision can be made by, e.g., selecting the output with the highest probability (greedy selection) or using ϵ-greedy selection, which selects the highest output with probability $1-\epsilon$. The calculations of a neural network are often represented by matrix and vector multiplications. Let $\boldsymbol{W}^{(1)}$ be the weight matrix connecting the input layer to the first hidden layer. Suppose an input vector $\boldsymbol{x}$ contains n features (the dimension of $\boldsymbol{x}$ is $\mathbb{R}^n$) and the first hidden layer contains k_1 neurons. Then, $\boldsymbol{W}^{(1)} \in \mathbb{R}^{k_1 \times n}$ (the matrix has size $k_1 \times n$). The weight matrices connecting two hidden layers, $\boldsymbol{W}^{(i)}$ for $i \in \{2, ..., l\}$, have dimension $k_{i+1} \times k_i$ for $i \in \{2, ..., l\}$. The weight matrix connecting the last hidden layer and output layer ($\boldsymbol{W}^{(l+1)}$) has dimension $k_o \times k_l$, where k_o is the number of neurons in the output layer. The equations of a FF neural network transforming the input to the output are defined as:

$$\text{input to hidden layer:} \qquad \boldsymbol{h_1} = \Phi\Big(\boldsymbol{W}^{(1)}\boldsymbol{x}\Big), \tag{3.17}$$

$$\text{hidden to hidden layer:} \qquad \boldsymbol{h^{(i)}} = \Phi\Big(\boldsymbol{W}^{(i)}\boldsymbol{h}^{(i-1)}\Big) \quad \forall i \in \{2, ..., l\}, \tag{3.18}$$

$$\text{hidden to output layer:} \qquad \hat{\boldsymbol{y}} = \Phi\Big(\boldsymbol{W}^{(l+1)}\boldsymbol{h}^{(l)}\Big), \tag{3.19}$$

where the activation functions are applied element-wise. The number of neurons must be carefully chosen, as too many neurons may result in overfitting. Overfitting means that the model performs well on training data, but generalizes poorly. This may occur when the number of weight parameters is too large compared to the training dataset, resulting in a model that 'remembers' the specific training instances without recognizing the patterns in the training dataset (Aggarwal, 2018).

3.3.3 Training DNNs

The output of a DNN is a function of all weight parameters of the network. Since the output of a DNN is used to determine the loss and update the model parameters, the influence of all parameters on the output must be determined. Therefore, to determine the gradient, a backpropagation algorithm is used which leverages the chain rule. The backpropagation algorithm utilizes dynamic programming to compute the gradient and consists of two phases: the forward phase and the backward phase. In the forward phase, the input is mapped into the output, by which the loss function is calculated and the derivative of the loss function w.r.t. the output. In the backward phase, the gradient of the loss function w.r.t. to each weight is determined, which is done by using the chain rule of derivatives.

The output of a neuron (sometimes called activation value) in the output layer is defined as:

$$a_1^{(l+1)} = \Phi\Big(w_{1,1}^{(l+1)}a_1^{(l)} + w_{1,2}^{(l+1)}a_2^{(l)} + ... + w_{1,k_l}^{(l+1)}a_{k_l}^{(l)} + b_1^{(l+1)}\Big), \tag{3.20}$$

where $w_{1,1}^{(l+1)}, ..., w_{1,k_l}^{(l+1)}$ represent the weights connecting the neurons in the preceding layer ($a_1^{(l)}, ...,$ $a_{k_l}^{(l)}$) to neuron $a_1^{(l+1)}$. Equation (3.20) helps to understand how the activation of a neuron can be changed. First of all, the activation value can be changed by changing the bias, for which the change is constant. Secondly, the activation value can be changed by changing the weights, for which a change of the weight connecting to a neuron with a higher activation value has a bigger effect. Thirdly, the activation value can be changed by changing the activation values of neurons in the previous layer. However, the activation values cannot be changed directly, but can only be changed by changing their weights and bias. The effect of a change of weights and bias determining the activation value of the neuron in the previous layer on the output neuron can be calculated with the chain rule (Sanderson, 2017).

Most often the output layer consists of multiple neurons and the goal is not to get the output of a single output to be as close as possible to its target, but to get all outputs as close as possible to its target. Therefore, each output neuron should be updated proportionally to the difference of its output target. Hence, for each output neuron, the most optimal update of its weights and bias is determined. In a fully connected network, the weights are connected to multiple outputs, and thus cannot be updated such that it satisfies all output neurons. Therefore, the desired updates are averaged. This process can also be done for the second to last hidden layer and all the other hidden layers. Moving backward through the network and repeating this process is referred to as backpropagation. Ultimately, this entire process is repeated for the entire batch (a subset of the training dataset) before updating the model parameters. Very deep networks may experience the vanishing gradient problem, which occurs during the backpropagation. In that case, the updates of model parameters in earlier layers are very very small.

3.4 Attention mechanisms

An attention mechanism is a very much used concept by DL methods for solving routing problems. Recurrent neural networks (RNNs) formed the foundation of attention mechanisms. Therefore, in this section a brief introduction to RNNs will be given, followed by a description of several attention mechanism architectures.

3.4.1 Recurrent neural networks

An RNN is a specialized neural network for processing sequential data (Goodfellow et al., 2016). The output of an RNN cell is a function of the preceding outputs and is formed by using the same network parameters in every step. Equation (3.21) is the basic equation of an RNN that is iteratively solved to compute a sequence of output values ($o_1, ..., o_n$) from sequence of input values ($x_1, ..., x_n$). The time step index t does not necessarily indicate a real-world time step, but can also indicate the position in the sequence.

$$\boldsymbol{h}_t = \Phi(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}) \tag{3.21}$$

Figure 5 depicts the process of an RNN that for each time step maps an input sequence $\boldsymbol{x}$ to an output sequence $\boldsymbol{o}$. The black square in the left diagram of Figure 5 indicates a delay of a single time step (from the state at time t to the state at time $t + 1$) between interactions. The right side of Figure 5 illustrates the unfolded computational graph. The unfolded graph size depends on the sequence length, i.e., the input and output sequence have the same length. The unfolded computational graph can be represented by the forward propagation equations (3.22) - (3.24):

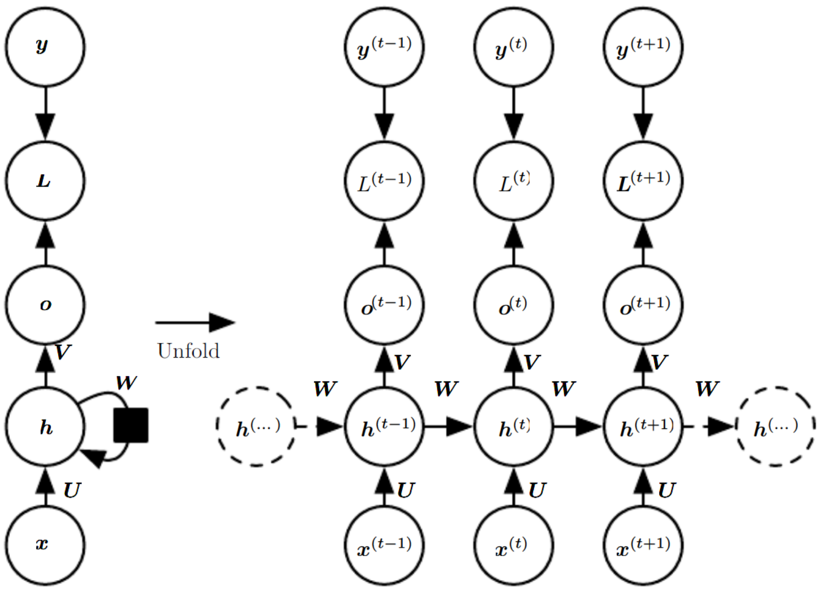


Figure 5: RNN with connections between hidden states. Left: recurrent connections where the black square indicates a delay of single time step (from the state at time t to the state at time $t+1$) between interactions. Right: unfolded computational graph where every node is associated with a particular time instance. Image retrieved from Goodfellow et al. (2016).

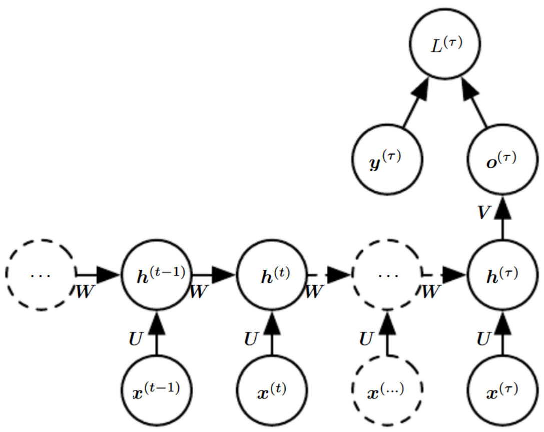
$$\boldsymbol{h}_t = \Phi(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{b}), \tag{3.22}$$

$$\boldsymbol{o}_t = \boldsymbol{V}\boldsymbol{h}_t + \boldsymbol{c}, \tag{3.23}$$

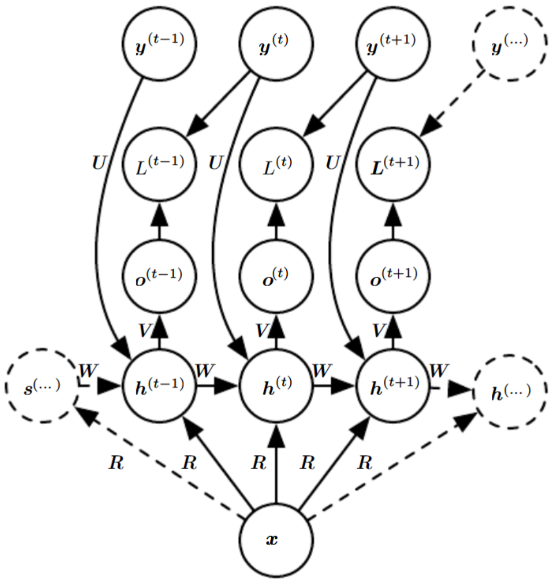
$$\hat{\boldsymbol{y}}_t = \text{softmax}(\boldsymbol{o}_t), \tag{3.24}$$

where $\boldsymbol{h}_t$ is a hidden state at time step t, $\boldsymbol{U}$ is a weight matrix connecting the input to the hidden states, $\boldsymbol{W}$ a weight matrix connecting the hidden states to hidden states, $\boldsymbol{V}$ a weight matrix connecting the hidden states to the output, and $\boldsymbol{b}$ and $\boldsymbol{c}$ are the bias vectors. The weight matrices and biases are shared by all the connections. The loss function uses $\hat{\boldsymbol{y}}$ to compare the normalized probabilities over the output to the target $\boldsymbol{y}$. The total loss is then defined by the sum of all losses over all time steps. The run time of this type of RNN scales with the sequence length, i.e., the run time is $\mathcal{O}(n)$, since the RNN structure is sequential and no parallelization can be applied. Since the calculated hidden states in the forward pass are needed during the backward pass, the memory cost is $\mathcal{O}(n)$ also. The algorithm used for the backpropagation is called the backpropagation through time (BPTT), which takes into account the sharing of parameters and temporal length. The vanishing gradient problem arises when RNNs are used on long sequences (for which the network is very deep), hence it is said that RNNs have a short-term memory. To tackle this problem, the long short-term memory (LSTM) and gated recurrent unit (GRU) were created. For details on the LSTM and GRU, see Appendix A.

There are various variants of recurrent neural networks of which two formed the basis for the sequence to sequence task, including routing problems. These two RNNs are depicted in Figure 6. The RNN that maps the input sequence to a fixed-length vector is illustrated by Figure 6a. The RNN that maps a fixed-length vector to a sequence is illustrated by Figure 6b.



(a) RNN that maps the input sequence to a fixed-size vector.



(b) RNN that maps an input vector to a sequence.

Figure 6: RNN variants that formed the basis for sequence to sequence task. Images retrieved from Goodfellow et al. (2016).

3.4.2 Encoder-decoder architecture

RNNs have the limitation that the output dimension is equivalent to the input dimension. The first who solved this problem were Cho et al. (2014), where after Sutskever et al. (2014) developed the model of Cho et al. (2014). These model architectures, illustrated by Figure 7, are contemporary known as encoder-decoder or sequence-to-sequence models. The general idea is to use two LSTMs, where the first LSTM (encoder) maps an input sequence to an output vector (similar to Figure 6a) and the second LSTM (decoder) maps an input vector (which is the output of the first LSTM) to an output sequence (similar to Figure 6b). The output vector of the encoder is often referred to as the context vector. This architecture allows to map a sequence to a sequence of different lengths and formed the foundation of many DRL approaches for solving the VRP.

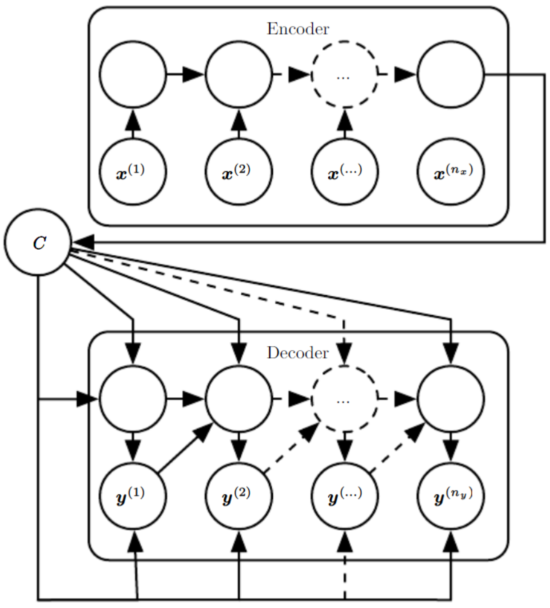


Figure 7: Encoder-decoder architecture. Image retrieved from Goodfellow et al. (2016).

3.4.3 Attention model

Bahdanau et al. (2014) observed that when the input dimension of the encoder is large, the output of the encoder cannot be summarized properly in a single context vector. To overcome this problem, Bahdanau et al. (2014) introduced an attention mechanism where the decoder can use any of the hidden states of the encoder. The encoder maps the input sequence to a sequence of vectors (the hidden states), after which a neural network is used to determine the most relevant hidden states for each decoding step. This model is mathematically defined, with encoder hidden states $(\boldsymbol{e}_1, ..., \boldsymbol{e}_n)$ and decoder hidden states $(\boldsymbol{d}_1, ..., \boldsymbol{d}_m)$, as:

$$u_j^i = \Phi(\boldsymbol{W_1}\boldsymbol{e}_j + \boldsymbol{W_2}\boldsymbol{d}_i) \quad j \in (1, ..., n), \tag{3.25}$$

$$a_j^i = \text{softmax}(u_j^i) \quad j \in (1, ..., n), \tag{3.26}$$

$$\boldsymbol{c}_i = \sum_{j=1}^{n} a_j^i \boldsymbol{e}_j, \tag{3.27}$$

where u_j^i scores to which extent the input around position j and the output at position i match, $\boldsymbol{a}$ is the attention vector, and $\boldsymbol{c}$ the context vector.

3.4.4 Pointer Network

Despite that the architecture of Bahdanau et al. (2014) makes it possible to efficiently map an input sequence to an output sequence of different lengths, the length of the output sequence is required to be fixed a priori. The Pointer Network (PN) of Vinyals et al. (2015b) was the breakthrough for solving combinatorial optimization problems where the size of the output is both variable and dependent on the length of the input. In other words, the PN allows predicting a sequence of outputs that is of equal length of the input, where the length of the input is defined during testing (instead of that the length must be known during testing). The attention mechanism of the PN is mathematically be expressed by:

$$u_j^i = \Phi(\boldsymbol{W_1}\boldsymbol{e}_j + \boldsymbol{W}_2\boldsymbol{d}_i) \quad j \in (1, ..., n), \tag{3.28}$$

$$\boldsymbol{a} = \text{softmax}(\boldsymbol{u}^i), \tag{3.29}$$

where u_j^i are the pointers to the input elements, and $\boldsymbol{a}$ is the normalized distribution of the vector $\boldsymbol{u}^i$ (with length n). Vinyals et al. (2015b) applied the PN framework to solve the TSP.

3.4.5 Transformer model

In the PN of Vinyals et al. (2015b), the main limitations are the inability to parallelize due to the sequential computations of RNNs and the loss of information for long sequences due to long path lengths of information. The Transformer architecture of Vaswani et al. (2017) improves on these aspects compared to the PN. With an RNN encoder, one element of the input sequence is embedded after the other. As explained earlier, the hidden state at the current time step has dependencies on the previous hidden states. In a Transformer encoder, all the input elements are embedded simultaneously. The embedding is done without RNNs and thus the positional information is lost, which was previously captured through the sequential nature of the RNN. Therefore, the embedding is provided with positional encoding. Instead of an attention mechanism where the decoder pays attention to the hidden states of the encoder (as in the PN), in the Transformer model a mechanism of self-attention is used. With self-attention, it is defined how relevant an input element is compared to all other elements of the input sequence. The decoder in the Transformer consists of three main components. First, a self-attention mechanism that determines the extent to which an element is related to all other elements in the so-far produced output. Secondly, a multi-head attention mechanism that combines the self-attention vectors of the encoder and the decoder and determines how related each self-attention vector is to each other. Third, a feed-forward neural network that processes the attention vectors. At each time step, the decoder outputs a probability distribution over the inputs for the next decision. In contrast to attention mechanisms that utilize RNNs, the Transformer architecture allows for more parallelization, which benefits the computation time. Additionally, less information is lost, since the maximum path length of the information is reduced from $\mathcal{O}(n)$ to $\mathcal{O}(1)$, where n is the input sequence length.

3.5 DRL algorithms

Deep reinforcement learning combines deep learning and reinforcement learning. The most popular DRL algorithms applied to solve routing problems are described in this section.

3.5.1 Value-based methods

Traditionally, the state-value function v_π and action-value function q_π were approximated through continuous iteration. The true value of a state used to be determined by averaging over a lot of collected rewards that followed a state. However, to encounter states sufficiently often, many iterations are needed. Similarly, averaging all collected rewards for each action taken in a certain state will converge to true action value in that state after a lot of iterations. A popular value-based RL algorithm is the Q-learning algorithm (Watkins and Dayan, 1992), which estimates $q_*(s,a)$ by creating a table with Q-values for all state-action combinations. This table is iteratively updated when the agent takes an action and receives a reward. The values in the table represent the cumulative reward. For large problems, the table can get very large and need many iterations. Hence, for very large state and action spaces, this type of tabular method is not very efficient. Mnih et al. (2015) proposed the deep Q-network (DQN) to overcome this bottleneck. The DQN uses a DNN to approximate the Q-values of state-action pairs. To find a trade-off between exploration and exploitation, the DQN algorithm uses a ϵ-greedy strategy for selecting an action. Hence, DQN is an off-policy algorithm. The agent's experiences are stored in an experience replay buffer, in which the experience $e_t = (S_t, A_t, R_T, S_{t+1})$ for each time step is stored until the buffer is full. Thereafter, the oldest experience is deleted to store new experiences. The model parameters (which are initialized randomly) are updated by using a random mini-batch from the experience replay buffer is sampled. This is done because of three reasons. First, learning from correlated consecutive experiences is not efficient. Second, this increases the data efficiency, since experiences are used multiple times to update the weights. Third, it avoids parameter oscillation or divergence by averaging over many experiences. Additionally, a parallel target DNN is used to evaluate the Q-value output by the DNN with parameters $\boldsymbol{\theta}$. The parameters of the target DNN $\boldsymbol{\theta'}$ only updated after several iterations, while $\boldsymbol{\theta}$ is updated each iteration. The DNNs are trained by minimizing the mean squared error between the current and target Q-value using gradient descent.

The target Q-value in DQN is determined using the DNN with parameters $\boldsymbol{\theta}'$ that a greedy action selection and action evaluation. The action selection and evaluation for the target Q-value are both based on the DNN with parameters $\boldsymbol{\theta'}$. This could overestimate the Q-value in the learning process. Therefore, Van Hasselt et al. (2016) proposed the double-DQN (DDQN) algorithm, where the action selection and evaluation are decoupled. The target Q-value in DQN and DDQN are calculated by, respectively:

$$y^{DQN} = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \boldsymbol{\theta'}), \tag{3.30}$$

$$y^{DDQN} = r + \gamma Q\Big(s', \arg\max_{a' \in \mathcal{A}} Q(s', a'; \boldsymbol{\theta}); \boldsymbol{\theta'}\Big). \tag{3.31}$$

3.5.2 Policy-based methods

Contrasting to value-based methods, policy-based methods learn a parameterized policy that does not use a value function to select actions. The parameterized policy $\pi_{\boldsymbol{\theta}}(a|s) \in (0,1)$ indicates the probability of selecting action a given in state s. REINFORCE is policy gradient (PG) method, introduced by Williams (1992), which is a Monte Carlo method since it relies on the full trajectory. The policy parameters using REINFORCE are updated by:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla_{\boldsymbol{\theta}_t} \pi_{\boldsymbol{\theta}_t}(A_t|S_t)}{\pi_{\boldsymbol{\theta}_t}(A_t|S_t)} \\ &= \boldsymbol{\theta}_t + \alpha G_t \nabla_{\boldsymbol{\theta}_t} \ln \pi_{\boldsymbol{\theta}_t}(A_t|S_t). \end{aligned} \tag{3.32}$$

This update equation is very intuitively as it increases the current parameters in the direction that most increases the probability of taking action A_t again on future visits to state S_t proportional to the return, i.e., it updates the parameter vector in the direction corresponding to the actions that yield the highest total expected reward. Secondly, the parameters are updated inversely proportional to the probability of selecting an action. This causes the agent to keep exploring for other high-yielding actions, instead of favoring frequently selected actions (which then ultimately might not even yield the highest reward anymore).

To reduce the variance of the gradient, one can subtract a baseline from the return in REINFORCE. The policy parameters using REINFORCE with baseline are updated by:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\Big(G_t - b(S_t)\Big)\nabla_{\boldsymbol{\theta}_t} \ln \pi_{\boldsymbol{\theta}_t}(A_t|S_t). \tag{3.33}$$

For a detailed overview of many policy gradient methods, the reader is referred to Weng (2018).

3.5.3 Actor-critic methods

Konda and Tsitsiklis (2000) proposed the actor-critic (AC) methods, which combine the benefits of both value-based and policy-based methods. The actor refers to the learned policy and the critic refers to the learned value function. Hence, AC methods consist of two models: the critic that updates the value function parameters and the actor that updates policy parameters in the direction implied by the critic.

In the advantage actor-critic (A2C) method, the critic learns the advantage value, which indicates how much better a specific action is than the average of all actions from that state.

Mnih et al. (2016) proposed the asynchronous advantage actor-critic (A3C) method, in which the multiple agents are trained in parallel on multiple instances of the environment. Therefore, the agents are likely to explore different parts of the environment and experiences are less likely to be correlated. Therefore, it is not needed to keep an experience replay buffer and sample from it, which reduces the training time.

For a detailed overview of more actor-critic algorithms, the reader is referred to Weng (2018).

4 Solving routing problems using machine learning

In this section, the reviewed literature on machine and deep learning models for vehicle routing problems is described. First, categories to classify the algorithmic structures are introduced. The second subsection describes the literature on static routing problems. The third and fourth subsection describe the literature on dynamic routing problems. Finally, a concluding subsection provides several overviews of the reviewed literature.

4.1 Categorization of algorithmic structures

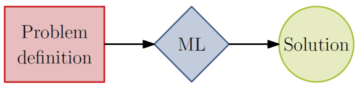
Bengio et al. (2018) surveyed the use of machine learning to solve combinatorial optimization problems and categorized the learning methods and algorithmic structure. They differentiated between three types of algorithmic structures (independent from the learning methods (un)supervised and reinforcement learning). It is mentioned that the distinctions made are sufficiently general such that the contributions of machine learning to combinatorial optimization can overlap. The three algorithmic structures are categorized as:

- end-to-end learning (Figure 8a):
  machine learning is utilized to output solutions directly from the input. In some end-to-end learning-based methods, the solution is generated with a constructive approach, where the solution is constructed from scratch by sequentially connecting the last visited node to another node in the next step.

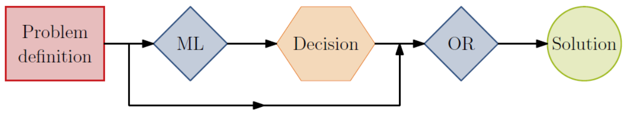
- learning to configure algorithms (Figure 8b):
  combinatorial optimization algorithms are provided with more information due to machine learning methods.

- machine learning alongside optimization algorithms (Figure 8c):
  machine learning methods guide combinatorial optimization algorithms repeatedly throughout the solution generation. Generally, these types of algorithms generate the solution by

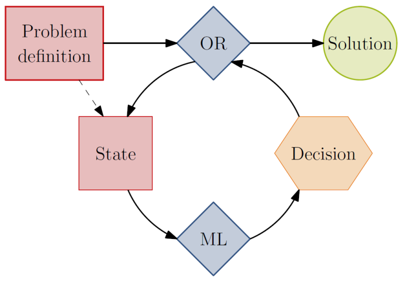
improving an initial solution iteratively, where the improvement operators are local search heuristics guided by ML. This machine learning structure is also called ‘learning to search’.



(a) End-to-end learning (E2E) illustrated.



(b) Learning to configure (L2C) algorithms illustrated.



(c) Machine learning alongside optimization illustrated, also called learning to search (L2S).

Figure 8: Three types of machine learning structures. Images retrieved from Bengio et al. (2018).

The reviewed literature on vehicle routing problems, covered in the following subsections, is first categorized as static or dynamic. The subsections for static routing problems are categorized by the algorithmic structures with subparagraphs for the routing variants. It must be noted that some literature studied multiple routing variants, but each work always has one routing variant that is most focused on. Additional studied variants of the routing problem by the authors are always mentioned. The dynamic routing problem is split into two stages, the dispatching and routing stage according to Zong et al. (2021). The dispatching stage involves the assignment of customers to vehicles (order matching) and the strategic repositioning of vehicles (fleet management). Thereafter, the best route for each vehicle to serve the assigned customers is determined in the routing stage. Despite these two stages, they are not rigidly separated. However, this categorization helps to identify the most important challenges in each stage. The dispatching stage is characterized by a high supply-demand ratio, meaning that the main objective is to optimize the assignment of demand to supply. Generally, the optimal in-loop routing is computationally cheap to determine, since the service loop of a vehicle contains few demand requests, such as in taxi or carpooling services. The routing stage is characterized by a low supply-demand ratio, meaning that each vehicle must serve many demand requests. Hence, the in-loop routing is computationally more expensive due to the NP-hard nature of VRPs. Unless otherwise stated, in this section ‘outperformed’ or ‘provided better results’ refers to outperforming in terms of the objective value.

4.2 Static routing

Several machine learning approaches for static routing variants are presented in this section. The literature for single-vehicle routing problems is discussed first, followed by the literature for multi-vehicle routing problems.

4.2.1 End-to-end learning (constructive)

The end-to-end learning approaches are categorized into single-vehicle and multi-vehicle routing problems. The two different approaches have in common that the problem is formulated as an MDP, that the solution is generated constructively, and that the vehicles are considered as agents. Starting from scratch, the routes are constructed by sequentially selecting the next node (customer or depot) to visit. In the machine learning approaches that solve single-vehicle problems end-to-end, e.g., (Hottung et al., 2021; Kool et al., 2019; Kwon et al., 2020; Li et al., 2021c; Nazari et al., 2018; Peng et al., 2019), only one vehicle is active at any time. The vehicle visits the depot multiple times to construct multiple routes, where the number of routes is not defined a priori. In the machine learning approaches that solve multi-vehicle problems end-to-end, e.g., (Falkner and Schmidt-Thieme, 2020; Li et al., 2021b; Zhang et al., 2020a), there are multiple vehicles active at the same time.

4.2.1.1 Single-vehicle routing problem

In this paragraph, several machine learning approaches to solve the static single-vehicle routing problem will be presented. From 2018, the research literature on machine learning for the VRP has rapidly expanded. Nazari et al. (2018) were the first to apply an attention mechanism to solve (among other COPs) the VRP. Although they proposed a model that also suits dynamic VRPs, it will still be presented in this section for two reasons. First, their main focus was on the static VRP. Secondly, it forms the basis for many other machine learning approaches to static VRPs. Hence, it is useful to introduce the model of Nazari et al. (2018) first.

All models in this section have several things in common. First of all, all models (except Nazari et al. (2018) consider static single-vehicle routing problems. Secondly, most approaches use concepts or ideas from others, i.e., they adapt or extend models from others in the research community. Therefore, it is convenient to present them in the same section. Thirdly, the same data distribution for instances is often used to evaluate the models. This makes it possible to easily compare model performances in terms of route length. Comparing the run times for training and testing the models is more difficult since computer specifications and the degree of parallelization heavily affect the run times. Lastly, the main focus of Nazari et al. (2018), Kool et al. (2019), Peng et al. (2019), Kwon et al. (2020) and Hottung et al. (2021) is on the CVRP.

CVRP

The first who applied an attention mechanism to solve the VRP were Nazari et al. (2018). They proposed models to solve the TSP, the CVRP, the CVRP with split deliveries (CVRPSD), and the dynamic and stochastic customer CVRP (DSCCVRP). They assumed a setting with only one vehicle and one depot. Customer and depot locations were uniformly generated in a unit square $[0,1] \times [0,1]$ with discrete customer demands randomly sampled from $\mathcal{U}(1,9)$ (uniformly distributed with lower and upper bounds 1 and 9, respectively). Their objective was to minimize the total tour length while ensuring that all demand is satisfied. Their study is a generalization of the work of Bello et al. (2016), who used the PN with an actor-critic algorithm to solve the TSP. In contrast to Vinyals et al. (2015b), Bello et al. (2016) used reinforcement learning instead of supervised learning to solve the TSP. In the TSP, a customer should only be visited once. Bello et al. (2016) addressed this by introducing additional computation steps (called glimpses proposed by Vinyals et al. (2015a)) to mask nodes already visited before each decoder step. The encoder in Nazari et al. (2018) calculates new attention vectors to determine the relevance of each node given a partially constructed tour at each time step. Their decoder determines a probability distribution over all possible nodes (masking some infeasible nodes). For the CVRP, a time step corresponds to a vehicle arrival at a node. For the DSCCVRP, a time step corresponds to either a new incoming customer or a vehicle arrival at a node. RNNs are especially useful for tasks where the input order is essential, e.g with text translation tasks. For the VRP, any permutation of the input nodes is identical to the original input. Therefore, Nazari et al. (2018) omitted the RNN encoder and created the node embeddings simultaneously, which allows for parallelization. The decoder outputs a probability distribution for the next node to visit. They used two different decoders, a greedy decoder (this model is named RL-greedy) and a beam search decoder (RL-BS($\cdot$), where ($\cdot$) indicates the beam width). Nazari et al. (2018) trained their models using on-policy actor-critic RL and A3C RL for the CVRP and DSCCVRP, respectively. After training the model for 13.5 hours, their model outperformed Google's OR-Tools in more than 60% and several heuristic algorithms in more than 99% for $n = 50$ and 100 (number of customers) in terms of total route length. Solving a single test instance required less than 0.5s.

Kool et al. (2019) use their model to solve several combinatorial optimization problems: TSP, CVRP, CVRPSD, orienting problem (OP), prize collecting TSP (PCTSP), and stochastic prize collecting TSP (SPCTSP). Their encoder architecture is inspired by the Transformer model but without the positional encoding such that the node embeddings are invariant to the input order. In contrast to the Transformer architecture, Kool et al. (2019) used different weight parameters for each layer (similar to the GAT model of Veličković et al. (2018)). Their decoder sequentially produces a probability distribution over feasible nodes. During training, a greedy decoder is used and the policy is trained using REINFORCE with baseline. The baseline is the total route length of a greedy roll-out of the best policy so far (which parameters may be updated after every epoch).

During testing, Kool et al. (2019) used a greedy decoder (AM-greedy) and a sampling decoder (AM-sampling). The sampling decoder samples 1280 solutions of which the best result is reported. Results based on 10000 test instances are compared to the LKH3 algorithm (Helsgaun, 2017), which is mentioned as the state-of-the-art solver, and Nazari et al. (2018) (which report the average tour length over 1000 test instances). For all tested CVRP sizes, LKH3 provided shorter route lengths than AM-greedy or AM-sampling but takes significantly more time to solve the test instances. AM-greedy and AM-sampling outperformed Nazari et al. (2018). Kool et al. (2019) noticed that their model constructs routes in which the order of stops in individual routes is suboptimal. They suggested that additional optimization (e.g., beam search, local search, and improving individual routes with a TSP solver) will improve the solution.

The AM model of Kool et al. (2019) used a static encoder-decoder architecture, i.e., the node embeddings (attention vectors) are fixed over time. The static embeddings are calculated once per instance and allow multiple solutions to be sampled per instance. Similar to Nazari et al. (2018), Peng et al. (2019) used a dynamic encoder-decoder architecture, where the attention vectors are updated during the solution construction. Since the model is not tested on a dynamic VRP variant, it is presented in this section for static VRPs. However, this model may be applied to dynamic VRPs. In the AM model (Kool et al., 2019), the attention vector of a node may assign high attention to infeasible nodes, which are later masked by the decoder. Since the attention weights are processed using softmax, this downgrades the attention to feasible nodes. Therefore, Peng et al. (2019) adjusted the AM such that only feasible nodes are considered by the encoder. Their model (AM-D) updates the attention vectors every time the vehicle returns to the depot. The policies are trained using REINFORCE with a greedy roll-out baseline. In addition to just a greedy decoder model, they also applied a 2-opt local search to improve the solution. Their results (averaged over 1000 test instances) indicated an improvement over AM-greedy, but not over AM-sampling (which uses a more extensive search strategy).

All the aforementioned constructive approaches use search strategies like sampling and beam search, which offer limited search guidance. Constructive approaches can also utilize an active search procedure, proposed by Bello et al. (2016), which adjusts the parameters of a trained model to a single instance during test time. Hottung et al. (2021) proposed efficient active search (EAS) that adjusts a subset of the model parameters and solved the TSP, CVRP, and the job shop scheduling problem (JSSP). Three different EAS implementations are proposed that adapt the model to a single instance by adjusting: 1) the (normally static) encoder embeddings, (2) the weights of decoder added instance-specific residual layers, and (3) the lookup table parameters that have a direct impact on the probability distribution output by model. EAS is not a complete model on its own, but can act as an extension to another model. Hottung et al. (2021) implement their model for the POMO model of Kwon et al. (2020). The model of Kwon et al. (2020), called policy optimization with multiple optima (POMO), is a constructive end-to-end approach and very similar to the AM model of Kool et al. (2019). The first main difference is that POMO explores solutions for an instance by selecting each node as a starting node (first node to visit after the depot), resulting in n different solutions. Secondly, in POMO the baseline used by REINFORCE is the average cost of a greedy roll-out of all n solutions per instance. Hottung et al. (2021) evaluate the performance of three POMO model variations: POMO-greedy (which uses a greedy action selection), POMO-sampling (which randomly samples $200 \cdot 8 \cdot n$ solutions and selects the best one), and POMO-AS (which uses active search of Bello et al. (2016) during testing). The first EAS model of Hottung et al. (2021), called EAS-Emb, updates instance-specific encoder embedding parameters. The second proposed strategy, called EAS-Lay, updates an instance-specific added layer to the trained model during the search. EAS-Emb and EAS-lay are on-policy DRL (policy-based) methods that use a loss function with an RL component and an imitation learning (IL) component. The third proposed strategy, called EAS-Tab, uses a lookup table to update the policy and is an on-policy Q-learning (value-based) method. The model is tested on the data distribution of Nazari et al. (2018) and the more challenging data distribution of Uchoa et al. (2017).

PDP

Li et al. (2021c) proposed a DRL approach for the pickup and delivery problem (PDP). The encoder-decoder architecture is similar to Kool et al. (2019), but with six additional heterogeneous attention layers. These layers learn the pairing and precedence relationship for pickup and delivery nodes. A masking scheme is used to guarantee a feasible solution. The problem is formulated as an MDP, with the (single) vehicle as an agent which selects the next node to visit. The objective was to minimize the travel time. The policy is trained with an actor-critic method. The model is compared to an exact solver, heuristics, and machine learning approaches. Results indicated that their model (for small problem sizes) is competitive with the exact solver, but much with much faster computation time. For somewhat larger problem sizes, their model provided the best performance for their objective.

4.2.1.2 Multi-vehicle routing problem

In end-to-end solved multi-vehicle routing problems, multiple vehicles are active at the same time, i.e., multiple routes are constructed simultaneously. The multi-vehicle routing problem is solved using ML by, e.g., Falkner and Schmidt-Thieme (2020), Li et al. (2021b), and Zhang et al. (2020a). Li et al. (2021b) and Zhang et al. (2020a) formulated the problem as single- and multi-agent MDP, respectively. Li et al. (2021b) studied the multi-CVRP with heterogeneous vehicles. Falkner and Schmidt-Thieme (2020) and Zhang et al. (2020a) studied the multi-CVRP with time windows.

CVRP
Li et al. (2021b) proposed a DRL constructive approach for the multi-CVRP with heterogeneous vehicles (different capacities and speeds). They considered both min-max and min-sum objectives. The former aims to minimize the longest travel time and the latter aims the minimize the total travel time. They used a Transformer encoder-decoder architecture similar to Kool et al. (2019), in which the encoder embeddings are calculated only once at $t = 0$. There is one agent (the fleet manager) that, at each time step, selects a vehicle and the next destination for that vehicle. Separate decoders select the vehicle and the next node to visit. The objective functions are minimized using REINFORCE with baseline, where the baseline is a greedy roll-out of the current best policy. The models are tested on an adapted data distribution of Nazari et al. (2018) and Uchoa et al. (2017) and compared to several heuristic and DRL methods. Their model provided superior results to most heuristics and DRL approaches and is only outperformed by a few percent by the state-of-the-art heuristic of Christiaens and Vanden Berghe (2020), which required significantly more time to solve the instances ($\pm 100\times$).

CVRPTW
Falkner and Schmidt-Thieme (2020) proposed a DRL approach for the multi-CVRP with time windows. Their joint attention model for parallel route construction (JAMPR) is an extension of Kool et al. (2019). Besides encoding the nodes, two additional encoders are used to embed the information of current routes and vehicles at each time step. The decoder selects which node to add to which route. A greedy and sampling approach is used during testing. They considered soft and hard time windows, where a violation of soft time windows induces a penalty. Their objective was to minimize the total route length plus penalties. The model parameters are optimized using REINFORCE with baseline, where the cost of a greedy roll-out policy is used as the baseline. They tested JAMPR for the CVRP on the data distribution of Nazari et al. (2018) and compared results to several reinforcement learning approaches. JAMPR performed slightly worse than AM, but the results indicated that JAMPR performs relatively well for the CVRP although intended for the CVRPTW. JAMPR with soft and hard time windows is evaluated on a dataset of Solomon (1987) and provided in most cases better results than their used benchmarks.

Zhang et al. (2020a) proposed a DRL approach for the MCVRPTW (with soft time windows). Their multi-agent attention model (MAAM) is an end-to-end constructive approach. Contrasting to Falkner and Schmidt-Thieme (2020), each vehicle is considered as an agent, which selects the next node to visit. The encoder is inspired by Kool et al. (2019) and is similar for all agents. The homogeneous agents perceive each other's state and then select actions sequentially within the same time step. During the training, they used a sampling decoding process, while during testing they used a greedy decoder. Their objective was to minimize the total travel costs, which is defined as a combination of total route length and a penalty term for time window violation. The model parameters are optimized using REINFORCE with baseline, where the cost of a greedy roll-out policy is used as baseline (similar to Kool et al. (2019)). MAAN is tested on instances with 20, 50, 100, and 150 customers and compared to two heuristic methods and Google's OR-tools. For most scenarios, MAAN provided the shortest route length in significantly less run time compared to the benchmarks.

4.2.2 Learning to configure

The aforementioned machine learning approaches for the single-vehicle routing problem all used reinforcement learning. Contrasting to this, Kool et al. (2021) used supervised learning to guide a dynamic programming (DP) algorithm. Although, the solution is constructed by sequentially adding a node to the partial solution, the model of Kool et al. (2021) is considered as learning to configure. This is because the ML algorithm is used once and the solution is constructed by a DP algorithm thereafter. According to the categorization of Bengio et al. (2018), this structure is learning to configure (Figure 8b).

CVRP

Kool et al. (2021) proposed the Deep Policy Dynamic Programming (DPDP) model, which combines a dynamic programming (DP) algorithm with supervised learning, to solve the TSP and the single-vehicle routing problem. DP is known as a solution method for the VRP, but scales badly with the number of customers. Therefore, the problem is first preprocessed by a graph convolution network into a heatmap of promising edges, which is proposed by Joshi et al. (2019). Thereafter, a solution is constructed by a restrictive dynamic programming algorithm (Gromicho et al., 2012) using beam search. The heatmap indicates the edges that have a high probability to be part of the best solution and reduces the number of possible routes by removing edges below a certain threshold. The target for each training instance is determined with the LKH3 solver. Models with different beam search sizes are used (10k, 100k, and 1M) and tested on the data distribution of Nazari et al. (2018) and the more challenging data distribution of Uchoa et al. (2017). DPDP-1M outperformed LKH3 in terms of total route length but required more time to solve a single instance (± 18s vs 5s).

4.2.3 Learning to search (improvement)

Whereas constructive approaches start from scratch, learning to search ML approaches to solve VRPs iteratively improve an initial solution. This is sometimes referred to as learning a heuristic since local search heuristics are guided by ML algorithms to improve the current solution. Generally, it is more challenging for these methods to solve richer VRPs (VRPs with constraints), since modifying solutions while maintaining feasibility is harder for these methods. This may be the reason why recent learning to search approaches to solve VRPs, e.g., Chen and Tian (2019); Hottung and Tierney (2020); Lu et al. (2020); Sultana et al. (2021), all studied the CVRP.

CVRP

The NeuRewriter model, proposed by Chen and Tian (2019), learns a policy to select heuristics and rewrite local parts of the current solution and applied it to solve expression simplification tasks, the JSSP, and the CVRP. The initial solution is constructed in the following way: starting at the depot, the vehicle visits the nearest node which either is an unrouted node whose demand can be completely satisfied, or is the depot node. This procedure is repeated until all demands are satisfied. The initial solution is incrementally improved (until convergence) by applying the 2-opt heuristic (an interroute method, where every node can swap with another node in the route). The model consists of a region-picking and rule-picking policy. The former is learned by using a parameterized action-value function by fitting it to the return sampled from the current policies. The latter is trained using the A2C algorithm with the learned action-value function as a critic. Results of the NeuRewriter model are compared to Nazari et al. (2018), Kool et al. (2019), Google's OR-tools, and several classic heuristics studied by Nazari et al. (2018). NeuRewriter provided for all problem sizes the shortest average tour length compared to the other machine learning approaches, but is not better than the LKH3 method.

Wu et al. (2020) extends the model of Chen and Tian (2019) by integrating the region-picking and action-picking policy into one policy. In this single-agent MDP, the action is to select a pair of nodes to swap. They used an actor-critic method, where the actor is a policy trained with an attention mechanism (similar to the Transformer) and the critic estimates the return of each state. They tested the model (called LIH) for the TSP and CVRP, where the CVRP model is tested on data distributions of Nazari et al. (2018) and Uchoa et al. (2017). Wu et al. (2020) compared their model with T = 1000, 3000, and 5000 iterations against LKH3, Google's OR-Tools, Chen and Tian (2019) and Kool et al. (2021), where the latter three were outperformed by LIH.

The results of the Chen and Tian (2019) and Wu et al. (2020) are very dependent on the initial solution as nodes are only swapped interroute. Therefore, Hottung and Tierney (2020) proposed to use destroy operators to destroy the solution, which is then repaired by repair operators. The model of Hottung and Tierney (2020) uses a large neighborhood search (LNS, see section 2.2.3) metaheuristic that guides the search and learns heuristics to repair incomplete solutions. Their model is called the neural large neighborhood search (NLNS) and iteratively tries to improve the initial solution by applying destroy and repair operations. An important feature of NLNS is that the repair operation complexity is mostly independent of the problem size, allowing the NLNS to efficiently deal with larger problem sizes. This is because only the relaxed nodes need to be repaired. Similar to the ALNS (Ropke and Pisinger, 2006), the NLNS model uses several destroy and repair operators. The initial solution is created using a greedy procedure, which adds the closest feasible node to the vehicle route. In case the demand of the closest node cannot be completely satisfied, the vehicle returns to the depot and a new route is started. This procedure is repeated until all demand is satisfied. A destroy operator determines which and how many nodes to relax (remove) from the current solution. Two destroy operator types are used: point-based destroy (which relaxes the closest nodes to a random point from all tours) and tour-based destroy (which relaxes the closest used edges to a random point). Both types of destroy operators can be set to different degrees of construction, indicating the percentage of nodes to remove. Whereas the destroy operators are random, for each destroy operator a repair operator is parameterized through a neural network. The repair operator sequentially repairs a destroyed solution by connecting, at each time step, a randomly selected incomplete tour (which can contain one or multiple nodes) endpoint to another node (depot or other incomplete tour endpoints). The repair operator uses an attention mechanism (similar to Nazari et al. (2018), but without the RNN decoder) to determine the probability distribution over the possible nodes to be connected. REINFORCE with baseline is used to update the model parameters. Two types of models with different search procedures are used: batch search (solves several instances in parallel) and single instance search (solves one instance by simultaneously applying destroy and repair operations to several solutions of the instance). Hottung and Tierney (2020) applied their model to solve the CVRP and CVRPSD. Results of the NLNS using batch search (denoted as NLNS-batch) are obtained using 10000 test instances and compared to Nazari et al. (2018) and Kool et al. (2019), which were both outperformed. Results for the CVRP of the NLNS using single instance search (denoted as NLNS-SIS) are compared to UHGS (Vidal et al., 2014), LKH3, and LNS. These methods are evaluated on the instances of Uchoa et al. (2017). The datasets are organized into 17 groups of 20 instances (with the same distribution per group) with the number of customers ranging from 100 to 297. For all instance groups, UHGS provided the best performance, with gaps of 0.04% to 3.10% to NLNS-SIS and with gaps of 0.08% to 3.38% LKH3. NLNS-SIS outperforms LNS on all instance groups by 0.47% to 10.93%. Performance of NLNS-SIS and LKH3 is comparable, with NLNS-SIS outperforming LKH3 on 11 of 17 instance groups.

Compared to the improvement approaches of Chen and Tian (2019) and Hottung and Tierney (2020), a much larger solution space is explored by Lu et al. (2020) resulting in better solution qualities for the CVRP. Their model is called learn to improve (denoted by L2I) consists of improvement and perturbation operators. Starting from an initial random solution, the reinforcement learned improvement operator improves the solution. If for six improvement steps the improvement controller cannot realize a cost reduction (i.e., a local optimum is reached), the solution is perturbed (randomly selected) and a sequence of improvement operators is applied again. At each iteration, the improvement controller tries to improve the solution (initial solution or perturbed solution) by applying one of several intraroute and interroute improvement operators. The improvement controller outputs a probability distribution over all the actions given a certain state (containing features from the problem instance, the current solution, and the history), calculated using the Transformer architecture. The actions are selected using ϵ-greedy, where with a probability of 0.05 a random action is selected. An improvement iteration consists of a sequence of actions and is restarted 40000 times. Six different policies are used, where the final solution is chosen as the best solution out of all discovered solutions. The policies are trained using REINFORCE with baseline. Lu et al. (2020) compared their model to Google's OR-Tools, LKH3, Nazari et al. (2018), Kool et al. (2019), and Chen and Tian (2019). For all problem sizes, L2I provided the shortest tour length compared to other methods and is the first to outperform LKH3 (averaged over 2000 instances) on the data distribution of Nazari et al. (2018).

The learning to guide local search (L2GLS) model of Sultana et al. (2021) is very similar to L2I (Lu et al., 2020), but instead of a set of perturbation operators uses penalty terms to escape local optima. L2GLS is tested for the TSP and CVRP up to 1000 nodes. L2GLS improves an initial random constructed solution by applying reinforcement learned improvement operators. The policy defines a probability distribution over actions given a certain state, where actions are selected according to ϵ-greedy (with $\epsilon = 0.05$). Each action consists of selecting a local search heuristic and parameters of the LSH. The parameters of an LSH operator are the nodes or edges to be used by the LSH operator. The state is defined by problem-specific (node coordinates and demand) and solution-specific features (neighboring nodes and their distances, recently taken actions and their effects, and remaining vehicle capacity in the route of a customer). These features are embedded and processed by an attention model (Transformer architecture), which produces the action probabilities. After six non-improving applied actions, a penalty term to the objective function is applied to escape the local optimum. This penalty term penalizes high-cost solution features, which are the high distances between consecutive nodes in a tour, and encourages the agent to improve these. After finding 40000 local optima, the best-found solution is selected. The policy is trained using REINFORCE with baseline. Results are obtained by averaging over 1000 test instances and are compared to (among others) LKH3, Nazari et al. (2018), Kool et al. (2019), Kwon et al. (2020), Chen and Tian (2019), and Lu et al. (2020). L2GLS outperformed all benchmark models and achieved state-of-the-art performance for the CVRP with data distributions of Nazari et al. (2018).

4.3 Dynamic dispatching

Urban environments often have a public transportation network that is focused on mass transportation. Public transportation services often are not able to bring people to their final destination, which is known as the 'last-mile problem'. Taxis can overcome this problem and traditionally drive around to find customers, i.e., they are not coordinated. Mobility-on-demand (MoD) transportation services, such as Uber, Lyft, and DiDi, offer huge potential to increase transportation efficiency by utilizing a centralized decision-making platform.

In this section, recent literature for three subtopics of MoD will be reviewed, with a focus on model-free reinforcement learning. First, solution methods for the order matching problem, where supply must be assigned to demand. Second, solution methods to increase transportation efficiency by repositioning vehicles (fleet management). Lastly, solution methods that integrate the order matching stage and fleet management.

4.3.1 Order matching

Order matching is the process of assigning demand to supply. In this section, two order matching for two MoD variants, where on-demand customers (orders) need to be assigned (matched) to vehicles. In on-demand ride-hailing problems, vehicles can only serve one customer at the same time. However, a group of people can be considered as one customer, as long as they have the same pickup and destination location. So more specifically, in on-demand ride-hailing problems, the vehicle cannot be matched with additional customers before it has completed an order (reached the destination location). The term 'on-demand' refers to dynamic incoming orders over time and is dropped in the remainder of this section since this entire section is about dynamic order matching. Contrasting to ride-hailing problems, in ride-sharing problems the vehicles may serve multiple customers (with different pickup and/or destination locations) simultaneously. Another term for ride-sharing is carpooling. In ride-sharing problems, it needs to be determined how many and which customers to serve simultaneously. Similar to ride-hailing problems, ride-sharing problems can be considered as order matching problems. The terms 'ride-hailing' and 'ride-sharing' are used interchangeably in the literature. In this report, ride-hailing and ride-sharing are solely considered as the above-described definitions.

Order matching for ride-hailing problems can be solved by the traditional Khun-Munkres (KM) bipartite matching algorithm (Munkres, 1957), but is computationally very costly for online problems and large problems. A greedy assignment method of the nearest vehicle, such as in Liao (2003), is computationally less costly, but is myopic (does not consider long-time rewards). Predicting demand based on historical data, such as in Sungur et al. (2010) Moreira-Matias et al. (2013), Tong et al. (2017), and Ke et al. (2017), improves over greedy assignment, but remains challenging in continuously updating environments due to the complex matching problem.

Order matching for ride-sharing problems can be solved by constrained programming, e.g., Ma et al. (2014) and Alonso-Mora et al. (2017), or by utilizing travel time estimation, e.g., Zhan et al. (2013), however, these methods do not consider the impact of the matching on future rewards. Other works used ADP which are model-based RL methods, Ulmer (2017) provides an overview of ADP solution methods or dynamic routing problems. Shah et al. (2020) indicated the limitations of traditional ADP and proposed an NN-based ADP algorithm, which significantly outperformed Alonso-Mora et al. (2017) for real-world ride-sharing experiments.

Intuitively, the order matching process can be formulated as an MPD, for which DRL methods recently have been leveraged. The advantage of DRL approaches is that they can learn the underlying probability distribution of the occurrence of dynamic events without explicitly specifying it. Additionally, DRL makes it possible to learn the long-term influence of actions by considering both spatial and temporal dynamics. The order matching problem is generally solved by accumulating the incoming orders and available vehicles for a fixed time window and matching them at the end of the time window.

4.3.1.1 Ride-hailing

There seems to be no consensus in the reviewed literature about single-agent or multi-agent reinforcement learning (MARL). Even if each vehicle is treated as an agent, some works, e.g., Wang et al. (2018) and Li et al. (2019), considered this as a single-agent setting if the agents are trained homogeneously. Other works, e.g., Xu et al. (2018) and Shi et al. (2019) considered homogeneous agents as a multi-agent setting. 'Homogeneous' refers to that the agents' state space, action space, and reward function are the same. The advantage of homogeneous MARL compared to heterogeneous MARL is that the action-value function can be trained for one agent and applied to all agents. In this report, the approach is considered to be multi-agent if there are multiple agents, regardless of whether they are homogeneous. In heterogeneous MARL, there are multiple agents which do not share the learned action-value function. The advantage of MARL compared to single-agent RL is that the state space can be reduced significantly, but comes at the cost of less coordination.

Homogeneous multi-agent methods
Works that used MARL with homogeneous agents are approaches are, e.g., Xu et al. (2018), Wang et al. (2018), Tang et al. (2019), and Shi et al. (2019).

Xu et al. (2018) proposed an ML approach for a ride-hailing platform with the objective to optimize the accumulated driver income (ADI). They define the matching process as an MDP, where the reward is based on immediate and future trip prices. In the proposed centralized multi-agent approach, each vehicle is considered an agent with a homogeneous state space, action space, and value function. The decision-making process for all agents is done in a centralized way. To capture the spatio-temporal patterns of supply and demand, an on-policy reinforcement learning algorithm determines the values for each spatio-temporal state. At each time step, the action for each vehicle is either to serve a customer or to stay idle (stay at location). The value (expected revenue of the vehicle) of being in a certain state, is used by an advantage function that calculates the expected gain of serving a specific order over the average value of that state. The advantage function is used by a KM algorithm to solve the matching process. The objective is to find the best actions for vehicles to maximize the platforms' profit, which corresponds to the aggregate profit of all vehicles. Therefore, the agents share the same policy that maximizes the aggregate action-value function. The policy is evaluated by using Q-learning with dynamic programming. The proposed method is evaluated on two simulator setups and a real-world environment, which showed promising results. This resulted in a real-world deployment of the method in DiDi Chuxing.

Wang et al. (2018) identified three main limitations of the approach of Xu et al. (2018). First, tabular methods (in general) do not generalize well beyond historical data, i.e., the agent has difficulties responding to real-time information. Second, each city requires its own model, despite that cities may share common properties. Third, the on-policy method requires a lot of evaluation-improvement iterations and is computationally very expensive. To overcome these limitations, Wang et al. (2018) proposed an off-policy deep reinforcement learning approach using DDQN (Van Hasselt et al., 2016) for transfer learning (Pan and Yang, 2009). Instead of sampling all actions, an action search strategy is used by the DDQN. The KM algorithm is used to solve the matching, where the value function is used to determine the edge weights of the bipartite graphs. Results indicated that the transfer learning model outperforms some benchmark models.

Tang et al. (2019) extend the approach of Xu et al. (2018), but formulate the order matching as a semi-MDP since the passenger assignments affect the state transitions over multiple time steps. They used memory-based neural networks for better feature representations and adapt the transfer learning approach of Wang et al. (2018). Besides long-term aggregate driver income, they also optimized for experiences from both sides (driver and customer). The matching process is solved by the KM algorithm, where the graph edges also are affected by a user experience term. However, this user experience term is not learned and the agents share the same value function. Hence, their approach is considered a homogeneous MARL method. The policy is evaluated using DDQN. Results on simulations indicated similar average performance but more consistent compared to Xu et al. (2018) and Wang et al. (2018).

If the ride-hailing service operates a fleet of electric vehicles (EV), the greenhouse gas emission can be reduced compared to gasoline vehicles. The challenge of EVs compared to gasoline vehicles is the requirement to relatively often recharge the battery. Shi et al. (2019) proposed an off-policy DRL algorithm for a fleet of EVs, with the objective to minimize the customer waiting time, electricity consumption, and operational costs. During the training phase, they used TD with an ϵ-greedy policy, whereas in the testing phase a greedy policy is used. Their method uses a centralized decision-making component, which coordinates the entire fleet and avoids conflicts, and a decentralized learning component through which homogeneous EVs (agents) can share their experiences. The problem is formulated as an MDP, where the actions are selected by the centralized decision-making component by utilizing an extension of the KM algorithm, proposed by (Bourgeois and Lassalle, 1971), that exploits the approximated value function to solve the matching problem. The state-value function is approximated by a deep neural network. The model is trained and tested on three cases: (1) a test case for a city center, (2) a test case where customers commute between two regions, and (3) a small-scale optimality test case for a single region. The model outperformed two benchmarks, which are a greedy algorithm (adapted from Chen et al. (2016) and Kang et al. (2016)) and the optimization algorithm of Bauer et al. (2018).

All the aforementioned homogeneous MARL approaches for the ride-hailing problem do not allow delayed matching. In other words, if there are sufficient idle vehicles to serve all new customers in the next time step, all new customers will be assigned. Some works do allow delayed matching or vehicles to reject customers, e.g., Ke et al. (2020) and Yang et al. (2021). Both customers and vehicles may benefit from delayed matching, since it may result in shorter pickup distances and shorter waiting times. Ke et al. (2020) and Yang et al. (2021) both considered each customer as an agent with demand for vehicles, instead of considering each vehicle as an agent with demand for customers. Ke et al. (2020) proposed several DRL approaches with state-values approximated by off-policy DQN, and on-policy A2C, PPO, and ACER. The matching process is formulated as a bipartite matching problem. Yang et al. (2021) proposed a DRL approach with a centralized decision-making process (the planning component). A many-to-many matching process utilizes the learned values. They took the vehicle's rejection behavior into account. The vehicle's ability to reject a customer is attractive for the suppliers.

Heterogeneous multi-agent methods
Contrasting to the MARL methods with homogeneous agents, Li et al. (2019) proposed a mean-field reinforcement learning (MFRL) approach for the ride-hailing problem. They stated that the homogeneous multi-agent approach is oversimplified and cannot completely model the complex interactions between vehicles and customers. Additionally, the centralized decision-making process may suffer from reliability issues, due to the potential single point of failure. To overcome these limitations, they adopted a mean-field approximation to capture the interaction between agents (vehicles) by taking an average action among neighborhoods. Each agent has its own critic that evaluates each agent's actions and updates their policies during training. The centralized critic (a DQN learned action-value function) is not used during testing, since agents then follow their own policy. Instead of using a matching algorithm that ensures a feasible solution, agents are allowed to immediately re-choose an order from an updated set of available customers when agents assign the same order.

Single-agent methods
Most works formulate a MARL framework for the ride-hailing problem. Contrary to that approach, Wang et al. (2019) proposed a single agent RL approach. The action-value function is approximated with restricted Q-learning, which guides a bipartite graph matching algorithm. The agent's (the platform's) state is the expected length of the time windows over which supply and demand are gathered. The actions of the agent are either to wait a time step or to stop the time window and assign the gathered customers to vehicles.

4.3.1.2 Ride-sharing

Jindal et al. (2018) proposed a DRL approach to predict the travel time and distance using NYC taxi data. Their objectives were to maximize transportation efficiency and minimize traffic congestion. They assumed that more served demand per trip results in more efficient transportation and less congestion. Therefore, the reward of vehicles is defined by the effective distance over a trip (more served demand per trip results in a higher reward). The action-value function is learned by a DDQN. The agents are considered as homogeneous, hence share the state space, action space, and action-value function. Results indicated that their model outperforms the prediction performance of Wang et al. (2014) and is more robust to outliers. Haliem et al. (2020) proposed a DRL approach where the agents (vehicles) can choose to assign customers to vacant seats or to reposition. Their objective is to minimize the supply and demand mismatch, the dispatch time, the repositioning time, and the number of vehicles and maximize fleet profit.

4.3.2 Fleet management

Traditionally, idle taxis drive around to find customers, which is a waste of fuel and causes extra congestion. Taxis may wait statically for a new assignment, but this is also not optimal for both customers and taxis. To tackle this problem, vehicles can be strategically repositioned. This is known as fleet management and improves transportation efficiency. Fleet management is an extensively studied problem, which traditionally used to be solved using supply and demand distributions, see e.g Yuan et al. (2012) and Qu et al. (2014). Some reinforcement learning approaches optimize the reposition strategy from the perspective of the single vehicle, e.g., Rong et al. (2016), Han et al. (2016) Wen et al. (2017), and Yu and Shen (2019). Model-based approaches of Rong et al. (2016) and Yu and Shen (2019) used value iteration-based dynamic programming to reposition the vehicle to optimize the long-term revenue.

Some model-free RL approaches used a fairly short time horizon, e.g., Han et al. (2016) and Wen et al. (2017). Both these works optimized an objective function from the perspective of a single-vehicle with episodes of length from being idle to completing an order and being idle again. Han et al. (2016) used Q-learning to maximize driver income and Wen et al. (2017) used DQN to maximize areawide service availability. With the increasing popularity of on-demand mobility services, vehicle repositioning strategies from the perspective of the entire fleet (the platform) with long-term objectives have also been more studied recently, e.g., Lin et al. (2018), Oda and Joe-Wong (2018), Liu et al. (2020), and Zhang et al. (2020b). All these four multi-agent DRL approaches considered both spatio-temporal and global supply and demand features, while the previously mentioned single-agent DRL approaches (Han et al., 2016; Wen et al., 2017) only considered local spatio-temporal and supply and demand information. Lin et al. (2018) developed DQN and A2C methods with a masking scheme for coordination. Oda and Joe-Wong (2018) used a heterogeneous fleet of vehicles (agents) that learn independent policies using DQN, which comes at the cost of coordination between vehicles but scales better compared to a model predictive control approach. Instead of considering a grid structure, Liu et al. (2020) divided the graph structure into zones by a road-connectivity aware clustering algorithm. The homogeneous agents are trained using DQN and take sequential actions. Their model outperformed, i.a., Lin et al. (2018), Oda and Joe-Wong (2018). Zhang et al. (2020b) used dueling-DQN (Wang et al., 2016) to train homogeneous agents, which actions are improved by another Q-learned agent.

4.3.3 Integrated dispatching

Fleet management can also be integrated in the order matching stage, which is studied in, e.g., Jin et al. (2019), Holler et al. (2019), Guo and Xu (2020), and (Liang et al., 2021). Jin et al. (2019) proposed a heterogeneous MARL approach utilizing DDPG (Lillicrap et al., 2015) for the integrated ride-hailing problem. They used a hierarchical structure, where each region cell is treated as a working agent that is coordinated by a manager agent (the district). To capture the impact on neighboring agents and learn the characteristics of each region and district, a multi-head attention mechanism is utilized. Vehicles in the same region are considered to be homogeneous and repositioning is done by introducing fake orders. Their model outperformed the order matching ride-hailing approaches of Xu et al. (2018), Wang et al. (2018), and Li et al. (2019) in terms of ADI and order response rate (ORR). Contrasting to the end-to-end approach of Jin et al. (2019), Guo and Xu (2020) used DDQN for the fleet management and the KM algorithm to solve the matching problem. They utilized a convolutional neural network (CNN) framework for the integrated ride-sharing problem to maximize the quality of service (QoS) minus the costs. While Jin et al. (2019) and Guo and Xu (2020) used a grid-based setting, Liang et al. (2021) preserved the graph-based topology of the supply and demand distribution. They proposed a homogeneous MARL framework for the ride-sharing problem. The matching is done by the KM algorithm that exploits constrained programming framework to filter out infeasible assignments and enhances vehicle cooperation. The edge weights for the KM algorithm are determined with TD-based DQN. Their model outperformed several adapted DRL frameworks in terms of ADI and ORR. The order of magnitude for the computation time per decision is milliseconds.

4.4 Dynamic routing

In the dynamic VRP, some information is only revealed during the execution of the routes. This could, e.g., be new incoming customers, a change in demands, or variable service and travel times. It is not obvious that a new request is accepted, since it may be too expensive or not feasible to serve the request. Whether a request is accepted or denied is referred to as service guarantee, introduced by Hentenryck and Bent (2006).

An initial solution to a static VRP may be adjusted at dynamic events by, e.g., inserting the new customer into existing routes or by solving the new problem instance. However, this approach for dynamic VRPs is not optimal, especially not if degree of dynamism (DoD) is high (Lund et al., 1996). The reason for this is that the underlying probability distribution of dynamic events is not taken into account. The degree of dynamism δ is defined by ratio of between dynamic and total requests: $\delta = \frac{n_d}{n_{tot}}$. To also include the time aspect, Larsen (2000) proposed the effective DoD for both without and with time windows: $\delta^e = \frac{1}{n_{tot}} \sum_{i \in \mathcal{R}} \frac{t_i}{T}$ and $\delta^e_{TW} = \frac{1}{n_{tot}} \sum_{i \in \mathcal{R}} \left(1 - \frac{l_i - t_i}{T}\right)$, respectively, where $\mathcal{R}$ is the set of requests, t_i the disclosure time of request $i \in \mathcal{R}$, T, the length of the planning horizon, and l_i the end of the time window.

The main difference between solving dynamic routing and dispatching problems is the supply/demand ratio (as already mentioned in the introduction of section 4). The dispatching stage is often associated with the transportation of people and the routing stage is often associated with the transportation of goods. In the latter case, vehicles typically should be loaded with goods prior to the delivery and once a delivery request has been accepted, it generally cannot be canceled afterward.

Contrasting to dynamic routing problems, dynamic dispatching problems (e.g., ride-hailing and -sharing) may primarily be concerned with determining the next customer, while for dynamic VRPs it is required to determine the entire route plan. Ulmer et al. (2017) proposed route-based MDP as a framework for dynamic VRPs, where dynamic events cause updates in the entire route plan. This route-based MDP approach allows to (1) determine the value of the projected route which helps to consider future requests and (2) communicate planning, which is important, e.g., (B2B) package or food delivery services.

Solving dynamic and deterministic VRPs used to be done with periodic re-optimization or continuous re-optimization approaches (Pillac et al., 2013). In the former approach, the static problem corresponding to its current state is solved when dynamic data is revealed reveal or at fixed time intervals. In the latter approach, the problem is solved when dynamic data is revealed using the information of good past solutions.

Solving dynamic and stochastic vehicle routing problems used to be done with sampling or stochastic modeling approaches (Pillac et al., 2013). In the former approach, scenarios are sampled using the stochastic probability distributions, where after the static and determined scenarios are solved. In the latter approach, stochastic knowledge is analytically integrated by solution methods, such as (ADP), see Ulmer (2017) for an overview of ADP solutions for dynamic routing problems.

The two main categories of solution methods for dynamic VRPs are offline and online approaches. In offline approaches, the policy for the decision-making process is defined prior to execution. A commonly used offline method is approximate value iteration (AVI), e.g., Ulmer et al. (2019). In such methods, the value function is approximated by a table. To obtain a good policy, a lot of iterations are required, since some states might not be encountered during training otherwise. In online methods, the policy that defines the actions to take if a certain state is encountered is not calculated a priori. Hence, the decisions (calculations) are done during execution. Popular online methods are sampling-based approaches, such as the multiple scenario approach (MSA) in, e.g., Bent and Van Hentenryck (2004a; 2004b; 2007) and Voccia et al. (2017). Conventional online methods are not very suitable when the decision-making needs to be done quickly, since a single decision can take the order of 100 seconds (Voccia et al., 2017).

There is much literature on model-free DRL for dynamic dispatching (section 4.3), but there is little literature on model-free DRL for dynamic routing. There is more literature on model-based or ADP methods for dynamic routing. However, there are many recent literature reviews for these approaches, for which the reader is referred to, e.g., Oyola et al. (2016b), Ritzinger et al. (2016), Psaraftis et al. (2016), Ulmer (2017), Ulmer et al. (2020), and Ojeda Rios et al. (2021). The model-free DRL approaches for dynamic routing problems of Nazari et al. (2018), Peng et al. (2019), James et al. (2019) and Joe and Lau (2020) are reviewed in the following sections. The former three applied an end-to-end learning approach and the latter one applied a learning to configure approach.

4.4.1 End-to-end learning

Nazari et al. (2018) and Peng et al. (2019) proposed end-to-end DRL methods for the dynamic CVRP, but mainly focused on the static routing variant. Hence, these methods are discussed in the section for static routing problems, see section 4.2.

James et al. (2019) proposed an end-to-end DRL method for the dynamic electric vehicle PDP with TW (DEVPDPTW), with the goal to make online decisions as quickly as possible. Their encoder-decoder framework is based on the PN with structural graph embeddings of Dai et al. (2016). The encoder sequentially embeds the nodes, whereafter the PN-based decoder determines a probability distribution for the next node to visit by a vehicle. The vehicle tours are created sequentially. The encoder is a GNN and the decoder is an LSTM. James et al. (2019) sample multiple solutions per instance of which the best is used and train the model parameters with A3C. They tested the model using real-world traffic data of 100 dynamic requests (with pickup and delivery locations) for 100 vehicles and compared results against MIP approaches of James and Lam (2017) and James (2018). The results indicated that the DRL approach outperformed the benchmark in both solution quality (total route length) and computation time.

4.4.2 Learning to configure

Joe and Lau (2020) used an on-policy DRL approach with simulated annealing (the model is called DRLSA) to solve a dynamic and stochastic customer CVRP with soft time windows (DSC-CVRPTW), with the objective to minimize total travel plus waiting times plus penalties for time window violations. They approximate the state-value function using on-policy TD learning with experience replay. The state-value function is utilized by a simulated annealing (SA) algorithm (Chiang and Russell, 1996; Osman, 1993), to insert new customers and swap existing customers in the route. Therefore, their approach can be categorized as 'learning to configure'. Joe and Lau (2020) used a route-based MDP approach, in which they assume that orders are not canceled and that customers cannot swap between routes when they have been assigned. At the start of the planning horizon, the known customer orders are assigned to a fixed number of vehicles, resulting in an initial solution. The state of the MDP is defined by a pre-decision state and a post-decision state, which both contain information on the time step, vehicle locations, remaining routes, and realized orders. At each time step, an action defines which vehicle(s) is/are assigned to serve the new incoming order(s). Additionally, an action may consist of swapping existing customers in the same route. The approximated state-value function takes expected rewards of future dynamic events into account. During training, a set of historical delivery plans are used with 2 vehicles that serve 22 out of 48 customers per day on average. Depending on the DoD, a percentage of the orders are randomly removed and later used as dynamic orders. The performance of DRLSA is compared to three benchmark algorithms: (1) the offline AVI method of Ulmer et al. (2019), (2) MSA with consensus of Bent and Van Hentenryck (2004a; 2004b), and (3) a myopic approach of DRLSA, which selects actions that minimize the immediate reward. After training three DRLSA models with DoD=0.3, 0.5, and 0.7 for a total of 700h, DRLSA outperformed AVI for DoD=0.5 and 0.7 and the myopic approach only for DoD=0.7 by 11.90%. The poor performance of DRLSA for low DoDs could be explained by the fact that it is more challenging to anticipate rarely occurring dynamic events (which is the case for low DoDs). The performance of AVI, for all DoDs, is close to the myopic approach (between -1.51% and 0.24%). AVI approximates values of encountered states during training and values of non-encountered states are set to 0, the latter is equivalent to myopic. This likely is the reason for the similar performance of AVI and the myopic approach. In general, AVI approaches require a lot of training instances for problem settings with large spaces (due to multi-vehicle and multi-constraints) to accurately approximate the value of many states. DRLSA significantly outperformed MSA by on average 7.5%. In terms of run time, the offline approaches DRLSA and AVI required < 10s per decision, while MSA required up to 10m per decision. This indicates that for situations where the decision-making should be very quick, offline approaches are more suitable. Hence, offline approaches for re-routing decisions are more suitable for practical applications.

4.5 Overviews

In this section, concise overviews will be given for the reviewed deep learning literature. First, an overview for literature associated with the routing stage is given followed by the literature associated with the dispatching stage, Table 1 and 2, respectively. Thereafter, the results for deep learning approaches for the static CVRP are summarized in Table 3.

4.5.1 Overview routing

In Table 1 the characteristics of the reviewed literature are summarized for ML-based solution methods for static and dynamic routing problems. The routing characteristics are split into five categories. First, it is indicated what type of problem, static and deterministic (SD) or dynamic and stochastic (DS), is solved by each literature. Second, the checkmarks indicate the routing variant considered by each literature. For the variants with time windows, S and H indicate soft and hard time window constraints, respectively. Third, the checkmarks for single-vehicle or multi-vehicle problems indicate if the corresponding literature proposed a model in which one or multiple vehicles are active simultaneously. For the end-to-end (E2E) and learning to configure (L2C) methods, the routes can be constructed by one or multiple active vehicles. For the learning to search (L2S) methods, naturally there are multiple vehicles active simultaneously. Fourth, the objective of each literature is indicated. Fifth, the maximum considered problem size is indicated for each literature. The ML characteristics are split into five categories. First, the algorithmic structure (E2E, L2C, L2S) is indicated. Second, the machine learning category is indicated. Third, it is indicated whether the literature explicitly formulates the problems as an MDP. Fourth, the used machine learning concepts (attention mechanisms and DRL) are indicated. It must be mentioned that there are many more machine learning concepts, which are not indicated in this overview (e.g., RNN, GNN, GAT, etc). Fifth, the used algorithm(s) for each literature is mentioned.

	Routing characteristics													**Machine learning characteristics**											
	SD	**DS**	**CVRP**	**CVRPSD**	**DSCCVRP**	**CVRPTW**	**PDP**	**DSCCVRPTW**	**DEVPDPTW**	**single-vehicle**	**multi-vehicle**	**objective**	**max. size**	**E2E**	**L2C**	**L2S**	**supervised**	**RL value-based**	**RL policy-based**	**RL actor-critic**	**MDP**	**attention**	**DRL**	**algorithm**
Nazari et al. (2018)	✓	✓	✓	✓	✓					✓		min. total route length	100	✓					✓	✓		✓	✓	REINFORCE w/b, A3C
Kool et al. (2019)	✓		✓	✓						✓		min. total route length	100	✓					✓			✓	✓	REINFORCE w/b
Peng et al. (2019)	✓	✓	✓							✓		min. total route length	100	✓					✓			✓	✓	REINFORCE w/b
Kwon et al. (2020)	✓		✓							✓		min. total route length	100	✓					✓			✓	✓	REINFORCE w/b
Hottung et al. (2021)	✓		✓							✓		min. total route length	300	✓				✓	✓			✓	✓	Q-learning, REINFORCE w/b
Li et al. (2021c)	✓						✓			✓		min. total travel time	120	✓						✓	✓	✓	✓	REINFORCE w/b
Li et al. (2021b)	✓		✓								✓	min./max. total travel time	160	✓					✓		✓	✓	✓	REINFORCE w/b
Falkner and Schmidt-Thieme (2020)	✓		✓			SH					✓	min. total route length + TW pen.	100	✓					✓			✓	✓	REINFORCE w/b
Zhang et al. (2020a)	✓					H					✓	min. total route length + TW pen.	150	✓							✓	✓	✓	REINFORCE w/b
Kool et al. (2021)	✓		✓							✓		min. total route length	100		✓		✓							DP
Chen and Tian (2019)	✓		✓								✓	min. total route length	100			✓				✓		✓	✓	A2C
Wu et al. (2020)	✓		✓								✓	min. total route length	200			✓				✓	✓	✓	✓	AC
Hottung and Tierney (2020)	✓		✓	✓							✓	min. total route length	300			✓			✓			✓	✓	REINFORCE w/b
Lu et al. (2020)	✓		✓								✓	min. total route length	100			✓			✓			✓	✓	REINFORCE w/b
Sultana et al. (2021)	✓		✓								✓	min. total route length	1000			✓			✓			✓	✓	REINFORCE w/b
Joe and Lau (2020)		✓						S			✓	min. total travel time + TW pen.	48		✓			✓			✓		✓	TD
James et al. (2019)		✓							S		✓	max. ORR - total route length	100	✓						✓		✓	✓	A3C

Table 1: Overview of ML-based literature for static and dynamic routing problems. See the list of abbreviations at the beginning of this report for the majority abbreviations in the table, furthermore is S: soft TW, H: hard TW, SH: soft and hard TW.

What becomes clear from Table 1, is that ML-based solution methods for routing problems are generally solved by RL policy-based methods. The primary reason for that could be due to the very complex state and action spaces for these problems, for which value function approximation is not suitable. However, policy-based methods have the disadvantage that every action leading to a good final reward is reinforced. Some performed actions may not be that good, nevertheless, if the outcome of the full trajectory was considered good, the probability of every selected action will be increased. Actor-critic methods combine the best of policy-based and value-based methods. In AC methods, each selected action by policy can be evaluated separately by the critic network. However, learning a good critic network remains challenging for high-dimensional state-action spaces. Another thing that stands out, is that the L2S approaches are unable to solve richer VRPs. This is due to the fact that rewriting the current solution makes it difficult to meet the constraints. Constructive (E2E) approaches can maintain feasibility by using a masking scheme. A commonly used method is to convert complex constraints into penalty terms in the objective function. This is done because training a model with complex constraints becomes more difficult if the number of constraints increases. In practice, more often than not the constraints (such as time windows) are not strict and penalties may reflect real-world practices. However, scenarios with strict constraints may also be applicable to practical applications and is a future challenge for deep learning models.

4.5.2 Overview dispatching

In Table 2 the characteristics of the reviewed literature are summarized for ML-based solution methods for dynamic dispatching problems. The dispatching characteristics are split into two categories. First, the type of problem is mentioned. Second, the objective of each literature is indicated. The machine learning characteristics are split into five categories. First, the algorithmic structure (E2E, L2C, L2S) is indicated. Second, the machine learning category is indicated and because none of the literature used supervised learning it is not included in the table. Third, the information on the MDP formulation is given. Fourth, the used machine learning concepts (attention mechanisms and DRL) are indicated. Fifth, the algorithm(s) used by each literature is mentioned. In Table 2, E2E does not imply a constructive approach, but just that solution is only based on the input and does not utilize other COP algorithms.

	Dispatching characteristics		**Machine learning characteristics**													
	type	**objective**	**E2E**	**L2C**	**L2S**	**RL value-based**	**RL policy-based**	**RL actor-critic**	**multi-agent homog.**	**multi-agent heterog.**	**single-agent**	**agent**	**actions**	**attention**	**DRL**	**algorithm**
Xu et al. (2018)	ride-hailing	max. gross merchandise volume		✓		✓			✓			vehicle	assign, idle			Q-learning
Wang et al. (2018)	ride-hailing	max. ADI		✓		✓			✓			vehicle	assign, idle		✓	DDQN
Tang et al. (2019)	ride-hailing	max. ADI and user experience		✓		✓			✓			vehicle	assign, idle		✓	DDQN
Shi et al. (2019)	ride-hailing (EV)	min. customer waiting time + C		✓		✓			✓			vehicle	assign, idle, charge		✓	TD
Li et al. (2019)	ride-hailing	max. ADI	✓					✓		✓		vehicle	assign, idle		✓	AC
Ke et al. (2020)	ride-hailing	max. ADI and user experience		✓		✓			✓			customer	assign, delay		✓	DQN
				✓				✓	✓						✓	A2C, PPO, ACER
Yang et al. (2021)	ride-hailing	min. average response time, cancellation rate		✓		✓			✓			customer	assign, delay		✓	TD
Wang et al. (2019)	ride-hailing	max. ADI		✓		✓					✓	platform	wait, stop time window			Q-learning
Jindal et al. (2018)	ride-sharing	max. effective trip distance	✓			✓			✓			vehicle	assign, idle		✓	DDQN
Han et al. (2016)	fleet management	max. vehicle income	✓			✓					✓	vehicle (taxi persp.)	move to adjacent grid cell		✓	Q-learning
Wen et al. (2017)	fleet management	max. service availability, min. rebalancing C	✓			✓					✓	vehicle (taxi persp.)	move to adjacent grid cell		✓	DQN
Lin et al. (2018)	fleet management	max. gross merchandise volume	✓			✓			✓			vehicle (platform persp.)	move to adjacent grid cell		✓	DQN
			✓					✓	✓						✓	A2C
Oda and Joe-Wong (2018)	fleet management	min. waiting time + repositioning C	✓			✓				✓		vehicle (platform persp.)	move to zone		✓	DQN
Liu et al. (2020)	fleet management	balance supply and demand	✓			✓			✓			vehicle (platform persp.)	move to neighbouring zone		✓	DQN
Zhang et al. (2020b)	fleet management	max. ORR	✓			✓			✓			vehicle (platform persp.)	move to adjacent grid cell		✓	dueling-DQN, Q-learning
			✓					✓	✓						✓	A2C
Jin et al. (2019)	integrated ride-hailing	max. ADI and ORR	✓					✓		✓		region cell, district	region cell: ranking features, manager: abstract goal	✓	✓	DDPG
Guo and Xu (2020)	integrated ride-sharing	max. QoS - C		✓		✓			✓			vehicle	assign, reposition		✓	DDQN
Liang et al. (2021)	integrated ride-hailing	max. ADI and ORR		✓		✓			✓			vehicle	assign, reposition		✓	DQN

Table 2: Overview of ML-based literature for dynamic dispatching problems. See the list of abbreviations at the beginning of this report for the majority abbreviations in the table, furthermore is ADI: accumulated driver income, C: costs, ORR: order response rate.

Contrasting to the ML-based solution methods for the routing problem (Table 1), for dynamic dispatching a learning to configure approach is more used. The matching process in the ride-hailing problem is generally solved by a bipartite graph matching algorithm that utilizes the approximated action-value function to quickly make online decisions. By formulating the problem as homogeneous multi-agent MDP, the state-action space can be reduced significantly and thus allows for a value-based approach. However, in this way the action space is effectively decomposed and thus some information is lost. The early ML-based approaches for fleet management optimized vehicle repositioning from the single-driver perspective. Later, with the increasing popularity of MoD services, the literature focused on fleet management problems from the perspective of the entire platform. Another noticeable thing is that the objectives are mostly in favor of the platform. Even though some literature try to maximize the ORR or minimize the waiting time, this is in the context of maximizing profit for the centralized platform. First, this could lead to large differences in profits between agents (vehicles) in the fleet, since maximizing joint profit does not guarantee any minimum profit for individual agents. Secondly, minimizing average waiting time or maximizing average ORR may disregard the service levels of individual customers.

4.5.3 Performance comparison static CVRP

Table 3 shows the results of deep learning solution methods for the static CVRP. As mentioned in the introduction of section 4.2, several deep learning approaches to solve the CVRP used the data distribution of Nazari et al. (2018). This allows for an easy comparison of the objective value since they all used the same objective function, which was to minimize the total route length. Run times are more difficult to compare since they depend on the used hardware. Hence, the run times in Table 3 should be interpreted as indications. The results of each method are reported identical to their original paper with two exceptions. First, the results of LKH3 are taken from Kool et al. (2019). Second, the results of Kwon et al. (2020) are taken from Hottung et al. (2021). The three bold headers in Table 3 indicate the problem size, where, e.g., $C_{30}VRP_{20}$ indicates a vehicle capacity of 30 and 20 customers. The percentage gap of each model indicates the performance gap compared to LKH3. The time (in seconds) indicates the average run time per instance.

		$C_{30}VRP_{20}$			**$C_{40}VRP_{50}$**			**$C_{50}VRP_{100}$**		
	Model name	Obj.	Gap (%)	Time (s)	Obj.	Gap (%)	Time (s)	Obj.	Gap (%)	Time (s)
Helsgaun (2017)	LKH3	6,14		0,720	10,38		2,520	15,65		4,680
Nazari et al. (2018)	RL-BS(10)	6,40	4,23%	0,162	11,15	7,42%	0,232	16,96	8,37%	0,445
	RL-greedy	6,59	7,33%	0,105	11,39	9,73%	0,156	17,23	10,10%	0,321
Kool et al. (2019)	AM-sampling	6,25	1,79%	0,036	10,62	2,31%	0,168	16,23	3,71%	0,720
	AM-greedy	6,40	4,23%	<0,001	10,98	5,78%	<0,001	16,80	7,35%	<0,001
Peng et al. (2019)	AM-D(greedy)	6,28	2,28%	<0,001	10,78	3,85%	<0,001	16,40	4,79%	0,016
	AM-D(2OPT)	6,25	1,79%	0,050	10,73	3,37%	0,340	16,27	3,96%	2,210
Falkner and Schmidt-Thieme (2020)	JAMPR-greedy	6,47	5,37%	0,110	11,44	10,21%	0,230			
	JAMPR-sampling	6,26	1,95%	0,840	10,84	4,43%	2,810			
Kwon et al. (2020)	POMO-greedy							15,76	0,70%	0,012
	POMO-sampling							15,67	0,13%	2,520
	POMO-AS							15,63	-0,13%	69,120
Hottung et al. (2021)	POMO-EAS-Emb							15,63	-0,13%	3,240
	POMO-EAS-Lay							15,61	-0,26%	4,320
	POMO-EAS-Tab							15,62	-0,19%	2,880
Kool et al. (2021)	DPDP-10k							15,832	1,16%	0,924
	DPDP-100k							15,694	0,28%	2,148
	DPDP-1M							15,627	-0,15%	17,502
Chen and Tian (2019)	NeuRewriter	6,16	0,33%	0,133	10,51	1,25%	0,211	16,10	2,88%	0,398
Wu et al. (2020)	LIH(T=1000)	6,16	0,33%	0,138	10,71	3,18%	0,288	16,30	4,15%	0,360
	LIH(T=3000)	6,14	0,00%	0,360	10,55	1,64%	0,720	16,11	2,94%	1,080
	LIH(T=5000)	6,12	-0,33%	0,720	10,45	0,67%	1,440	16,03	2,43%	1,800
Hottung and Tierney (2020)	NLNS-batch	6,19	0,81%	0,043	10,54	1,54%	0,145	15,99	2,17%	0,374
Lu et al. (2020)	L2I	6,12	-0,33%	0,360	10,35	-0,29%	0,510	15,57	-0,51%	0,720
Sultana et al. (2021)	L2GLS	5,85	-4,72%	0,420	10,30	-0,77%	0,600	14,67	-6,26%	1,020

Table 3: Overview of results for ML-based approaches for the static CVRP, where the gap is calculated with respect to the results of the LKH3 heuristic.

A few conclusions from the performance comparison can be made. First, improvement methods generally achieve better performance, which can be explained by the fact that the solution space is explored more. Constructive methods use a less efficient search strategy (e.g., sampling) and explore the solution to a lesser extent. Second, because the search strategy of improvement methods is more extensive than of constructive methods, improvement methods require more execution time. Despite that comparing execution time between literature is hard due to different hardware used, Li et al. (2021a) also concluded this after running different models on the same hardware.

Another conclusion not specifically tied to the performance comparison in Table 3 is that improvement methods tend to perform better on data with a different distribution than during training, i.e, improvement methods generalize better. This is again related to the less extensive search exploration of constructive methods.

5 Challenges and opportunities for future research

Throughout the previous section, the advantage of DRL methods compared to conventional solution methods for routing problems has become clear. DRL methods can produce high-quality solutions in short run times, but there remain some challenges to these approaches. In this section, the major challenges of DRL solution methods for routing problems are mentioned. Throughout the following sections, opportunities for future research will also be discussed.

5.1 Scalability

Deep learning approaches for the dispatching problem are able to handle big problem sizes (with thousands of vehicles) by using centralized control and considering homogeneous agents that share model parameters. However, in reality, the agents are more likely to be heterogeneous. Heterogeneity remains challenging due to the rapidly growing state and action spaces.

The current deep learning methods for the routing phase are limited to solve instances of a few hundred customers. Training a model for larger problem sizes comes with two difficulties. First, the long training time is a bottleneck for quickly improving and evaluating the model performance, which might be the reason that the literature only considered relatively small problem sizes. Second, larger problems sizes require larger models with more parameters and consequently, more data is required to train the parameters. However, the sizes of real-world data sets are generally not very large for deep learning models, making overfitting another bottleneck. Li et al. (2021a) concluded that improvement methods require less training data compared to constructive methods as the former converges faster in terms of average route length on the training data.

Deep learning improvement methods seem to be better scalable than deep learning constructive methods. This can be explained by the fact that in the former method the complexity of the improvement is mostly determined by the degree of destruction (e.g., if in an iteration five nodes are relaxed from the current solution, new locations for only these five nodes must be determined). Most deep learning improvement methods use randomly created initial solutions so creating the initial solution is not a bottleneck for large-scale problems. On the other hand, for deep learning constructive methods the complexity of determining the next node to visit is directly affected by the problem size.

5.2 Generalization

Deep learning algorithms can outperform several heuristics when the models are tested on the same data distribution and problem size as during training. However, performance decreases relatively fast for deep learning models as settings differ from those during training. End-to-end constructive methods tend to generalize worse compared to learning to search improvement methods, which was indicated by, e.g., Wu et al. (2020) and Li et al. (2021a), due to the less extensive search exploration of constructive methods compared to improvement methods. For practical purposes, it may be useful to train multiple models for different settings and automatically detect which model can best be used to solve the problem at hand. Another potential way to tackle this issue is to use active search in constructive methods, as in Hottung et al. (2021), which adapts some of model parameters during testing to the specific instance at hand.

5.3 Adaptability

Each specific problem setting requires an individual trained DL model, whereas heuristic methods of the OR community are able to solve problems with broader data characteristics. Nonetheless, several machine learning approaches showed that the general concept of a model can be utilized for multiple types of problems with similarities. Whereas pure heuristic methods need a lot of expert knowledge and understanding of the specific problem, e.g., Nazari et al. (2018) and Kool et al. (2019) showed that a DL model can easily be adapted to suit a different type of problem with minor changes.

Constructive solution methods have much greater adaptability than improvement solution methods. First of all, by constructively building the solution they can adapt to dynamic events. Second, by using a simple masking scheme, complex constraints can be taken into account, whereas improvement methods have much more difficulties to maintain feasibility while locally rewriting the solution. Therefore, constructive approaches have more potential for dynamic and constrained routing problems.

5.4 Training and testing time

Comparing training and testing time is tricky since the hardware (GPU vs CPU) and language (C++ vs Python) highly influence the running times. Whereas the OR community usually implemented algorithms in the more efficient language C++, the CS community implemented deep learning models in Python. Also, the degree of parallelization has a big effect on the run times. Heuristic algorithms are most often executed on CPUs using a single thread, while ML approaches are often executed in parallel on CPUs. To facilitate a fair comparison, an indicative measure of the speedup due to CPU vs GPU could be given.

Nonetheless, it can be concluded that DRL approaches in general are able to reduce the inference run time compared to heuristic approaches. However, training deep learning models can take quite a large time. For dynamic problems, where the decision-making needs to be quick, learning a value function that learns to configure a combinatorial optimization algorithm seems the best approach.

5.5 Inconsistencies

There are several inconsistencies in the literature in terms of reported run times, objective values, and data distributions. First, Hottung and Tierney (2020) reported a total run time of 6d for LKH3 for all 10000 test instances, while others e.g., Kool et al. (2021; 2019) reported 780m. Second, the run times reported by Hottung and Tierney (2020) for RL-BS(10) are inconsistent with what is reported by Nazari et al. (2018). Third, Lu et al. (2020) mentioned different vehicle capacities as Nazari et al. (2018), but reported identical tour lengths for the other machine learning approaches as in their original papers. Fourth, the reported run time by Kool et al. (2021) for the model of Lu et al. (2020) is inconsistent as in its original paper. Fifth, Sultana et al. (2021) reported the average run time per instance for LKH3 as 780m, while others, e.g., Kool et al. (2021; 2019) reported this as the total run time for 10000 test instances. More care should be given reporting results consistent with other approaches to facilitate a fair comparison.

5.6 Benchmark datasets and algorithms

In the introduction was mentioned that the CS community recently has become more interested in solving COPs, like the VRP. A common way for the CS community is to generate data using an arbitrarily defined data distribution from which instances are sampled. Contrasting to this, the OR community has generated benchmark instances, e.g., Solomon (1987) and Uchoa et al. (2017), that represent more relevant practices. While some deep learning methods tested models on these benchmark instances, e.g., Hottung et al. (2021) and Kool et al. (2021), the majority of the CS community primarily used the data distribution of Nazari et al. (2018). For a fair comparison with highly optimized heuristics, the CS community could consider evaluating their models on these benchmark instances. Additionally, the CS community often considered LKH3 or Google's OR-Tools to be the state-of-the-art solvers. However, heuristic solvers of Vidal (2021), Accorsi and Vigo (2021), and Christiaens and Vanden Berghe (2020) and the exact solver of Pessoa et al. (2020) outperform the two aforementioned solvers (Accorsi et al., 2021). Future deep learning models for VRPs created by the CS community should be compared against these state-of-the-art VRP solvers created by the OR community.

6 Conclusion

In this literature review, ML-based solution methods for the vehicle routing problem are surveyed. The literature is categorized according to the dispatching and routing stage. The dispatching stage corresponds to MoD services, such as the ride-haling problem, in which the supply-demand ratio is high and thus is more like an order matching problem. For these dynamic problems, value-based DRL solution methods are most common. The literature shows that deep learning models are able to learn the underlying spatio-temporal probability distribution and that with the use of approximated value functions order matching algorithms can determine good decisions for large-scale applications in a matter of milliseconds. The routing stage corresponds more to the traditional vehicle routing problem, where the supply-demand ratio is low. Due to the NP-hard nature of VRPs, DRL policy-based methods are the most applied. The ML-based literature for the VRP is still limited to a maximum of a few hundred customers. Deep learning methods with an improvement approach, also called learning to search, explore the solution space more extensively compared to constructive approaches and hence generally achieve better results, generalize better to other data distributions than during training, but also require more run time.

Considering that the CS community only have focused on VRPs for a few years and some outperformed highly optimized heuristics is less computation time, the potential of DRL approaches for routing problems looks very promising. However, there remain some challenges with these solution methods in terms of scalability, generalization, and adaptability.

Future research for the dispatching stage could be focused on automatically detecting abrupt changes in supply and demand dynamics and updating the model parameters or the learned value function corresponding to the abrupt changes due to, e.g., traffic accidents. Furthermore, the fair distribution of profits between platforms, agents (vehicle drivers), and customers should be taken into account rather than maximizing total profit for the platform.

Future research for the routing problem stage be focused on larger and richer VRPs with more practical constraints. Future DRL approaches should compare results against state-of-the-art heuristic solvers created by the OR community on more realistic data sets rather than to outdated heuristics and on self-created data distributions. There are few DRL solution methods for dynamic VRPs, so future research may consider extending existing models for static VRPs to dynamic VRPs. Several end-to-end constructive approaches mentioned that the final solution may benefit from an additional optimization step. Applying learning to search DRL methods or other improvement algorithms on top of the final solution of end-to-end models could also be interesting for future research.

For both the dispatching and routing stage, taking into account heterogeneous agents (vehicles) remains challenging due to the quickly increasing state and action spaces, but is an important feature for real-world applications.

References

Accorsi, L., Lodi, A., & Vigo, D. (2021). Guidelines for the computational testing of machine learning approaches to vehicle routing problems. *arXiv preprint arXiv:2109.13983*.

Accorsi, L., & Vigo, D. (2021). A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems. *Transportation Science*, *55*(4), 832–856.

Aggarwal, C. C. (2018). *Neural networks and deep learning*. Springer. https://doi.org/10.1007/978-3-319-94463-0

Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., & Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, *114*(3), 462–467.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv*, *1409.0473*. https://arxiv.org/abs/1409.0473

Bai, R., Chen, X., Chen, Z.-L., Cui, T., Gong, S., He, W., Jiang, X., Jin, H., Jin, J., Kendall, G. et al. (2021). Analytics and machine learning in vehicle routing research. *arXiv preprint arXiv:2102.10012*.

Baker, B., & Ayechew, M. (2003). A genetic algorithm for the vehicle routing problem. *Computers and Operations Research*, *30*(5), 787–800. https://doi.org/10.1016/S0305-0548(02)00051-5

Baldacci, R., Mingozzi, A., & Roberti, R. (2012). Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, *218*(1), 1–6.

Bauer, G. S., Greenblatt, J. B., & Gerke, B. F. (2018). Cost, energy, and environmental impact of automated electric taxi fleets in manhattan. *Environmental Science & Technology*, *52*(8), 4920–4928. https://doi.org/10.1021/acs.est.7b04732

Bellman, R. (1957). A Markovian Decision Process. *Indiana Univ. Math. J.*, *6*, 679–684. https://doi.org/10.1512/iumj.1957.6.56038

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Bengio, Y., Lodi, A., & Prouvost, A. (2018). Machine learning for combinatorial optimization: A methodological tour d'horizon. *CoRR*, *abs/1811.06128*. http://arxiv.org/abs/1811.06128

Bent, R., & Van Hentenryck, P. (2004a). Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, *52*, 977–987.

Bent, R., & Van Hentenryck, P. (2004b). The Value of Consensus in Online Stochastic Scheduling. *ICAPS*, *4*, 219–226.

Bent, R., & Van Hentenryck, P. (2007). Waiting and relocation strategies in online stochastic vehicle routing. *IJCAI*, *7*, 1816–1821.

Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., & Laporte, G. (2007). Static pickup and delivery problems: A classification scheme and survey. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, *15*, 1–31. https://doi.org/10.1007/s11750-007-0009-0

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Bourgeois, F., & Lassalle, J.-C. (1971). An extension of the munkres algorithm for the assignment problem to rectangular matrices. *Communications of the ACM*, *14*(12), 802–804.

Braekers, K., Ramaekers, K., & Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. *Computers Industrial Engineering*, *99*, 300–313. https://doi.org/10.1016/j.cie.2015.12.007

Bräysy, O., & Gendreau, M. (2005a). Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science*, *39*, 104–118. https://doi.org/10.1287/trsc.1030.0056

Bräysy, O., & Gendreau, M. (2005b). Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science*, *39*, 119–139. https://doi.org/10.1287/trsc.1030.0057

Chen, T. D., Kockelman, K. M., & Hanna, J. P. (2016). Operations of a shared, autonomous, electric vehicle fleet: Implications of vehicle charging infrastructure decisions. *Transportation Research Part A: Policy and Practice*, *94*, 243–254. https://doi.org/10.1016/j.tra.2016.08.020

Chen, X., & Tian, Y. (2019). Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, *32*. https://proceedings.neurips.cc/paper/2019/file/131f383b434fdf48079bff1e44e2d9a5-Paper.pdf

Chiang, W.-C., & Russell, R. A. (1996). Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Annals of Operations Research*, *63*(1), 3–27.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734. https://doi.org/10.3115/v1/D14-1179

Christiaens, J., & Vanden Berghe, G. (2020). Slack induction by string removals for vehicle routing problems. *Transportation Science*, *54*(2), 417–433.

Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, *12*(4), 568–581. https://doi.org/10.1287/opre.12.4.568

Dai, H., Dai, B., & Song, L. (2016). Discriminative embeddings of latent variable models for structured data. *International conference on machine learning*, 2702–2711.

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, *6*(1), 80–91.

Desrochers, M., & Laporte, G. (1991). Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *Operations Research Letters*, *10*(1), 27–36. https://doi.org/10.1016/0167-6377(91)90083-2

EEA. (2020). *Greenhouse gas emissions from transport in Europe*. https://www.eea.europa.eu/data-and-maps/indicators/transport-emissions-of-greenhouse-gases/transport-emissions-of-greenhouse-gases-12

Eksioglu, B., Vural, A. V., & Reisman, A. (2009). The vehicle routing problem: A taxonomic review. *Computers Industrial Engineering*, *57*(4), 1472–1483. https://doi.org/10.1016/j.cie.2009.05.009

Falkner, J. K., & Schmidt-Thieme, L. (2020). Learning to solve vehicle routing problems with time windows through joint attention. *arXiv: 2006.09100*.

Farazi, N. P., Ahamed, T., Barua, L., & Zou, B. (2020). Deep reinforcement learning and transportation research: A comprehensive review. *arXiv preprint arXiv:2010.06187*.

Gers, F., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, *12*, 2451–71. https://doi.org/10.1162/089976600300015015

Gillett, B. E., & Miller, L. R. (1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations research*, *22*(2), 340–349.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence [Applications of Integer Programming]. *Computers Operations Research*, *13*(5), 533–549.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. http://www.deeplearningbook.org

Greff, K., Srivastava, R., Koutník, J., Steunebrink, B., & Schmidhuber, J. (2015). LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*. https://doi.org/10.1109/TNNLS.2016.2582924

Gromicho, J., van Hoorn, J., Kok, A., & Schutten, J. (2012). Restricted dynamic programming: A flexible framework for solving realistic vrps. *Computers Operations Research*, *39*(5), 902–909. https://doi.org/j.cor.2011.07.002

Guo, G., & Xu, Y. (2020). A deep reinforcement learning approach to ride-sharing vehicles dispatching in autonomous mobility-on-demand systems. *IEEE Intelligent Transportation Systems Magazine*.

Haliem, M., Mani, G., Aggarwal, V., & Bhargava, B. (2020). A distributed model-free ride-sharing algorithm with pricing using deep reinforcement learning. *Computer Science in Cars Symposium*. https://doi.org/10.1145/3385958.3430484

Han, M., Senellart, P., Bressan, S., & Wu, H. (2016). Routing an autonomous taxi with reinforcement learning. *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2421–2424.

Haydari, A., & Yilmaz, Y. (2020). Deep reinforcement learning for intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*.

Helsgaun, K. (2017). An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems.

Hentenryck, P. V., & Bent, R. (2006). *Online stochastic combinatorial optimization*. The MIT Press.

Hildebrandt, F. D., Thomas, B., & Ulmer, M. W. (2021). Where the action is: Let's make reinforcement learning for stochastic dynamic vehicle routing problems work! *arXiv preprint arXiv:2103.00507*.

Hillier, H., & Lieberman, G. (2014). *Introduction to Operations Research* (10th edition). Mc Graw Hill Education (Uk).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*, 1735–80. https://doi.org/10.1162/neco.1997.9.8.1735

Holler, J., Vuorio, R., Qin, Z., Tang, X., Jiao, Y., Jin, T., Singh, S., Wang, C., & Ye, J. (2019). Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. *2019 IEEE International Conference on Data Mining (ICDM)*, 1090–1095.

Hottung, A., Kwon, Y.-D., & Tierney, K. (2021). Efficient active search for combinatorial optimization problems. *arXiv preprint arXiv:2106.05126*. https://arxiv.org/abs/2106.05126

Hottung, A., & Tierney, K. (2020). Neural large neighborhood search for the capacitated vehicle routing problem. *arXiv preprint arXiv:1911.09539*.

James, J. (2018). Two-stage request scheduling for autonomous vehicle logistic system. *IEEE Transactions on Intelligent Transportation Systems*, *20*(5), 1917–1929.

James, J., & Lam, A. Y. (2017). Autonomous vehicle logistic system: Joint routing and charging strategy. *IEEE Transactions on Intelligent Transportation Systems*, *19*(7), 2175–2187.

James, J., Yu, W., & Gu, J. (2019). Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, *20*(10), 3806–3817.

Jin, J., Zhou, M., Zhang, W., Li, M., Guo, Z., Qin, Z., Jiao, Y., Tang, X., Wang, C., Wang, J. et al. (2019). Coride: Joint order dispatching and fleet management for multi-scale ride-hailing platforms. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 1983–1992.

Jindal, I., Qin, Z. T., Chen, X., Nokleby, M., & Ye, J. (2018). Optimizing taxi carpool policies via reinforcement learning and spatio-temporal mining. *2018 IEEE International Conference on Big Data (Big Data)*, 1417–1426.

Joe, W., & Lau, H. C. (2020). Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers. *Proceedings of the International Conference on Automated Planning and Scheduling*, *30*, 394–402. https://ojs.aaai.org/index.php/ICAPS/article/view/6685

Joshi, C. K., Laurent, T., & Bresson, X. (2019). An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*. https://arxiv.org/abs/1906.01227

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. *International conference on machine learning*, 2342–2350.

Kang, N., Feinberg, F. M., & Papalambros, P. Y. (2016). Autonomous Electric Vehicle Sharing System Design. *Journal of Mechanical Design*, *139*(1). https://doi.org/10.1115/1.4034471

Ke, J., xiao feng, f., Yang, H., & Ye, J. (2020). Learning to delay in ride-sourcing systems: A multi-agent deep reinforcement learning framework. *IEEE Transactions on Knowledge and Data Engineering*, 1–1.

Ke, J., Zheng, H., Yang, H., & Chen, X. ((2017). Short-term forecasting of passenger demand under on-demand ride services: A spatio-temporal deep learning approach. *Transportation Research Part C: Emerging Technologies*, *85*, 591–608. https://doi.org/10.1016/j.trc.2017.10.016

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by simulated annealing. *Science (New York, N.Y.)*, *220*, 671–80.

Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-critic algorithms. *Advances in neural information processing systems*, 1008–1014.

Kool, W., van Hoof, H., Gromicho, J., & Welling, M. (2021). Deep policy dynamic programming for vehicle routing problems. *arXiv preprint arXiv:2102.11756*. https://arxiv.org/abs/2102.11756

Kool, W., van Hoof, H., & Welling, M. (2019). Attention, learn to solve routing problems! *International Conference on Learning Representations*. https://openreview.net/forum?id=ByxBFsRqYm

Kwon, Y.-D., Choo, J., Kim, B., Yoon, I., Gwon, Y., & Min, S. (2020). Pomo: Policy optimization with multiple optima for reinforcement learning. *arXiv preprint arXiv:2010.16011*.

Laporte, G. (2009). Fifty years of vehicle routing. *Transportation Science*, *43*, 408–416. https://doi.org/10.1287/trsc.1090.0301

Laporte, G., & Semet, F. (2002). 5. Classical Heuristics for the Capacitated VRP. *The Vehicle Routing Problem* (pp. 109–128).

Larsen, A. (2000). *The dynamic vehicle routing problem* (Doctoral dissertation). PhD thesis, Institute of Mathematical Modelling, Technical University of Denmark.

Li, B., Wu, G., He, Y., Fan, M., & Pedrycz, W. (2021a). An overview and experimental study of learning-based optimization algorithms for vehicle routing problem. *arXiv preprint arXiv:2107.07076*.

Li, J., Ma, Y., Gao, R., Cao, Z., Lim, A., Song, W., & Zhang, J. (2021b). Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem. *IEEE Transactions on Cybernetics*.

Li, J., Xin, L., Cao, Z., Lim, A., Song, W., & Zhang, J. (2021c). Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*.

Li, M., Qin, Z., Jiao, Y., Yang, Y., Wang, J., Wang, C., Wu, G., & Ye, J. (2019). Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. *The World Wide Web Conference*, 983–994.

Liang, E., Wen, K., Lam, W. H., Sumalee, A., & Zhong, R. (2021). An integrated reinforcement learning and centralized programming approach for online taxi dispatching. *IEEE Transactions on Neural Networks and Learning Systems*.

Liao, Z. (2003). Real-time taxi dispatching using global positioning systems. *Commun. ACM*, *46*(5), 81–83. https://doi.org/10.1145/769800.769806

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, K., Zhao, R., Xu, Z., & Zhou, J. (2018). Efficient large-scale fleet management via multi-agent deep reinforcement learning. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, 1774–1783. https://doi.org/10.1145/3219819.3219993

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, *44*(10), 2245–2269. https://doi.org/10.1002/j.1538-7305.1965.tb04146.x

Liu, Z., Li, J., & Wu, K. (2020). Context-aware taxi dispatching at city-scale using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*.

Lu, H., Zhang, X., & Yang, S. (2020). A learning-based iterative method for solving vehicle routing problems. https://openreview.net/forum?id=BJe1334YDH

Lund, K., Madsen, O., & J.M., R. (1996). Vehicle Routing Problems with Varying Degrees of Dynamism.

Ma, S., Zheng, Y., & Wolfson, O. (2014). Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, *27*(7), 1782–1795.

Mazyavkina, N., Sviridov, S., Ivanov, S., & Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers Operations Research*, *134*, 105400. https://doi.org/10.1016/j.cor.2021.105400

Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM*, *7*(4), 326–329. https://doi.org/10.1145/321043.321046

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *Proceedings of The 33rd*

International Conference on Machine Learning, *48*, 1928–1937. https://proceedings.mlr.press/v48/mniha16.html

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–33. https://doi.org/10.1038/nature14236

Montoya-Torres, J. R., López Franco, J., Nieto Isaza, S., Felizzola Jiménez, H., & Herazo-Padilla, N. (2015). A literature review on the vehicle routing problem with multiple depots. *Computers Industrial Engineering*, *79*, 115–129. https://doi.org/10.1016/j.cie.2014.10.029

Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., & Damas, L. (2013). Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, *14*(3), 1393–1402.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, *5*(1), 32–38.

Nazari, M., Oroojlooy, A., Takáč, M., & Snyder, L. (2018). Reinforcement learning for solving the vehicle routing problem. *Advances in Neural Information Processing Systems*, 9839–9849.

Oda, T., & Joe-Wong, C. (2018). Movi: A model-free approach to dynamic fleet management. *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, 2708–2716.

Ojeda Rios, B. H., Xavier, E. C., Miyazawa, F. K., Amorim, P., Curcio, E., & Santos, M. J. (2021). Recent dynamic vehicle routing problems: A survey. *Computers Industrial Engineering*, *160*, 107604. https://doi.org/10.1016/j.cie.2021.107604

Olah, C. (2015). *Understanding lstm networks*. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, *41*(4), 421–451.

Oyola, J., Arntzen, H., & Woodruff, D. L. (2016a). The stochastic vehicle routing problem, a literature review, part i: Models. *EURO Journal on Transportation and Logistics*, *7*(3), 193–221. https://doi.org/10.1007/s13676-016-0100-5

Oyola, J., Arntzen, H., & Woodruff, D. L. (2016b). The stochastic vehicle routing problem, a literature review, part ii: Solution methods. *EURO Journal on Transportation and Logistics*, *6*(4), 349–388. https://doi.org/10.1007/s13676-016-0099-7

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345–1359.

Peng, B., Wang, J., & Zhang, Z. (2019). A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems, 636–650.

Pessoa, A., Sadykov, R., Uchoa, E., & Vanderbeck, F. (2020). A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, *183*(1), 483–523.

Phi, M. (2018). *Illustrated guide to lstm's and gru's: A step by step explanation*. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, *225*(1), 1–11. https://doi.org/10.1016/j.ejor.2012.08.015

Potvin, J.-Y., & Rousseau, J.-M. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, *66*(3), 331–340. https://doi.org/10.1016/0377-2217(93)90221-8

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers Operations Research*, *31*(12), 1985–2002. https://doi.org/10.1016/S0305-0548(03)00158-8

Psaraftis, H. N. (1980). A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, *14*, 130–154. https://doi.org/10.1287/trsc.14.2.130

Psaraftis, H. N., Wen, M., & Kontovas, C. A. (2016). Dynamic vehicle routing problems: Three decades and counting. *Networks*, *67*(1), 3–31. https://doi.org/10.1002/net.21628

Qin, Z., Zhu, H., & Ye, J. (2021). Reinforcement learning for ridesharing: A survey. *arXiv preprint arXiv:2105.01099*.

Qu, M., Zhu, H., Liu, J., Liu, G., & Xiong, H. (2014). A cost-effective recommender system for taxi drivers. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 45–54.

Ritzinger, U., Puchinger, J., & Hartl, R. F. (2016). A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, *54*(1), 215–231. https://doi.org/10.1080/00207543.2015.1043403

Rochat, Y., & Taillard, E. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, *1*(1), 147–167. https://doi.org/10.1007/BF02430370

Rong, H., Zhou, X., Yang, C., Shafiq, Z., & Liu, A. (2016). The rich and the poor: A markov decision process approach to optimizing taxi driver revenue efficiency. *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2329–2334. https://doi.org/10.1145/2983323.2983689

Ropke, S., & Pisinger, D. (2006). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, *40*, 455–472.

Sanderson, G. (2017). *Neural networks*. https://www.3blue1brown.com/topics/neural-networks

Shah, S., Lowalekar, M., & Varakantham, P. (2020). Neural approximate dynamic programming for on-demand ride-pooling. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(01), 507–515.

Shaw, P. (1997). A New Local Search Algorithm Providing High Quality Solutions to Vehicle Routing Problems.

Shi, J., Gao, Y., Wang, W., Yu, N., & Ioannou, P. A. (2019). Operating electric vehicle fleet for ride-hailing services with reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, *21*(11), 4822–4834.

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*, 484–489. https://doi.org/10.1038/nature16961

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, *550*, 354–359. https://doi.org/10.1038/nature24270

Soeffker, N., Ulmer, M. W., & Mattfeld, D. C. (2021). Stochastic dynamic vehicle routing in the light of prescriptive analytics: A review. *European Journal of Operational Research*.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, *35*(2), 254–265. http://web.cba.neu.edu/~msolomon/problems.htm

Sultana, N., Chan, J., Sarwar, T., Abbasi, B., & Qin, A. K. (2021). Learning enhanced optimisation for routing problems. *arXiv preprint arXiv:2109.08345*.

Sungur, I., Ren, Y., Ordóñez, F., Dessouky, M., & Zhong, H. (2010). A model and algorithm for the courier delivery problem with uncertainty. *Transportation science*, *44*(2), 193–205.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 3104–3112.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Taillard, É., Badeau, P., Gendreau, M., Guertin, F., & Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, *31*(2), 170–186. https://doi.org/10.1287/trsc.31.2.170

Tang, X., Qin, Z. (, Zhang, F., Wang, Z., Xu, Z., Ma, Y., Zhu, H., & Ye, J. (2019). A deep value-network based approach for multi-driver order dispatching. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*. https://doi.org/10.1145/3292500.3330724

Thompson, P. M., & Psaraftis, H. N. (1993). Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations Research*, *41*(5), 935–946.

Tong, Y., Chen, Y., Zhou, Z., Chen, L., Wang, J., Yang, Q., Ye, J., & Lv, W. (2017). The simpler the better: A unified approach to predicting original taxi demands based on large-scale online platforms. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1653–1662. https://doi.org/10.1145/3097983.3098018

Toth, P., & Vigo, D. (2002). *The vehicle routing problem*. SIAM.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., & Subramanian, A. (2017). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, *257*(3), 845–858. https://doi.org/10.1016/j.ejor.2016.08.012

Ulmer, M. W., Goodson, J. C., Mattfeld, D. C., & Thomas, B. W. (2017). *Route-based markov decision processes for dynamic vehicle routing problems* (tech. rep.). Technical report, Braunschweig.

Ulmer, M. W., Goodson, J. C., Mattfeld, D. C., & Thomas, B. W. (2020). On modeling stochastic dynamic vehicle routing problems. *EURO Journal on Transportation and Logistics*, *9*(2), 100008.

Ulmer, M. W., Thomas, B. W., & Mattfeld, D. C. (2019). Preemptive depot returns for dynamic same-day delivery. *EURO journal on Transportation and Logistics*, *8*(4), 327–361. https://doi.org/10.1007/s13676-018-0124-0

Ulmer, M. W. (2017). *Approximate dynamic programming for dynamic vehicle routing*. Springer.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. *Proceedings of the AAAI conference on artificial intelligence*, *30*(1).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 6000–6010.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph attention networks. *International Conference on Learning Representations*.

Vesselinova, N., Steinert, R., Perez-Ramirez, D. F., & Boman, M. (2020). Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, *8*, 120388–120416. https://doi.org/10.1109/ACCESS.2020.3004964

Vidal, T. (2021). Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 105643.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, *234*(3), 658–673. https://doi.org/10.1016/j.ejor.2013.09.045

Vinyals, O., Bengio, S., & Kudlur, M. (2015a). Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015b). Pointer networks. *Advances in Neural Information Processing Systems*, 2692–2700.

Voccia, S. A., Campbell, A. M., & Thomas, B. W. (2017). The same-day delivery problem for online purchases. *Transportation Science*, *53*(1), 167–184.

Wang, Y., Tong, Y., Long, C., Xu, P., Xu, K., & Lv, W. (2019). Adaptive dynamic bipartite graph matching: A reinforcement learning approach. *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 1478–1489.

Wang, Y., Zheng, Y., & Xue, Y. (2014). Travel time estimation of a path using sparse trajectories. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 25–34.

Wang, Z., Qin, Z., Tang, X., Ye, J., & Zhu, H. (2018). Deep reinforcement learning with knowledge transfer for online rides order dispatching. *2018 IEEE International Conference on Data Mining (ICDM)*, 617–626.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. *International conference on machine learning*, 1995–2003.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), 279–292. https://doi.org/10.1007/BF00992698

Wen, J., Zhao, J., & Jaillet, P. (2017). Rebalancing shared mobility-on-demand systems: A reinforcement learning approach. *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 220–225. https://doi.org/10.1109/ITSC.2017.8317908

Weng, L. (2018). Policy gradient algorithms. https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

Williams, R. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256. https://doi.org/10.1007/BF00992696

Wren, A., & Holliday, A. (1972). Computer scheduling of vehicles from one or more depots to a number of delivery points. *Journal of the Operational Research Society*, *23*(3), 333–344. https://doi.org/10.1057/jors.1972.53

Wu, Y., Song, W., Cao, Z., Zhang, J., & Lim, A. (2020). Learning improvement heuristics for solving routing problems.

Xu, Z., Li, Z., Guan, Q., Zhang, D., Li, Q., Nan, J., Liu, C., Bian, W., & Ye, J. (2018). Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, 905–913. https://doi.org/10.1145/3219819.3219824

Yang, L., Yu, X., Cao, J., Liu, X., & Zhou, P. (2021). Exploring deep reinforcement learning for task dispatching in autonomous on-demand services. *ACM Trans. Knowl. Discov. Data*, *15*(3). https://doi.org/10.1145/3442343

Yu, X., & Shen, S. (2019). An integrated decomposition and approximate dynamic programming approach for on-demand ride pooling. *IEEE Transactions on Intelligent Transportation Systems*, *21*(9), 3811–3820.

Yuan, N. J., Zheng, Y., Zhang, L., & Xie, X. (2012). T-finder: A recommender system for finding passengers and vacant taxis. *IEEE Transactions on knowledge and data engineering*, *25*(10), 2390–2403.

Zhan, X., Hasan, S., Ukkusuri, S. V., & Kamga, C. (2013). Urban link travel time estimation using large-scale taxi data with partial information. *Transportation Research Part C: Emerging Technologies*, *33*, 37–49. https://doi.org/10.1016/j.trc.2013.04.001

Zhang, K., He, F., Zhang, Z., Lin, X., & Li, M. (2020a). Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach. *Transportation Research Part C: Emerging Technologies*, *121*, 102861. https://doi.org/10.1016/j.trc.2020.102861

Zhang, W., Wang, Q., Li, J., & Xu, C. (2020b). Dynamic fleet management with rewriting deep reinforcement learning. *IEEE Access*, *8*, 143333–143341.

Zong, Z., Feng, T., Xia, T., Li, Y. et al. (2021). Deep reinforcement learning for demand driven services in logistics and transportation systems: A survey. *arXiv preprint arXiv:2108.04462.*

Appendices

A LSTM and GRU

Goodfellow et al. (2016), Olah (2015), and Phi (2018) are consulted for information, equations, and images about LSTMs and GRUs.

Hochreiter and Schmidhuber (1997) introduced the LSTM model. Similar to RNNs, LSTMs processes information sequentially. An LSTM is able to control the flow of information with its internal mechanisms (called gates). The gates are NNs that can learn whether some information is important or not. The gates can decide which information to keep and to throw away. The operations within an RNN cell (Figure 9) and an LSTM cell (Figure 10) is what differentiates them. The activation functions may be chosen differently from the examples given below.

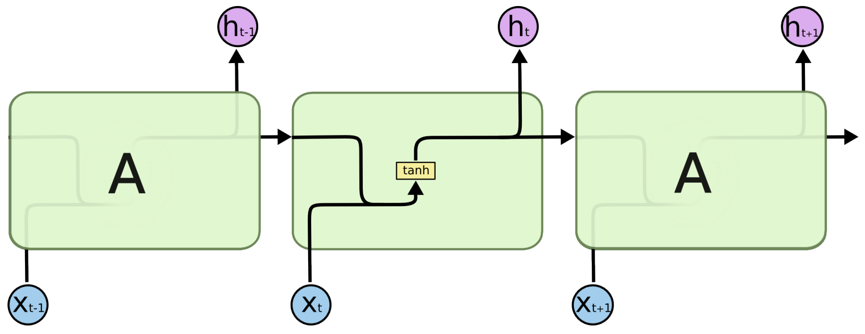


Figure 9: image retrieved from Olah (2015). The repeating module in a standard RNN contains a single layer.

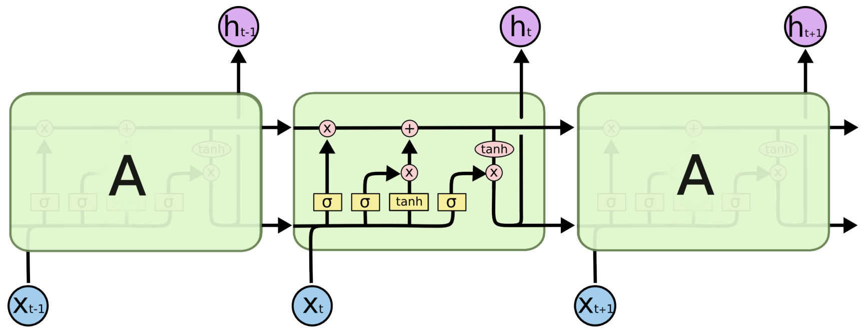


Figure 10: image retrieved from Olah (2015). The repeating module in an LSTM contains four interacting layers.

The LSTM cells consists of a state and gates. The cell state transfers information throughout the sequence processing, i.e. its the memory of the network. Gates are NNs that determine which information is added or removed to the cell state and are trained to learn what information is relevant. There are three gates in an LSTM cell: the forget gate, the input gate, and the output gate.

As the name suggest, the forget gate (see Figure 11) determines which information should be kept and disregarded. The input ($\boldsymbol{x}_t$) and the previous hidden state ($\boldsymbol{h}_{t-1}$) is processed by a sigmoid layer, which outputs values between 0 (forget all information) and 1 (keep all information). The forget gate operation f_t^j (for time step t and cell j) is mathematically expressed as:

$$\boldsymbol{f}_t^j = \sigma\left(\boldsymbol{b}_f^j + \boldsymbol{U}_f^j \boldsymbol{x}_t + \boldsymbol{W}_f^j \boldsymbol{h}_{t-1}^j\right), \tag{A.1}$$

where $\boldsymbol{b}_f$, $\boldsymbol{U}_f$, and $\boldsymbol{W}_f$ are biases, input weights, and recurrent weights respectively.

The input gate and a vector of new candidate values (see Figure 12) are used to update the cell's internal state. For the input gate, the input and previous hidden state are processed by a sigmoid layer to determine which information needs to be updated. For the vector of new candidates, the input and previous hidden state is processed by a hyperbolic tangent layer, which output values between -1 and 1. The output of the sigmoid and tanh layer are multiplied. The output of the sigmoid layer determines which information from the tanh output layer is important. The input gate $\boldsymbol{i}_t^j$ and the vector of new candidate values $\tilde{\boldsymbol{C}}_t^j$ operations are mathematically expressed as:

$$\boldsymbol{i}_t^j = \sigma\left(\boldsymbol{b}_i^j + \boldsymbol{U}_i^j \boldsymbol{x}_t + \boldsymbol{W}_i^j \boldsymbol{h}_{t-1}^j\right), \tag{A.2}$$

$$\tilde{\boldsymbol{C}}_t^j = \tanh\left(\boldsymbol{b}_C^j + \boldsymbol{U}_C^j \boldsymbol{x}_t + \boldsymbol{W}_C^j \boldsymbol{h}_{t-1}\right). \tag{A.3}$$

Now the cell state can be calculated as the sum of the previous cell state multiplied by the forget layer output and the scaled new candidate values (see Figure 13):

$$\boldsymbol{C}_t^j = \boldsymbol{h}_{t-1}^j \boldsymbol{f}_t^j + \boldsymbol{i}_t^j \tilde{\boldsymbol{C}}_t^j. \tag{A.4}$$

The output of LSTM cell is the hidden state ($\boldsymbol{h}_t^j$, see Figure 14), which is calculated by the product of the output gate ($\boldsymbol{o}_t^j$) and the cell's state:

$$\boldsymbol{o}_t^j = \sigma\left(\boldsymbol{b}_o^j + \boldsymbol{U}_o^j \boldsymbol{x}_t + \boldsymbol{W}_o^j \boldsymbol{h}_{t-1}^j\right), \tag{A.5}$$

$$\boldsymbol{h}_t^j = \boldsymbol{o}_t^j \tanh\left(\boldsymbol{C}_t^j\right). \tag{A.6}$$

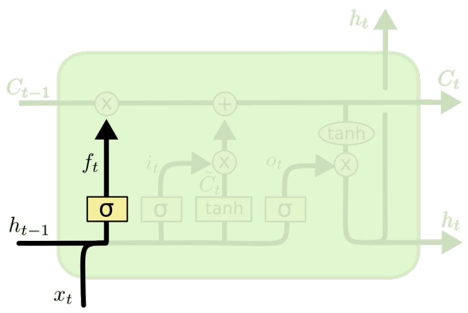


Figure 11: image retrieved from Olah (2015). Forget gate operations.

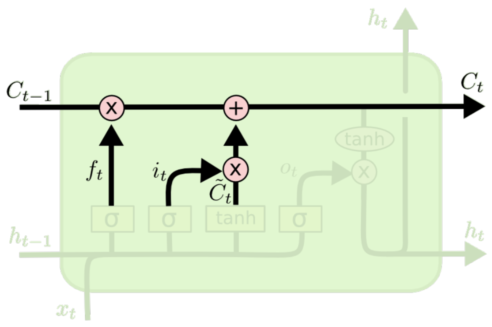


Figure 13: image retrieved from Olah (2015). Calculating cell state.

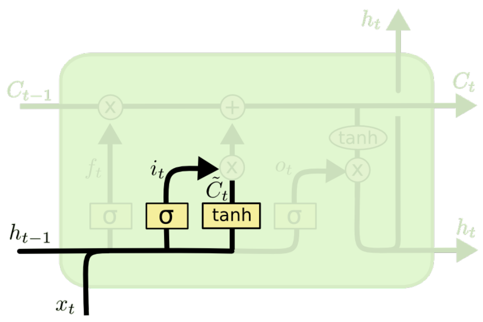


Figure 12: image retrieved from Olah (2015). Input gate operations.

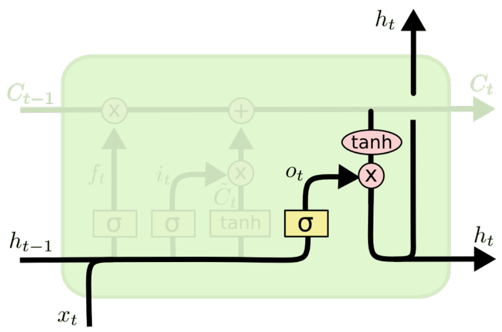


Figure 14: image retrieved from Olah (2015). Output gate operations.

Gers et al. (2000) expanded the LSTM model by connecting the cell state to the gates (so called peephole connection). Cho et al. (2014) introduced a variant of the LSTM model, the Gated Recurrent Unit (GRU) with two gates: the update gate and reset gate. In the GRU model the forget gate input gate are merged into an update gate, which controls which information to forget and to keep. The GRU does not contain a cell state, but only a hidden state. The reset gate decides which information is used to compute the next hidden state. For a review on LSTMs variant and their performance, see Greff et al. (2015) and Jozefowicz et al. (2015).