Offline-Online re-optimization using a hybrid method

S. de Vringer

TU Delft

Offline-Online reoptimization using a hybrid method

by

S. de Vringer

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday January 28, 2022 at 10:00 AM.

Student number: 4374851
Project duration: April 19, 2021 – January 28, 2022
Thesis committee: Dr. N. Yorke-Smith, TU Delft, supervisor
Dr. J. Alonso Mora, TU Delft
Dr. F. Broz, TU Delft

TU Delft

Preface

This project, and I suppose the entirety of my studies here at TU Delft, have been quite the experience. I am grateful for the opportunity to have experienced all that I have.

I would like to thank my father for his continued support. I would like to thank my friends, both offline and online, for keeping me sane throughout it all. More on the professional side, I would very much like to thank my supervisor Professor Neil Yorke-Smith, and Professor Wendelin Böhmer, for their excellent guidance.

S. de Vringer
Delft, December 2021

Contents

1 Introduction

Vehicle routing problems are their own entire field of study. They are a representation of an idea at the core of logistics, which is the transportation of goods through a number of vehicles or transportation units. Hence, vehicle routing problems can be used to formulate problems in a wide variety of contexts, ranging from automated warehouses to taxi rides and cargo delivery.

With the regards to the specific context of restaurant meal delivery, this thesis examines the central issue of the optimization of delivery routes. Delivery routes are determined by a centralized algorithm, which means that the problem parameters fit especially well when concerning online food delivery platforms. Recent years have seen strong growth in the use of online food delivery platforms (Ahuja et al., 2021). One recent development is the modelling of realistic problem settings in food delivery through the use of dynamic vehicle routing problems (M. Ulmer et al., 2020) (Ritzinger et al., 2016).

1.1. Problem relevance

The optimization of routes is essential for online food delivery platforms. Profit margins in the industry are low and competition is fierce (Ahuja et al., 2021). Better optimized routes lead to a higher amount of deliveries per vehicle over the same period of time, which has the potential to reduce costs in a number of areas. For example, it might reduce labour costs by reducing the number of drivers employed or the amount paid per delivery due if said pay is linked to travel distances. If vehicles are owned by the platform itself, it can reduce their operational cost by reducing fuel costs amongst others.

Other stakeholders can benefit from optimized routes as well. Restaurants typically need to pay a cost for orders received through online food delivery platforms. Better optimized routes increase the value of a platform to a restaurant, by increasing customer satisfaction through quicker deliveries. Delivery drivers can benefit greatly from optimized routes due to the fact that increased customer satisfaction can lead to higher tips.

From a perspective of academic research, vehicle routing problems are combinatorial optimization problems. Combinatorial optimization has been a field of study for many decades, and the same holds for many of the logistical problems that can be formulated as combinatorial optimization problems (Schrijver, 2005). Far from archaic, combinatorial optimization problems have even increased in relevance in recent years (Kacem et al., 2021). Autonomous vehicles, automated traffic regulation and scheduling are very active areas of research that are often at the forefront of algorithmic development. This work makes its contribution to the field of combinatorial optimization.

1.2. Problem statement and approach

This thesis tackles the *adapted restaurant meal delivery problem* (ARMDP). This is a vehicle routing problem in the class of pickup and delivery problems. It is an online planning problem that requires reoptimization over the planning horizon to account for the realization of stochastic information about

new customers and restaurant ready times. The employed solution method is a hybridization of deep reinforcement learning and adaptive large neighbourhood search. Deep reinforcement learning is employed to learn state values through temporal difference learning. Adaptive large neighbourhood search dynamically assigns weights to heuristics which function as actions for deep reinforcement learning during the online phase.

This work adds to body of work regarding the hybridization of learning and heuristics in the context of dynamic vehicle routing problems (DVRPs). It investigates the integration of both deep reinforcement learning and local search heuristics into a hybrid algorithm. Furthermore, it analyses the impact of the different method components on the performance of the hybrid algorithm. Afterwards, it gives an initial assessment of the practicality of the hybrid algorithm. These objective are expressed through the main research question and sub questions, which are formulated as follows:

How can a combination of reinforcement learning and handcrafted search heuristics be used to tackle a restaurant meal delivery problem?

1. How can the problem be formulated in a suitable manner?
2. In what manner can reinforcement learning and search heuristics be combined into a single hybrid algorithm for this problem setting?
3. What is the impact of different method components on the performance of the hybrid algorithm?
4. Could the hybrid algorithm be used in practice?

This thesis successfully adapts a model from the literature to fit the ARMDP. This model allows for the realization of stochastic information and the reoptimization of routes. The model has successfully been implemented for the purposes of algorithmic analysis. This work formulates a novel method for integrating both deep reinforcement learning and adaptive large neighbourhood search into a hybrid algorithm for the chosen problem setting. The algorithm combines learning with handcrafted search heuristics, and it combines offline computation with online computation. It concludes that the deep reinforcement learning component does currently not improve overall performance and establishes that the hybrid algorithm can be used to tackle small scale reoptimization problems.

1.3. Thesis structure

Chapter 2 establishes the context of both the ARMDP and the hybrid solution method. It establishes where the ARMDP fits into the existing literature and how it relates to other DVRPs. Furthermore, it examines prior work related the method components used, with an emphasis on hybridization and deep reinforcement learning. Chapter 3 establishes problem formulations for the ARMDP. A two-stage MDP is set forth as the basis for reinforcement learning and simulation of the online planning horizon. Chapter 4 delves into the details with regards to the hybrid solution method and its components. For reinforcement learning, it establishes states, actions, state transitions and rewards amongst others. For ALNS it examines the different handcrafted local search heuristics used and the manner in which heuristic weights are updated. For hybridization, this section establishes how the different components are integrated and the different solution methods compared. Chapter 5 deals with the experimental setup and experimental results. It discusses findings on the performance of the different solution methods. Finally, Chapter 6 aims to answer the research questions and provide directions for future work.

2

Related Work

This chapter addresses the most relevant related work that the author has found. Due to the extensive, diverse and rapidly-evolving body of DVRP literature, it is not possible to guarantee that all highly relevant works are included.

2.1. Dynamic vehicle routing problems

The vehicle routing problem (VRP) and its variants have been extensively studied for several decades (Psaraftis et al., 2016). Dynamic vehicle routing problems (DVRPs) are a problem class that more accurately model real transportation problems than static VRPs by addressing the need for reoptimization of routes. There is a great number of dynamic vehicle routing problem variants that can be used to accommodate different real world settings. One method of classifying DVRPs is by their constraints and their available information (Ritzinger et al., 2016).

2.1.1. The restaurant meal delivery problem

In this thesis, the specific problem variant examined is based on the restaurant meal delivery problem (RMDP) as formulated by M. Ulmer et al., 2020. M. Ulmer et al., 2020 classify the RMDP as a dynamic pickup and delivery problem (DPDP) and place it in the context of already existing works. Additionally, they have formulated the problem as an MDP. This MDP is convenient in that it allows one to easily adapt or extend their established problem formulation. Furthermore, it can be used in the application of deep reinforcement learning (deep RL).

Even though the RMDP has been introduced rather recently, there already exist a number of papers addressing this problem variant. A good overview can be found in F. Hildebrandt and Ulmer, 2020.

2.1.2. Other DVRP variants

As mentioned in the previous subsection, the problem formulation examined in this paper is based on the RMDP as formulated in M. Ulmer et al., 2020. However, DVRPs seem to vary in name based on specific details, so it is prudent to mention other names for problem classes that are closely related. The class of DPDPs is the class of VRPs concerned with transporting goods from origin to destination; and the same day delivery problem has each order originate from the same location (M. Ulmer et al., 2020). The dial a ride problem refers to a vehicle routing problem for the transportation of passengers from an origin to their destinations (Cordeau and Laporte, 2007).

Rather than try to provide an exhaustive overview of problem variants, subsequent sections in the related work will focus on the existing applications of the solution techniques used in this paper in the general area of DVRPs. For an elaborate overview of DVRPs, consult for example Psaraftis et al., 2016 or Ritzinger et al., 2016.

2.2. Hybrid solution methods

One important distinction in DVRP solution approaches is whether methods are offline or online. In the online phase there is a need for tentative routes (F. D. Hildebrandt et al., 2021). In this thesis, we aim to employ a hybrid approach consisting of both offline and online computation, as well both learning and a search algorithm. As such, this section focuses on such hybrid approaches in DVRPs, and combinatorial optimization in general.

Joe and Lau, 2020 provides a small overview of offline, online and hybrid approaches to DVRPs. They mention the limits of offline MDP-based approaches when it comes to tackling realistic problem variants. Secondly, they assert that pure online approaches might not function well when there is a limit on computation time. M. Ulmer et al., 2020 questions the scalability of a variable neighbourhood search in the RMDP given its online nature and the limited computation time.

F. Hildebrandt and Ulmer, 2020 mentions the surprising scarcity of online-offline methods incorporating machine learning in the field of VRPs. M. W. Ulmer, Goodson, et al., 2019 combines value function approximation with an online rollout algorithm.

Radaideh and Shirvan, 2021 is a relatively recent paper that combines reinforcement learning with search algorithms to tackle a real life combinatorial optimization problem. They find that RL can be used to guide a search procedure to promising areas in the search space, and that information can then be used by search algorithms.

In the field of DVRP, Ilin et al., 2015 provides an overview of approaches that combine AI with search algorithms. However, this overview seems limited given developments in recent years. One of the algorithm they mention formulates an initial greedy solution which is then used by variable neighbourhood search (VNS).

2.3. Reinforcement learning in vehicle routing

There are several ways in which RL can be used to solve VRPs in the literature, of which a number of approaches are listed in this section. A significant portion of the literature employing RL in VRPs aims to construct routes, not taking any tentative routes into account (Peng et al., 2020) (Zhang et al., 2020). An overview of RL in DVRPs is given in F. D. Hildebrandt et al., 2021, which also highlights current boundaries to employing said method.

In 2017, an MDP model was formulated for a DVRP that incorporates tentative routes into its formulation (M. W. Ulmer et al., 2017). This model has been the basis for several value approximation based approaches to solving VRP variants, including at least one that uses reinforcement learning (M. W. Ulmer, Thomas, et al., 2019) (Joe and Lau, 2020).

A prevalent use of RL in VRPs is in learning route orders, as in for example Zhang et al., 2020. Once again, these methods are often used in static settings. Furthermore, Sultana et al., 2021 uses RL to learn the value of vehicle item pairs, which subsequently aids in assignment decisions.

Lastly, of particular interest to this thesis, considering both the problem variant and employed solution method, is Jahanshahi et al., 2021. They use RL on a meal delivery problem and learn the best action out of a limited set from a vehicle perspective.

2.4. Adaptive Large Neighbourhood Search in vehicle routing

Adaptive large neighbourhood search (ALNS) is one of many search algorithms used in the context of VRPs. Ropke and Pisinger, 2006 uses it specifically in the context of a pickup and delivery problem. Since then, Lutz, 2014 used ALNS on a DVRP. Variable neighbourhood search has also been used for closely related DVRPs (Schilde et al., 2011) (Schilde et al., 2014). Grangier et al., 2021 uses a matheuristic based on large neighbourhood search to tackle a pickup and delivery problem.

3

Problem Formulation

This chapter explains the problem examined in this thesis. First, Section 3.1 give an explanation of a basic dynamic vehicle routing problem (DVRP). Section 3.2 an overview of the specific problem variant studied in this thesis. Section 3.3 covers an exact definition of the studied problem. Finally, Section 3.4 concludes with some implementation details connected to the definition of the problem.

3.1. The dynamic vehicle routing problem

The basic vehicle routing problem (VRP) can be thought of as an extension to the TSP, in which there are multiple vehicles that can visit the different cities.

The basic vehicle routing problem consists of n vehicles, c customers to visit, and one depot d. A solution to this problem consists of a route for each of the vehicles such that all customers are visited by one vehicle. A route consists of an ordered list of customers for the associated vehicle to visit. The depot d is the initial location from which all vehicles depart and the location to which each vehicle returns after it has visited its last customer.

In the literature, there are many different names associated with the locations to visit before returning to the depot, such as orders, destinations or customers. Given the context of the RMDP, this thesis will generally refer to an order as a request by a customer to have food delivered, or the food itself. A destination refers to a locations to visit, while customers and restaurants can refer to destinations where orders are either picked up or delivered respectively.

Travelling between the different locations or from/to the depot incurs a cost, which can be thought of as for example travel time or travel distance. The cost of a route taken by a vehicle is therefore determined by which customers it visits in what order. In the basic VRP, the objective is to minimize this cost in some manner. For example, to minimize the highest cost of any route, or to minimize the combined cost of all routes.

A basic dynamic vehicle routing problem extends the basic VRP by introducing changes to the problem over time. While vehicles travel, information about orders is revealed and routes must be adapted to account for these new orders. The crux is that new orders have an impact on the objective function which one aims to optimize the routes for. If the objective is atleast partially time related, the progression of time can both realise potentially stochastic information about routes, as well as introduce new elements that affect the objective function.

The DVRP has a need for re-optimization. Re-optimization of routes means that the DVRP is an online planning problem. The objective in re-optimization is to minimize the additional cost made by assigning the new customers to the vehicles. For example in a basic DVRP, this additional cost may be thought of as additional travel costs.

In the context of a DVRP, the exact coordinates of vehicles may be unknown at the time new orders are introduced or reoptimization is performed. As such, the location of a vehicle is often expressed in terms of either at a location or travelling between locations with an expected amount of time until arrival at the next location.

3.2. The adapted restaurant meal delivery problem

The previous section dealt with an introduction to a basic DVRP. This section discusses both the RMDP and the problem variant studied in this thesis. For convenience, the problem examined in this thesis will be referred to as the *adapted restaurant meal delivery problem*, or ARMDP for short.

The ARMDP is based on, and nearly identical to, the restaurant meal delivery problem as formulated by M. Ulmer et al., 2020. The RMDP was adapted to create the ARMDP in order to facilitate different experiments, with different parameters, constraints, and a different decision scope. This thesis is not in collaboration with any external partners, which has as an advantage that the problem can be formulated as desired.

An overview of the significant differences and similarities between the RMDP and ARMDP can be found in Table 3.1. The remainder of this section elaborates.

Problem characteristic	RMDP	ARMDP	Difference
Problem setting	Restaurant meal delivery	Restaurant meal delivery	No
Problem class	DPDP	DPDP	No
Planning horizon	480 minutes	400 minutes	No
Objective	Min. delivery window violations	Min. delivery window violations	No
Decision scope	Assignment decisions; irreversible	Full driver control	Yes
Driver waiting and repositioning	Basic strategy	None	Yes
Stochastic information	Customer realizations; order ready times	Customer realizations; order ready times	No
Capacity constraint	None	Uniform vehicle capacity	Yes

Table 3.1: Overview of the problem characteristics associated with the RMDP and ARMDP

In the RMDP, there is a number of drivers employed by an online delivery platform to pickup food orders from affiliated restaurants and deliver the orders to customers. Therefore it is sensible to define the planning horizon as encompassing an evening in which customers place orders that need to be added to the already-existing routes of the drivers making deliveries that evening. Orders cannot arrive in the last 30 minutes. This allows drivers to wrap up their already assigned deliveries.

The ARMDP is a dynamic pickup and delivery problem. The difference between a basic DVRP and a DPDP is that fulfilling a customer order requires visiting two locations. At the first location an order is picked up, after which it is delivered at the second location. For the ARMDP, these two locations are the restaurant and the customer respectively. Technically, it is possible that the same restaurant is visited once for multiple orders if the restaurant occurs multiple times successively in the planned route.

The same as in M. Ulmer et al., 2020, the objective in the ARMDP is to minimize the violation of the time window constraints. The time windows define the preferred moments of delivery of food to the customers. It is assumed there only exist maximal delivery window times. Minimal delivery window times would be illogical in this problem context. Additionally, it would complicate heuristics and not work well with the absence of any waiting strategy. It is assumed that restaurants do not have time windows for picking up food after it has been prepared.

Costs between locations are travel times. In addition to travelling from location to location, picking up

an order or delivering is also assumed to take constant time.

Each vehicle is operated by a driver. As such, it is assumed that the starting locations of vehicles for the initial problem are not assumed to be equal, which is the same as in M. Ulmer et al., 2020.

It is assumed that it is possible to control the route taken by drivers. However, the first destination in any route should never change, as this might create confusion for the drivers. This is a different approach from M. Ulmer et al., 2020, where routes are assumed to follow a general heuristic the drivers use. The assumption of full driver control can be justified through the idea that is always possible to only reveal the next location for a driver to travel to, even if other locations have already been assigned to them behind the scenes.

Furthermore, M. Ulmer et al., 2020 assumes that waiting and repositioning after orders are completed is done at the discretion of the drivers. In this thesis, vehicles do not wait nor reposition themselves after deliveries have been made.

There are many different types of information that can be realized over time. The RMDP as defined in M. Ulmer et al., 2020, as well this thesis, deals with stochastic customer realization and stochastic order ready times. The stochastic order ready times imply that it is possible for a driver to visit a restaurant to pick up an order before that order has been prepared. Whether an order is ready or not only becomes known after that order is successfully prepared at a restaurant. M. Ulmer et al., 2020 poses the additional restriction that a vehicle must be at the restaurant for any ready orders to be observed.

Finally, the formulation by M. Ulmer et al., 2020 does not include capacity constraints for the vehicles. In the ARMDP it is assumed that there is a maximum number of orders a driver can transport at once.

3.3. Mathematical formulation

This section contains a mathematical formulation of the reoptimization problem faced each time the routes are updated. There are plenty of formulations for vehicle routing problems already present in the literature. The following mathematical formulation is based on the formulation by Ramaekers et al., 2015.

Please note that this mathematical formulation on its own is not the foundation for the solution method. The mathematical formulation does not incorporate knowledge that can be extracted from the already-existing route plan, nor does it learn about the stochastic problem elements. Chapter 4 introduces an MDP formulation that allows for RL to tackle these issues.

Vehicles:
There are k vehicles. K denotes the set of vehicles and k refers to the k-th vehicle.

Relevant locations:
Let the set of locations be defined as $N = U \cup O \cup P \cup D$. $U = \{n_0\}$ is a dummy location that is the last location visited by each vehicle. Travel time to it will always be defined as zero. $O = \{n_1, ..., n_k\}$ is the set set of current locations for each of the k vehicles. $P = \{n_{k+1}, ..., n_{k+n}\}$ is the set of restaurant locations for uncompleted orders. $D = \{n_{k+n+1}, ..., n_{k+2n}\}$ is the set of customers associated with the orders, such that $n_j \in D$ belongs to the same order as $n_{j-n} \in P$. It should be noted that only orders for which a vehicle has not arrived at the customer are included in the problem formulation.

Decision variables and implied variables:
The set X contains $X_{i,j}^k \in \{0,1\} \quad \forall i \in N \setminus U, \forall j \in N \setminus O, \forall k \in K$. $X_{i,j}^k$ equals 1 if vehicle k travels from location i to location j, and 0 otherwise.

The set Y contains $Y_i^k \in \{0,1\} \quad \forall i \in N, \forall k \in K$. Y_i^k equals 1 if vehicle k visits location i, and 0 otherwise.

The set T contains $T_i^k \geq 0 \quad \forall i \in N, \forall k \in K$. If vehicle k visits location i, T_i^k denotes the expected

arrival time.

The set L contains $Q \geq L_i^k \geq 0 \quad \forall i \in N, \forall k \in K$. If vehicle k visits location i, L_i^k denotes the load it is carrying after a visit to that location.

Expected travel time:
$c_{i,j} \geq 0$ denotes the expected difference in arrival time between location i and location j. $c_{i,j} = wait(i) + at(i) + tt(i,j)$. $wait(i)$ is the expected waiting time before the pickup action at a restaurant can be performed. $at(i)$ is the time that still needs to be spent at a location to complete a pickup/delivery action. This is a known constant for both restaurant and customers. $tt(i,j)$ is the defined travel time between two locations.

Time windows:
β_i is given and denotes the maximal delivery window time for an order i.

Vehicle capacity:
It is assumed every vehicle has equal capacity Q. $Q \geq q_i \geq -Q$ denotes the integer change in load at location i, which is known. l_k denotes the initial load of vehicle k at its starting location.

Pickup status:
For some orders, the restaurant may already have been visited. $v_i \in V$ equals 1 when N_{k+i} should be visited, and is 0 otherwise. If $v_i = 1$, then $w_i^m \in W$ denotes that customer related to location N_{k+n+i} needs to be visited by vehicle m.

Set first locations:
If a vehicle is not idle and has planned destinations, the first destination should remain the same. G is the set containing $g_{k,j}$ if j is the first previously planned destination of vehicle k.

Other variables:
$f_{b,i}^k \geq 0$ and $T_{b,i}^k \geq \beta_i$ help define the objective function.

Objective and constraints:
Upon reoptimization, the objective is too minimize the additional time window constraint violations. As there are no minimal delivery time windows, this means minimizing the expected summed delay of all orders.

$$\min \sum_k \sum_{i \in D} (T_{b,i}^k - \beta_i) * Y_i^k \tag{3.1}$$

An arrival time should add to the objective value when it is outside the time windows.

$$T_i^k + f_{b,i}^k = T_{b,i}^k \quad \forall i \in D, \forall k \tag{3.2}$$

$$T_{b,i}^k \geq \beta_i \tag{3.3}$$

Arrival times are dependent on the arrival times at previous locations if they exist. This also eliminates subtours.

$$T_k^k = 0 \quad \forall k \in K \tag{3.4}$$

$$T_i^k = \sum_j X_{j,i}^k * c_{j,i} \quad \forall i \in N \setminus U, \forall k \in K \tag{3.5}$$

A solution sees each vehicle leave its current location, visit a number of locations related to orders, and finish at a final dummy location.

$$\sum_j X_{i,j}^k = Y_i^k \quad \forall i \in N \setminus U, \forall k \in K \tag{3.6}$$

$$\sum_{j \in N \setminus \{i\}} X_{i,j}^k - \sum_{w \in N \setminus \{i\}} X_{w,i}^k = 0 \quad \forall i \in N \setminus U \setminus O, \forall k \in K \tag{3.7}$$

$$\sum_i X_{i,0}^k = Y_0^k \quad \forall k \in K \tag{3.8}$$

Each vehicle starts at its initial location. Each location that needs to be visited is visited exactly once by a vehicle, except the dummy location to which all vehicles return (travel time to this location is defined as always 0, and vehicles are not required to move).

$$Y_k^k = 1 \quad \forall k \in K \tag{3.9}$$

$$\sum_k Y_i^k = 1 \quad \forall i \in N \setminus U \setminus P \tag{3.10}$$

$$\sum_k Y_i^k = v_{i-k} \quad \forall i \in P \tag{3.11}$$

$$\sum_k Y_0^k = k \tag{3.12}$$

If a restaurant still needs to be visited, a vehicle visits either both the restaurant and customer belonging to a certain order, or neither. These two locations needs to be visited in order. If a restaurant is already visited, the customer needs to be visited by the vehicle that picked up the order.

$$v_{i-k} * T_i^k \leq T_{(i+n)}^k \quad \forall i \in P, \forall k \in K \tag{3.13}$$

$$v_{i-k} * Y_i^k = v_{i-k} * Y_{(i+n)}^k \quad \forall i \in P, \forall k \in K \tag{3.14}$$

$$Y_i^k * (1 - v_{i-k}) = w_i^k \quad \forall w_i^k \in W \tag{3.15}$$

At no point on its route can a vehicle exceed its capacity.

$$L_k^k = l_k \quad \forall k \in K \tag{3.16}$$

$$L_i^k = \sum_j X_{j,i}^k * (L_j + q_i) \quad \forall i \in N \setminus O, \forall k \in K \tag{3.17}$$

$$L_i^k \leq Q \quad \forall i \in N \setminus O, \forall k \in K \tag{3.18}$$

The first destinations of the vehicles, if any, should remain as planned.

$$X_{k,j}^k = g_{k,j} \quad \forall g_{k,j} \in G \tag{3.19}$$

3.4. Implementation and simulation details

Python is used to implement the ARMDP and simulate the planning horizon. This section aims to answer possible questions with regards to its implementation that might arise given the problem formulation.

Vehicles are either idle, moving to a new location, waiting at a restaurant for an order, or performing the pickup or delivery action at a restaurant or customer respectively. The status of a vehicle is updated every time step.

Each location is given by a set of coordinates. The travel time between two locations is the Euclidean distance multiplied by 1.4 and rounded up. The 1.4 multiplier is taken from M. Ulmer et al., 2020. The frequency in which orders appear is determined by a Poisson distribution. This is also what is done in M. Ulmer et al., 2020. Furthermore, it is assumed that orders can only appear at locations that no order has appeared yet during the planning horizon. This is based on the idea that the same address probably will not order takeout multiple times a day. Similarly, orders cannot appear at the restaurant locations.

There is a minimal travel time between any two consecutive destinations in a route of one time step, even when those two destinations are the same restaurant. Furthermore, consecutive destinations at the same restaurant in a route do not change the action time the vehicle requires at each destination.

The time a vehicle spends at a location $at(i)$ to perform either a pickup or delivery action is constant. The time a vehicle expects to wait at a restaurant when an order is not yet ready is determined by the time it expects to arrive at the restaurant:

$$wait(i) = \min(\max(t_{order} + 10, t_{curr} + 1) - t_{arrival}, 0)$$

In case an order is ready, the waiting time is always zero. The actual unknown ready time is from a uniform distribution between 5 time steps and 15 steps after an order has been received.

4

Solution Methods

This section describes the algorithmic solution methods examined in this thesis. Section 4.1 discusses the experimental simulation, reoptimization procedure and solution methods. Section 4.2 and Section 4.3 discuss the role of the deep RL component and ALNS component respectively.

4.1. Full reoptimization procedure overview

Recall that the aim of this thesis is to examine different algorithmic solution methods that reoptimize an initial route plan for ARMDP into a different routing plan. During an experimental simulation, new orders appear over time. These orders must be added to the routing plan immediately, as no orders can remain unassigned. Reoptimization is the decision to use a solution method to reoptimize the routes, and the decision to reoptimize is separate from the initial insertion of new orders into the existing routes. This section gives an overview of the entire reoptimization process. Figure 4.1 illustrates the reoptimization process in broad terms.

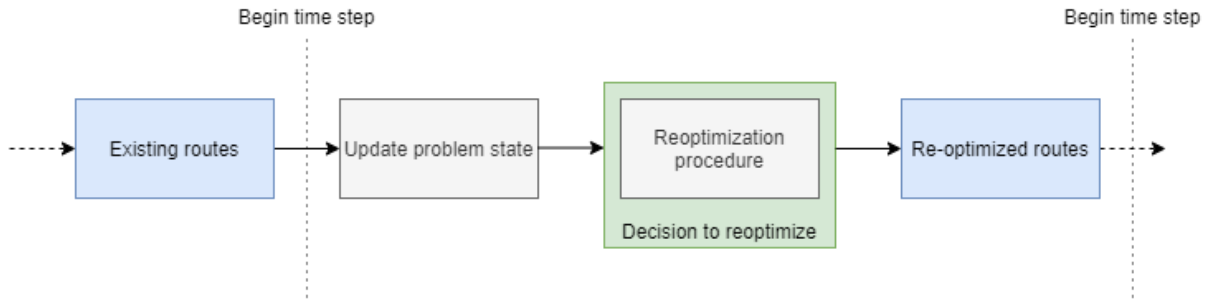

Figure 4.1: A time step in a simulation

The need for reoptimization springs forth from changes in the problem state. In this thesis, that is the arrival of new orders. While it is theoretically possible that realized waiting times lead to the need for re-optimization, this will usually not be the case in the performed experiments, as is discussed in Chapter 5.

The reoptimization problem can be viewed as consisting of two stages, based on the MDP formulation by M. Ulmer et al., 2020. In one stage, new information is realized through the progression of time. Vehicles perform actions and stochastic information is realized. In the other stage, routes may be re-optimized given the updated problem state.

In the first stage, vehicles move. The remaining travel time to the next destination might change, as might the expectation of how long a driver will remain at a destination. Orders might be completed, or become permanently assigned to certain drivers due the pickup of food at restaurants. Furthermore, stochastic information will be realized. Recall from Section 3.2 that this means that new orders appear and have to be assigned to drivers. To prepare for the reoptimization of routes, these new orders are

inserted in routes according to a simple insertion heuristic. Additionally, the expected ready times for pickups at restaurants might change as well.

In the second stage, one of the solution methods is employed to reoptimize the routes. A solution method consists of some form of simulated annealing (SA), which might use one or both of the major algorithmic components (deep RL or ALNS).

The aim is to have two major algorithmic components that are capable of complementing each other. The related work discussed the identified limitations of the reinforcement learning and search algorithms individually, and why the hybrid method is a promising alternative. The deep RL component is concerned with state evaluation, and requires a lot of offline computation. The ALNS component is concerned with the manner is which states are searched, and computes knowledge on the effectiveness of possible heuristics dynamically during the online phase. The idea is have deep RL function more globally, while ALNS allows for more progress locally.

The different solution methods employed in this thesis are discussed in Table 4.1. While the details of the individual components are discussed in subsequent sections, the ways in which the different components are combined can already be explained. It should be apparent from the description of the two algorithmic components given that they can operate both jointly and independently. Furthermore, distinction can be made on whether the best evaluation so far is determined by the neural network evaluation of the corresponding state, or simply the actual objective function.

Name	Description
None	No reoptimization.
Simple	SA using the heuristics (that are also used for ALNS).
ALNS	SA using the heuristics. The choice of heuristics is influenced by dynamic weights.
RL	SA using the heuristics. The best found solution found so far and the current solution in the search are both determined by the NN evaluation.
RL2	SA using the heuristics. The best found solution found so far is determined by the actual evaluation according to the weighted objective function. The current solution in the search is determined by NN evaluation.
FULL	SA using the heuristics. The choice of heuristics is influenced by dynamic weights. The best solution found so far, and the current solution in the search are both determined by the NN evaluation.
FULL2	SA using the heuristics. The choice of heuristics is influenced by dynamic weights. The best found solution found so far is determined by the actual evaluation according to the weighted objective function. The current solution in the search is determined by NN evaluation.
CONSEC	Functions as FULL2 until search step 100, and then transitions to ALNS.

Table 4.1: Different solution methods

In the case where deep RL and ALNS are used consecutively, it can be specified what type of state is a desirable starting starting state for ALNS through the deep RL reward function. In theory, this allows one to directly test how different starting states affect ALNS behaviour. Another way of examining how the deep RL component affects ALNS behaviour is by observing the difference in the probabilities assigned to heuristics in otherwise identical solution methods.

There is literature with approaches that are comparable in to the work done in this thesis. Joe and Lau, 2020 searches through state values learned via deep RL using simulated annealing. However, their actions do not concern the assignment of orders to vehicles, and instead only change the route of a single vehicle at a time. Radaideh and Shirvan, 2021 uses outcomes of the RL component as input to the second component of their hybrid methods in a sequential manner.

4.2. Reinforcement learning

This section discusses the deep reinforcement learning component used by several of the solution methods. A neural network is trained via reinforcement learning to learn the value of the problem states. In other words, the trained neural network acts as a value function approximator that evaluates problem states. The value function approximator can be used online by SA in order to evaluate states during the online search procedure. The work done is heavily based on Joe and Lau, 2020. The implementation of this algorithmic component is done using PyTorch.

4.2.1. The rule-based approach

The chosen type of reinforcement learning is rule-based reinforcement learning. This type of reinforcement learning allows specification of the reward function as a function of extracted input parameters. This means it is easy to customize what the neural network aims to learn. Alternative methods that observe rewards directly from the environment do not seem feasible here. One reason for this that the decisions made by a method rely on assumptions about stochastic problem elements. The expected result of such decisions changes when stochastic information is realised, or when further reoptimization procedures are performed. The outcome of a reoptimization procedure is a plan that will not necessarily be executed. Furthermore, the intention is to use the neural network as a component in different hybrid solution methods. A rule-based approach allows the search procedure used in the training of the neural network to differ from the search procedure used during the online planning phase.

The approach used is a form of deep reinforcement learning, for which the reward function is differentiable with respect to the input parameters. Of course, the trained value function approximator will never give state values as accurately as directly calculating the objective function and capacity constraint violations. However, the evaluations of the trained neural network incorporate information about the neighbourhood of a state. The idea is to have the RL component guide the search procedure to highly valued regions of the search space. As previously mentioned in the related work, Radaideh and Shirvan, 2021 has previously demonstrated that rule-based RL can be used successfully for this purpose.

As argued by Joe and Lau, 2020, the set of actions is dependent on the state of the problem. Furthermore, the set of actions is quite large and depends on how the designer wishes to the define the neighbourhood of a state. Therefore, value function approximation is preferable over methods that learn the values of state-action pairs.

4.2.2. States

The full state representation consists of static problem information as well as planned routes. It is of such complexity and detail that it cannot function as the input to the proposed neural network. Therefore, state simplification is necessary and decisions have to be made with regards to what features are extracted from the state information.

The full state of a problem consists of the elements listed in Table 4.2. The majority of elements have been previously defined in the mathematical formulation. Nevertheless the notation for many elements has been altered a bit for the sake of convenience.

The stochastic information on both customers and order ready times at restaurants is included in this formulation as follows. The expected arrival times at the locations is dependent on the expected ready times of orders such that $T_k = \sum_{i=0}^{k-1} = wait(i) + at(i) + tt(i,j)$. A problem state already includes any newly realized customers, whom are inserted using the sequential insertion heuristic (H-I0) explained in Section 4.3 below.

Feature sets are extracted from the problem state given in Table 4.2. Different feature sets and associated reward functions have been experimented with. The details regarding this as well as the results can be found in Section 5.4. All feature sets work under the assumption that it is important that the extracted features support a reward function that can evaluate the state with respect to the objective function and constraints.

$o_k \in O$	The current locations of all K vehicles.
$p_i \in P$	The locations of restaurants for all n uncompleted orders.
$d_i \in D$	The locations of customers for all n uncompleted orders.
$v_i \in V$	$v_i = 1$ if the restaurant for an uncompleted order i still needs to be visited; and $v_i = 0$ otherwise.
w_i	If $v_i = 0$, w_i denotes the vehicle with which to visit the customer associated with order i.
$r_k \in R$	The currently planned route for each of the vehicles (implies the assignments and order of visits). Each route consists of a sequence of elements p_i and d_i. For example, $r_1 = (p_1, d_1, p_5, p_9, d_9, d_5)$.
$y_i \in Y$	$y_i = 0$ if an order i is unassigned. Otherwise $y_i = k$, where k is the vehicle to which order i is assigned (can be derived from the routes).
$T_k \in T$	T_k contains the expected arrival time at each location in the route of vehicle k.
$L_k \in L$	V_k contains the vehicle load after each location in the route of vehicle k.
$\beta_i \in B$	The maximal delivery time window for all orders i
M	The constant maximum vehicle capacity.
$q_j \in Q$	The change in load after a visit to location j in $P \cup D$.

Table 4.2: Full state representation of the ARMDP

4.2.3. Actions and state transitions

The set of possible actions needs to be defined for all states. Deciding on an action set entails making trade-offs. Complex actions or an expansive action set increase the computational effort required to perform a single action or all available actions. However, the upside of more complex actions is the amount of work performed per action. Elaborate actions allow us to move across the state space more quickly per action, reaching different parts of the state space faster.

In light of the hybrid approaches examined in this thesis, the action set is equivalent to the action set used in simple SA and ALNS. This means that the action set consists of all combinations of destroy and repair heuristics also used in the ALNS component of the algorithm.

There are potential disadvantages to the use of this set of actions. Firstly, with this action set actions are not necessarily reversible by other actions. Backtracking to previous solutions, or choosing solutions that are suboptimal in the short term to maximize long term reward might be more difficult. Furthermore, states might not be encountered often during the same reoptimization procedure, as each action aims to find a state that is better than the previous one.

An advantage of this action set is that is respects all conditions that ensure an action results in a feasible solution, without the need to incorporate any constraints in the reward function. The capacity constraint is a soft constraint. The following conditions have to be, and are, respected by the action set:

- No first destinations for any vehicle should be changed if that vehicle is already en route
- Pickups must happen before deliveries of the same order
- If both pickup and delivery have yet to be done, they must be planned to be done by the same vehicle
- If a vehicle has already picked up an order, that vehicle must also deliver that order

4.2.4. Reward function

The reward function for rule-based reinforcement learning is closely related the chosen extracted features. They work in conjunction, and as such are part of the experiments regarding feature sets as discussed in Section 5.4. For all reward functions holds that the observed reward is the difference between the evaluation of the new state and the previous state. Therefore, the difference between reward functions is the difference in how each state is evaluated.

In order to facilitate backpropagation, the reward function is a function of the input parameters to the neural network. This means that the reward function is a function of at least the input parameters related to the objective function and capacity constraint. The neural network will need to learn to respect the capacity constraint through the use of the reward function.

4.2.5. Offline and online computation

During the offline phase, the neural network is trained using temporal difference learning. This method allows us to observe the difference in the value of states involved in state transitions according to the specified reward function. The neural network architecture is simple and consists of linear transformations and rectifiers.

The outcome is a neural network used that can be used to evaluate states during the online phase. Details with regards to the training procedure are discussed in Section 5.4. There are a myriad of ways in the neural network can be used in conjunction with other methods. The ones examined in this thesis see RL and ALNS used both sequentially and concurrently.

4.3. Adaptive large neighbourhood search

This section discusses the adaptive large neighbourhood search (ALNS). This is the second major algorithmic component used by several of the solution methods.

4.3.1. Search algorithm design considerations

Recall that the aim of this thesis is to develop hybrid algorithms that incorporate both learning and heuristics. The reinforcement learning component deals with how solutions are evaluated. Therefore, the other major component should influence the way is which new solutions are searched for.

There is a nearly endless amount of search algorithms, many of which have already been applied to the broader domain of DVRPs (Psaraftis et al., 2016). This includes the adaptive large neighbourhood search and a number of closely related algorithms. In fact, Lutz, 2014 is a work that provides an extensive set of heuristics that are used on a closely-related problem variant. The ALNS employed in this thesis is based on Lutz, 2014.

One major advantage of ALNS is its explainability. The heuristics used are hand-crafted, which allows us to customize the action set for hybrid solution methods that incorporate deep reinforcement learning. The set of heuristics can easily to adapted or extended. Furthermore, the performance of individual heuristics is expressed directly by the weights and probabilities assigned to them during the optimization procedure.

4.3.2. ALNS design

ALNS functions similar to a local search algorithm. The search procedure starts by considering a set of routes as the initial solution. The initial solution is altered through a number of steps. In each step, the current solution is changed through the combination of a destroy heuristic and a repair heuristic to create a new, slightly different solution. This new solution can replace the current solution as the starting point for the remainder of the search steps if certain conditions are met.

In other words, for a number of steps the algorithm tries to adapt a solution into a new solution. During the experiments performed in this thesis, the number of steps was set to 400. After this process has been completed, the best found solution as determined by the relevant evaluation metric is chosen as the resulting routing plan.

The condition that determines whether to use a newly-found solution for the next search step is as in simulated annealing. In the basic ALNS solution method, if the evaluation of the newly-found solution is higher with regards to the function given by Formula 4.1, the new solution is used. In a hybrid approach, the Formula 4.1 can be replaced with the evaluation of the state given by the neural network. If the newly found solution is evaluated worse as the current solution used for search, they might still replace them. The chance of this happening is decreases linearly over time.

$$eval(S) = -(100 * \sum_{k} violation(r_k) + \sum_{k} expectedDelay(r_k)) \tag{4.1}$$

The crux of the ALNS algorithm is the dynamic manner in which the current solution is repeatedly adapted to a new solution. A number of destroy heuristics and repair heuristics have been defined. To adapt one solution into another solution, one destroy and one repair heuristic are both randomly selected. The chance that a particular destroy or repair heuristic is selected is equal to its share of the total weight for all destroy heuristics or repair heuristics respectively.

At the start of the search procedure, all destroy heuristics have equal weight, as do all repair heuristics. Any improvement in the evaluation of a state that results from the adaptation of a solution through a combination of a destroy and repair heuristic will be recorded. After 70 steps of the algorithm, the weights of the different heuristic are updated by taking into account the summed improvement that they have resulted in. The weights for destroy heuristics and repair heuristics are updated separately, but similarly using Formula 4.2. In Formula 4.2, $W_{h,n}$ is the weight of heuristic h after weight update n, OVS is the parameter that determines how much the new value resembles the old value and how much it is

dependent on the recently collected weight, and $share_{h,n}$ is the share of improvement belonging to a destroy or repair heuristic.

$$W_{h,n+1} = OVS * W_{h,n} + (1 - OVS) * share_{h,n} \tag{4.2}$$

4.3.3. Heuristics

The heuristics used are heavily based on the work by Lutz, 2014. As in Lutz, 2014, there is a set of destroy heuristics and repair heuristics. Destroy heuristics are heuristics that remove some of the planned destinations from the current routing plan. Repair heuristics reinsert these destinations into the vehicle routes. Each heuristic satisfies the requirements for valid tentative routes as laid out at the end of Section 4.2.3. One of the heuristics might introduce further capacity constraint violations, while one repair heuristic can deal with solving capacity constraint violations.

The complexity of the set of heuristics is dependent on the choice of heuristics. For the destroy heuristics, one should expect the computational complexity to atleast be linear with regards to the total number of destinations. This is due to the fact that calculating criteria for different destinations involves iterating over said destinations. Lutz, 2014 discusses punishing complexity by incorporating execution times into the weight updates, which is not done in this thesis. For repair heuristics, the complexity scales with the amount of insertion options available. It has been implemented that should the amount of insertion options at any insertion exceed 100, a random subset of 100 insertion options will be considered instead of all options.

Destroy heuristics:
The following heuristics are the destroy heuristics. They remove a number of destinations from the routes. For all destroy heuristics, n has been set to 3. For all destroy heuristics it holds that only requests associated with atleast one destination that can be removed may count towards n.

Worst deadline violation removal (H-D1)
Removes all destinations related to the n requests with the worst deadline violations given the expected customer destination arrival times. If there are less than n requests for which delay is expected, only destinations related to those requests for which delay is expected are removed.

Worst capacity violation removal (H-D2)
Removes the destinations for which the capacity constraint is violated by the highest amount immediately after arrival. The destinations that it attempts to remove are related to n different orders, or less if there are less orders with destinations that violate the capacity constraint. This is the only heuristic that explicitly deals with solving capacity constraint violations.

Worst waiting time removal (H-D3)
Removes the destinations for which the estimated waiting time upon arrival is the highest. The destinations that it attempts to remove are related to n different orders, or less if there are less orders. It can happen that destinations associated with an estimated waiting time of 0 are removed.

Worst associated travel time removal (H-D4)
Removes all destinations for which the associated travel time (consisting of time-to, estimated waiting time, and time-from) is the largest. This metric excludes the first and last destinations from any routes. The destinations that it attempts to remove are related to n different orders, or less if there are less orders.

Related location removal (H-D5)
Removes a number of randomly chosen destinations which are located within a randomly chosen 0.4 by 0.4 area on the map. The destinations that it attempts to remove are related to at most n different orders, less if there are less destinations in the chosen area.

Related departure time removal (H-D6)
Removes a number of randomly chosen destinations of which the expected arrival times are close to

that of a randomly chosen destination that is not first in a route. The destinations that it attempts to remove are related to at most n different orders, less if there are less destinations that meet our definition of close. A destination is considered to have a expected arrival time that is close to another expected arrival time if there is a difference of less than twice the time it takes to cross the map.

Related restaurant removal (H-D7)
Removes all destinations related to n orders of which the pickup location is that of a randomly chosen restaurant. If there are less than n orders related to the chosen restaurant, less destinations related to orders are removed.

Repair heuristics:
The following heuristics are the repair heuristics. Repair heuristics reinsert a number of destinations removed by one of the destroy heuristics. Each repair heuristic can be used to insert any number of orders into the routes.

Balanced objectives (H-I1)
While there are still orders left to be inserted, this heuristic calculates the cost of a number of possible insertions of relevant locations for each order. It then performs the insertion that leads to the lowest additional cost. The cost function is the same as is used to evaluate a solution in ALNS, given by Formula 4.1. This means that the insertions by this heuristic could lead to capacity constraint violations.

Minimize regret (H-I2)
While there are still orders left to be inserted, this heuristic calculates the largest capacity constraint violations and expected route delays resulting from a number of possible order insertions. If there are less than $n + 1$ options with no capacity constraint violation for an order to be inserted into a route, the order with the lowest amount of such options is inserted. If all orders have atleast $n + 1$ insertion options without capacity constraint violations, then for each order the summed difference in route delay between the best option and n next best options is calculated. The order for which this value is the largest is then inserted at its insertion location(s) given by the best option. If it is possible to insert locations related to the orders without any capacity constraint violation, this heuristic will do so.

Minimal cost first (H-I3)
While there are still orders left to be inserted, this heuristic calculates the largest capacity constraint violation and expected route delays for a number of possible insertions. It then performs the insertion which is associated with the lowest largest capacity constraint violations, followed by the lowest route delay. If it is possible to insert locations related to the order without any capacity constraint violation, this heuristic will do so.

New order insertion heuristic:
The following heuristic is not used in ALNS, but is used to initially insert newly generated orders into the tentative routes.

Sequential insertion (H-I0)
Functions similar to the minimal cost first heuristic (H-I3), except it only calculates insertion options for a single order at a time. The order in which orders are inserted into the routes is the order in which they are given in a list.

5

Experiments

In this chapter, the performed experiments are discussed in detail. In summary, the experimental procedure consists of setting experimental parameters, generating training and test data, training the neural network, and running a number of experiments using each proposed optimization procedure.

5.1. Experimental parameters

Table 5.1 shows the simulation experimental parameters. *Map detail* refers to the number of locations that can be distinguished along a dimension of the map. In these experiments, the map is a 2D grid. The *number of vehicles* is the number of vehicles that travel around the map delivering orders. Each vehicle has its own planned route. The *number of restaurants* determines the different locations at which an order might be picked up for delivery. The *maximum capacity* determines the combined size of orders a vehicle can carry without incurring a penalty for exceeding the capacity constraint. All orders are between size 1 and 3. The *action time* is the number of time steps a vehicle stays at each destination it visits. This aims to imitate the vehicle performing the action of either picking up an order or delivering it. The *cross time* determines how many time steps it would take for a vehicle to travel from one end of the map to the other along one of the dimensions. The *maximum order arrival time* determines until what time step new orders can arrive. The *planning horizon* determines the minimum amount of time steps for which the simulation runs. The simulation will run for additional time steps if not all orders have been completed once the planning horizon has been reached. The *reoptimization frequency* determines how often the routes are reoptimized using the chosen solution method. Routes are no longer reoptimized after the maximum order arrival time has passed. The *number of initial orders* determines how many orders exist at time step 0, and therefore how many orders are present during the initial reoptimization procedure at time step 0. The *customer expectation* sets the expectation for the Poisson distribution that determines how many new orders appear at each time step. The *minimum and maximum order ready times* determine how long it can take for an order to become ready at a restaurant after it appears in the system.

Through experimentation, the parameters have been set such that the problem instances are algorithmically interesting. The number of initial orders has to be sufficiently large such that the performance of the different optimization procedures is distinguishable. The cross time needs to be sufficiently large such that paths between different locations are generally of different lengths. The downside of higher cross times is that vehicles cannot process a lot of orders per timestep, meaning the planning horizon needs to be increased such that the number of destinations in the planned routes is still significantly large. The decision has been made to have new orders appear at a frequency which means that the length of the routes stays roughly the same until the maximum order arrival time.

Another consideration was the expected cost of a simulation according to the objective function. Generally, one would aim to plan deliveries such that the deadline is rarely exceeded. However, in these experiments the deadline is very strict such that the performance of difference algorithms can be compared. Note that if deadlines for the orders were at exactly the moment they appeared, minimizing the

Simulation experimental parameter	value
Map detail	100
Number of vehicles	6
Number of restaurants	6
Maximum capacity	5
Action time	5
Cross time	20
Maximum order arrival time	200
Planning horizon	400
Delivery window size	60
Reoptimization frequency	80
Number of initial orders	60
Customer expectation	$2/15$
Minimum order ready time	5
Maximum order ready time	15

Table 5.1: The simulation experimental parameters

expected total deadline violation would be equivalent to minimizing the expected length of the routes. A disadvantage of high deadline violations is that vehicles pickup orders long after they have been prepared, meaning the influence of waiting times is almost non-existent.

Table 5.2 shows the parameters used in the search procedures. The *maximum search steps* is the number of search steps typically performed by a solution method. This parameter is relevant to all solution methods. A solution method incorporating ALNS may stop after a lower number of steps if between two ALNS weight updates no better solution is found at all. The *CONSEC switch step* is the search step at which that particular solution method switches how it functions. The different solution methods were discussed in Table 4.1. The *weight update frequency* determines how often the rewards collected by the ALNS heuristics are used to update the heuristic weights. This value is the result of a trade-off. It needs to be large enough such that each heuristic is used a significant number of times. However, a smaller value means the weights adapt to the situation more quickly. The *old value significance* determines the influence of the old weights of the heuristics when assigning new weights. If this value is 1, nothing ever changes. If this value is 0, the weights are dependent completely on the recently collected rewards. Therefore, this parameter must be set to a value in between these two extremes.

ALNS experimental parameter	value
Maximum search steps	400
CONSEC switch step	100
Weight update frequency	70
Old value significance	0.5

Table 5.2: The basic and ALNS experimental parameters

Table 5.3 shows parameters used in the training of the neural networks. The *batch size* determines the number of states used in an update of the policy network. *Gamma* determines the discount factor in the state value update equation. The *episode maximum length* determines the maximum amount of search steps during one episode, and is set equal to the maximum search steps parameter seen in Table 5.2. The *target network update frequency* determines how often the target network is updated in number of episodes. The *number of episodes* determines the maximum amount of episodes used for training.

NN training experimental parameter	value
Batch size	128
Gamma	0.95
Episode maximum length	400
Target network update frequency	10
Number of episodes	200

Table 5.3: Parameters used in training the different neural networks

5.2. Generating training and test data

Problem instances need to be generated for both training and testing purposes. The state of a problem instance can be visualized. An example of a state of the problem is given in Appendix A in Figure A.1.

In order to train the neural network, training data is generated. This training data is generated as follows. A number of simulations is run without reoptimization. The state of the problem is recorded at the time steps where reoptimization would normally occur. Then, during training random states are chosen from different simulations as the starting states of the episodes. This means that all starting states for training are not reoptimized. This is for simplicity's sake. During the actual simulations, states to reoptimize after time step 0 are influenced by earlier reoptimization decisions.

In terms of test data, a number of scenarios is created. The term scenarios in this instance refers to the list of orders that will appear at the different time steps. The insertion heuristic for orders uses a fixed random seed. As such, each solution method will have observe the same initial routing plan at time step 0 during the actual experiments.

The one difference between training data and testing data is that for the first simulations have to be performed, while that is not necessary for the latter. Scenarios have fixed vehicle starting locations and fixed restaurant locations. The orders that will appear during the simulation are generated beforehand using a Poisson distribution with the expectation as mentioned in Table 5.1.

5.3. Offline computations: ALNS

The set of heuristics used is not optimized with respect to its parameters. This would require significant effort, while the main point of this thesis is an experimental comparison between different types of solution for which the used heuristics merely form a basis. The *maximum amount of search steps* as given in Table 5.2 is similarly set. Experiments have been conducted for different values of *weight update frequency* and *old value significance* using the concept of successive halving, but it was different to draw conclusions on the performance of different parameter settings.

5.4. Offline computations: Training the neural network

As previously discussed in the RL section, the neural network aims to learn state values. It is trained based on the principle of temporal difference learning. The reward is the difference in evaluation of the two states involved in a state transition. Experiments have been conducted in order to observe the effect of different feature sets and methods used for finding the next state.

Two different methods have been tested for find the next state:

- **Arg max** - There is a small and decreasing chance that a random combination of destroy and repair heuristic is performed to find the next state. Otherwise, all combinations of destroy and repair heuristic are tried. In the latter case, the new state is either the resulting state with the highest discounted value plus reward, or the episode ends if no new state has a better evaluation as the old state.
- **ALNS step** - Determine the next state as the ALNS procedure would during the online phase, except using the evaluations by the neural network.

It has been found that the *Arg max* method leads to much lower training losses for feature sets FS1 through FS4 as shown in Table 5.4. This might be due to the inherent randomness of the *ALNS step* method. One could say it is the method that gives more information about what is the best that is possible with a state locally. As the way in which states are searched is already handled by the ALNS in hybrid solution methods, the *Arg max* method has been chosen for further experiments.

Table 5.4 shows the different feature sets. They are based on Joe and Lau, 2020. Descriptions of the different feature sets are given in the table itself. The four feature sets are similar in that they focus immensely on giving the actual performance of the vehicles with respect to the objective function. Initial experiments with another feature set that gives additional input features that do not directly influence the reward function were conducted, but regrettably there was not enough time to do so properly and results are not included in this work.

Each feature set corresponds to its own reward function, which is based on how states are evaluated as shown in Table 5.5. CM refers to a multiplier for constraint violations, and it is set to 500. C_k is the constraint violation input feature for a vehicle k. D_k is the expected delay input feature for a vehicle k. CCD_k sums both the capacity violations and expected delays into a single input for vehicle k.

Feature set	Description
FS1	This set of input features consists of the average capacity constraint violation per vehicle, as well as its maximum and standard deviation. Furthermore, it includes the average expected delay per vehicle, as well as its maximum and standard deviation. This means there is always 6 input features.
FS2	This set of input features consists of the expected delays and capacity constraint violations of the planned routes per vehicle. The number of input features is twice the number of vehicles.
FS3	This set of input features consists of the input features found in FS2, and an input feature which is the number of steps already performed in the search procedure. This is very similar to what is done in Joe and Lau, 2020.
FS4	This set of input features combines the expected route delay and number of capacity constraint violations (with a multiplier) into a single input feature per vehicle. Hence the number of input features is equal to the number of vehicles.

Table 5.4: Different feature sets used for training

Feature sets FS1 through FS4 show comparable training losses. A reason for this might be that all the aforementioned feature sets aim to predict the state value from input features that directly give information about the value of the current state as specified by the reward function.

Feature set	Corresponding reward function
FS1	$-CM * (C_{avg} + 2 * C_{max} + 2 * C_{stdev}) - (D_{avg} + 2 * D_{max} + 2 * D_{stdev})$
FS2	$-CM * \sum_k C_k - \sum_k D_k$
FS3	Same as for FS2
FS4	$-\sum_k CCD_k$

Table 5.5: State evaluations for reward functions corresponding to feature sets

5.5. Solution method performance comparison

The solution methods as discussed in Table 4.1 have been compared through a reoptimization experiment. The parameters of this experiment are as discussed in Section 5.1. A number of different metrics have been recorded, compared and analyzed. Each solution method has been used to solve the same set of 200 problem instances.

Figure A.2 shows the run times of the different solution methods at the different time steps, as well as their standard deviations. The standard deviations complicate interpreting the graph, so Table 5.6 shows the average portion of search steps in which the method listed in the row is faster than the method listed in the column, as well as the standard deviation of that value. It is shown that no reoptimization whatsoever is the fastest, and that the simple method tends to be faster than any other of the solution methods. Beyond that, it is difficult to distinguish the performance of the different methods. Additionally, recording other data for analysis might influence the already small differences. However, the data does allow us to conclude that the use of a trained NN does not hinder the computational performance of a solution method to a detrimental degree.

(avg, stdev)	None	Simple	ALNS	RL	RL2	FULL	FULL2	CONSEC
None	-	1.0, 0.0	1.0, 0.0	1.0, 0.0	1.0, 0.0	1.0, 0.0	1.0, 0.0	1.0, 0.0
Simple	0.0, 0.0	-	0.86, 0.03	0.59, 0.09	0.64, 0.05	0.91, 0.03	0.92, 0.04	0.85, 0.06
ALNS	0.0, 0.0	0.14, 0.03	-	0.06, 0.03	0.06, 0.05	0.76, 0.01	0.78, 0.01	0.56, 0.05
RL	0.0, 0.0	0.4, 0.09	0.94, 0.04	-	0.57, 0.07	0.94, 0.04	0.96, 0.03	0.91, 0.07
RL2	0.0, 0.0	0.36, 0.05	0.94, 0.05	0.41, 0.07	-	0.94, 0.05	0.96, 0.03	0.9, 0.07
FULL	0.0, 0.0	0.09, 0.03	0.24, 0.01	0.06, 0.04	0.06, 0.05	-	0.53, 0.01	0.29, 0.0
FULL2	0.0, 0.0	0.08, 0.04	0.22, 0.01	0.03, 0.03	0.03, 0.03	0.47, 0.01	-	0.26, 0.01
CONSEC	0.0, 0.0	0.14, 0.05	0.43, 0.05	0.09, 0.07	0.1, 0.07	0.71, 0.01	0.73, 0.01	-

Table 5.6: How often the column methods is faster as the row method

Next, a number of figures have been created to show the performance of the different solution methods in terms to the actual objective function. Figures 5.1, A.3 and A.4 show, over the different search steps and time steps, the performance with respect to the sum of the realised delay and expected delay as well as capacity constraints violations.

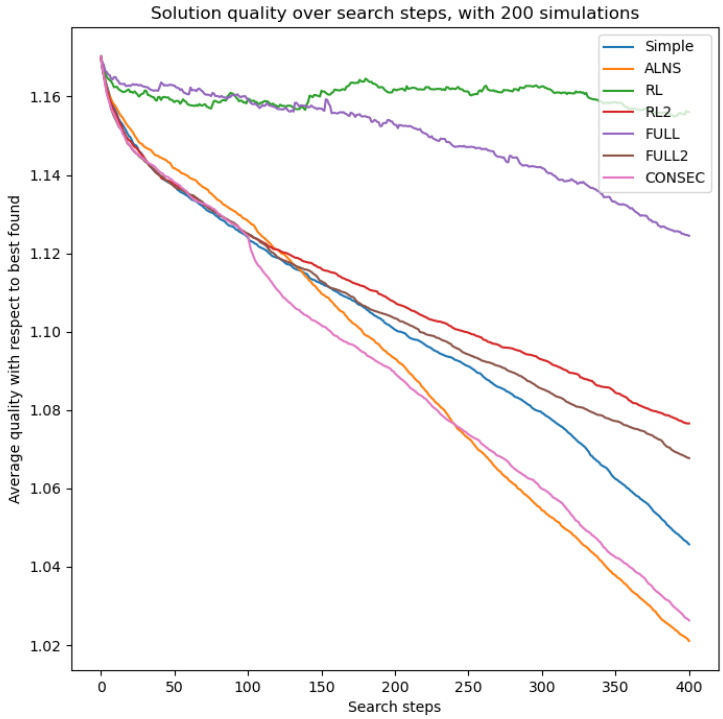

Figure 5.1: Solution quality in terms of expected delay and capacity constraint violations over search steps at time step 0

These graphs show only averages and no standard deviations, as these are enormous given the variety of problem instances and randomness inherent in choosing heuristics during a search procedure. The figures seems to suggest that the ALNS method is the most successful, together with CONSEC which functions identical to ALNS after the first 100 search steps. After that is the Simple method. Following that are the methods that track the best found solution using the actual objective function, but use the neural network in searching for possible next states. Last, and borderline detrimental, are the methods that use the neural network to both search for possible next states and keep track of the best found state.

As useful as Figures 5.1, A.3 and A.4 are for visualizing where differences arise in the search process, it is still necessary to examine the effect of the solution methods on both the realized delay and realized capacity constraint violations separately and in more detail. Table 5.7 shows how often a solution method results in lower realized delays when compared to another solution method. This information supports the ideas about the efficiency of different methods as supported by Figure 5.1. Figure A.5 show the realised delay for the different methods at the end of the planning horizon, but is less useful for interpreting as a result of the standard deviations.

	None	Simple	ALNS	RL	RL2	FULL	FULL2	CONSEC
None	-	0.0	0.0	0.30	0.0	0.15	0.0	0.0
Simple	1.00	-	0.28	0.98	0.77	0.95	0.68	0.28
ALNS	1.00	0.71	-	1.00	0.92	0.98	0.88	0.51
RL	0.70	0.01	0.0	-	0.06	0.34	0.05	0.0
RL2	1.00	0.23	0.08	0.94	-	0.83	0.42	0.09
FULL	0.84	0.05	0.02	0.67	0.17	-	0.13	0.03
FULL2	1.00	0.32	0.12	0.95	0.57	0.87	-	0.13
CONSEC	1.00	0.71	0.49	1.00	0.92	0.97	0.87	-

Table 5.7: How often the column methods result in lower realized delay as the row method at the end of the planning horizon

Figure A.6 shows the performance of the solution methods with respect to the number of capacity constraint violations. It clearly illustrates that methods which keep track of the best found solution through evaluations done by the neural network result in tentative routes that have more capacity constraint violations. Through experimentation, it has also been made apparent that the size of the penalty associated with each capacity constraint violation can influence how often they occur in the other methods, which is only natural. Figure A.7 shows the average frequency per search step interval for which which the different methods find new solutions at time step 0. Again, standard deviations are very high and the graph is messy, but it does seem to suggest that ALNS and CONSEC find the highest amount of new best solutions.

Under the assumption that ALNS performs the best, and better as FULL2, the weights given to the different heuristics over time have been compared between these two methods. It has been found that H-D1 is the best destroy heuristic, followed by H-D5 and afterwards H-D4 and H-D6. Destroy heuristics H-D2, H-D3, H-D4 do not seem relevant given their decreasing weights. In terms of repair heuristics, H-I1 and H-I3 are preferred over H-I2. To illustrate this, heuristic weights for the well-performing heuristics over the search steps at time step 160 have included in the Appendix in Figures A.8, A.9, A.10, A.11, A.12 and A.13.

5.6. Discussion

The performed experiments have demonstrated a way to incorporate deep reinforcement learning evaluations into a search procedure using heuristics. While more experiments would need to be performed to give exact numbers, run times for moderately sized instances are not detrimentally effected to a significant degree. This result applies given the complexity of the heuristics used during each search step. If implementation of the different heuristics is optimized further, or if changes to a solution are of a simpler nature, the evaluations performed may be a more significant factor in turns of the total run time.

Experimental analysis has demonstrated the use of reoptimization in the problem setting. It has shown

that simple simulated annealing is outperformed by dynamically updating the weight of the used though an adaptive large neighbourhood search. ALNS is the best performing method in terms of the objective function, outperforming any method that incorporates RL. In light of these results, further experiments that aim to optimize the heuristics and weight update rule used by ALNS, as well as additional experiments with regards to the run times of different heuristics seem worthwhile.

Differences in performance between different methods that incorporate RL also highlight the difference in performance between methods that use neural network evaluations for both searching the next state and keeping track of the best solution found so far, or only the former. It suggests it might be prudent to always have the best found solution evaluation via the actual objective function. Planned routes reoptimized by methods that record the best found solution according to neural network evaluations have much more capacity constraint violations as other reoptimized routes. Of course, tentative routes could be altered to solve capacity constraint violations in newly reoptimized routes at the end of the reoptimization procedure.

6

Conclusions and Future Work

6.1. Conclusions

The thesis has examined a restaurant meal delivery problem through the lens of a dynamic vehicle routing problem. It has formulated the adapted restaurant meal delivery problem (ARMDP), a problem at the crossroads of recent developments in both online food delivery and combinatorial optimization. A combination of reinforcement learning and adaptive large neighbourhood search has been used to create a different hybrid solution methods to tackle the ARMDP.

Chapter 2 contextualized the problem and the proposed hybridization within the existing body of work. Chapter 3 showed a two-stage MDP formulation that has been used as the foundation for formulating the solution method and implementing the problem simulation. Chapter 4 laid out the implementation of the different method components and their hybridizations. Chapter 5 compared different solution methods and components. Furthermore, it discusses the performance of the used solution methods. Finally, Chapter 6 will now proceed by answering the research questions and formulating possible future work. The sub-questions will be answered first, after which an overall answer is given to the main research question.

Recall the main research question:
How can a combination of reinforcement learning and handcrafted search heuristics be used to tackle a restaurant meal delivery problem?

Sub question 1:
How can the problem be formulated in a suitable manner?
This thesis has formulated a problem variant for the established restaurant meal delivery problem, referred to as the adapted restaurant meal delivery problem (ARMDP). It has adapted a mathematical problem formulation and an MDP-based formulation from the existing literature to suit the ARMDP. The ARMDP incorporates customers that appear during the online planning phase, and allows for reoptimization of the existing routes including the reassignment of orders to different drivers. An online planning simulation has been created in Python, such that vehicles are updated for the planning horizon, generated customers appear over time, and reoptimization of vehicle routes can occur at specified time steps.

Sub question 2:
In what manner can reinforcement learning and search heuristics be combined into a single hybrid algorithm for this problem setting?
A novel method for combining reinforcement learning and local search heuristics for ARMDP has been formulated. It has been shown that rule-based reinforcement learning can be combined with adaptive large neighbourhood search in several hybrid algorithms for this problem domain. It is possible to use reinforcement learning to make non-sequential decisions that might result in the reassignment of orders during an online planning phase. This reaffirms the literature in that reinforcement learning can

be used be used in the context of tentative routes. It also demonstrates the possibility of online order reassignment using reinforcement learning in this context. Furthermore, it has been shown that it is possible to distinguish between how a problem state can be evaluated as a point from where to search, and how is evaluated as a solution.

Sub question 3:
What is the impact of different method components on the performance of the hybrid algorithm?
It has been demonstrated that reoptimization can indeed improve the routes in terms of the objective over the planning horizon. Full ALNS outperforms SA using the handcrafted search heuristics. In the case where the input features are strictly related to the reward function using during training, state evaluations learned through reinforcement learning do not improve the solution quality when compared to ALNS.

Sub question 4:
Could the hybrid algorithm be used in practice?
Experimental results indicate that hybrid solution methods incorporating both ALNS and neural network evaluations can be used to tackle small scale online reoptimization problems. The added cost of neural network evaluations and adaptive weights are not factors that influence the run-time to a detrimental degree. Hence, this also means that ALNS, which is the best performing algorithm found, can be used for small scale problems in this context.

From the answers to the sub questions it can be concluded that it is possible to combine reinforcement learning and handcrafted search heuristics for the ARMDP in the described manner. However, the reinforcement learning component does not learn sufficiently from only input features directly related to the objective function as expressed by the reward function through the method employed in this thesis.

6.2. Future work

The conclusions indicate the following avenues for further research:

Other hybridization structures. As a hybridization of reinforcement learning and local search heuristics is relatively new to this problem domain, there are still many unexplored hybridization methods and aspects to these methods. Different hybridization methods using the established components could yield interesting results, and so could varying methods aspects such as the neural network architectures, training method or local search heuristics. Possible limitations of the search heuristics used as actions for deep RL are that they all aim to improve the existing solution and that they are not directly reversible by other heuristics. Furthermore, eliminating the requirement of having tentative routes during each step of the search procedure can be done if one assumes that the procedure will finish in the the time allocated for reoptimization. This might allow for more possibilities in designing a state and action space.

Other hybridization methods. There is a plethora of solution techniques in the field of learning and search algorithms. New combinations of dynamic vehicle routing problems and hybrid solution techniques could yet yield scientifically interesting results.

Working on practicality. There are still different questions with regards to the practicality of using a hybrid approach such as presented in this thesis. Additional analysis that focuses on the generalizability of the learned state values across different types of instances is required. Furthermore, the scalability of the method could be examined closer. It would be beneficial to cooperate with an actual food delivery platform for a number of reasons. It would allow for the collection of real data, facilitate the exchange of ideas and expertise, and present the opportunity to put the developed ideas into practice. Another recommendation would be to experiment with dynamically deciding when reoptimization should occur.

Human expertise. As mentioned in Joe and Lau, 2020, the reward function could incorporate human expertise in order to facilitate learning. Doing so requires the necessary expertise, and this could be a project in and of itself.

Bibliography

Ahuja, K., Chandra, V., Lord, V., & Peens, C. (2021). Ordering in: The rapid evolution of food delivery. https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/ordering-in-the-rapid-evolution-of-food-delivery

Cordeau, J.-F., & Laporte, G. (2007). The dial-a-ride problem (darp): Models and algorithms. *Annals OR*, *153*, 29–46. https://doi.org/10.1007/s10479-007-0170-8

Grangier, P., Gendreau, M., Lehuédé, F., & Rousseau, L. (2021). The vehicle routing problem with cross-docking and resource constraints. *J. Heuristics*, *27*(1-2), 31–61. https://doi.org/10.1007/s10732-019-09423-y

Hildebrandt, F., & Ulmer, M. (2020). Supervised learning for arrival time estimations in restaurant meal delivery.

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

Ilin, V., Simić, D., Tepić, J., Stojić, G., & Saulić, N. (2015). A survey of hybrid artificial intelligence algorithms for dynamic vehicle routing problem. In E. Onieva, I. Santos, E. Osaba, H. Quintián, & E. Corchado (Eds.), *Hybrid artificial intelligent systems* (pp. 644–655). Springer International Publishing.

Jahanshahi, H., Bozanta, A., Cevik, M., Kavuk, E. M., Tosun, A., Sonuc, S. B., Kosucu, B., & Basar, A. (2021). A deep reinforcement learning approach for the meal delivery problem. *CoRR*, *abs/2104.12000*. https://arxiv.org/abs/2104.12000

Joe, W., & Lau, H. C. (2020). Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers. In J. C. Beck, O. Buffet, J. Hoffmann, E. Karpas, & S. Sohrabi (Eds.), *Proceedings of the thirtieth international conference on automated planning and scheduling, nancy, france, october 26-30, 2020* (pp. 394–402). AAAI Press. https://aaai.org/ojs/index.php/ICAPS/article/view/6685

Kacem, P. D. I., Kellerer, H., & Mahjoub, A. (2021). Preface: New trends on combinatorial optimization for network and logistical applications. *Annals of Operations Research*, *298*. https://doi.org/10.1007/s10479-021-03957-w

Lutz, R. (2014). Adaptive large neighbourhood search. https://oparu.uni-ulm.de/xmlui/handle/123456789/3264

Peng, B., Wang, J., & Zhang, Z. (2020). A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. *CoRR*, *abs/2002.03282*. https://arxiv.org/abs/2002.03282

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

Radaideh, M. I., & Shirvan, K. (2021). Rule-based reinforcement learning methodology to inform evolutionary algorithms for constrained optimization of engineering applications. *Knowledge-Based Systems*, *217*, 106836. https://doi.org/https://doi.org/10.1016/j.knosys.2021.106836

Ramaekers, K., Caris, A., Maes, T., & Janssens, G. (2015). Pickup and delivery selection: Problem formulation and extension to problem variants. *Information Technology and Management Science*, *18*. https://doi.org/10.1515/itms-2015-0013

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

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, *40*(4), 455–472. https://doi.org/10.1287/trsc.1050.0135

Schilde, M., Doerner, K., & Hartl, R. (2011). Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. *Computers & Operations Research*, *38*(12), 1719–1730. https://doi.org/https://doi.org/10.1016/j.cor.2011.02.006

Schilde, M., Doerner, K., & Hartl, R. (2014). Integrating stochastic time-dependent travel speed in solution methods for the dynamic dial-a-ride problem. *European Journal of Operational Research*, *238*(1), 18–30. https://doi.org/https://doi.org/10.1016/j.ejor.2014.03.005

Schrijver, A. (2005). On the history of combinatorial optimization (till 1960). In K. Aardal, G. Nemhauser, & R. Weismantel (Eds.), *Discrete optimization* (pp. 1–68). Elsevier. https://doi.org/https://doi.org/10.1016/S0927-0507(05)12001-5

Sultana, N. N., Baniwal, V., Basumatary, A., Mittal, P., Ghosh, S., & Khadilkar, H. (2021). Fast approximate solutions using reinforcement learning for dynamic capacitated vehicle routing with time windows. *CoRR*, *abs/2102.12088*. https://arxiv.org/abs/2102.12088

Ulmer, M., Thomas, B., Campbell, A., & Woyak, N. (2020). The restaurant meal delivery problem: Dynamic pick-up and delivery with deadlines and random ready times. *Transportation Science*. https://doi.org/10.1287/trsc.2020.1000

Ulmer, M. W., Goodson, J., Mattfeld, D., Thomas, B. W., & May. (2017). Route-based markov decision processes for dynamic vehicle routing problems.

Ulmer, M. W., Goodson, J. C., Mattfeld, D. C., & Hennig, M. (2019). Offline–online approximate dynamic programming for dynamic vehicle routing with stochastic requests. *Transportation Science*, *53*(1), 185–202. https://doi.org/10.1287/trsc.2017.0767

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

Zhang, K., Li, M., Zhang, Z., Lin, X., & He, F. (2020). Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach. *CoRR*, *abs/2002.05513*. https://arxiv.org/abs/2002.05513

A

Other Figures

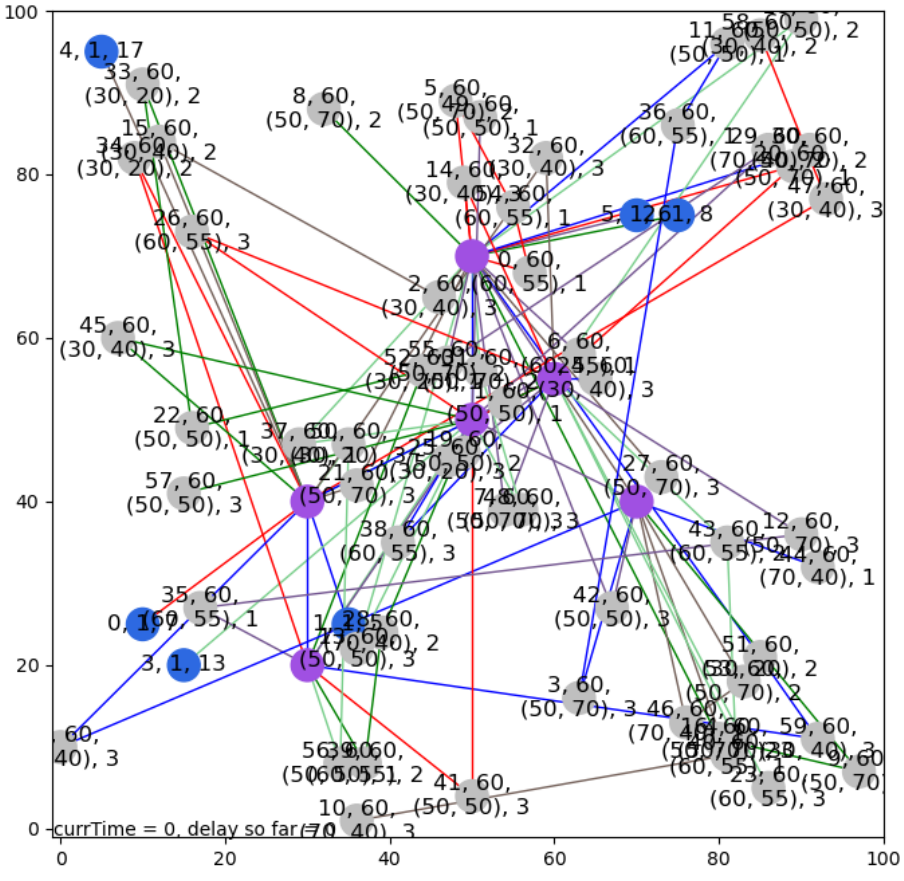

Figure A.1: Example of the state of a problem. Grey dots are customer locations. Blue dots are last known vehicle locations. Purple dots are restaurant locations. Different coloured lines are different planned routes

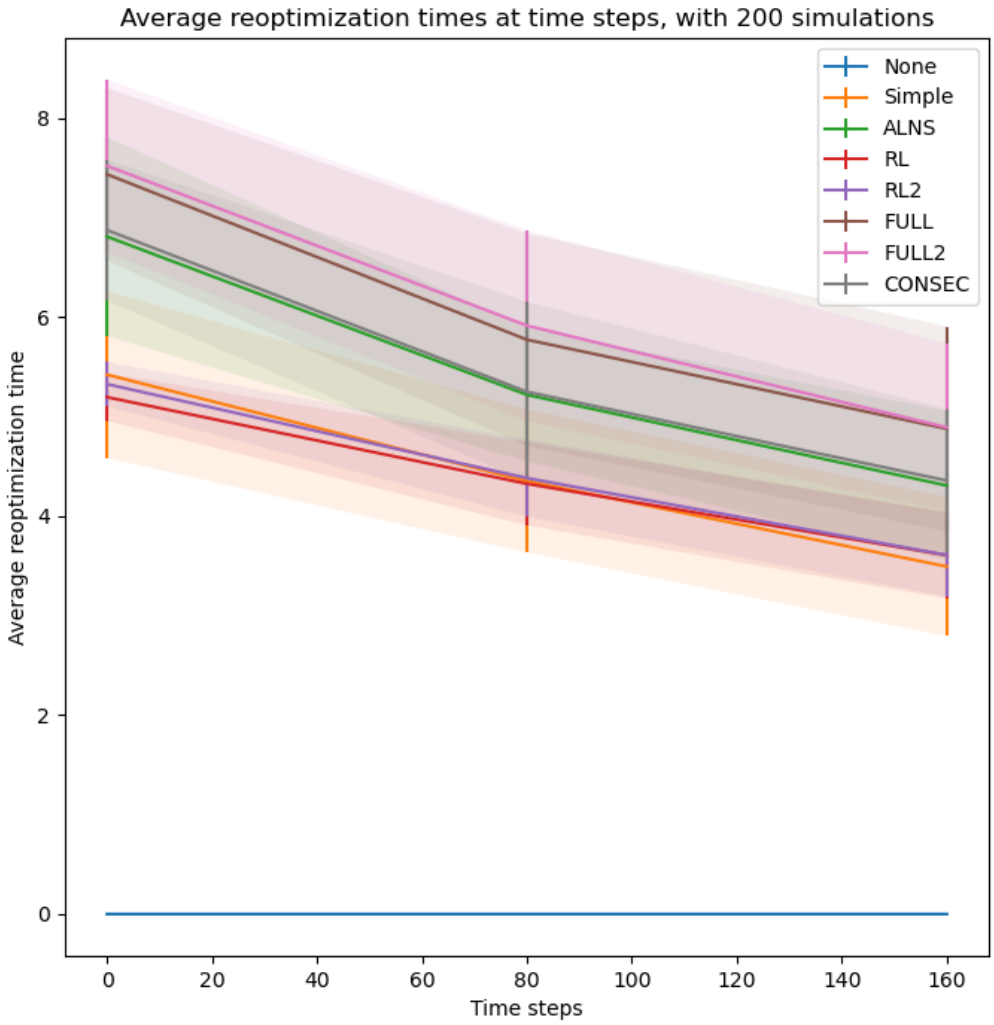

Figure A.2: Average reoptimization times for solutions at different time steps

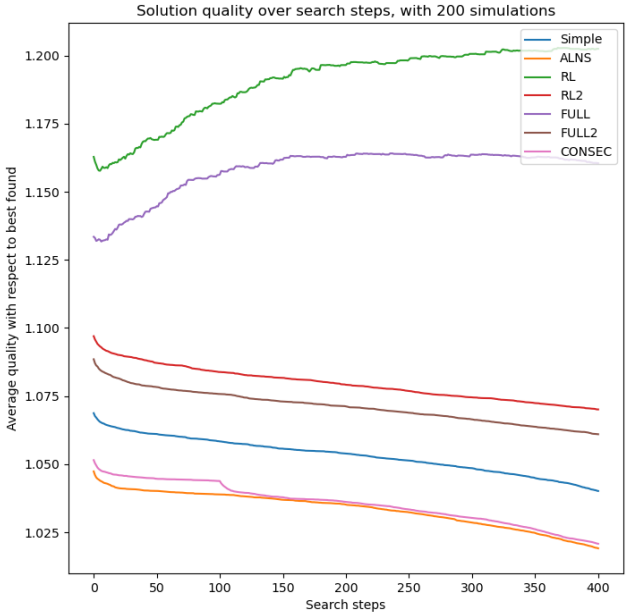

Figure A.3: Solution quality in terms of expected delay and capacity constraint violations over search steps at time step 80

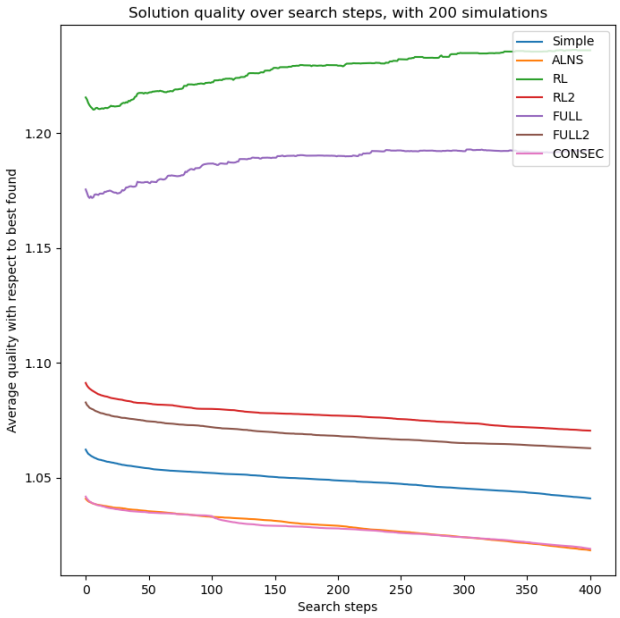

Figure A.4: Solution quality in terms of expected delay and capacity constraint violations over search steps at time step 160

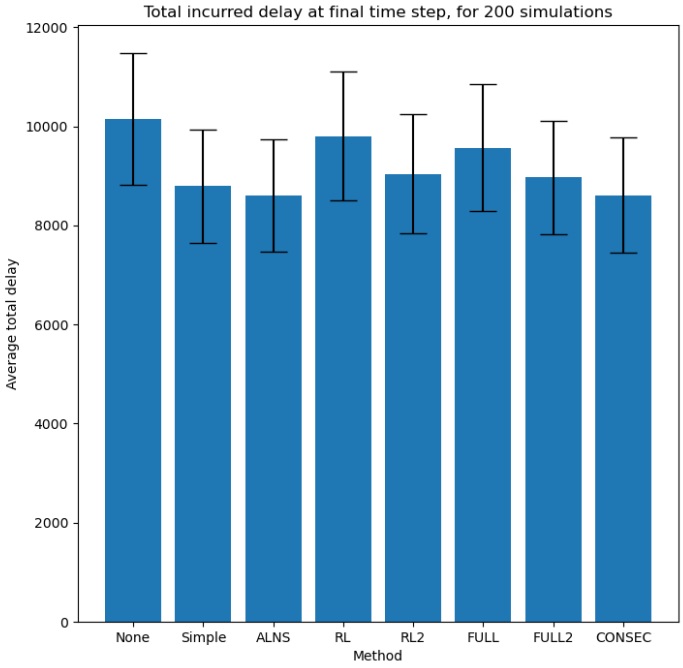

Figure A.5: Average realized delay and standard deviations for solution methods

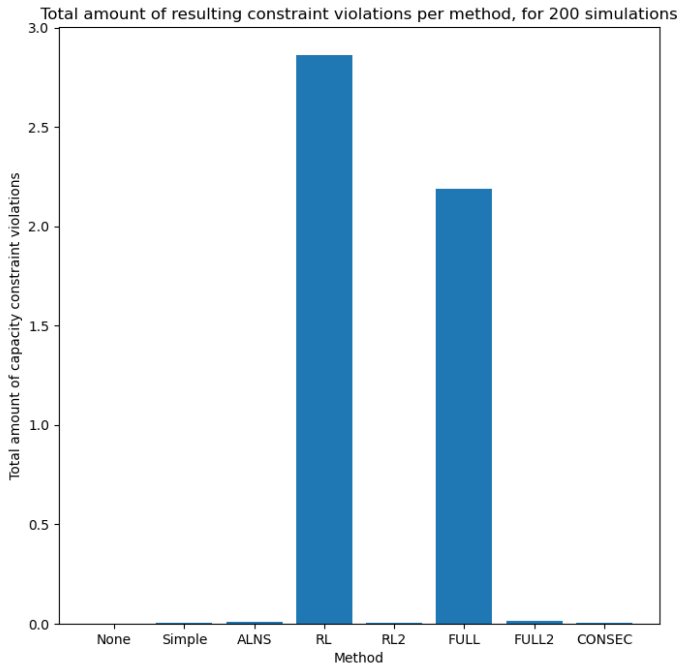

Figure A.6: Average total capacity constraint violation at the end of a reoptimization procedure

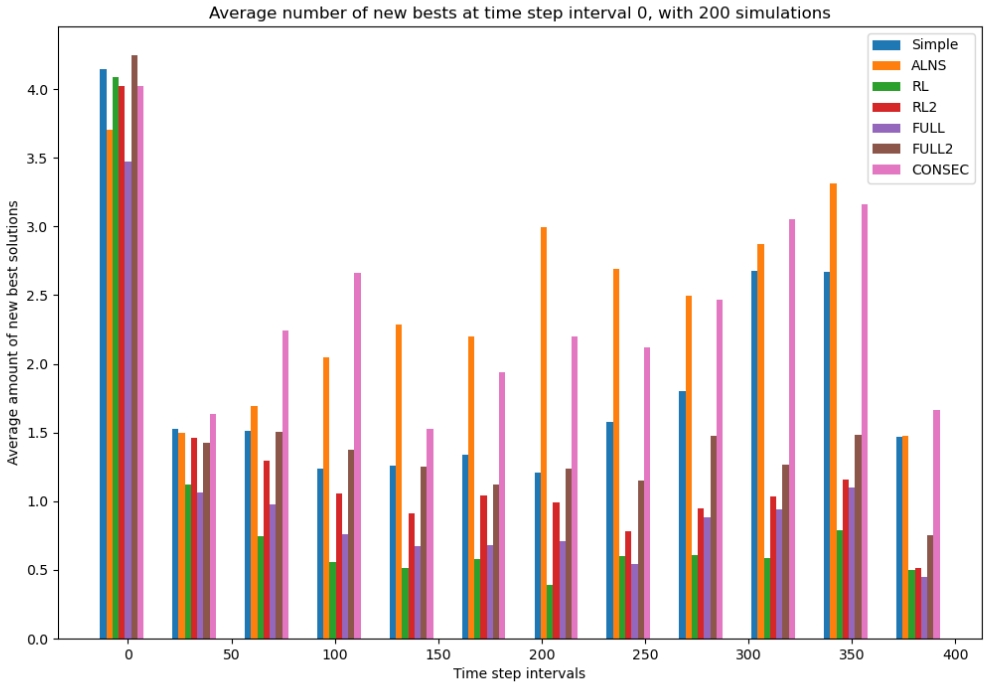

Figure A.7: Frequency at which different solution methods find new best solutions at time step 0

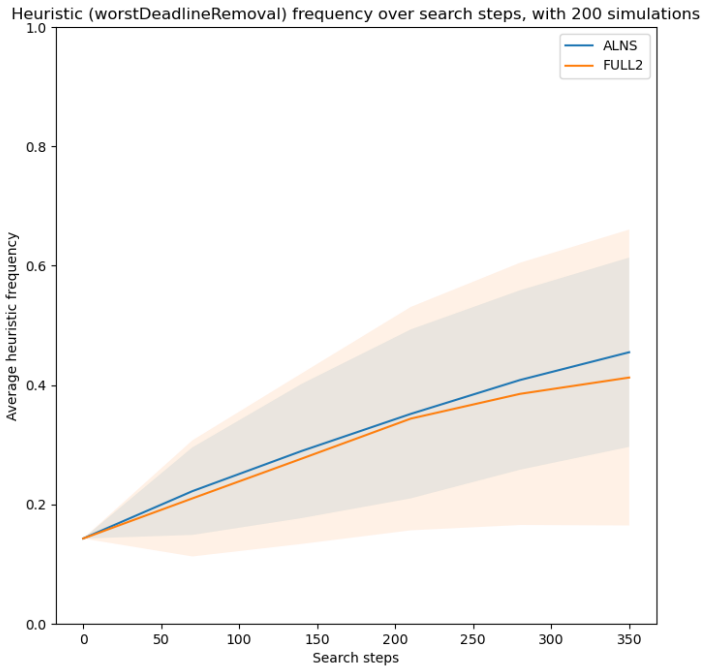

Figure A.8: Average weight of heuristic H-D0 over the search steps at time step 160

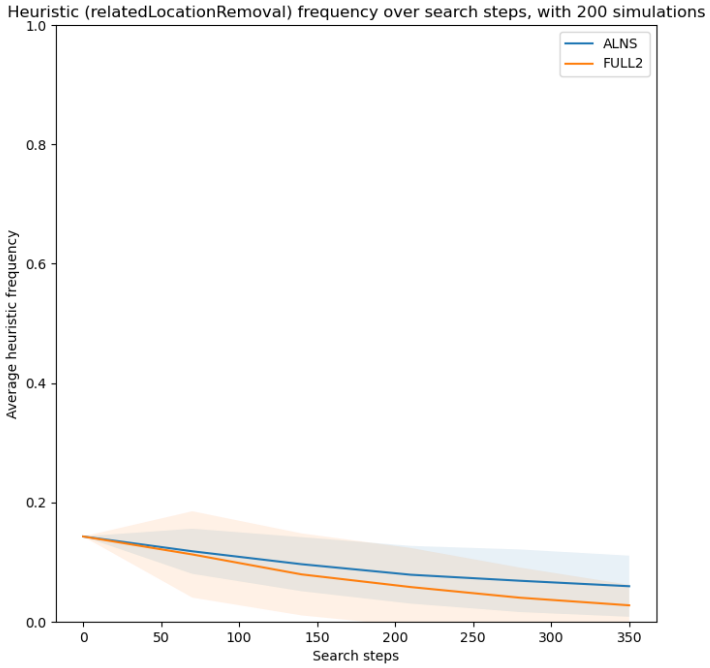

Figure A.9: Average weight of heuristic H-D4 over the search steps at time step 160

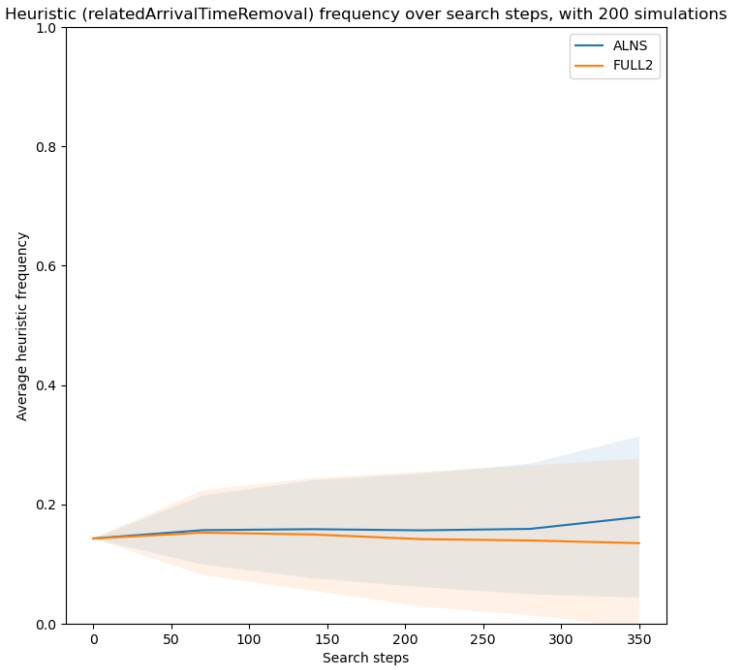

Figure A.10: Average weight of heuristic H-D5 over the search steps at time step 160

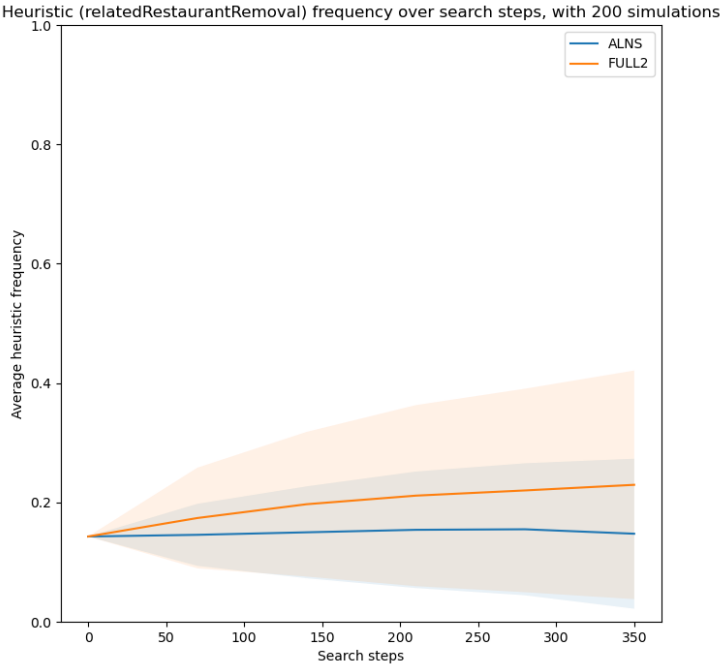

Figure A.11: Average weight of heuristic H-D6 over the search steps at time step 160

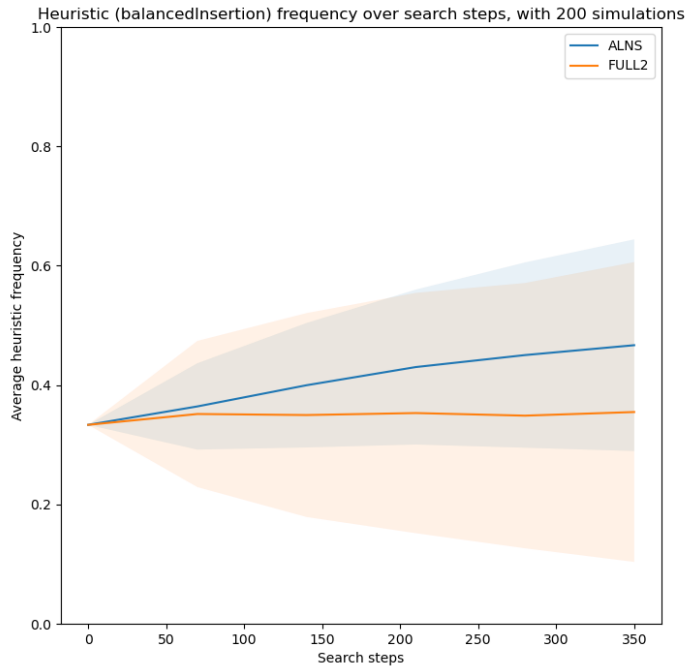

Figure A.12: Average weight of heuristic H-I1 over the search steps at time step 160

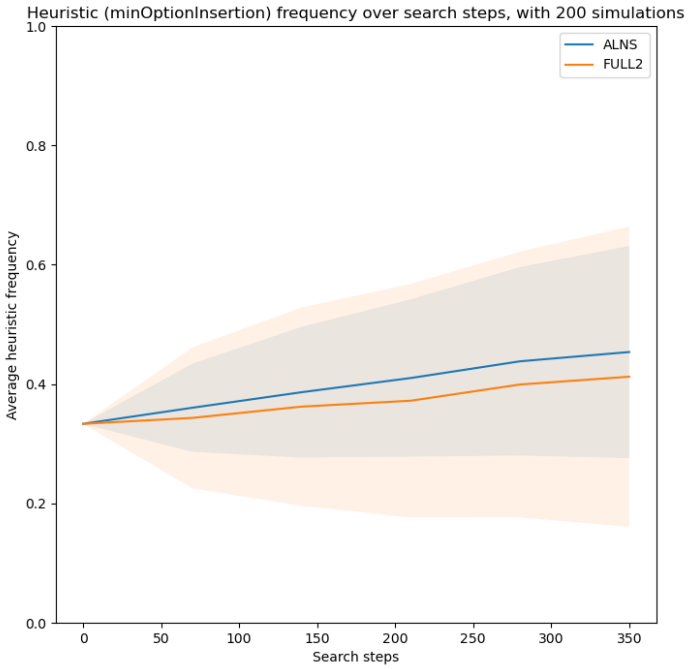

Figure A.13: Average weight of heuristic H-I3 over the search steps at time step 160