ERASMUS SCHOOL OF ECONOMICS

Master Thesis - Operations Research and Quantitative Logistics

---

The dynamic Pickup and Delivery Problem with Inside Transfers

---

Author:
Romaaike J.M. Schellekens
485136

Supervisor:
Dr. P.C. Bouman

Second reader:
Dr. T.A.B. Dollevoet

April 30, 2020

Abstract

In the dynamic pickup and delivery problem with inside transfers transportation requests must be executed by modular autonomous vehicles. A request is defined as the transportation of several passengers from a pickup point to a delivery point. As modular autonomous vehicle can couple while en-route, passengers can use an inside transfer while the vehicles are driving. Outside transfers, in which passengers are transferred between vehicles at a transfer point, are also considered.

The objective is to construct vehicle routes that execute as many requests as possible, such that the sum of the total energy consumption of the vehicles and the passenger inconvenience, coming from outside transfers, is minimized. The vehicle routes must satisfy time windows and capacity constraints as well.

In this thesis an adaptive large neighbourhood search solution method is formulated to solve this problem. This solution method is included in a rolling horizon approach, to handle the dynamic announcement of requests during the scheduling horizon. The solution method is tested on five instances of 50 requests on five different tree networks. The computational experiments show that inside and outside transfers are used in situations that they are beneficial.

Contents

1. Introduction

As the global urban population is growing, the need for urban mobility is increasing. Nowadays, 55 per cent of the world population lives in urban areas. This percentage is projected to increase to 68 per cent by 2050 (United Nations, 2019). In 2017, 77.2 per cent of the urban population in the European Union was exposed to concentrations of fine particulate matter above the WHO guideline value. For particulate matter and ozone, these percentages are 44,4 and 95 per cent, respectively (European Environment Agency, 2019). Globally, 25 per cent of urban ambient air pollution from fine particulate matter is caused by traffic (Karagulian et al., 2015). Traffic not only causes greenhouse gas emissions but also contributes to poor air quality and therefore poses a threat to health in urban areas.

Average privately owned car occupancy in The Netherlands was about 1.4 passengers per car in 2017 (Centraal Bureau voor de Statistiek, 2019). There are a little over five hundred passenger cars per thousand inhabitants in the European Union in 2017 (Eurostat, 2019). This high private car ownership, combined with low vehicle utility, claims a large part of public space in urban areas, as passenger cars are parked the majority of the time. As there is limited space in urban areas, car ownership competes with housing opportunities. Adopting more efficient and sustainable means of transportation could relieve part of the pressure on public space and improve air quality.

Due to advances in information and communication technologies, like the internet and smartphones, a shift in mass transit has already begun. Where traditionally mass transit only meant public transportation, nowadays many on-demand shared mobility services have emerged. Some of the services qualifying as on-demand shared mobility services are Transportation Network Companies (TNCs - like Uber and Lyft), car-sharing (like MyWheels, GreenWheels, SnappCar), ride-sharing (like BlaBlaCar) and scooter- and bike-sharing. However, these services do not address the ever-increasing traffic congestions in urban areas, they likely contribute to it. The average vehicle occupancy in TNCs is only 1.5 people, including the driver, according to a report based on the US situation (Schaller, 2018). In this same report, it has been found that only 40 per cent of passengers use the TNC services as an alternative for their own car or a taxi, 60 per cent of passengers would otherwise have walked, biked or used public transportation. Concluding, on-demand shared mobility services reduce the usage of other options that are more space-efficient and sustainable.

There is a need for systems in which passengers actually share their ride, to increase vehicle utility and decrease traffic congestion. However, as there are many alternatives, these systems will only be adopted by a broad public if the discomfort is minor. This means that the travel time compared to direct door-to-door transportation should be as short as possible. The uncertain waiting time when switching vehicles should be minimized as well. According to Garcia-Martinez et al. (2018) the penalty for using a single transfer is equivalent to 15.2-17.7 minutes of additional drive time. The passenger transportation concept introduced by Next Future Transportation Inc.

(2017) could improve comfort in a ride-sharing system. The modular autonomous vehicles of this concept can couple and decouple while en-route. Passengers can, therefore, transfer from one vehicle to another without any waiting time at all.

The problem considered in this thesis is how these vehicles can be routed to transport all the passengers in a way that minimizes the total energy consumption of the vehicles and the inconvenience for the passengers. This thesis will focus on developing a methodology to solve this vehicle routing problem. The remainder of this thesis is divided as follows: First, a review of relevant literature is presented in Chapter 2. The specific problem, a new variant on the pickup and delivery problem with transfers, is described in Chapter 3. A heuristic algorithm, based on the Adaptive Large Neighbourhood Search, to solve the problem, is proposed in Chapter 4. The results of some computational experiments, conducted to evaluate the performance of the algorithm, are given in Chapter 5. Finally, the thesis is concluded in Chapter 6 with a reflection on improving the developed algorithm and suggestions for further research.

2. Literature Review

This chapter describes some relevant literature related to pickup and delivery problem with inside transfers.

2.1 Vehicle Routing Problem

In the Vehicle Routing Problem (VRP) n customers need to be serviced from a single depot by m homogeneous vehicles. It can be represented as the following problem. Let $G = (V, A)$ be a complete and directed graph, where $V = \{v_0, ..., v_n\}$ is the set of vertices. Vertex v_0 represents the depot, and the remaining vertices represent the customers that need to be serviced. A is the arc set, defined as $A = \{(v_i, v_j) : v_i, v_j \in V, v_i \neq v_j\}$. There is also a cost matrix defined over A, where each arc $(v_i, v_j) \in A$ has non-negative costs c_{ij}, usually equal to the distance, travel time or travel cost. In many cases, these costs satisfy the triangle inequality (Toth & Vigo, 1998). The VRP consists of constructing one route for each of the m homogeneous vehicles based at the depot, such that each of the vertices is visited exactly once with minimum overall costs. A route could also entail that a vehicle does not leave the depot at all.

Since the introduction of the Truck Dispatching Problem by Dantzig & Ramser (1959), which became known as the VRP, it has been extensively studied. As this most simple vehicle route problem is already NP-hard (Lenstra & Rinnooy Kan, 1981), most studies have concentrated on developing heuristics and/or meta-heuristic to solve several variants of the problem. In these variants, additional side constraints are introduced with respect to the classical VRP. Extensive surveys on the solution approaches for the most common variants can be found.

- The capacitated VRP, in which each vehicle has a capacity Q (Laporte, 2009),
- The VRP with time windows, where customers should be serviced within a given time interval (Gendreau & Tarantilis, 2010).
- The VRP with multiple depots (Montoya-Torres et al., 2015).
- The VRP with heterogeneous vehicles (Baldacci et al., 2010).
- The periodic VRP, where customers require service on one or multiple moments within the planning period (Campbell & Wilson, 2014).

Of course, this is not an exhaustive list and combinations of variants are also common. For all these variants some parameters can be stochastic, common examples are stochastic demands, stochastic travel times or an uncertain set of customers (Gendreau et al., 1996). All these variants can also occur in a dynamic setting, this means that part of the input or all of the input is unknown at the start of the planning period and will become known during the planning period (Pillac et al., 2013).

The class of problems mostly related to the problem considered in the thesis are the Pickup and Delivery Problems. They either deal with the transportation of goods from the depot to customers and back, known as *one-to-many-to-one* problems, or the transportation of goods or people between pickup and delivery points. These points can be unpaired, known as *many-to-many* problems, or paired, known as *one-to-one* problems (Berbeglia et al., 2007). As the problem described in this thesis falls into this last category, this category will be examined more closely.

2.2 Pickup and Delivery Problem

The one-to-one pickup and delivery problem (PDP) can be defined on a complete and directed graph $G = (V, A)$. A set of n requests is considered. In this problem each request has to transported from a pickup point to a delivery point. The vertex set is now partitioned into $V = \{v_0\} \cup P \cup D$, where $P = \{v_1, ..., v_n\}$ is the set of pickup points and $D = \{v_{n+1}, ..., v_{2n}\}$ is the set of delivery points. Here it holds that pickup point v_j is paired with delivery point v_{n+j}, where $1 \leq j \leq n$. The depot is represented by v_0. To each vertex $v_i \in V$ a load q_i is associated, satisfying $q_i = 0$ for v_0, $q_i > 0$ for $v_i \in P$ or $q_i = -q_{i-n}$ for $v_i \in D$. This load is for instance the number of passengers that is to be transported, or the amount of parcels. A is the arc set, defined as $A = \{(v_i, v_j) : v_i = v_0, v_j \in P,$ or $v_i, v_j \in P \cup D, v_i \neq v_j$ and $v_i \neq v_{n+j},$ or $v_i \in D, v_j = v_0\}$. Each arc $(v_i, v_j) \in A$ has non-negative costs c_{ij} and travel time t_{ij}.

The PDP consists of constructing one route for each of the m homogeneous vehicles with capacity Q based at the depot, such that each of the vertices is visited exactly once with minimum overall costs. A route could also entail that a vehicle does not leave the depot at all. In addition to the constraints of the classic VRP, some additional constraints are introduced. The pairing constraint states that vertex v_i and vertex v_{n+i} should be visited by the same vehicle. The precedence constraint states that vertex v_i should be visited before vertex v_{n+i}. Also, the vehicle capacity should never be exceeded (Cordeau et al., 2008).

In dial-a-ride problems (DARPs) the transportation of people is considered. In these problems, usually, user inconvenience is also taken into account. This could, for instance, be included in the form of customer ride time, customer waiting time or the difference between actual and desired delivery times. (Cordeau & Laporte, 2007)

2.2.1 Static Problems

Due to the difficulty of the PDP most early work focused on the single-vehicle PDP. One of the first studies on the single-vehicle PDP was conducted by Stein (1978). They considered a variant with infinite vehicle capacity and proposed a simple heuristic. The heuristic consists of building a Travelling Salesman Problem (TSP) tour for the pickup request and a separate TSP tour for the delivery requests and then concatenating the two tours. Psaraftis (1983a) also proposed a heuristic for this problem based on a TSP tour, which has a worst-case performance ratio of 3. Psaraftis (1983b) introduced one of the first local search methods for this problem, the k-interchange procedure and showed how to find the best k-interchange in $O(n^k)$ time. Kalantari et al. (1985) proposed branch-and-bound procedures for both the capacitated and uncapacitated variants. They were able to solve instances up to 15 requests. Desrosiers et al. (1986) proposed an exact forward dynamic programming algorithm for the capacitated variant of the problem with time windows. They were able to solve instances with up to 40 vertices.

Psaraftis (1980) carried out one of the first studies on the single-vehicle DARP. In the variant they examined users wish to be serviced as soon as possible. A dynamic programming approach was developed to solve the problem exactly, but with a complexity of $O(n^2 3^n)$ it can only solve small instances. It was later extended in Psaraftis (1983b) to handle hard time windows.

For the multi-vehicle PDP with time windows also several exact algorithms were introduced. The column generation algorithm of Dumas et al. (1991) can solve instances with up to 55 requests. Xu et al. (2003) proposed a column generation algorithm that can deal with several real-life restrictions, like driver work rules, multiple time windows and compatibility constraints between carriers, vehicles and requests.

Li & Lim (2003) proposed a tabu-embedded simulated annealing algorithm. They also contributed by generating six different data sets, with in total 56 100-customer problem instances. Later, also 200-,400-,600-,800- and 1000-customer instances were generated, making a total of 354 instances. This is referred to as the (Li & Lim, 2003) benchmark, which has been used by many studies to compare the performance of algorithms.

Nanry & Barnes (2000) use a reactive tabu search approach, where first a simple insertion algorithm creates an initial solution. They designed three types of neighbourhood moves. The first move attempts to move a pickup-delivery pair from one route to another. The second move swaps pairs between two different routes. The third route is a polishing move and reorders customers within a route. A hierarchical search strategy based on the tightness of the time windows was developed to select the neighbourhood move. The first stage in the two-stage heuristic of Bent & Van Hentenryck (2006) consists of a simulated annealing algorithm to minimise the number of routes. In the second stage a large neighbourhood search (LNS) is applied. Here neighbourhoods are defined as the solutions the can be achieved by relocating at most p requests. This algorithm has produced 73 best solutions on the Li & Lim (2003) benchmark. Ropke & Pisinger (2006) also proposed a LNS heuristic. Their Adaptive LNS is embedded within a simulated annealing framework, meaning that the algorithm keeps track of the score for each neighbourhood operator during the search. Several removal and insertion operators are used during the same search. This heuristic has improved more than half of the solutions in the Li & Lim (2003) benchmark en generally outperforms the heuristic of Bent & Van Hentenryck (2006). Guided Ejection Search (GES), proposed by Nagata & Kobayashi (2010), only focuses on minimising the number of vehicles used. GES allows for partial solutions during the search by unassigning all pd-pairs from a randomly selected route. Combining the work of Ropke & Pisinger (2006), Nanry & Barnes (2000) and Nagata & Kobayashi (2010), Curtois et al. (2018) developed an algorithm which was able to find 142 new best-known solutions to the Li & Lim (2003) benchmark.

2.2.2 Transfers

In the Pickup and Delivery Problem with Transfers (PDPT) it is possible to change vehicle at certain transfer points. This means that the coupling constraint is relaxed. This is the vehicle routing problem that is most closely related to the problem that is considered in this thesis. However, in most implementations of the PDPT only a limited number of locations is assigned as a transfer location before solving the problem. In the problem considered in this thesis, all locations can be a transfer location and transfers can also take place while en-route. Shang & Cuff (1996) were the first to include transfers in the pickup and delivery problem. They proposed a heuristic method with a multi-criteria objective. Several studies developed exact methods to solve the PDPT. Kerivin et al. (2008) presented two mixed-integer linear programming formulations, both based on a space-time graph. They developed a branch-and-cut algorithm for the problem. A branch-and-cut algorithm was also developed by Cortés et al. (2010). In their formulation transfer nodes are split into two nodes, to handle the arrival and departure of a vehicle separately. Rais et al. (2014) proposed a new formulation, with a four-index binary variable to track demand. They distinguished between vehicle flows and request flows. With this algorithm, they could solve instances with up to 7 requests.

Since the PDPT is also a NP-hard problem, most studies focus on heuristic algorithms. According to Masson et al. (2014) feasibility checks become more difficult in the PDPT, since it is an interdependence problem due to the synchronisation at transfer points. They show that the feasibility problem can be modelled as a Simple Temporal Problem and find necessary and sufficient conditions to solve it in at most $O(n^2)$. In their problem the introduction of transfer points leads to savings of up to 8.24%. Sampaio et al. (2018) handle the feasibility check in a different way. After the insertion of a request in a route, they use two separate procedures to check the earliest times and latest times of all affected routes. This procedure stops when the insertion does not affect the earliest or latest time. In their research they find that transfers can provide significant benefits. This is especially true when requests with large distances and short vehicle shifts are combined.
Both studies extend on the ALNS of Ropke & Pisinger (2006) to also include transfer points. Qu & Bard (2012) combine ALNS with a greedy randomised adaptive search procedure. In the construction phase a request is added to a partially solution at random, selecting from the ones that increase the cost the least. Then in the optimization phase ALNS is used. In the studies mentioned above, the locations of the transfer points are selected before solving the problem. Deleplanque & Quilliot (2013) considered one dynamic transfer point, meaning that the location of the transfer point is not selected before solving the problem and can change from solution to solution.

More recently, PDPT's are studied that look at more innovative ways of transportation. Masoud & Jayakrishnan (2017) developed a decomposition algorithm for the multi-hop Peer-to-Peer ridematching problem. In this problem, drivers are not professionals, but private car owners who offer to share their car. This problem is essentially the same as the heterogeneous multi-depot Dial-A-Ride Problem. It is different in the way that the set of vehicles is not fixed, and as drivers are peers quality measures should also be extended to the drivers. They formulated it as a binary program in a time-expanded network. They introduced a pre-process procedure to limit the input, with this they could solve instances with up to 400 participants, including both drivers and passengers. Another way to include transfers in a PDPT is to combine peer-to-peer ride-sharing with public transit. This is done by Stiglic et al. (2018). The algorithm consists of a match identification phase followed by an optimal matching phase. In the first phase all feasible matches between drivers and passengers are determined and in the second phase the best combination is selected. Instances with 2000 participants were solved within a minute. Lotfi et al. (2019) investigate a variant of the PDPT, where passengers can choose if they are willing to ride-share and/or transfer. They used a modified column generation algorithm. They were able to solve problems with up to 70 passengers and show that number of served passengers goes up when the percentage of passengers willing to ride-share and/or transfer goes up.

2.2.3 Dynamic Problems

For a problem in the dynamic context, a part of the input is revealed or updated during the execution of the routes. For instance, the travel time can change, or vehicles break down. However, the most common source of dynamism is the arrival of requests during the execution of the routes. So we will look specifically at studies with this form of dynamism, with the emphasis on how they handle the arrival of these requests.

Wilson & Colvin (1977) were first to study a dynamic vehicle routing problem. Their insertion heuristic is used on the single-vehicle DARP. Psaraftis (1980) introduced the dynamic programming approach to periodic re-optimization of the DARP. It finds the optimal route every time a request arrives.

Most studies use a two-phase heuristic algorithm. First a request is accepted or rejected quickly and then a optimization phase is started. Caramia et al. (2001) proposed a two-phase heuristic approach for the PDP. A dynamic programming algorithm is used to solve a single-vehicle routing problem. This problem consist of finding the shortest path through all pickup and delivery points already assigned to this vehicle. Caramia et al. (2001) use the A^*-Algorithm proposed by Hart et al. (1968) for this phase. When a request arrives the single-vehicle problem is solved for all vehicles. The request is assigned to the vehicle with minimum additional costs. When no feasible solution exists, the request is rejected. As no waiting times are allowed and passengers have to give a stretch-factor, the maximum allowed deviation from shortest route, Fabri & Recht (2006) adapted this algorithm to a situation where waiting times are allowed and a time window for pickup and delivery is given. In addition any free computational capacity is used for a local search algorithm to improve the assignment. Horn (2002) also use a two-phase heuristic for the PDP. When a request arrives a lowest-cost insertion is performed, immediately followed by a local improvement step. Periodically,

after every k requests, a steepest-decent improvement strategy is applied. Attanasio et al. (2004) use parallel implementations of the Tabu Search algorithm, developed by Cordeau & Laporte (2003) for the static DARP. At the start of the planning horizon a static solution is constructed for the already known requests. When a new request arrives, it is first decided whether to accept the request or not, then post-optimization is performed by running the TS algorithm. Khelifi et al. (2013) also deploy a Tabu Search algorithm in the second phase. Lois & Ziliaskopoulos (2017) use a fast insertion algorithm and sequentially a more advanced regret-based algorithm for the DARP. However, they show that there is a certain turning point, where using a more advanced optimization technique is no longer more profitable, as more new requests are lost.
The strategy deployed by Gendreau et al. (2006) is a little different from the rest. In an adaptive memory they store the best visited solutions. When a requests arrives, it is inserted into all solution in the adaptive memory, as well as in the incumbent solution. A optimization procedure runs as long as no 'event' occurs. An 'event' is not only the arrival of a request, but also the end of service at a pickup or delivery node, in this case the node is deleted from all solutions in the adaptive memory and the next node of the vehicle is fixed in all solutions.

Some studies also include stochastic information about future requests in their algorithm. Bent & Van Hentenryck (2004) introduce the Multiple Plan Approach (MPA) in which a solution pool is maintained of all previously found solutions. At the arrival of a request it is inserted into every solution in the solution pool and an incumbent solution is chosen. Incompatible solutions are discarded from the pool. Pool updates are done periodically and when vehicles deliver a request. They also introduce the Multiple Schedule Approach (MSA), which is similar to MPA, but stochastic information on future requests is also taken into account. Schilde et al. (2011) apply the MPA and MSA to the dial-a-ride-problem and show that using stochastic information on future requests significantly improves solution quality.

In the dynamic context, waiting strategies become more important. However, not many studies have been conducted on this subject. Mitrović-Minić & Laporte (2004) consider four different waiting strategies: Drive-First, Wait-First, Dynamic Waiting and Advanced Dynamic Waiting. The best results are achieved with the Advanced Dynamic Waiting Strategy.

There have not been many studies on the dynamic PDPT. Thangiah et al. (2007) extend on the work of Shang & Cuff (1996) and implement their method for the PDPT on real-time data sets. However, in their real-time data sets, most information is already known before-hand and a maximum of seven real-time events occur. Bouros et al. (2011) propose a solution based on a graph-based formulation of the problem, where each request is handled independently. Compared to conventional two-phase local search algorithms, they find solutions significantly faster at only marginally increased costs. For the dynamic ride-sharing problem, there has been some research recently. Agatz et al. (2011), Kleiner et al. (2011), Ben Cheikh et al. (2015), Schreieck et al. (2016), Najmi et al. (2017) and Ma et al. (2019) have all studied a variant of the dynamic ride-sharing problem.

2.3 Contribution

Modular autonomous vehicles have the ability to couple when on-route. Since this ability is not common in vehicles, there have not been many studies focused on vehicle routing problems specifically using this type of vehicle. However, there have been some studies that focus on different aspects that arise when trying to utilise a system with this kind of vehicles.
Gecchelin & Webb (2019) provide a description on how a system consisting of these vehicles compares to other kinds of transport. They mostly focus their comparison to car sharing, where they remark that car sharing could potentially cannibalise public transport. They then propose that systems with modular autonomous vehicles should preferably function as feeder to train based rapid transit, as an alternative to bus transport and even as individual transport in off-peak periods. Guo et al. (2017) look at a system that can either operate with fixed or flexible routes. In their model the modular autonomous vehicles can either operate as single vehicle, or as a coupled vehicle. So either all vehicles act like a bus, or all vehicles act as individual transport. It can determine when the system should change between these two modes, depending on several features like speed of the vehicle and future demand. However, by making this choice for all vehicles at the same time, a lot of flexibility is lost.
The master thesis of Caros (2019) focuses on day-to-day operator service to maximise profit for the operator. The cost function is a combination of three weighted components: passenger travel time, passenger wait time and travelled distance of the vehicle. The system works in a dynamic way, as requests are released during the routing period. However, there are no time windows for the requests. The system has two types of actors: the centralized operator and the individual agents that can choose to use the system. On each day agents can make the decision whether to use the system and what their departure time should be, based on the results of the previous days. A multinomial logit model is used for the agent mode choice. The relative magnitudes of the cost components are adapted each day to try to maximise profit. As the simulation of the day-to-day operation continues, the learning mechanism converges toward an optimal profit, as proven in the master thesis.
The routing algorithm uses a simple insertion algorithm for each request. Each request is inserted in the minimum cost vehicle route. After insertion, a transfer assignment step is performed. For each other vehicle the costs are calculated for the transfer of any feasible combination of passengers at any possible point in the routes of these vehicles. The transfer that reduces the total cost the most, if any exists, is inserted into the routes of both vehicles.

The contribution of this thesis will be a more advanced routing algorithm for modular autonomous vehicles. In my approach they will be used as a taxi service, not coupled with another form of public transport.

3. Problem Definition

In the pickup and delivery problem with inside transfers, a set of requests R is given, that have to be executed by a fleet of homogenous modular autonomous vehicles M, that have a maximum capacity Q per vehicle. It is assumed that an unlimited amount of vehicles is available. Each request $r \in R$ is defined as the transport of q_r passengers from a pickup point r^+ to a delivery point r^-. The pickup point and the delivery point are locations on an underlying road network. This road network is a directed graph $G_L(L, E_L)$. Here $L = \{L_1, L_2, ..., L_n\}$ is the set of n locations in this network and E_L are the roads between these locations. In this variant of the problem, the underlying road network has a tree structure, meaning that any two locations are connected by exactly one path of unique locations. Thus, it is possible to determine a single request route for every request. This request route consists of unique locations on the road network that must be visited one after the other by the vehicles that execute this request, to reach its delivery point. It is possible that multiple vehicles execute a part of a request, as transfers are allowed. Naturally, each part of a request can only be executed by one vehicle.

There are an earliest pickup time E_r and a latest delivery time L_r associated with each request. The difference between the latest delivery time and the earliest pickup time must be larger than the travel time from the requests pickup point to its delivery point. Also, a request must be announced at time $A_r < E_r$. Announcing a request means that from the time of announcement, the request is known to the system. A vehicle is allowed to arrive at a pickup point before the earliest pickup time, but then it must wait. It is not allowed to arrive at the delivery point after the latest delivery time. Passengers can share a vehicle, as long as this does not cause an exceedance of the vehicle capacity. Furthermore, all vehicles originate from a single depot where they start and end their route.

Once a request is announced, this request is preprocessed to exploit the specific advantage of the road networks with a tree structure. Every request $r \in R$ is split up in a set of sub-requests $\{r_1, r_2, \ldots, r_{m_r}\}$. The number of sub-requests depends on the request, so the value of m_r is different for every request. For every request $r \in R$ exactly one request route exists from its pickup point r^+ to its delivery point r^-. This request route consists of unique locations on the road network that must be visited one after the other by the vehicles that execute this request to reach its delivery point. Each sub-request is a part of the request; the transportation of the passengers from one location in the request route to the next. Each sub-request $r_j, j \in \{1, 2, ..., m_r\}$ has a pickup point r_j^+ and a delivery point r_j^-, which correspond to two sequential locations of the request route. The introduction of sub-requests provides the possibility for an outside transfer to occur between each sub-request, as each sub-request can be executed by a different vehicle. Figure 3.1 gives an example of how a request is split up in several sub-requests.

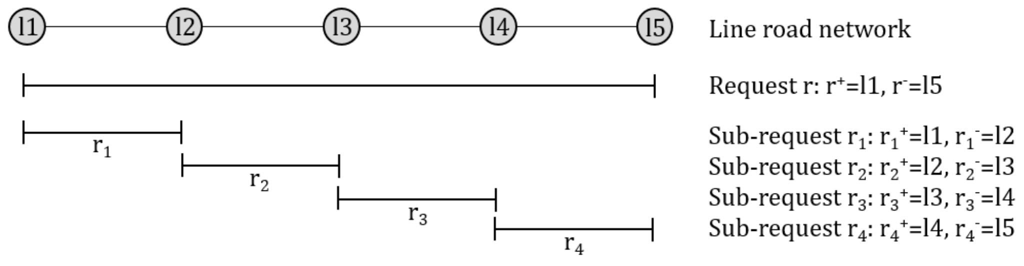

l1
l2
l3
l4
l5
Line road network
Request r: r^+=l1, r^-=l5
r_1
r_2
r_3
r_4
Sub-request r_1: r_1^+=l1, r_1^-=l2
Sub-request r_2: r_2^+=l2, r_2^-=l3
Sub-request r_3: r_3^+=l3, r_3^-=l4
Sub-request r_4: r_4^+=l4, r_4^-=l5

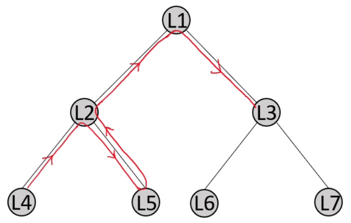
Figure 3.1: Request turned into sub-requests. For the request and each sub-request the pickup point and delivery point are given.

As modular autonomous vehicles are considered, passengers can transfer from one vehicle to another, while these vehicles are coupled. This type of transfer is called an inside transfer. Outside transfers are also considered, in which a passenger is transferred from one vehicle to another at a transfer point. A transfer point can be any location on the underlying road network. Entering or leaving a vehicle takes a passenger t_s amount of time, regardless of the number of passengers that enter or leave the vehicle at this location. So using an outside transfer takes $2t_s$ amount of time for a passenger since the passenger needs to leave one vehicle and then enter another vehicle. Using an inside transfer takes no time, as switching vehicles is done while driving. The vehicles that participate in an inside transfer need to drive in a coupled manner from the location where the inside transfer is initiated to the next. Doing so gives the passengers enough time to switch vehicle. This type of coupled driving is called platooning.

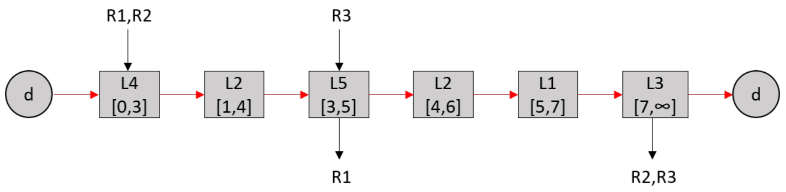
In a solution, S, each request must be executed by a vehicle. A request can also be partially executed by multiple vehicles if it uses one or more transfers. This means that the sub-requests are executed by different vehicles. It is allowed to reject and not execute a request, but in this case a very high penalty, p_r, per passenger is imposed. A solution is defined as a set of vehicle routes, V. An example of a vehicle route is depicted in Figure 3.2. Each vehicle route $v \in V$ must be executed by a single vehicle.

A vehicle route consists of locations the vehicle must visit sequentially; these are called vehicle locations. A vehicle location in a vehicle route is a combination of the physical location on the road network G_L and a time window in which the vehicle must depart from the vehicle location. This can be seen in Figure 3.2, here vehicle location L2[1,4] is a different location in the vehicle route than vehicle location L2[4,6].

At every vehicle location, multiple situations can occur. One or more passengers can enter the vehicle, leave the vehicle or transfer to another vehicle, or nothing happens. In the latter, the vehicle drives by the vehicle location without needing to stop there, whereas in the first three situations the vehicle needs to stop. Note that the first three situations can coincide in any given combination at



(a) road network with vehicle route, each road has length 1.



(b) vehicle route

Figure 3.2: In (a) a road network is given, with the physical locations of the vehicle route connected by the solid red line. In (b) the corresponding vehicle route, that starts and ends at depot d, is given, including time windows at every vehicle location and requests R1, R2 and R3 that are executed by this vehicle route. The time windows start with a slack of 3 time units, which is decreased to 2 time units at location L5 as leaving and entering the vehicle takes $t_s = 1$ time units. The latest time at location L3 is equal to infinity as no requests have to be executed anymore. The vehicle that executes this vehicle route must stop at vehicle locations L4, L5 and L3, but can drive by vehicle locations L2[1,4], L2[4,6] and L1.

one vehicle location. Naturally, none of them can be combined with the situation in which nothing happens.

Every vehicle location has a time window in which the vehicle must depart from this location. This time window consists of an earliest departure time and a latest departure time. The earliest departure time is such that no request, which is either fully or partially executed by this vehicle, leaves its pickup point before its earliest pickup time. The latest departure time is such that every request, which is either fully or partially executed by this vehicle, arrives at its delivery point before its latest delivery time. All vehicles routes are needed to determine the earliest and latest departure times, to incorporate transfers accurately. In a feasible solution, the earliest departure time is always smaller than or equal to the latest departure time.

Note that a vehicle route is not merely a path through locations, but a vehicle route also keeps track of the requests that enter and/or leave the vehicle at these locations and the time windows in which the vehicle must depart from these locations.

The dynamic variant of the problem is considered, in which requests are announced at arbitrary times during the scheduling horizon. In this dynamic setting relatively fast response times are required, according to Attanasio et al. (2004). Once a request has been accepted for execution, it is not allowed to reject this request later on. When a request is rejected, it will not be executed by any vehicle. In a dynamic setting, vehicle routes are being updated during the execution of these routes. This means that every vehicle route has an implemented part and a non-implemented part. The implemented part of a vehicle route is the part of the vehicle route that is already executed or fixed to be executed in the near future. No vehicle locations can be removed from or added to the implemented part of the vehicle routes. Requests that are partially or fully executed in the implemented part of the vehicle route, can not be removed from this vehicle route. Thus, once a request has been picked up, the assignment of the vehicles that execute this request can not be changed. The non-implemented part of the vehicle route is not yet executed and can still be changed.

The aim is to construct feasible vehicle routes, such that total costs are minimized. The objective value of a solution, $z(S)$, is the combination of two components; the total energy consumption of the vehicles and the total inconvenience for the passengers. The total energy consumption is determined by the total distance driven by each vehicle. For each vehicle, the total distance driven, d_v, is calculated by summing the distances between each sequential vehicle location the vehicle has visited. An amount of p_u is paid per distance unit the vehicle is driving. Per outside transfer used, per passenger, a penalty p_{ot} is given, to account for the discomfort for the passenger. Per vehicle a fixed cost p_v is incurred for the deployment of this vehicle. It is assumed there are no costs for the amount of time a vehicle is being used, since these vehicle are autonomous and therefore do not require a driver. Waiting time is not included in the cost calculation. Lastly, there is a penalty for each rejected passenger, p_r, as the aim is to serve as many passengers as possible. The total objective value is the sum of the aggregate costs of all deployed vehicles, the outside transfer penalties and the rejected passenger penalties:

$$z(S) = p_{ot}(\text{number of outside transfers}) + p_r(\text{number of rejected passengers}) + \sum_{v \in V}(p_u d_v + p_v)$$

4. Algorithm Design

In this section, a heuristic method is proposed to solve the dynamic variant of the pickup and delivery problem with inside transfers. First, a heuristic method is introduced in which in each iteration several requests are removed from the solution and then reinserted into the solution, the main steps of this method are described. Then it is explained how this heuristic method fits into a framework to handle the dynamic announcement of the requests. Next, a closer look is given to the removal and insertion of the requests. Finally, the update procedure of the vehicle routes after the removal and insertion requests is explained, as well as a feasibility check for the newly constructed solution.

4.1 Adaptive Large Neighbourhood Search

Since the Pickup and Delivery Problem with Transfers is NP-hard, solving instances with an exact algorithm is only possible for small instances. Pisinger & Ropke (2007) show that the adaptive large neighbourhood search (ALNS) framework is capable of solving many variants of the vehicle routing problem. This framework is an extension of the large neighbourhood search (LNS) framework introduced by Shaw (1998). Many studies have made use of the ALNS framework to solve a variant of the PDPT, see Section 2.2.2. Therefore, this thesis explores an ALNS solution method to solve this problem, as well.

In this section, the main steps in the ALNS solution method are explained. The steps are given in Algorithm 1. The calculations for the objective value of a solution and a method to avoid getting trapped in local minima are given as well.

To successfully apply an ALNS solution method, an initial solution is constructed (lines 1,2). This can be achieved easily by executing each request with a separate vehicle, as it is assumed there is an infinite amount of vehicles available. The initial solution consists of all requests that are already known before the start of the scheduling horizon.

During the search, many iterations of this ALNS solution method are applied. Removal and insertion operators are central to the ALNS solution method. These operators modify the current solution. In each iteration, a removal and insertion operator are chosen (line 5), based on their weight at that moment, which is determined by past performance (line 20).

At the beginning of each iteration, a check is done for newly announced requests (line 6). They are added to the incumbent solution using an insertion operator (line 9). The set of new requests can either be accepted or rejected. They are accepted when the added costs to the objective value of the incumbent solution for executing this set of requests are lower than the costs for rejecting the requests (lines 10, 11). Otherwise, they are all rejected (lines 13,14).

After the insertion of newly announced requests, the solution is modified. This is done by first

Algorithm 1: Main steps in the ALNS solution method

```
    Input: InitialSolution
    Output: BestSolution
 1  BestSolution ← InitialSolution;
 2  CurrentSolution ← InitialSolution;
 3  while Termination criteria not satisfied do
 4      $S$ ← CurrentSolution;
 5      Selection of a removal ($O^-$) and an insertion ($O^+$) operator;
 6      Check for newly announced requests, $R'$;
 7      if $R' \neq \emptyset$ then
 8          $S$ ← BestSolution;
 9          $S \leftarrow O^+(R' \cup S)$;
10          if $z(S) \leq z(BestSolution)+|R'|RejectionPenalty$ then
11              Accept all newly announced requests;
12          else
13              Reject all newly announced requests;
14              $S$ ← CurrentSolution;
15          end
16      end
17      $S \leftarrow O^+(O^-(S))$;
18      if $z(S) \leq z(CurrentSolution)$ then
19          CurrentSolution ← $S$;
20          if $z(S) \leq z(BestSolution)$ then
21              BestSolution ← $S$;
22          end
23      else
24          CurrentSolution ← simulatedAnnealingAcceptanceCriteria(S,CurrentSolution)
25      end
26      Update scores of used operators;
27  end
    Result: BestSolution
```

removing specific requests from the solution, using the selected removal operator (line 17). When a request is removed from the solution, all sub-requests are removed. Afterwards, an insertion operator is used on all the removed requests (line 17). After the insertion of every sub-request, a feasibility check is performed. After a certain amount of iterations, the weights of the operators are adjusted, depending on how well they have scored recently (line 26). In this way, it is attempted to select the best operators for each stage of the search.

Improved solutions are always accepted (lines 18-22). To avoid getting trapped in a local minimum, solutions that have a higher objective value than the current solution can be accepted as well (line 24). An acceptance criterium from simulated annealing is used for this cause. A solution (S') that has a higher objective value than the current solution (S) is accepted with probability $e^{-\frac{z(S')-z(S)}{T}}$, where $z(S)$ the objective value and T is the temperature. This temperature is decreased every iteration with a cooling rate c, where $0 < c < 1$. The initial temperature (T_0) is set such that a solution that has an objective value that is $(1 + \omega)$ times higher than the objective value of the current solution, where $(0 < \omega < 1)$, has a probability of 0.5 of being accepted. $T_0 = \frac{w}{-\ln 0.5} z_0$, where z_0 is the objective value of the initial solution. The exact values for c and w are determined by experimentation. The value given to the acceptance probability is irrelevant. If the value would be higher than 0.5, the selected value of w would probably turn out lower, giving approximately the same effect. After a fixed amount of iterations in which no improved solution has been found, the temperature is reset to $\frac{w}{-\ln 0.5} z*$, here $z*$ is the objective value of the incumbent solution. This is done to advance the search when it is getting stuck.

Recall the formula for the objective value of a solution: $z(S) = p_{ot}(\text{number of outside transfers}) + p_r(\text{number of rejected passengers}) + \sum_{v \in V}(p_u d_v + p_v)$. For each vehicle the total travel distance, d_v, is calculated, by summing the distances between each sequential vehicle location the vehicle has visited. The cost for travelling one distance unit is p_u. Per vehicle a fixed cost p_v is incurred for the deployment of this vehicle. Per used outside transfer, per passenger, a penalty p_{ot} is given, to account for the discomfort for the passenger. There is a penalty p_r for each rejected passenger.

Insertion costs are the costs to insert a request into an already existing vehicle route. Insertion costs can also be defined for requests that are already included in a vehicle route. In this case, the costs are determined by removing the request from the vehicle route, and than calculating the costs to reinsert it at the exact same place. Insertion costs are calculated to determine the best place of insertion for a request. Since these costs have to be calculated many times during an iteration of the ALNS solution method, it is important that they are calculated in an efficient manner. Let $c_{(v_l, v_{l+1})} = p_u * \tau_{(v_l, v_{l+1})}$ be the cost to drive from vehicle location v_l to v_{l+1}. These costs are determined by multiplying the distance between these two vehicle locations $\tau_{(v_l, v_{l+1})}$, by the cost per distance unit p_u. Remember that p_{ot} is the penalty for using an outside transfer per passenger and q_r is the number of passengers in request r. Let r be a request with sub-requests $r_1, ..., r_{m_r}$. Each sub-request r_j has a pickup point r_j^+ and a delivery point r_j^-. The same vehicle will visit the pickup and delivery point of the sub-request. The insertion costs d_{r_j} to insert sub-request r_j in vehicle route v, between vehicle locations v_l and v_{l+1}, are as follows:

$$d_{r_j, v_l} = \begin{cases} c_{(v_l, r_j^+)} + c_{(r_j^+, r_j^-)} + c_{(r_j^-, v_{l+1})} - c_{(v_l, v_{l+1})} & \text{if } j = 1 \text{ or vehicle route } v \\ & \text{also executes sub-request } r_{j-1} \\ c_{(v_l, r_j^+)} + c_{(r_j^+, r_j^-)} + c_{(r_j^-, v_{l+1})} - c_{(v_l, v_{l+1})} + p_{ot} * q_r & \text{otherwise} \end{cases}$$

To calculate the total insertion cost of a request, in specific vehicle routes and at specific vehicle locations, the insertion costs of the sub-requests are added together: $\sum_{j=1}^{m_r} d_{r_j, v_l}$.

4.2 Dynamic announcement of requests

In most implementations of dynamic variants of vehicle routing problems, a two-phase heuristic algorithm is used. First a request is accepted or rejected quickly and then a optimization phase is started. The idle time between events is used to deploy a post-optimization step. For instance, Attanasio et al. (2004) and Gendreau et al. (2006) have used a tabu-search as post-optimization step and Lois & Ziliaskopoulos (2017) used a regret optimization algorithm. An events is often defined as the announcement of a new request or the end of service at a location in the vehicle route. A big advantage of this strategy is the immediate acceptance or rejection of a request, as the response time is crucial in a dynamic setting. However, as the number of events increases, the time for the post-optimization step decreases.

Therefore, in this thesis, a methodology is implemented that is similar to the routing part of the on-line solution methodology as described in Mitrović-Minić & Laporte (2004). In their methodology, a cheapest insertion procedure is applied every k minutes to insert the new requests accumulated in the previous k minutes. In the meantime, a tabu search procedure is applied to the part of the current solution starting k minutes from the current time t. This is essentially some form of a rolling horizon implementation. In this thesis, it is proposed to use the ALNS solution method in a similar rolling horizon setting. However, requests are not accumulated, but inserted into the current solution directly after their announcement, using an insertion operator from the ALNS solution method. This is possible as it is imposed that requests must be announced at least $k+1$ minutes before their earliest departure time. In this way, the requests can always be scheduled before their earliest departure time.

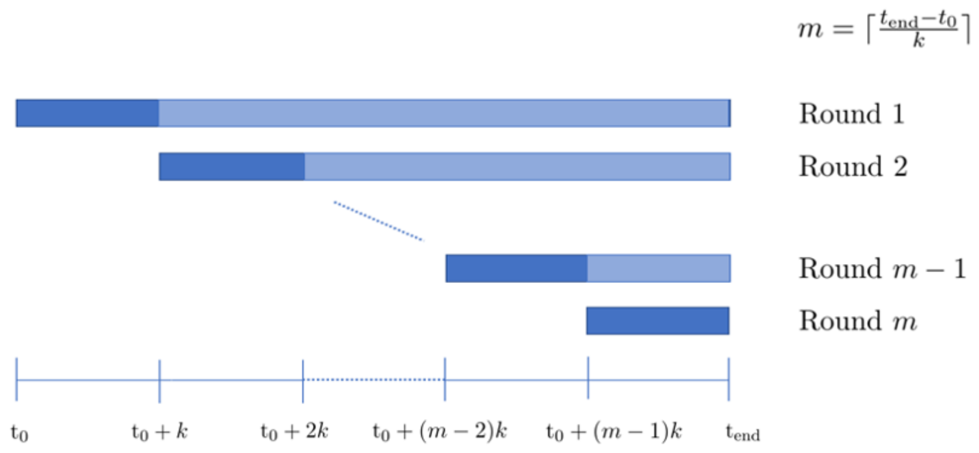


Figure 4.1: The rolling horizon solution methodology. Each optimization round lasts k minutes. In each round, the dark blue part represents the part of the solution which is implemented at the start of the round. The light blue part represents the part of the solution that can still be improved. In the last round, no improvements can be made.

Figure 4.1 shows this rolling horizon setting. There is a total of m optimization rounds. At the beginning of optimization round j at time $t = t_0 + (j-1)k$, the incumbent solution for the next

k minutes is implemented. The execution of all requests for which the pickup happens before time $t+k$ is fixed. Thus, these requests can not be removed from the solution in the following iterations of the search anymore. The vehicle routes are fixed too, until time $t+k$. This means that all vehicle locations in a vehicle route, for which the earliest departure time is earlier than time $t+k$, become part of the implemented part of the vehicle route. In the implemented part of the vehicle route, the vehicle departs from the vehicle locations at the earliest departure time. No vehicle locations can be removed from or added to the implemented part of a vehicle route.

4.2.1 Removal Operators

In this section, several removal operators are introduced. To modify the solution, several requests are removed from the solution. This is done by a removal operator (O_k^-), which removes a predefined number (q) of requests from the current solution. Only requests that are completely executed by the non-implemented part of the vehicle routes, can be removed from the solution. This leaves the set $R^* \subseteq R$, containing the available requests that can be removed from the current solution. q is controlled by two parameters $0 < \psi_1 < \psi_2 < 1$. Every iteration a new value for q is drawn uniformly from $[\psi_1|R^*|, \psi_2|R^*|]$. In this way, more variation is included in the search.

The ALNS solution method includes the following removal operators:

1. *Worst request removal (O_1^-)*: The operator deletes q requests with the highest insertion costs.

2. *Random request removal (O_2^-)*: The operator randomly selects q requests to be deleted from the current solution.

3. *Route removal (O_3^-)*: The operator selects the non-implemented part of a route to be deleted from the current solution. Routes are selected in increasing order of efficiency of the non-implemented part. Every request in the non-implemented part of the route is removed from the solution. If the non-implemented part of a route contains a transferred request, this request is also removed from the route it is transferred to or from. Routes are being deleted until the number of removed requests exceeds q, or until there exist no more routes that have non-implemented parts. Efficiency is calculated as in Algorithm 2.

4. *Transfer-based request removal (O_4^-)*: This operator exclusively looks at requests that use an outside transfer in the current solution and then removes requests that seldom use outside transfers in incumbent solutions encountered so far. Let I be the number of incumbent solutions encountered so far. Let t_r^* be the number of times request r has used at least one outside transfer in incumbent solutions. The probability that the operator will select request r is proportional to $1 - \frac{t_r^*}{I}$.

Algorithm 2: Calculating the efficiency of the non-implemented part of a route

```
Input: Vehicle route $v$
Output: Efficiency score
1  $v^* \leftarrow$ non-implemented part of $v$;
2  PreviousLocation $\leftarrow$ First vehicle location $v_l \in v*$;
3  PreviousTime $\leftarrow e_{v_l}$ (earliest departure time at $v_l$);
4  TotalTime $\leftarrow 0$;
5  TotalScore $\leftarrow 0$;
6  for Each vehicle location $v_l \in v^*$ do
7      CurrentLocation $\leftarrow v_l$;
8      CurrentTime $\leftarrow e_{v_l}$;
9      Score $\leftarrow v_l(q) * \tau_{(\text{PreviousLocation, CurrentLocation})}$ ; // (occupancy * traveltime)
10     TotalScore $+=$ Score;
11     ElapsedTime = CurrentTime - PreviousTime;
12     TotalTime $+=$ ElapsedTime;
13     PreviousLocation $\leftarrow$ CurrentLocation;
14     PreviousTime $\leftarrow$ CurrentTime;
15 end
16 Efficiency $\leftarrow \frac{\text{TotalScore}}{\text{TotalTime}}$;
Result: Efficiency
```

4.2.2 Insertion Operators

In this section, several insertion operators are introduced. An insertion operator reinserts the requests that have been removed by the removal operator into the current solution. The insertion operators are also used to insert the recently announced requests. If a request can not feasibly be inserted into an already existing vehicle route, a new vehicle route is created. This is always possible as the assumption is made that an infinite amount of vehicles is available. The ALNS solution method includes the following insertion operators:

1. *Travel time (O_1^+)*: Requests are sequenced with the longest direct travel time from pickup to delivery point first.

2. *Insertion cost (O_2^+)*: Requests are inserted sequentially in the least-cost position and route. After a request is inserted, the insertion cost is recalculated for the remaining requests, for all insertion possibilities.

3. *Regret (O_3^+)*: Requests are sequenced in decreasing order of regret. Regret is defined as the difference between the least insertion cost and the second least insertion cost. After a request is inserted, the insertion cost is recalculated for the remaining requests, for all insertion possibilities.

4. *Outside transfer (O_4^+)*: Requests are inserted in the same sequence in which they were removed

from the solution. Requests are forced to be inserted with an outside transfer if there is a feasible outside transfer option.

5. *Inside transfer (O_5^+)*: Requests are inserted in the same sequence in which they were removed from the solution. Requests are forced to be inserted with an inside transfer if there is a feasible inside transfer option.

4.2.3 Operator Selection

In every iteration of the ALNS solution method, a removal and insertion operator are selected. These operators are selected based on the roulette wheel selection principle. For instance when there are m insertion operators with weight $w_i, i \in 1, 2, ..., m$, then insertion operator j is chosen with probability $\frac{w_j}{\sum_{i=1}^{m} w_i}$. The removal operator is selected independently from the insertion operator. There are situations in which a removal operator can not remove enough requests from the solution. For instance, when the transfer-based request removal operator is chosen, but there are less than q requests that make use of an outside transfer. Or when the route removal operator is chosen, but large parts of most vehicle routes are already implemented. When this happens, another removal operator is chosen to finish the removal phase. At the start of the search, all operators have equal weights. The search is divided into several segments. One segment is, for instance, 150 iterations. After each iteration, the score π_i of the selected operator is updated. All scores are reset to zero, at the beginning of each segment. Every iteration, the score can be increased by one of the three following amounts:

1. The operator combination has found an overall best solution: increase the score of with the amount σ_1.

2. The operator combination has resulted in an accepted solution with a lower cost than the current solution: increase the score with the amount σ_2.

3. The operator combination has resulted in an accepted solution with a higher cost than the current solution: increase the score with the amount σ_3.

Here it should hold that $\sigma_1 > \sigma_2 > \sigma_3$. Both the removal and insertion operator are rewarded the same amount. This only happens when an unvisited solution is found since we want to explore as many different solutions as possible. Let a_i be the number of times operator i is used in the last segment. The weight of operator i is updated at the end of each segment as follows $w_i \leftarrow w_i(1-\rho) + \frac{\pi_i}{a_i}\rho$. Here the parameter ρ determines how much past segments contribute to the score.

4.2.4 Removal of Requests

Several removal operators were introduced. Here it is explained how requests are removed from the solution. Algorithm 3 shows the general steps of a removal operator.

Algorithm 3: Steps performed by a removal operator (O_k^-)

```
Input: Feasible solution $S$
Output: Partial solution $S^-$, and set of removed requests $R^-$
1  $q \leftarrow$ Number of requests to remove;
2  $R^- \leftarrow \emptyset$;
3  $S^- \leftarrow S$;
4  for $1,2,\ldots,q$ do
5      Select a request $r \in R^* \backslash R^-$ to be removed, according to $O_k^-$;
6      for $j$ in $\{1,2,\ldots,n\}$ do
7          Identify vehicle route $v$ that executes sub-request $r_j$;
8          Identify pickup location $v_{r_j^+}$ and delivery location $v_{r_j^-}$;
9          Remove $r_j$ from $v_{r_j^+}$ and $v_{r_j^-}$;
10         Check if $v_{r_j^+}$ should be removed from vehicle route $v$;
11         Check if $v_{r_j^-}$ should be removed from vehicle route $v$;
12         Update earliest departure times;
13         Update latest departure times;
14         $S^- \leftarrow S^- \backslash r_j$;
15     end
16     $R^- \leftarrow R^- \cup r$;
17 end
Result: $S^-, R^-$
```

To remove a request from the solution (line 5), each sub-request is removed separately. To remove a sub-request r_j from the solution (lines 7-14), the vehicle route v that executes this sub-request is identified, as well as the pickup location $v_{r_j^+}$ and the delivery location $v_{r_j^-}$ of this sub-request in this vehicle route (lines 7,8). Remove the sub-request from the relevant part of the vehicle route (line 9).

Recall that at every vehicle location in a vehicle route, multiple requests can either enter, leave, or remain in the vehicle. If no more requests enter, leave, or remain the vehicle at a location after the removal of the sub-request, the entire vehicle location is removed from vehicle route v (lines 10,11). This means that the vehicle does not have to drive to this location. This also happens when solely requests remain in the vehicle, for which the physical location of this vehicle location is not part of their request route, meaning this location is now a detour. There is one exception. If an inside transfer is initiated at vehicle location v_{l-1}, vehicle location v_l cannot be removed from the route, as the vehicle needs to platoon to this location.

When the sub-request is removed, earliest departure times are updated starting from vehicle location $v_{r_j^+}$, or $v_{r_j^+ +1}$ if $v_{r_j^+}$ is removed, with the procedure as explained in Algorithm 7 (line 12). Here $v_{r_j^+ +1}$ is the vehicle location that is visited directly after $v_{r_j^+}$. Latest times are updated starting from vehicle location $v_{r_j^-}$, or $v_{r_j^- -1}$ if $v_{r_j^-}$ is removed, with the procedure as explained in Algorithm 8 (line 13). Here $v_{r_j^- -1}$ is the vehicle location that is visited directly preceding $v_{r_j^-}$. Algorithms 7 and 8 are discussed in Section 4.3.2.

4.2.5 Insertion of Requests

In this section, the insertion of requests is examined. There are two distinct insertion procedures for the insertion of a request with or without transfers.

Insertion without transfers

In the case that a request is inserted without transfers, the request is fully executed by one vehicle. Thus, the pickup point and delivery point of this request have to be inserted into the same vehicle route. Every vehicle location in every vehicle route is considered for the insertion of the pickup point and the delivery point of this request. In this way, all possible insertion options are evaluated. In Algorithm 4 the insertion procedure is given for reinserting all requests, previously removed by the removal operator, into the solution, making use of the selected insertion operator without transfers.

Algorithm 4: Steps performed by an insertion operator that does not consider transfers (O_1^+, O_2^+ or O_3^+)

```
Input: Requests $R'$ to be inserted, partial solution $S'$, insertion operator $O_i^+, i \in \{1,2,3\}$
Output: New Solution $S^*$
1  $R'' \leftarrow R'$; // Request of $R'$ ordered according to inserion operator criteria
2  while $R'' \neq \emptyset$ do
3      for Each request $r \in R''$ do
4          $I_r \leftarrow \emptyset$; // list of feasible routes for request $r$
5          for Each vehicle route $v$ in $S'$ do
6              for $v_l, l \in \{0,1,2,\dots,N\}$ vehicle locations in route $v$ do
7                  check feasibility of inserting pickup point $r^+$ directly following $v_l$;
8                  if feasible insertion then
9                      for $v_k, k \in \{l,\dots,N\}$, vehicle locations in route $v$ do
10                         check feasibility of inserting delivery point $r^-$ directly following $v_k$;
11                         if feasible insertion then
12                             $v^* \leftarrow$ insert all sub-requests $\{r_1, r_2, \dots, r_n\}$ in route $v$;
13                             $I_r \leftarrow I_r \cup v^*$;
14                         end
15                     end
16                 end
17             end
18         end
19     end
20     Select request $r^*$ to insert according to the insertion operator $O_i^+$;
21     $v^* \leftarrow$ route with lowest insertion costs of $r^*$ in $I_r^*$;
22     $v \leftarrow$ original route of $v^* \in S'$; // so the same route, without request $r^*$ inserted into it
23     $S' \leftarrow (S' \backslash v) \cup v^*$;
24     $R'' \leftarrow R'' \backslash r^*$;
25 end
26 $S^* \leftarrow S'$
Result: $S^*$
```

All requests that were removed from the solution, have to be reinserted (line 3). Assume that the pickup point of request $r \in R''$, with time window $[E_r, L_r]$ is to be inserted into vehicle route v, behind vehicle location v_l (line 6) and the delivery point behind vehicle location v_k (line 9).

The insertion of the pickup point behind v_l is considered first. First, a check is done for the remaining capacity of the vehicle at the desired vehicle location. If this is sufficient, a check is done for time windows. There are two options for the insertion of the pickup point of request r: either the vehicle location v_l has the same physical location on the road network as the pickup point, or not. If the physical location is the same, the pickup point does not have to be inserted into the vehicle route; the first sub-request only needs to be added to the list of sub-requests that enter the vehicle at vehicle location v_l. If the physical location is different, the pickup point needs to be inserted into vehicle route v. There are two situations when the insertion is immediately considered infeasible. If the next vehicle location in the vehicle route has the same physical location as the pickup point, this option is not considered at this time. Otherwise, it would be considered twice. Also, when another request uses an inside transfer to or from vehicle route v at location v_l, no other vehicle location can be inserted after v_l, because vehicle v needs to platoon to location v_{l+1} with the other vehicle that participates in this inside transfer to make it possible.

Let v_{r^+} be the vehicle location of the pickup point in vehicle route v and let v_{r^++1} be the next vehicle location in the vehicle route. All earliest departure times of all affected vehicle locations in the solution, starting from v_{r^+} are updated by the procedure explained in Algorithm 7. The latest time $l_{v_{r^+}}$ is updated as follows: $l_{v_{r^+}} = \min\{l_{v_{r^+}}, L_r - \tau_{r^+,r^-}, l_{v_{r^++1}} - \tau_{v_{r^+},v_{r^++1}}\}$, where τ_{r^+,r^-} is the shortest travel time from the pickup point r^+ to the delivery point r^- and $\tau_{v_{r^+},v_{r^++1}}$ the travel time from the pickup point to the next vehicle location in the route. Since the insertion place of the delivery point in vehicle route v is not yet known, the direct route from the pickup point to the delivery point is used. This means that the calculated latest departure time at the pickup point might be too late, as the exact route could include a detour. This is not a problem as the actual latest departure time will be calculated after the insertion of the delivery point. Only if every affected vehicle location still has a valid time window, the insertion of the pickup point behind v_l is feasible, and the insertion of the delivery point can be considered (line 8).

For the insertion of the delivery point r^- directly behind vehicle location v_k, the same procedure holds as for the insertion of the pickup point. Let v_{r^-} be the vehicle location of the delivery point in vehicle route v. The earliest and latest departure time of all affected vehicle locations in the solution are updated, starting from v_{r^-}, as explained in Algorithms 7 and 8. If every affected vehicle location still has a valid time window, the insertion is feasible.

Now all vehicle locations in vehicle route v are updated, but we still need to assign the separate sub-requests to the correct vehicle location, as it is necessary to keep track of the separate sub-requests. The procedure to assign the sub-requests to the correct vehicle locations is shown in Algorithm 5. For every vehicle location, it is indicated which sub-request leaves the vehicle at this

location and which sub-request enters the vehicle at this location. The first sub-request r_1 enters the vehicle at the pickup point v_r^+ (line 3). The next sub-request r_2 enters the vehicle at the next vehicle location v_{r^++1}, if this v_{r^++1} has the same physical location on the road network as its pickup point r_2^+ and the next vehicle location v_{r^++2} has the same physical location as its delivery point r_2^- (line 9). If this is not the case the previous sub-request stays in the vehicle (line 15) and the vehicle location v_{r^++2} is considered as the location where sub-request r_2 enters the vehicle (line 6). This essentially means that going to vehicle location v_{r^++1} is a detour for this request.
This continues until all sub-request are assigned to the correct vehicle location.

Algorithm 5: Assigning sub-requests to correct the correct vehicle location in the vehicle route.

```
   Input: Vehicle route $v$ after insertion of the pickup and delivery point.
   Request $r \in R$ that is to be inserted.
   Output: Updated vehicle route $v$
 1 locations ← $\{v_{r^+}, \dots, v_{r^-}\}$;
 2 nextLocation ← locations.Next();
 3 $r_1$ enters the vehicle at nextLocation;
 4 $j \leftarrow 2$;
 5 while $j \leq n$ do
 6     while locations.hasNext() do
 7         Location ← nextLocation;
 8         nextLocation ← locations.Next();
 9         if Location == $r_j^+$ and nextLocation == $r_j^-$ then
10             $r_j$ enters the vehicle at Location;
11             $r_{j-1}$ leaves the vehicle at Location;
12             $j++$;
13             break out of while loop;
14         else
15             $r_{j-1}$ stays in the vehicle at Location;
16         end
17     end
18 end
   Result: Updated vehicle route $v$
```

Insertion with transfers

In the case that a request is inserted with transfers, the separate sub-request can be executed by different vehicles. Thus, the insertion of each sub-request is considered separately. The pickup point and delivery point of a sub-request have to be inserted in the same vehicle route. Figure 4.2 illustrates the insertion of sub-request r_j in vehicle route v, directly after vehicle location v_{l-1}.

However, not all insertion possibilities are considered. The maximum number of insertion possibilities for a request $r \in R'$ is calculated. Let n be the number of sub-requests for request r, m the number of available vehicle routes, and v_N the number of vehicle locations in a vehicle route v.

Then

$$(\sum_{v=1}^{m} v_N)^n$$

is the total number of insertion possibilities. So, when an unlimited amount of transfers is allowed, the number of insertion possibilities quickly explodes. The first measure to limit the number of insertion possibilities is to allow only two transfers per request. To limit the number of allowed transfers also makes sense when looking from a customer convenience viewpoint, as it is more convenient to remain in the same vehicle. The second measure to limit the number of insertion possibilities is to only continue with the five best insertion possibilities after the insertion of every sub-request. The number five is chosen at random.

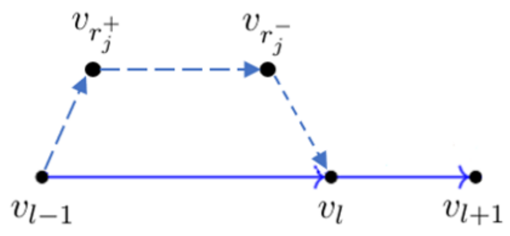


Figure 4.2: The insertion of sub-request r_j in vehicle route v, directly after vehicle location v_{l-1}

In Algorithm 6 the insertion procedure is given for inserting all removed requests back into the solution, making use of the selected insertion operator with transfers. Note that a request is either inserted using inside transfers or using outside transfers, but never with a combination of the two types. During the insertion procedure, each sub-request is inserted sequentially from the first to the last sub-request.

The requests are reinserted in the same sequence in which they were removed from the solution (line 1). As only 2 transfers per request are allowed, it is necessary to keep track of the number of transfers per request in each solution that is considered (line 5). All sub-requests are inserted sequentially (line 6). For each sub-request all feasible insertions are stored (line 4). To limit the size of the neighbourhood for the search, for each sub-request only the five solutions with the lowest costs so far are considered (line 7). Each sub-request can either be inserted with a transfer, or without transfer. However, to be inserted with a transfer the total number of transfers in the considered solution has to be smaller than 2 (line 10). In the case inside transfers are considered, it makes no sense to insert the last sub-request with an inside transfer, because the two vehicle participating in the inside transfer need to platoon (line 10).

Algorithm 6: Steps performed by an insertion operator that does consider transfers (O_4^+ or O_5^+)

```
Input: Requests $R'$ to be inserted, partial solution $S'$, insertion operator $O_4^+$ or $O_5^+$
Output: New Solution $S^*$
1  $R'' \leftarrow R'$; // Request of $R'$ ordered according to inserion operator criteria
2  $S \leftarrow S'$;
3  for Each request $r \in R''$ do
4      $L \leftarrow \{S\}$; // list of possible solutions
5      numtrans($S$) $\leftarrow 0$ // number of transfers used by request $r$ in solution $S$
6      for $j \in \{1, 2, \ldots, n\}$ do
7          $L' \leftarrow$ {the 5 solutions of $L$ with lowest detour costs for request $r$};
8          $L \leftarrow \emptyset$;
9          for $S \in L'$ do
10             if numtrans($S$) $< 2$ and $!(j == n$ and $O_5^+)$ then
11                 for each vehicle route $v$ in $S$ do
12                     for $v_l, l \in \{1, 2, \ldots, N\}$ vehicle locations in route $v$ do
13                         $v^* \leftarrow$ insert pickup and delivery point of sub-request $r_j$ in route $v$
                            directly behind $v_l$;
14                         if feasible insertion then
15                             $S^* \leftarrow (S \backslash v) \cup v^*$;
16                             $L \leftarrow L \cup S^*$;
17                             numtrans($S^*$) = numtrans($S$);
18                             if transfer then
19                                 numtrans($S^*$) $++$;
20                             end
21                         end
22                     end
23                 end
24             else
25                 determine route $v'$ into which sub-request $r_{j-1}$ is inserted;
26                 for $v'_l, l \in \{1, 2, \ldots, N\}$ vehicle locations in route $v'$ do
27                     $v^* \leftarrow$ insert pickup and delivery point of sub-request $r_j$ in route $v'$
                        directly behind $v'_l$;
28                     if feasible insertion then
29                         $S^* \leftarrow (S \backslash v') \cup v^*$;
30                         $L \leftarrow L \cup S^*$;
31                         numtrans($S^*$) = numtrans($S$);
32                     end
33                 end
34             end
35         end
36     end
37     $S \leftarrow$ least-cost solution in $L$, for which numtrans($S$) $> 0$;
38 end
39 $S^* \leftarrow S$;
Result: new solution $S^*$
```

If a transfer is still allowed, it is attempted to insert the sub-request into each vehicle route in the considered solution (line 11). Every vehicle location in the vehicle route is considered for insertion (line 12,13). How this insertion is executed and how the feasibility is checked, will be explained later on in this section. If the insertion at a vehicle location is feasible, a new solution with the sub-request inserted at this vehicle location is created (line 15), this solution is then added to the list with feasible solutions for this sub-request (line 16). In the case the sub-request uses a transfer, the number of transfers for this solution is increased (line 19).
If a transfer is not allowed, it is only attempted to insert this sub-request in the same route as the previous sub-request was inserted (line 25). The procedure thereafter is the same as when transfers are still allowed (lines 25-31).

When all sub-requests are inserted, a single least-cost solution, in which the request is inserted with at least one transfer, is selected from the list with feasible solutions (line 37). If there exists no solution in which the request is inserted with at least one transfer, the least-cost solution without transfers is selected. This solution is then used as the starting solution for the insertion of the next request (line 5).

As the insertion with outside transfers differs from the insertion with inside transfers, these two options are explained separately in closer detail.

Insertion with outside transfers
In the situation where a request is inserted into a route with outside transfers, each sub-request is either inserted without a transfer or with an outside transfer. There is essentially no difference in these two situations, only in the updating of the time windows, as explained in Section 4.3.1. To insert the pickup point and delivery point of a single sub-requests r_j in route v directly behind vehicle location v_{l-1}, as illustrated in Figure 4.2, the following procedure is followed: First a check is done for the remaining capacity of the vehicle at the desired location. If the capacity is sufficient, a check is done for the time windows.

There are two options for the insertion of the pickup point r_j^+: either the vehicle location v_{l-1} has the same physical location on the road network as the pickup point, or not. If the physical location is the same, the pickup point does not have to be inserted into the vehicle route; the sub-request only needs to be added to the list of sub-requests that enter the vehicle at vehicle location v_{l-1}. If the physical location is different, the pickup point needs to be inserted into vehicle route v. There are two situations when this is immediately considered infeasible. If the next location in the vehicle route has the same physical location as the pickup point, this option is not considered at this time, because otherwise it would be considered twice. Also, when another request uses an inside transfer to or from vehicle route v at vehicle location v_{l-1}, no other vehicle location can be inserted after v_{l-1}, because vehicle v needs to platoon with the other vehicle that participates in this inside transfer to make it possible.

Let $v_{r_j^+}$ be the vehicle location of the pickup point in vehicle route v. Sub-request r_j will be added to the list of sub-requests that enter the vehicle at this vehicle location. Earliest departure times at all affected vehicle locations in the solution are updated starting from $v_{r_j^+}$, with the procedure as explained in Algorithm 7. Latest departure times at all affected vehicle locations are updated too, with the procedure as explained in Algorithm 8. The latest departure time at $v_{r_j^+}$ is calculated in a different manner: $l_{v_{r_j^+}} = \min\{l_{v_{r_j^+}}, L_r - \tau_{(r_j^+, r^-)}, l_{v_{l+1}} - \tau_{(v_{r_j^+}, v_{l+1})}\}$, where $\tau_{(r_j^+, r^-)}$ is the direct travel time from the pickup point of sub-request r_j to the delivery point of request r and $\tau_{(v_{r_j^+}, v_{l+1})}$ the travel time from the pickup point to the next vehicle location in the route.

Only if the updated time windows of all affected vehicle locations are still feasible, the insertion of the delivery point r_j^- is considered. For the insertion of the delivery point directly behind vehicle location $v_{r_j^+}$, the same procedure holds as for the insertion of the pickup point. Now let $v_{r_j^-}$ be the vehicle location of the pickup point in vehicle route v. Sub-request r_j will be added to the list of sub-requests that leave the vehicle at this location. Let $v_{r_j^-+1}$ the next vehicle location in the vehicle route. Earliest departure times at all affected vehicle locations in the solution are updated starting from $v_{r_j^-}$, with the procedure as explained in Algorithm 7. Latest departure times at all affected vehicle locations are also updated, with the procedure as explained in Algorithm 8. The latest departure time at $v_{r_j^-}$ is calculated in a different manner: $l_{v_{r_j^-}} = \min\{l_{v_{r_j^-}}, L_r - \tau_{(r_j^-, r^-)}, l_{v_{r_j^-+1}} - \tau_{(v_{r_j^-}, v_{r_j^-+1})}\}$. If all updated time windows are feasible, the insertion of sub-request r_j directly behind vehicle location v_{l-1} is feasible.

Insertion with inside transfers

In the situation where a request is inserted into a route with inside transfers, each sub-request is either inserted without a transfer or with an inside transfer. The insertion procedure for a sub-request without a transfer is the same as the insertion procedure for a sub-request with an outside transfer. Now the insertion of a sub-request with an inside transfer is considered.

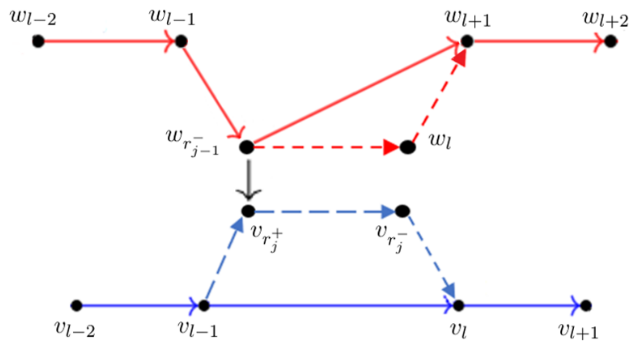


Figure 4.3: The inside transfer of request r from vehicle route w to vehicle route v

Figure 4.3 illustrates the insertion of sub-request r_j in vehicle route v, with an inside transfer from vehicle route w. The pickup point and delivery point of sub-request r_j are inserted between vehicle locations v_{l-1} and v_l. Vehicle v and w have to platoon, for vehicle v from vehicle location $v_{r_j^+}$ to $v_{r_j^-}$ and for vehicle w from vehicle location $w_{r_{j-1}^-}$ to w_l. The solid lines represent the vehicle routes before insertion of sub-request r_j into route v. For each of the three vehicle locations $v_{r_j^+}, v_{r_j^-}$ and w_l, it could be the case that they have the same physical location on the road network as a vehicle location in the respective vehicle route, or that they are a recently inserted vehicle location.

$v_{r_j^+}$:

- Not recently inserted : $v_{r_j^+}$ has the same physical location on the road network as v_{l-1}
- Recently inserted: $v_{r_j^+}$ does not have the same physical location as v_{l-1}. A new vehicle location has to be inserted, this is infeasible if there is already an inside transfer at the previous vehicle location v_{l-1}. This is also considered infeasible if $v_{r_j^+}$ has the same physical location as v_l, because then this option would be considered twice.

$v_{r_j^-}$:

- Not recently inserted: $v_{r_j^-}$ has the same physical location on the road network as v_l
- Recently inserted: $v_{r_j^-}$ does not have the same physical location as v_l. A new vehicle location for $v_{r_j^-}$ has to be inserted, this is infeasible if there is already an inside transfer at the previous location $v_{r_j^+}$. Of course $v_{r_j^+}$ can only already have an inside transfer when it is the same vehicle location as v_{l-1}.

w_l:

- Not recently inserted: w_l has the same physical location on the road network as w_{l+1}
- Recently inserted: w_l does not have the same physical location as w_{l+1}. A new vehicle location has to be inserted, this is infeasible if there is already another inside transfer at the previous vehicle location $w_{r_{j-1}^-}$.

After the sub-request is inserted, the time windows are updated in the following manner. First, the earliest departure times at all affected vehicle locations, starting from $v_{r_j^+}$, are updated as explained in Algorithm 7. If the resulting time windows are feasible the latest departure times at all affected vehicle locations, starting from $v_{r_j^-}$, are updated as explained in Algorithm 8. If the resulting time windows are feasible again, the earliest and latest departure times of all affected vehicle locations are updated starting from w_l. If all updated time windows are feasible, the insertion of sub-request r_j at vehicle location v_{l-1}, with an inside transfer from vehicle location $w_{r_{j-1}^-}$, is feasible.

4.3 Updating Vehicle Routes

To be able to determine which solution is the best, several aspects have to be taken into account. The best solution should be a feasible solution against the lowest costs. In this section, the method to determine feasibility is discussed. When a certain vehicle location is updated, that is, a sub-request is added to or removed from the vehicle at this location, or this vehicle location is added to or removed from the vehicle route, the time windows of other vehicle locations can be affected as well. First, the calculations for the time windows at a single vehicle location are explained. Then it is explained which other vehicle locations must also be checked after updating a certain vehicle location.

4.3.1 Time Windows

For a solution S, the time windows of each vehicle route should be calculated. For each vehicle route v and for each vehicle location v_l in this route, the earliest time e_{v_l} and the latest time l_{v_l} the vehicle is allowed to depart from this vehicle location should be determined. Each request r has a time window $[E_r, L_r]$, which is the earliest time the request can be picked up from its pickup point and the latest time it can be delivered at its delivery point. When a vehicle has to stop to pick up or deliver a request, this takes t_s time, no matter how many requests enter or leave the vehicle at this location. $\tau_{(v_{l-1},v_l)}$ is the travel time from vehicle location v_{l-1} in vehicle route v to vehicle location v_l in vehicle route v. Let R_{v_l} be the set of all requests that are present in vehicle v at the time it departs from vehicle location v_l.

Earliest time

Starting from the first vehicle location in a vehicle route and continuing along the route, for each vehicle location v_l the earliest departure time the vehicle can depart from the vehicle location is determined. The vehicle itself has an earliest departure time, $e_{v_l} = e_{v_{l-1}} + \tau_{(v_{l-1},v_l)} + I_l t_s$. Naturally, the vehicle can not depart earlier than the earliest time it can arrive at the vehicle location; namely the earliest time it can depart from the previous vehicle location, v_{l-1}, in the route plus the travel time between these vehicle locations. Also, an indicator function is added, which is equal to one if the vehicle needs to stop at vehicle location v_l and equal to zero if it does not need to stop. The indicator function is multiplied with the time it takes for the vehicle to stop. The vehicle needs to stop when a request leaves the vehicle at vehicle location v_l. This can either be because the request uses an outside transfer or it has reached its destination.

It is also necessary to include all the requests that are in the vehicle at the time it departs from vehicle location v_l. For each request $r \in R_{v_l}$, the earliest time the vehicle can depart from vehicle location v_l, if it was only executing this request, is calculated. This time is denoted E'_r.

- Option 1: The request enters the vehicle at v_l. The vehicle can depart from v_l at time $E'_r = E_r + t_s$. In this situation also $I_l = 1$.

- Option 2: The request entered the vehicle at a vehicle location prior to v_l. The vehicle can depart at time $E'_r = e_{v_{l-1}} + \tau_{(v_{l-1},v_l)}$.
- Option 3: The request has used an outside transfer from another vehicle w at vehicle location w_l to vehicle v at v_l. Vehicle v can depart at time $E'_r = e_{w_{l-1}} + \tau_{(w_{l-1},w_l)} + 2t_s$. Note that v_l and w_l have the same physical location on the underlying road network. Now $I_l = 1$ as well.

Finally, all the requests that use an inside transfers at v_l, either to or from vehicle v are considered. If a request uses an inside transfer to or from vehicle w, then vehicle v and w must platoon from the physical location of v_l to the physical location of v_{l+1}. Let W be the set of all vehicles that are platooning with vehicle v from the physical location of v_l to the physical location of v_{l+1}. Then it holds that $e_{v_l} = \max\{e_{v_l}, \max_{w\in W}\{e_{w_l}\}\}$.
Compounding the above, the earliest depart time vehicle v can leave from vehicle location v_l is:

$$e_{v_l} = \max\{(e_{v_{l-1}} + \tau_{(v_{l-1},v_l)} + I_l t_s), \max_{w\in W}\{e_{w_l}\}, \max_{r\in R_{v_l}}\{E'_r\}\}$$

For every vehicle $w \in W$ the condition $e_{w_l} < e_{v_l}$ is checked. If this condition holds the earliest departure time of vehicle w at w_l needs to be updated as well, since it should be the same as the earliest departure time of vehicle v at v_l, as the two vehicles are platooning. The update procedure will be explained in more detail in Section 4.3.2.

Latest time

Starting from the last vehicle location in a vehicle route and working backwards, the latest time a vehicle is allowed to depart from a vehicle location v_l is determined. The vehicle itself has a latest time it is allowed to leave the vehicle location; $l_{v_l} = l_{v_{l+1}} - \tau_{(v_l,v_{l+1})} - I_{(l+1)}t_s$. Of course, the vehicle needs to depart at such a time that it is present at the next vehicle location, v_{l+1}, at the time service needs to start there; namely the latest time it is allowed to depart at v_{l+1}, subtracted with the travel time between the two vehicle locations. Also, an indicator function is added, which is equal to one if the vehicle needs to stop at v_{l+1} and equal to zero if it does not need to stop. The indicator function is multiplied with the time it takes for the vehicle to stop. The vehicle should stop when a request enters the vehicle at v_{l+1}. This can either be because the request is being picked up there or uses an outside transfer.

It is also necessary to include all the requests that are in the vehicle at the time it departs from v_l. For each request $r \in R_{v_l}$, the latest time the vehicle can leave vehicle location v_l, if it was only executing this request, is calculated. This time is denoted L'_r. To this end the status of the request at vehicle location v_{l+1} is determined. For each request there are several options:

- Option 1: The delivery point of the request has the same physical location as v_{l+1}. The time at which the request has to leave vehicle location v_l is: $L'_r = L_r - t_s - \tau_{(v_l,v_{l+1})}$. Now also $I_{(l+1)} = 1$.

- Option 2: The request will use an outside transfer to another vehicle w at v_{l+1} and w_{l+1}. The request has to leave v_l at: $L'_r = l_{w_{l+1}} - 2t_s - \tau_{(v_l,v_{l+1})}$. Note that v_{l+1} and w_{l+1} have the same physical location on the underlying road network. Now also $I_{(l+1)} = 1$.

- Option 3: The request will remain in the vehicle at v_{l+1}. The request has to leave v_l at: $L'_r = l_{v_{l+1}} - \tau_{(v_l,v_{l+1})}$

Finally, all the requests that use an inside transfer at v_l, either to or from vehicle v, are considered. If the request uses an inside transfer to or from vehicle w, then vehicle v and w are platooning from the physical location of v_l to the physical location of v_{l+1}. Let W be the set of all vehicles that are platooning with vehicle v from the physical location of v_l to the physical location of v_{l+1}. Then it holds that $l_{v_l} = \min\{l_{v_l}, \min_{w\in W}\{l_{w_l}\}\}$.
Adding everything together, the latest time vehicle v can depart from vehicle location v_l is

$$l_{v_l} = \min\{(l_{v_{l+1}} - \tau_{(v_l,v_{l+1})} - I_{(l+1)}t_s), \min_{w\in W}\{l_{w_l}\}, \min_{r\in R_{v_l}}\{L'_r\}\}$$

For every vehicle $w \in W$ it is checked if $l_{w_l} > l_{v_l}$. If this is the case the latest departure time of vehicle w at vehicle location w_l needs to be updated as well, since it should be the same as the latest departure time of vehicle v at v_l, because the two vehicles are platooning. The update procedure will be explained in more detail in Section 4.3.2.

When the time window of a vehicle location v_l is updated, time windows of vehicle locations in the same route, as well as in other routes might be affected. The update procedures are explained in the following section.

4.3.2 Updating After an Insertion

When a vehicle location is inserted, or updated, in the route of vehicle v this can affect vehicle locations before and after it in the route. Inserting, or updating, vehicle location v_l between v_{l-1} and v_{l+1} might modify earliest departure times for vehicle locations $v_{l+1}, v_{l+2}, ...,$ and v_{v_N} and latest departure times for vehicle locations $v_1, v_2, ...,$ and v_{l-1}. When one or more transfers are included in this route, the vehicle routes to or from which these transfers happen can be affected as well.

Figure 4.4 illustrates an example with two vehicle routes, v and w (blue and red routes, respectively). There is a transfer of request r from vehicle v to w at the physical location of v_{l+2} and w_{l-2}. If this transfer is an outside transfer and vehicle location v_l is inserted between v_{l-1} and v_{l+1}, then vehicle w can only leave vehicle location w_{l-2} after the arrival of vehicle v. Thus, the earliest departure times for vehicle locations visited by vehicle w in the route from vehicle location w_{l-2} onwards might have to be updated. If vehicle w would transfer a request in this part of the route to a vehicle z, then the route of this vehicle might need to be updated too, etcetera.

If it is an inside transfer, vehicle w has to leave vehicle location w_{l-2} at the same time as vehicle v has to leave v_{l+2}. Similarly, when vehicle location w_l is inserted in the route of vehicle w, this

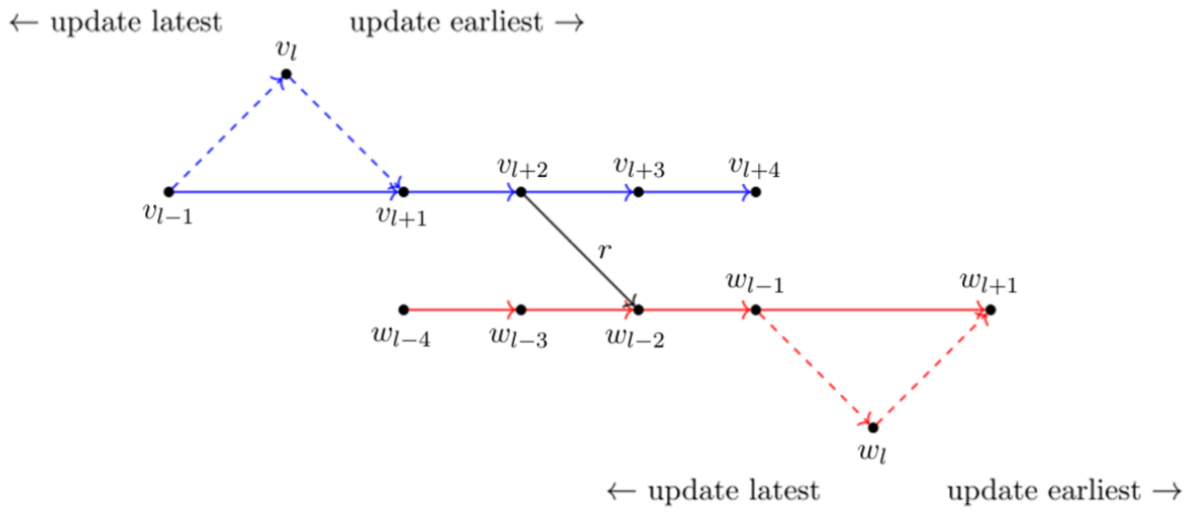


Figure 4.4: Insertion and update procedure. Adapted from (Sampaio et al., 2018). The solid lines are the vehicle routes before the insertion of vehicle locations v_l and w_l. A request r is transferred from vehicle route v at v_{l+2} to vehicle route w at w_{l-2}. Note that vehicle locations v_{l+2} and w_{l-2} have the same physical location on the road network.

could affect latest times for vehicle locations visited by vehicle v, since the latest time vehicle v can arrive at v_{l+2} is bounded by the latest departure time of vehicle w at w_{l-2}. If a vehicle location was inserted into vehicle route v after the transfer of a request to vehicle w, so after vehicle location v_{l+2}, then this insertion would not affect the earliest departure times of vehicle w.

Algorithm 7 gives the pre-order tree traversal updating procedure that can be used for updating the earliest departure times. The updating procedure for the earliest departure times starts at the vehicle location where the earliest departure time needs to be updated (line 1). This can be a recently inserted vehicle location, or a vehicle location to which a request is added or removed. For this vehicle location, the new earliest departure time is calculated (line 2) as explained in Section 4.3.1. If the updated earliest departure time is the same as the old earliest departure time, the updating procedure can stop, as nothing has changed (line 3 and 29). If the updated earliest departure time is greater than the latest departure time, the new solution is infeasible, and the updating procedure can stop (lines 4-7). If the updated earliest departure time is feasible, the affected vehicle locations are determined (lines 8 - 23). The first vehicle location that needs to be updated too is the next vehicle location in the vehicle route (lines 9-11). When there are requests that are transferred from the current vehicle v to another vehicle w, the earliest departure time at the vehicle location of this other vehicle also needs to be updated (lines 12-17). When an outside transfer takes place, vehicle w can only leave this vehicle location when the request is present. When an inside transfer takes place, vehicle w needs to leave at the same time as vehicle v. Also, when there are requests that are transferred from another vehicle w to the current vehicle v, with an inside transfer, the earliest departure time of vehicle w at vehicle location w_l needs to be updated (lines 18-23). As vehicles

Algorithm 7: Tree traversal function to update earliest times

```
Input: Location $v_l$ in vehicle route $v$ that is to be updated.
Current Earliest departure time at all vehicle locations: $E_{v_l}$.
Current Latest departure time at all vehicle locations: $L_{v_l}$.
All vehicle routes in the current solution.
Output: Updated earliest departure time at all vehicle locations.
1  Function PreOrderTraversalEarliest(Vehicle location $v_l$):
2      $e_{v_l} = \max\{(e_{v_{(l-1)}} + \tau_{(v_{l-1},v_l)} + I_s t_s), \max_{w \in W}\{e_{w_l}\}, \max_{r_j \in v_l}\{E_{r_j}\}\}$; // See Section 4.3.1
3      if $e_{v_l} \neq E_{v_l}$ then
4          if $e_{v_l} > L_{v_l}$ then
5              Solution not feasible;
6              Stop updating procedure;
7          else
8              $C \leftarrow \emptyset$; // list of vehicle locations
9              if $v_l$ is not the last vehicle location in vehicle route $v$ then
10                 $C \leftarrow C \cup v_{l+1}$;
11             end
12             if One or more requests are transferred away (outside or inside) from vehicle $v$
                at $v_l$ then
13                 for Each transferred request do
14                     determine vehicle location $w_l$ in vehicle route $w$ to which the request is
                        tranferred to;
15                     $C \leftarrow C \cup w_l$;
16                 end
17             end
18             if One or more requests are transferred (only inside) to vehicle $v$ at $v_l$ then
19                 for Each transferred request do
20                     determine vehicle location $w_l$ in vehicle route $w$ from which the request is
                        tranferred from;
21                     $C \leftarrow C \cup w_l$;
22                 end
23             end
24             for each vehicle location in $C$ do
25                 PreOrderTraversalEarliest(vehicle location);
26             end
27         end
28     else
29         Stop updating procedure;
30     end
31 end
```

v and w are platooning, they need to leave at the same time. When all vehicle locations that are affected are identified, the updating procedure is repeated for all these vehicle locations (lines 24-26).

A similar procedure can be constructed for updating the latest times. This procedure is given in Algorithm 8.

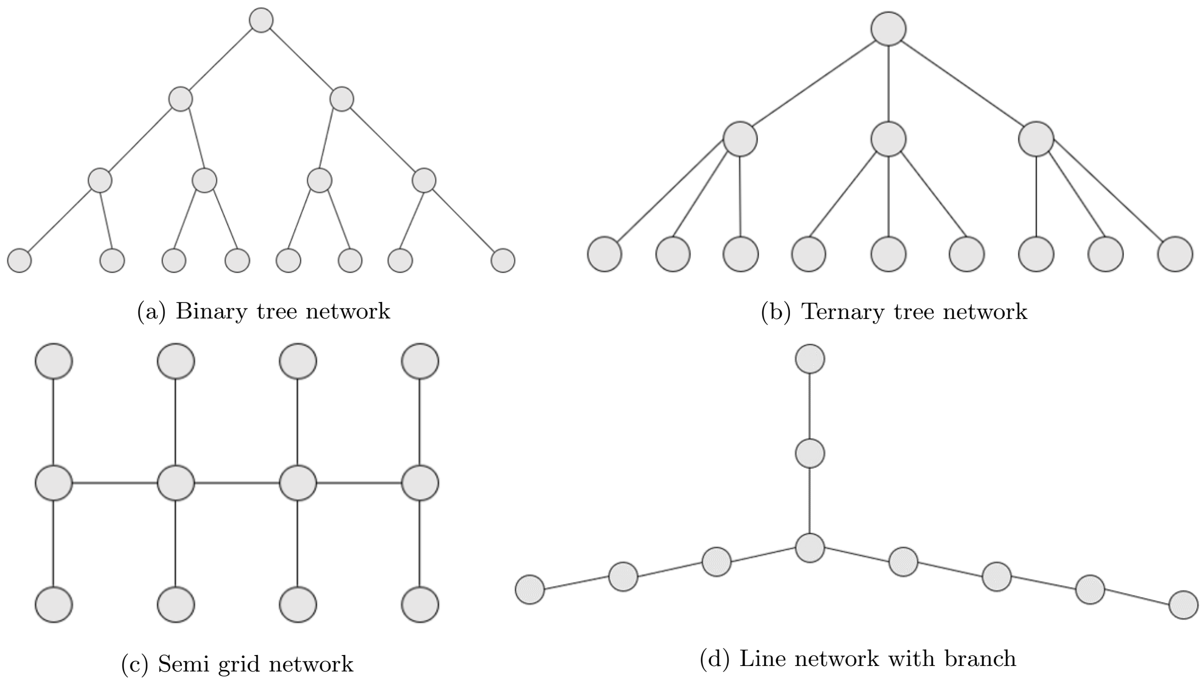
```
Algorithm 8: Tree traversal function to update latest times
   Input: Vehicle location $v_l$ in vehicle route $v$ that is to be updated.
   Current Earliest time of all vehicle locations: $E_{v_l}$.
   Current Latest time of all vehicle locations: $L_{v_l}$.
   All vehicle routes in the current solution.
   Output: Updated Latest departure time at all vehicle locations.
 1 Function PreOrderTraversalLatest(Vehicle location $v_l$):
 2     $l_{v_l} = \min\{(l_{v_{l+1}} - \tau_{(v_l, v_{l+1})}), \min_{w \in W}\{l_{w_l}\}, \min_{r_j \in v_l}\{L_{r_j}\}\}$; // See Section 4.3.1
 3     if $l_{v_l} \neq L_{v_l}$ then
 4         if $l_{v_l} < E_{v_l}$ then
 5             Solution not feasible;
 6             Stop updating procedure;
 7         else
 8             $C \leftarrow \emptyset$; // list of vehicle locations
 9             if $l_{v_l}$ is not the first vehicle location in vehicle route $v$ then
10                 $C \leftarrow C \cup v_{l-1}$;
11             end
12             if One or more requests are transferred (outside or inside transfer) to $v$ at $v_l$
                then
13                 for Each transferred request do
14                     determine vehicle location $w_l$ in vehicle route $w$ from which the request is
                        transferred from;
15                     $C \leftarrow C \cup w_l$;
16                 end
17             end
18             if One or more requests are transferred away (only inside transfer) from $v$ at $v_l$
                then
19                 for Each transferred request do
20                     determine vehicle location $w_l$ in vehicle route $w$ to which the request is
                        transferred to;
21                     $C \leftarrow C \cup w_l$;
22                 end
23             end
24             for Each vehicle location in $C$ do
25                 PreOrderTraversalLatest(vehicle location);
26             end
27         end
28     else
29         Stop updating procedure;
30     end
31 end
```

5. Computational experiments

To test the performance of the developed ALNS algorithm, several computational experiments were conducted. Five different tree network types were designed. One of these network types was designed to be beneficial for inside transfers, especially. Several instances were created for this network type, to test whether inside transfers would actually be used. The instances were solved statically and dynamically, to evaluate the performance of solving the instances dynamically. When an instance is solved statically, all requests are known before the execution of the routes, and the final solution is found before the execution of the routes starts. When an instance is solved dynamically, only some requests are known before the execution of the routes, others become known during execution. The solution is adapted during the execution of the routes. The four other network types are designed to test the performance of the algorithm in different situations.

5.1 Parameter Tuning

The developed ALNS algorithm has a large number of parameters. Parameter tuning was performed on a set of four instances. For each of the network types illustrated in Figure 5.1 an instance with 50 requests was created.



(a) Binary tree network

(b) Ternary tree network

(c) Semi grid network

(d) Line network with branch

Figure 5.1: Several tree networks, each edge has length 1.

These 50 requests all had a slack of 10 minutes and were distributed equally over a two-hour horizon. In retrospect, these instances were not the best choice. As these instances have sparsely distributed requests, both over the locations on the network and in time, most of the instances did not benefit from using transfers at all, so the selected parameter values are probably less effective on more dense instances.

The following strategy for parameter tuning is used. First, the parameter setting of Sampaio et al. (2018) is used as the default setting. This setting is improved by varying the values of a single parameter at a time and keeping the rest fixed. For each value, the set of test instances is solved statically four times. The parameter value yielding the best results is chosen. Each solution is awarded a score to determine which parameter value gives the best result. For each instance, the lowest objective value (z^*) is determined, as well as the highest number of iterations ($t*$), in the fixed runtime of 10 minutes. Then, for each solution, the score is determined as follows: $0.9\frac{z*}{z} + 0.1\frac{t}{t*}$, where z is the objective value of this solution and t the number of iterations. The number of iterations is also taken into account, as this increases the likelihood that a better solution can be found. The next parameter is tuned using the values found so far and using the default values for the parameters that have not yet been tuned. This continues until all relevant parameters have been considered. The values that are considered for each parameter (set) are given in Table 5.2. In Table 5.1 the values of some problem-specific parameters are given.

Table 5.1: Problem-specific parameters

Parameter	Value
Outside transfer penalty (p_{ot})	1
Vehicle cost (p_v)	10
Driving cost per time unit (p_{du})	1
Rejection penalty (p_r)	100
Vehicle stoptime (t_s)	1
Vehicle capacity (Q)	5

The first parameters considered are ψ_1 and ψ_2, which control the fraction of available requests R^* to remove each iteration. The number of requests to be removed each iteration is drawn uniformly from $[\psi_1|R^*|, \psi_2|R^*|]$, so it must hold that $0 < \psi_1 < \psi_2 < 1$. The results indicate that $(\psi_1, \psi_2) = (0.05, 0.1)$ performs best. When more requests are removed per iteration, the newly found solution differs more from the previous solution than when fewer requests are removed per iteration. Thus, removing more requests each iteration diversifies the search more. Probably the other implemented techniques for diversification are already effective enough, and thus removing more requests each iteration is not needed. Removing more requests each iteration increases the time that is needed to reinsert all the requests. This can clearly be seen in Figure 5.2. Thus, fewer iterations can be done in the fixed time. Especially for the cost insertion operator and the regret insertion operator, the time needed increases a lot, when the number of removed requests increases. The reason for this is that with these operators, after the insertion of one request all other requests are reinserted again.

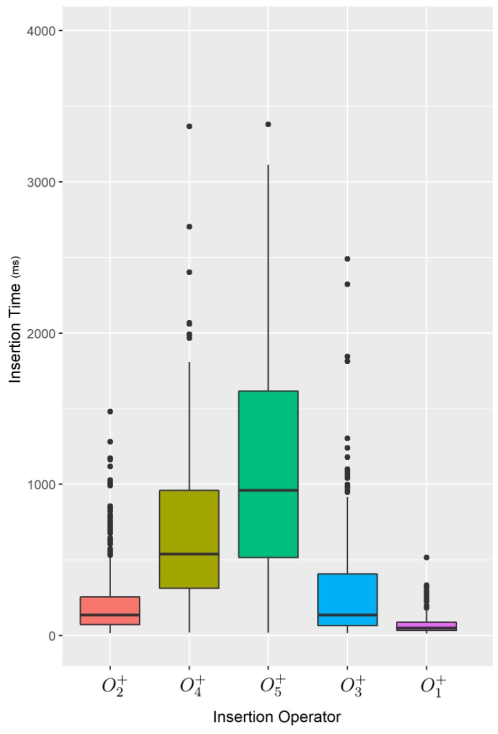
Table 5.2: Calibration of parameters

Parameter	Value	Instance			
		1	2	3	4
(ψ_1, ψ_2)	**(0.05,0.1)**	0.98	0.98	0.91	0.97
	(0.1,0.2)	0.94	0.94	0.87	0.95
	(0.2,0.3)	0.91	0.92	0.82	0.93
ρ	0.2	0.88	0.93	0.90	0.91
	0.5	0.89	0.96	0.89	0.95
	0.8	0.97	0.96	0.89	0.96
ω	**0.1**	0.93	0.96	0.89	0.99
	0.15	0.93	0.94	0.91	0.96
	0.2	0.85	0.92	0.89	0.97
ν	**50**	0.92	0.97	0.86	0.93
	150	0.90	0.97	0.86	0.88
	250	0.86	0.96	0.87	0.84
l	**50**	0.97	0.95	0.91	0.98
	150	0.93	0.92	0.92	0.96
	250	0.92	0.93	0.91	0.96

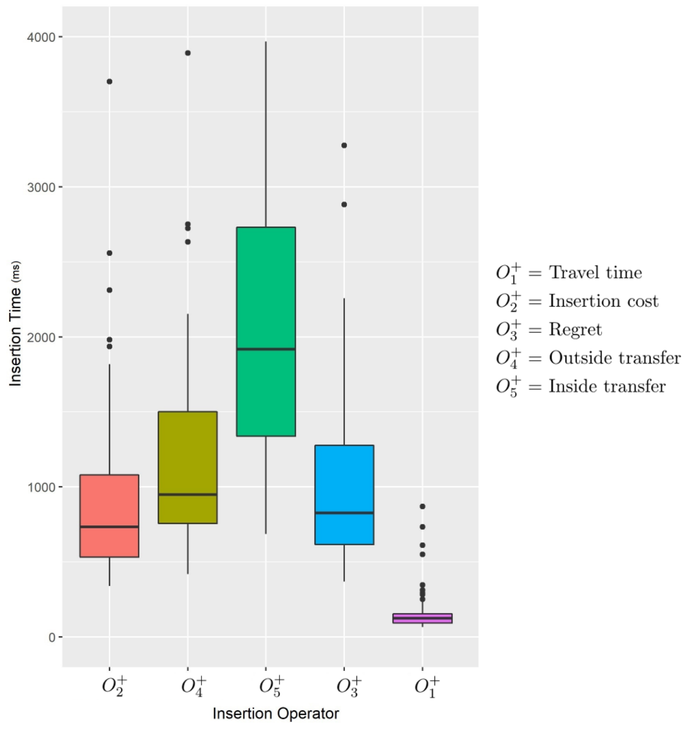
So when the number of requests increases, the time increases more than linear.

The reaction factor, ρ, in the weight-adjustment of the operators is considered second. Here $\rho = 0.8$ performs best. As explained earlier, the selected instances do not benefit from transfers a lot, so the parameter value that enables not selecting these operators the fastest is probably most beneficial. All operators start with the same weight, with $\rho = 0.2$ the weights in the new period depend mostly on the weights in the previous period, thus the insertion operators that use transfers are being selected for quite a long time. These operators also take the longest time to complete the insertion (see Figure 5.2). Thus, as the search runs for a fixed amount of time, the useful operators have less time. This is probably the reason that $\rho = 0.8$ performs best, as more useful iterations can be executed and more iterations can be executed in the fixed time as well. This argument is validated by the fact that for instance 3, the difference in performance between $\rho = 0.2$ and $\rho = 0.8$ is negligible. Instance 3 was the only instance for which the final solution did have transfers.

Next, the initial temperature adjustment factor ω is considered. Now the results indicate that $\omega = 0.1$ performs best. With a smaller omega, the allowed increase in the objective value is smaller. However, it can be seen in Figure 5.3 that the increase in objective value with $\omega = 0.1$ is already quite big at the moments that the temperature is reset. This indicates that $\omega = 0.1$ allows for enough diversification.



(a) $(\psi_1, \psi_2) = (0.05, 0.1)$



(b) $(\psi_1, \psi_2) = (0.2, 0.3)$

Figure 5.2: Box plot for each insertion operator. Showing the time the insertion operators needs to insert all requests previously removed from a solution by a removal operator. Instance 1 is used as an example.

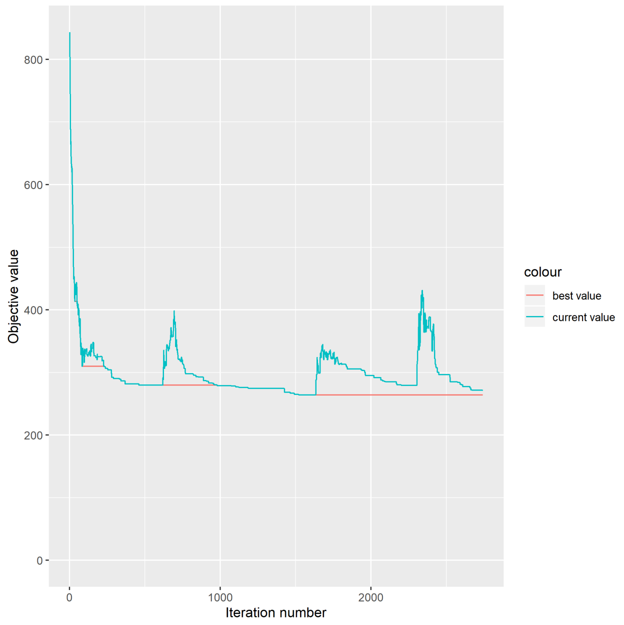


Figure 5.3: Objective value during the search. Instance 1 is used as an example.

The fourth parameter considered is the maximum number of iterations in which no improving solution value can be found; ν. $\nu = 50$ gives the best performance. When allowing for more iterations without improvement, more intensification can be achieved. However, it could also be that in a situation where there are already 50 iterations without an improvement, there will still be no improvement after 150 or even 250 iterations. In that case, it is better to continue the search after 50 iterations, especially since the total number of executed iterations in the fixed runtime of 10 minutes is not very large.

The last parameter is the level length l. This is the number of iterations after which the weight of the operators is updated. The argument given for the reaction factor also holds here. A smaller level length makes sure that the inefficient transfer operators decrease in weight faster, and thereby increase the number of useful iterations with other insertion operators.

Parameters not considered for tuning are the parameters that control the weight-adjustment of the operators. These parameters are set to values used in Sampaio et al. (2018), $\sigma_1 = 40, \sigma_2 = 10, \sigma_1 = 1$. Also the simulated annealing cooling rate $\alpha = 0.98$ is set to the value used in Sampaio et al. (2018).

5.2 Use of inside transfers

To demonstrate the use of inside transfers first a small, stylized instance is created. This instance is depicted in Figure 5.4. This instance consists of nine requests that have a very tight time window

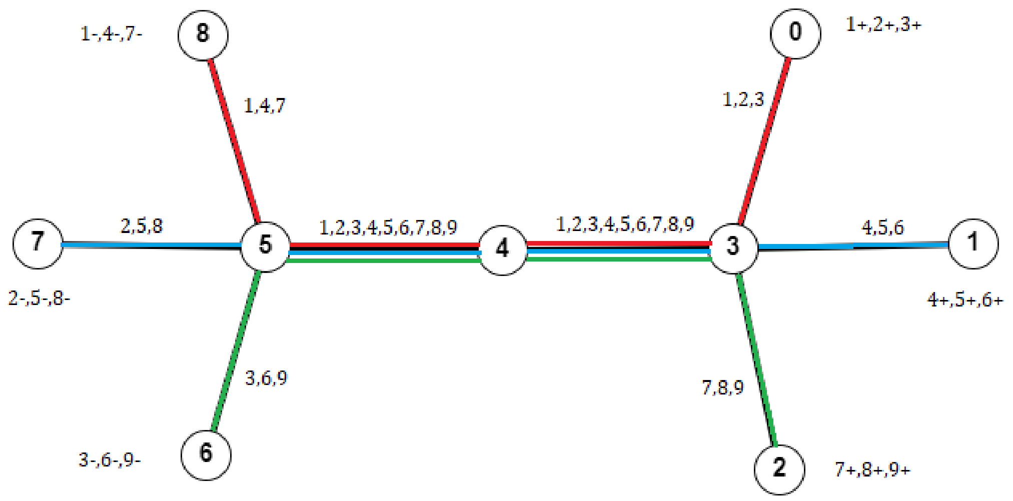


Figure 5.4: Instance demonstrating the use of inside transfers. A network containing nine locations is given. Every road has length 1, and every request has time window [0,5]. The numbers next to a location indicate which requests use this location as a pickup (e.g. 1^+), or as a delivery point (e.g. 1^-). A coloured line indicates a vehicle using that road. The numbers next to a vehicle indicate which requests are transported by this vehicle.

without slack, each having a unique pair of locations as pickup point and delivery point. In a situation without transfers, nine vehicles would be needed to execute these nine requests. As an outside transfer takes a certain amount of time, in a situation with only outside transfers, there would be nine vehicles needed still. When inside transfers are allowed, there are only three vehicles needed, as the vehicles can drive in a coupled manner from location 3 tot location 5 and the requests can transfer during this time. From now on the name ‘small network’ will be used to refer to the network of this instance.

On this same small network, also two larger instances are created. Requests are only allowed to be transported from locations 0, 1 or 2 to locations 6, 7 and 8 (or the other way around). The requests are equally distributed on a horizon of 30 minutes, each request with a slack of 5 minutes. The first instance has 50 requests, the second instance has 100 requests. Each instance is solved in eight different ways. It is solved statically and dynamically. In the static way, the search runs for thirty minutes. For the instance with 50 requests, 17 requests are known beforehand, and for the instance with 100 requests, 26 requests are known beforehand. In the dynamic way, the search runs for ten minutes on the already visible requests and then during the entire planning horizon. In both approaches, the instances is solved in four different settings. In the first setting both transfers types are allowed, in the second setting only inside transfers are allowed, in the third setting only outside transfers are allowed and in the last setting no transfers are allowed at all. Every setting is solved six times. In Table 5.3 the results of the best of these six solutions are reported, as well as the average of these six solutions.

Table 5.3: Results for the two generated instances on the small network

Setting	50 requests				100 requests			
	Static		Dynamic		Static		Dynamic	
	Best	Average	Best	Average	Best	Average	Best	Average
No transfers								
Obj	392	398	355	370	717	760	641.1	675
Veh	14	14.1	10	11.4	36	30.3	20	22.3
In	0	0	0	0	0	0	0	0
Out	0	0	0	0	0	0	0	0
All transfers								
Obj	293	311	293	322	529	618	309	362
Veh	12	13.4	11	12.25	22	25.5	12	13.8
In	1.22	1.06	0.82	0.60	0.9	0.52	1.25	1.17
Out	0	0	0	0.02	0.03	0.22	0	0.03
Only inside								
Obj	299	307	269	311	481	530	330	360
Veh	12	13.25	11	12.25	19	21.3	12	14.1
In	0.92	1.07	0.82	0.72	1.03	1.04	1.17	1.18
Out	0	0	0	0	0	0	0	0
Only outside								
Obj	389	395	344	376	575	631	505.8	529
Veh	14	14.6	11	12.6	22	24.5	19	20.5
In	0	0	0	0	0	0	0	0
Out	0.1	0.13	0.28	0.17	0.68	0.64	0.69	0.64

Obj = Objective value, Veh = number of vehicles, In (Out) = average number of inside (outside) transfers per request

There are several observations to be made. First of all, the ‘only inside transfers’ setting performed better than the ‘no transfers’ setting. This was expected, as the generated network is one where inside transfers are clearly beneficial. The ‘only inside transfers’ setting performed much better than the ‘only outside transfers’ setting as well. This is also reflected in the ‘all transfers’ setting, as in this setting mostly inside transfers are executed. This effect is bigger for the instance with 100 requests, as there are more requests that can be scheduled at the same time. This is an indication that the algorithm includes the right type of transfers when they are beneficial.

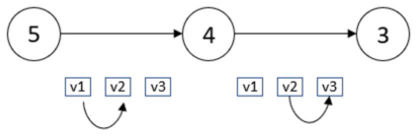
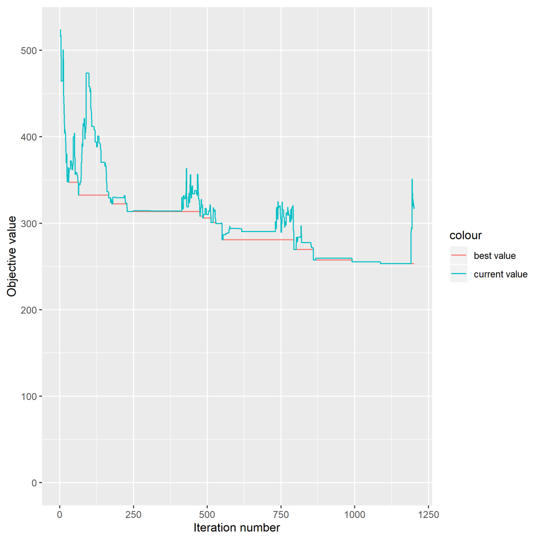


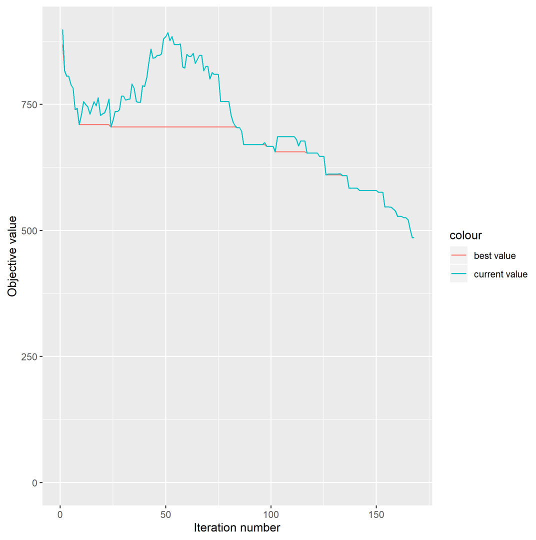
Figure 5.5: A request that uses two inside transfers in the small network

It should also be noted that the number of inside transfers per request is sometimes bigger than 1. The explanation is illustrated in Figure 5.5. Here a request travels from location 5 to location 3 on the small network. The requests start in vehicle 1, but at the same time there are two other vehicles also driving from location 5 to location 3. As an inside transfer comes at no costs, it can happen that the request is first transferred from vehicle 1 to vehicle 2 and then from vehicle 2 to vehicle 3, instead of directly from vehicle 1 to vehicle 3. Of course, this is not a desirable situation, and some sort of check could be incorporated to eliminate these superfluous inside transfers.

Another undesirable fact is that the objective values of the six different solutions of one setting are far apart, especially for the instance with 100 requests. For instance, the average objective value of the ‘all transfers allowed, dynamic’ setting is 17% higher than the objective value of the best solution. This indicates that the algorithm does not reach a good solution in every search because it takes too much computational time. This observation is further supported by the fact that the objective values of the static settings are higher than the objective values of the dynamic settings, even though the search of the static setting has already ran for thirty minutes. Solving an instance statically should give lower objective values than solving an instance dynamically since in the static way all requests are known. However, as can be seen in Figure 5.6 fewer than 200 iterations are executed on the instance with 100 requests. This is not enough to reach a good solution every time, as can be seen from the current value line, which is still decreasing. For the instance with 50 requests, the current value line seems to be flattening somewhat, but 1250 iterations is still on the low side. In the dynamic setting, fewer requests have to be handled each iteration, as some are not visible yet, and a part of the solution is already implemented. When fewer requests have to be handled, there is a higher probability that the heuristic finds a good solution.



(a) instance 50 requests



(b) instance 100 requests

Figure 5.6: Objective value during the search of the static setting, with only inside transfers allowed

For each of the networks used in the parameter tuning, five new instances with 50 requests were created. All requests were planned simultaneously, with a slack of 10 minutes. The pickup and delivery point of each request was selected at random. Since the requests are close to each other in time, solving these instance dynamically makes no sense. Therefore, they are only solved statically. This scenario is further referred to as the 'Cost 1' scenario. For the small network, the instance with 50 requests is included. The searches ran for 30 minutes, executed with four settings; no transfers allowed, all transfers allowed, only inside transfers allowed, and only outside transfers allowed. Each instance is solved five times for each setting. The results of this search for one instance of the small network are shown in Table 5.4. The results of this search for one instance of the Ternary tree network and one instance of the Semi grid network are shown in Table 5.5. For one instance of the Line with branch network and one instance of the Binary tree network, the results are shown in Table 5.5. Each setting is solved five times, the results of the best solution of these five solutions are reported, as well as the average of these five solutions. The results of all the instances can be found in the Appendix.

As for most instances fewer than 2000 iterations were executed in 30 minutes, the search was executed for the same instances for a longer time (4 hours) as well. This scenario is referred to as the 'Cost 1, long' scenario. Finally, a search was also executed in which the cost of an outside transfer was set to zero. This scenario is referred to as the 'Cost 0' scenario. For all instances, when comparing the objective values for the 'Cost 1, long' scenario to the corresponding 'Cost 1' scenario, the objective values (both the best and average) decrease significantly when the running time is increased. This is a further indication that not enough iterations are executed, and that the algorithm runs too slow, even for 50 requests.

Table 5.4: Results for different scenarios and settings for an instance of the small network

Setting	Small network			
	Cost 0		Cost 1	
	Best	Average	Best	Average
No transfers				
Obj	397	402	392	398
Veh	14	14.1	14	14.1
In	0	0	0	0
Out	0	0	0	0
All transfers				
Obj	292	315	293	311
Veh	12	13.4	12	13.4
In	1.08	0.88	1.22	1.06
Out	0	0.013	0	0
Only inside				
Obj	287	309	299	307
Veh	12	13	12	13.25
In	0.98	0.995	0.92	1.07
Out	0	0	0	0
Only outside				
Obj	373	387.2	389	395
Veh	14	14.25	14	14.6
In	0	0	0	0
Out	0.24	0.19	0.1	0.13

On the instance of the small network, the ‘only inside transfers’ setting performed best, both in terms of the objective value and number of vehicles used, for the ‘Cost 1’ scenario. Decreasing the cost of an outside transfer to zero has a different effect for each network type. For the small network, the effects are minimal. In the ‘all transfers’ setting, still only inside transfers are executed. For the ‘only outside transfers’ setting, the average number of outside transfers per request increased from 0.13 to 0.19. This minor increase was expected, as it was already determined that this network benefits most from inside transfers.

On the instance of the Ternary tree network and the instance of the Semi grid network, the ‘only outside transfers’ setting performed best, both in terms of the objective value and number of vehicles used, for the ‘Cost 1’ scenario. For the instance of the Ternary tree network, where outside transfer already performed best in the ‘Cost 1’ scenario, the number of outside transfers per request increases significantly in the ‘Cost 0’ scenario. In the ‘only outside transfers’ setting the number of outside transfers per request increased from 0.71 to 0.96. In the ‘all transfers’ setting it increased from 0.615 to 0.928. For the Semi grid network the number of outside transfers per request increase from 0.6325 to 0.91 for the ‘only outside transfers’ setting and from 0.595 to 0.88 for the ‘all transfers’ setting. This increase in outside transfers is as expected, as the outside transfers were already dominant in the ‘Cost 1’ scenario, for both these instances.

Table 5.5: Results for different scenarios and settings for an instance of the Ternary tree network and the Semi grid network

Setting	Ternary tree network						Semi grid network					
	Cost 0		Cost 1		Cost 1, long		Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average	Best	Average	Best	Average	Best	Average
No transfers												
Obj	-[1]	-	300	309	-[2]	-	-[1]	-	297	304	-[2]	-
Veh	-	-	15	15.75	-	-	-	-	15	15.375	-	-
In	-	-	0	0	-	-	-	-	0	0	-	-
Out	-	-	0	0	-	-	-	-	0	0	-	-
All transfers												
Obj	180	195	232	246	213	218	189	196	225	242	207	217
Veh	11	11.625	13	12.375	11	10.667	11	11.125	11	11.875	9	10.4
In	0	0.005	0.42	0.0725	0.26	0.1933	0	0.0125	0	0.025	0	0.044
Out	1.04	0.9275	0.38	0.615	0.48	0.6067	0.86	0.88	0.74	0.595	0.84	0.712
Only inside												
Obj	-[1]	-	268	280	243	250	-[1]	-	253	260	233	238
Veh	-	-	15	15.125	13	13.4	-	-	14	14.125	12	12.8
In	-	-	0.5	0.33	0.36	0.392	-	-	0.38	0.3525	0.32	0.312
Out	-	-	0	0	0	0	-	-	0	0	0	0
Only outside												
Obj	185	196	220	238	220	223	171	192	205	232	206	213
Veh	11	11.75	11	11.75	11	11	10	11.125	10	11.5	10	10.2
In	0	0	0	0	0	0	0	0	0	0	0	0
Out	0.96	0.96	0.72	0.71	0.72	0.74	0.9	0.91	0.54	0.6325	0.7	0.732

[1] For these settings the ‘Cost 0’ Scenario was not executed, since outside transfers are not allowed.
[2] For these settings the ‘Cost 1, long’ Scenario was not executed, since the were > 10.000 iterations in the ‘Cost 1’ Scenario.

When looking at the ‘all transfers’ setting of the ‘Cost 1’ scenario for the Ternary tree network, the best solution has a higher number of inside transfers than outside transfers, even though the ‘only outside transfers’ setting performed better than the ‘only inside transfers’ setting. This is probably just a coincidence, as when looking at the ‘Cost 1, long’ scenario, the ‘only inside transfers’ setting achieved a best objective value of 243. So probably in the considered solution, some good inside transfers were found early on in the search.

On the instance of the Line with branch network and the instance of the Binary tree network, the ‘only inside transfers’ setting and the ‘only outside transfers’ setting performed equally well when only the objective value is considered, for the ‘Cost 1’ scenario. When looking at the number of used vehicles, the ‘only outside transfers’ setting performs better. This pattern also holds for the ‘Cost 1, long’ scenario.
For the instance of the Line with branch network, the number of outside transfers per request increases significantly for the ‘only outside transfers’ setting, from 0.675 in the ‘Cost 1’ scenario to 1.0125 in the ‘Cost 0’ scenario. However, this is not reflected in the ‘all transfers’ setting. In this setting the number of outside transfers per requests only increases from 0.08 to 0.19. The majority of transfers are still inside transfers. An explanation could lie in the fact that there is no removal

Table 5.6: Results for different scenarios and settings for an instance of the Line with branch network and the Binary tree network

Setting	Line with branch network						Binary tree network					
	Cost 0		Cost 1		Cost 1, long		Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average	Best	Average	Best	Average	Best	Average
No transfers												
Obj	-[1]	-	421	428	-[2]	-	-[1]	-	433	446	-[2]	-
Veh	-	-	17	17	-	-	-	-	19	19.625	-	-
In	-	-	0	0	-	-	-	-	0	0	-	-
Out	-	-	0	0	-	-	-	-	0	0	-	-
All transfers												
Obj	233	255	217	266	228	240	285	300	291	317	273	280
Veh	12	13.25	11	13.5	11	12.4	14	15	14	15.625	14	15
In	0.74	0.6325	0.82	0.6525	0.74	0.796	0.08	0.3675	0.72	0.7225	0.96	1
Out	0.18	0.19	0	0.0775	0	0.004	0.82	0.5275	0.26	0.1175	0.04	0.02
Only inside												
Obj	-[1]	-	266	289	240	247	- [1]	-	308	337	277	285
Veh	-	-	14	14.875	13	13.75	-	-	16	16.625	14	14.4
In	-	-	0.8	0.735	0.76	0.825	-	-	0.92	0.75	0.92	0.908
Out	-	-	0	0	0	0	-	-	0	0	0	0
Only outside												
Obj	220	239	270	286	256	264	284	303	306	334	298	305
Veh	10	11.875	11	12.25	11	11	14	15	13	14.75	13	13
In	0	0	0	0	0	0	0	0	0	0	0	0
Out	0.76	1.0125	0.6	0.675	0.76	0.72	0.86	0.8625	0.7	0.665	0.56	0.712

[1] For these settings the ‘Cost 0’ Scenario was not executed, since outside transfers are not allowed.
[2] For these settings the ‘Cost 1, long’ Scenario was not executed, since the were $>$ 10.000 iterations in the ‘Cost 1’ Scenario.

operator that only removes inside transfers, while such a operator does exist for outside transfers. Also, in the ‘Cost 1 scenario’ the best solution of the ‘all transfers’ setting only has inside transfers, with an objective value of 217, while the objective value of the best solution of the ‘only inside transfers’ setting is 266. This could be just a coincidence, as it is already established that good solutions are not found every time the search is executed. The average objective values are much closer to each other.

For the instance of the Binary tree network, the number of outside transfers per request increased from 0.665 to 0.8625 for the ‘only outside transfers’ setting. For the ‘all transfers‘ setting it increased from 0.1175 to 0.5275. It is not very surprising that in the ‘all transfers’ setting in the ‘Cost 0’ scenario there are still 0.3675 inside transfers per requests, even though the ‘only outside transfers’ setting performed better than the ‘only inside transfers‘ setting. The combination of the fact that there is no specific removal operator for inside transfers and that the objective value of the best solution in the ‘only inside transfers’ setting (308) is only 8.5% higher than the objective value of the best solution in the ‘only outside transfers‘ setting (284) of the ‘Cost 0’ scenario, makes it likely that inside transfers are still used.

The objective values shown in the tables, are the total costs. The total costs consist of two parts, the driving costs and the customer inconvenience, which are the total costs of all the outside transfers. In Table 5.7 the savings coming from ignoring customer inconvenience for each network type are given. These are calculated by comparing solely the average driving costs of the 'all transfers' settings of the 'Cost 1' scenario and the 'Cost 0' scenario.

Table 5.7: Savings from ignoring customer inconvenience in the objective value. In the column 'savings', the savings of the instances discussed in this section are given. The column 'min. (max.) savings' gives the minimal (maximal) savings of all the instances of the respective network type.

	savings (%)	min. savings (%)	max. savings (%)
Ternary tree network	9.3	5.4	12.4
Semi grid network	7.5	4.7	13.9
Line with branch network	2.7	0.1	5.8
Binary tree network	3.5	1.8	9.7

In Table 5.7 it can be clearly seen that instances which benefit more from outside transfers, like the instances of the Ternary tree network and the Semi grid network, have higher savings coming from ignoring the customer convenience. It must be noted that for all these instances, the requests are distributed over a relatively short time horizon. This means that the fixed vehicle costs, 10 per deployed vehicle, are around half of the driving costs, so the savings are likely to go up when the lengths of the vehicle routes increase. The maximum number of outside transfers is already limited to 2 per passenger, so customer convenience is not completely ignored. Considering this, it might be beneficial to set the cost of an outside transfer to zero.

6. Discussion

In this thesis, I have implemented an Advanced Large Neighbourhood Search method for solving the dynamic Pickup and Delivery Problem with Inside Transfers. First, the design of this algorithm is summarized and discussed, next the results of the computational experiments are discussed, and finally some suggestions for further research are given.

The solution method is included in a rolling horizon approach to handle the dynamic announcement of the requests. The decision to accept requests that are announced simultaneously is made by comparing their added costs to the combined rejection penalty. If the combined added costs are lower than the combined penalties, all requests are accepted. It would be better to consider every request separately.

The search starts from an initial solution. This initial solution is easily created by executing each request by a separate vehicle. Of course, this is only possible under the assumption that an unlimited number of vehicles is available. This assumption is not very realistic, but in the objective value a fixed cost per deployed vehicle is included to limit the number of vehicles used. The solution method is composed of several removal and insertion operators which modify the current solution. The removal operators remove requests from the solution, based on specific criteria. A removal operator is included that solely removes requests which make use of an outside transfer. However, I did not include a similar removal operator for inside transfers. The insertion operators reinsert all requests that have been removed, in a specific way; without transfers, with outside transfers or with inside transfers. A combination of the two types of transfers is currently not possible. Thus, fewer options in total are possible, and the solution could be further improved if more combinations were possible.

Parameter tuning was done on instances with a low amount of requests per time unit. As only 50 requests with a maximum duration of 10 minutes were distributed over a two-hour horizon on networks with a minimum of 10 locations, most vehicles could only execute one request at a time. So in none of these instances, it was beneficial to use outside or inside transfers. Therefore, there is no way to be sure if the selected parameters would also perform well on instances in which transfers can occur more frequently. When looking at the behaviour of the algorithm during the entire search, sharp peaks in the objective value are observed after a time in which no improved value was found. This is done to escape local minima, but it would be better if the peaks were lower. This could be achieved by increasing the value of α, the decrease in temperature in every iteration, while simultaneously decreasing ω.

In the computational experiments, I found that in the instance of the small network only inside transfers were used. This was to be expected, as this network was designed to be beneficial for inside transfers. Also, the dynamic setting performed better than the static setting. This was not expected, as all requests are known in the static setting, while in the dynamic setting, initially only a subset of all requests is known. In the instances of two of the other four network types, mostly outside transfers were used. In the other two instances, both transfer types were used. It is currently unclear why these network types used these type of transfers. Ignoring the customer inconvenience leads to maximum savings of 13.9% for the operator of the vehicle fleet.

I also found that the solution method performs poorly when considering computational time. Several factors contribute to the high computational time. The solution method spends most time in the insertion of the requests. In the implementation, the solution is copied for every insertion possibility that is considered. Of course, copying an entire solution means that every location in every vehicle route must be visited, which takes much computational time. Also, the cost insertion operator and the regret insertion operator are not implemented efficiently. When one request is inserted, the insertion possibilities of all other requests are recalculated. This is not necessary when the best insertion possibility of another request is in a different vehicle route than the vehicle routes that execute the request that has just been inserted. It could save much computational time if only the insertion possibilities for the updated vehicle route are recalculated. The inside transfer insertion operator and outside transfer insertion operator take the most time. The number of sub-requests contributes significantly to the number of insertion possibilities. Therefore it could make sense to split a request in a smaller number of sub-requests. This means that transfers are forbidden at some places beforehand. Some flexibility will be lost, but less calculation time is needed for determining the optimal insertion. In the current way in which the neighbourhood size is decreased, I have selected the 5 best solutions each round. This number was chosen at random; I could have included this in the parameter tuning, as this number has considerable influence on the computational time of the transfer insertion operators.

Using sub-requests can not be easily extended to other non-tree network types, as the passenger route can not be determined beforehand. This limits the application of the heuristic to a tree network, whereas in reality the road network of a city is not merely build up by a tree network.

Not considered in this thesis is that the modular autonomous vehicles are electric vehicles and thus have a limited action radius. This could be included in further research, for instance, by including maximum ride times per vehicle.

From the conducted experiments, it can be concluded that certain network types benefit more from inside transfers and other network types benefit more from outside transfers. The reason for this is that both transfer types have their own benefits and disadvantages. For outside transfers a significant disadvantage is that both vehicles need to stop at the transfer location, that the passenger has to leave the first vehicle, maybe wait for the second vehicle to arrive and then enter the second vehicle. However, even though an inside transfer does not cost any time, one of the two vehicles still needs to wait for the other vehicle. Then the vehicles need to platoon to the next location on the route, limiting the flexibility of the system. The time windows of all platooning vehicles are shortened, as all earliest departure times of the vehicles are now equal to the latest earliest departure time, and all latest departure times are now equal to the earliest latest departure time. Especially in situations where the number of vehicles is limited, this could have a significant impact. More experiments should be conducted to evaluate which specific situations benefit from which transfer type. For instance, the slack in the time window could have a considerable influence on the preferred type of transfer, as I expect that inside transfers become more relevant with tighter time windows, as inside transfers do not cost any time. Also varying the number of requests per time unit might be interesting, as transfers are probably more beneficial when there are more requests since there are more combinations and opportunities for a transfer.

Another suggestion for further research is to look at modular autonomous vehicles as a substitute for bus transport. It could be interesting to examine the situation where part of the routes are fixed like a bus route and the flexible part of the route consist of transporting passengers from their pickup points to this fixed route, and from the fixed route to their delivery points. In this situation, passengers could choose to use only the fixed part of the route.

References

Agatz, N. A., Erera, A. L., Savelsbergh, M. W., & Wang, X. (2011). Dynamic ride-sharing: A simulation study in metro Atlanta. *Transportation Research Part B: Methodological*, *45* '(9), 1450–1464. doi: 10.1016/j.trb.2011.05.017

Attanasio, A., Cordeau, J.-F., Ghiani, G., & Laporte, G. (2004). Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, *30*(3), 377 - 387. doi: https://doi.org/10.1016/j.parco.2003.12.001

Baldacci, R., Toth, P., & Vigo, D. (2010). Exact algorithms for routing problems under vehicle capacity constraints. *Ann Oper Res*, *175*(4), 213–245. doi: 10.1007/s10479-009-0650-0

Ben Cheikh, S., Hammadi, S., & Tahon, C. (2015). Agent-based evolutionary cooperative approach for dynamic multi-hop ridematching problem. *IFAC-PapersOnLine*, *28*(3), 887–892. doi: 10.1016/j.ifacol.2015.06.195

Bent, R., & Van Hentenryck, P. (2004). Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, *52*(6), 977–987. doi: 10.1287/opre.1040.0124

Bent, R., & Van Hentenryck, P. (2006). A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, *33*(4), 875–893. doi: 10.1016/j.cor.2004.08.001

Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., & Laporte, G. (2007, may). Static pickup and delivery problems: a classification scheme and survey. *TOP*, *15*(1), 45–47. doi: 10.1007/s11750-007-0015-2

Bouros, P., Sacharidis, D., Dalamagas, T., & Sellis, T. (2011). Dynamic Pickup and Delivery with Transfers. In D. Pfoser et al. (Eds.), *Advances in spatial and temporal databases. sstd 2011. lecture notes in computer science, vol 6849.*

Campbell, A. M., & Wilson, J. H. (2014, jan). Forty years of periodic vehicle routing. *Networks*, *63*(1), 2–15. doi: 10.1002/net.21527

Caramia, M., Italiano, G. F., Oriolo, G., Pacifici, A., & Perugia, A. (2001). Routing a Fleet of Vehicles for Dynamic Combined Pick-up and Deliveries Services. In *Operations research proceedings 2001* (pp. 3–8). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-50282-8_1

Caros, N. S. (2019). *Dynamic operations of a mobility service with en-route transfers (master's thesis)* . doi: 10.13140/RG.2.2.13098.47040

Centraal Bureau voor de Statistiek. (2019). *Personenmobiliteit naar vervoerswijze.* https://www.cbs.nl/nl-nl/maatschappij/verkeer-en-vervoer/transport-en-mobiliteit/mobiliteit/personenmobiliteit/categorie-personenmobiliteit/personenmobiliteit-naar-vervoerswijze.

Cordeau, J. F., & Laporte, G. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, *37*(6), 579–594. doi: 10.1016/S0191-2615(02)00045-0

Cordeau, J. F., & Laporte, G. (2007). The dial-a-ride problem: Models and algorithms. *Annals of Operations Research*, *153*(1), 29–46. doi: 10.1007/s10479-007-0170-8

Cordeau, J. F., Laporte, G., & Ropke, S. (2008). Recent models and algorithms for one-to-one pickup and delivery problems. In B. Golden, S. Raghavan, & E. Wasil (Eds.), *The vehicle routing problem: Latest advances and new challenges* (Vol. 43, pp. 327–357). Springer. doi: 10.1007/978-0-387-77778-$8_1$5

Cortés, C. E., Matamala, M., & Contardo, C. (2010). The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. *European Journal of Operational Research*, *200*(3), 711–724. doi: 10.1016/j.ejor.2009.01.022

Curtois, T., Landa-Silva, D., Qu, Y., & Laesanklang, W. (2018). Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows. *EURO Journal on Transportation and Logistics*, *7*(2), 151–192. doi: 10.1007/s13676-017-0115-6

Dantzig, G. B., & Ramser, J. H. (1959, oct). The Truck Dispatching Problem. *Management Science*, *6*(1), 80–91. doi: 10.1287/mnsc.6.1.80

Deleplanque, S., & Quilliot, A. (2013, jan). Dial-a-Ride Problem with time windows, transshipments, and dynamic transfer points. *IFAC Proceedings Volumes*, *46*(9), 1256–1261. doi: 10.3182/20130619-3-RU-3018.00435

Desrosiers, J., Dumas, Y., & Soumis, F. (1986). A Dynamic Programming Solution of the Large-Scale Single-Vehicle Dial-A-Ride Problem with Time Windows. *American Journal of Mathematical and Management Sciences*, *6*(3-4), 301–325. doi: 10.1080/01966324.1986.10737198

Dumas, Y., Desrosiers, J., & Soumis, F. (1991). The pickup and delivery problem with time windows. *European Journal of Operational Research*, *54*(1), 7–22. doi: 10.1016/0377-2217(91)90319-Q

European Environment Agency. (2019, jul). *Exceedance of air quality standards in urban areas.* https://www.eea.europa.eu/data-and-maps/indicators/exceedance-of-air-quality-limit-3/assessment-5.

Eurostat. (2019). *Passenger cars in the EU.* https://ec.europa.eu/eurostat/statistics-explained/index.php/Passenger_cars_in_the_EU#Overview.

Fabri, A., & Recht, P. (2006). On dynamic pickup and delivery vehicle routing with several time windows and waiting times. *Transportation Research Part B: Methodological*, *40*(4), 335–350. doi: 10.1016/j.trb.2005.04.002

Garcia-Martinez, A., Cascajo, R., Jara-Diaz, S. R., Chowdhury, S., & Monzon, A. (2018). Transfer penalties in multimodal public transport networks. *Transportation Research Part A: Policy and Practice*, *114*, 52-66.

Gecchelin, T., & Webb, J. (2019, march). Modular dynamic ride-sharing transport systems. *Economic Analysis and Policy*, *61*, 111-117. doi: 10.1016/j.eap.2018.12.003

Gendreau, M., Guertin, F., Potvin, J.-Y., & Séguin, R. (2006). Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, *14*(3), 157–174. doi: 10.1016/j.trc.2006.03.002

Gendreau, M., Laporte, G., & Séguin, R. (1996). Stochastic vehicle routing. *European Journal of Operational Research*. doi: 10.1016/0377-2217(95)00050-X

Gendreau, M., & Tarantilis, C. D. (2010). Solving large-scale vehicle routing problems with time windows: The state-of-the-art. *Cirrelt*(4), 1–45.

Guo, Q. W., Chow, J. Y., & Schonfeld, P. (2017). Stochastic dynamic switching in fixed and flexible transit services as market entry-exit real options. *Transportation Research Procedia*, *23*, 380–399. doi: 10.1016/j.trpro.2017.05.022

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Horn, M. E. (2002). Fleet scheduling and dispatching for demand-responsive passenger services. *Transportation Research Part C: Emerging Technologies*, *10*(1), 35–63. doi: 10.1016/S0968-090X(01)00003-1

Kalantari, B., Hill, A. V., & Arora, S. R. (1985). An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, *22*(3), 377–386. doi: 10.1016/0377-2217(85)90257-7

Karagulian, F., Belis, C. A., Dora, C. F. C., Prüss-Ustün, A. M., Bonjour, S., Adair-Rohani, H., & Amann, M. (2015, nov). Contributions to cities' ambient particulate matter (PM): A systematic review of local source contributions at global level. *Atmospheric Environment*, *120*, 475–483. doi: 10.1016/j.atmosenv.2015.08.087

Kerivin, H., Lacroix, M., Mahjoub, A., & Quilliot, A. (2008). The splittable pickup and delivery problem with reloads. *European Journal of Industrial Engineering*, *2*(2), 112-133.

Khelifi, L., Zidi, I., Zidi, K., & Ghedira, K. (2013). A hybrid approach based on Multi-Objective Simulated Annealing and Tabu Search to solve the Dynamic dial a Ride Problem. In *2013 international conference on advanced logistics and transport, icalt 2013.* doi: 10.1109/ICAdLT.2013.6568464

Kleiner, A., Nebel, B., & Ziparo, V. A. (2011). A mechanism for dynamic ride sharing based on parallel auctions. *IJCAI International Joint Conference on Artificial Intelligence*, 266–272. doi: 10.5591/978-1-57735-516-8/IJCAI11-055

Laporte, G. (2009). Fifty years of vehicle routing. *Transportation Science*, *43*(4), 408–416. doi: 10.1287/trsc.1090.0301

Lenstra, J. K., & Rinnooy Kan, A. H. G. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, *11*(2), 221–227. doi: 10.1002/net.3230110211

Li, H., & Lim, A. (2003). A Metaheuristic for the Pickup and Delivery Problem with Time Windows. *International Journal on Artificial Intelligence Tools*, *12*(2), 173–186. doi: 10.1142/S0218213003001186

Lois, A., & Ziliaskopoulos, A. (2017, jan). Online algorithm for dynamic dial a ride problem and its metrics. *Transportation Research Procedia*, *24*, 377–384. doi: 10.1016/J.TRPRO.2017.05.097

Lotfi, S., Abdelghany, K., & Hashemi, H. (2019). Modeling Framework and Decomposition Scheme for On-Demand Mobility Services with Ridesharing and Transfer. *Computer-Aided Civil and Infrastructure Engineering*, *34*, 21–37. doi: 10.1111/mice.12366

Ma, T. Y., Rasulkhani, S., Chow, J. Y., & Klein, S. (2019). A dynamic ridesharing dispatch and idle vehicle repositioning strategy with integrated transit transfers. *Transportation Research Part E: Logistics and Transportation Review*, *128*(June), 417–442. doi: 10.1016/j.tre.2019.07.002

Masoud, N., & Jayakrishnan, R. (2017). A decomposition algorithm to solve the multi-hop Peer-to-Peer ride-matching problem. *Transportation Research Part B: Methodological*, *99*(May), 1–29. doi: 10.1016/j.trb.2017.01.004

Masson, R., Lehuédé, F., & Péton, O. (2014, jan). The Dial-A-Ride Problem with Transfers. *Computers & Operations Research*, *41*, 12–23. doi: 10.1016/J.COR.2013.07.020

Mitrović-Minić, S., & Laporte, G. (2004, aug). Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, *38*(7), 635–655. doi: 10.1016/J.TRB.2003.09.002

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

Nagata, Y., & Kobayashi, S. (2010). Guided Ejection Search for the Pickup and Delivery Problem with Time Windows. In P. Cowling & P. Merz (Eds.), *Evolutionary computation in combinatorial optimization. evocop 2010. lecture notes in computer science, vol 6022.* (1st ed., pp. 202–203). Springer, Berlin, Heidelberg. doi: 10.1007/978-3-642-12139-5$_1$8

Najmi, A., Rey, D., & Rashidi, T. H. (2017). Novel dynamic formulations for real-time ride-sharing systems. *Transportation Research Part E: Logistics and Transportation Review*, *108*(November), 122–140. doi: 10.1016/j.tre.2017.10.009

Nanry, W. P., & Barnes, J. (2000). Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological*, *34*(2), 107–121. doi: 10.1016/S0191-2615(99)00016-8

Next Future Transportation Inc. (2017). https://www.next-future-mobility.com. (Accessed on 30-05-2019)

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

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers and Operations Research*, *34*(8), 2403–2435. doi: 10.1016/j.cor.2005.09.012

Psaraftis, H. N. (1980). Dynamic Programming Solution To the Single Vehicle Many-To-Many Immediate Request Dial-a-Ride Problem. *Transportation Science*, *14*(2), 130–154. doi: 10.1287/trsc.14.2.130

Psaraftis, H. N. (1983a). Analysis of an O(N2) heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem. *Transportation Research Part B: Methodological*, *17*(2), 133–145.

Psaraftis, H. N. (1983b). Exact Algorithm for the Single Vehicle Many-To-Many Dial-a-Ride Problem With Time Windows. *Transportation Science*, *17*(3), 351–357. doi: 10.1287/trsc.17.3.351

Qu, Y., & Bard, J. F. (2012). A GRASP with adaptive large neighborhood search for pickup and delivery problems with transshipment. *Computers and Operations Research*, *39*(10), 2439–2456. doi: 10.1016/j.cor.2011.11.016

Rais, A., Alvelos, F., & Carvalho, M. S. (2014). New mixed integer-programming model for the pickup-and-delivery problem with transshipment. *European Journal of Operational Research*, *235*(3), 530–539. doi: 10.1016/j.ejor.2013.10.038

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. doi: 10.1287/trsc.1050.0135

Sampaio, A. H., Savelsbergh, M. W., Veelenturf, L. P., & van Woensel, T. (2018). The Benefits of Transfers in Crowdsourced Pickup-and-Delivery Systems. *Optimization Online*.

Schaller, B. (2018). *The New Automobility: Lyft, Uber and the Future of American Cities* (Tech. Rep.). Retrieved from `www.schallerconsult.com`

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

Schreieck, M., Safetli, H., Siddiqui, S. A., Pflügler, C., Wiesche, M., & Krcmar, H. (2016). A Matching Algorithm for Dynamic Ridesharing. *Transportation Research Procedia*, *19*(June), 272–285. doi: 10.1016/j.trpro.2016.12.087

Shang, J. S., & Cuff, C. K. (1996). Multicriteria pickup and delivery problem with transfer opportunity. *Computers and Industrial Engineering*, *30*(4), 631–645. doi: 10.1016/0360-8352(95)00181-6

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Cp-98, fourth international conference on principles and practice of constraint programming, lecture notes in computer science* (Vol. 1520, pp. 417–431). doi: 10.1007/3-540-49481-2_30

Stein, D. M. (1978). An asymptotic, probabilistic analysis of a routing problem. *Mathematics of Operations Research*, *3*(2), 89–101.

Stiglic, M., Agatz, N., Savelsbergh, M., & Gradisar, M. (2018). Enhancing urban mobility: Integrating ride-sharing and public transit. *Computers and Operations Research*, *90*, 12–21. doi: 10.1016/j.cor.2017.08.016

Thangiah, S. R., Fergany, A., & Awan, S. (2007). Real-time split-delivery pickup and delivery time window problems with transfers. *Central European Journal of Operations Research*, *15*(4), 329–349. doi: 10.1007/s10100-007-0035-x

Toth, P., & Vigo, D. (1998). Exact solution of the vehicle routing problem. In T. G. Crainic & G. Laporte (Eds.), *Fleet management and logistics* (pp. 1–31). New York: Springer US. doi: 10.1007/978-1-4615-5755-5_1

United Nations. (2019). *World Urbanization Prospects 2018 Highlights* (Tech. Rep.). Department of Economic and Social Affairs, Population Division.

Wilson, N. H. M., & Colvin, N. J. (1977). *Computer control of the rochester dial-a-ride system* (No. 77). Massachusetts Institute of Technology, Center for Transportation Studies.

Xu, H., Chen, Z.-L., Rajagopal, S., & Arunapuram, S. (2003). Solving a practical pickup and delivery problem. *Transportation Science*, *37*(3), 347–364. doi: 10.1287/trsc.37.3.347.16044

Appendix

Table 6.1: Binary tree network, instance 1

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	433	446	433	446	433	446
Veh	19	19.625	19	19.625	19	19.625
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
All transfers						
Obj	285	300	291	317	273	280
Veh	14	15	14	15.625	14	15
In	0.08	0.367	0.72	0.72	0.96	1
Out	0.82	0.5272	0.26	0.1175	0.04	0.02
Only inside						
Obj	308	337	308	337	277	285
Veh	16	16.625	16	16.625	14	14.4
In	0.92	0.75	0.92	0.75	0.92	0.908
Out	0	0	0	0	0	0
Only outside						
Obj	284	303	306	334	298	305
Veh	14	15	13	14.75	13	13
In	0	0	0	0	0	0
Out	0.86	0.86	0.7	0.665	0.56	0.712

Table 6.2: Binary tree network, instance 2

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	419	425	419	425	419	425
Veh	19	19.5	19	19.5	19	19.5
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	262	279	290	315	276	282
Veh	13	14.285	14	15	15	14.6
In	0.16	0.15	0.36	0.2825	0.54	0.68
Out	0.84	0.75	0.44	0.46	0	0.088
Only inside						
Obj	310	339	310	339	287	299
Veh	16	17.125	16	17.125	15	15.4
In	0.76	0.632	0.76	0.632	0.78	0.756
Out	0	0	0	0	0	0
Only outside						
Obj	263	275	310	323	283	291
Veh	14	14.25	14	14.5	12	12.6
In	0	0	0	0	0	0
Out	1.02	0.9075	0.66	0.65	0.82	0.812

Table 6.3: Binary tree network, instance 3

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	422	433	422	433	422	433
Veh	19	19.375	19	19.375	19	19.375
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	260	292	313	334	265	272
Veh	13	14.625	15	16	13	14.2
In	0	0.1425	0.66	0.585	0.82	0.86
Out	0.98	0.845	0.24	0.215	0	0.064
Only inside						
Obj	310	343	310	343	282	285
Veh	17	17.75	17	17.75	14	14.33
In	0.96	0.8475	0.96	0.8475	0.98	0.926
Out	0	0	0	0	0	0
Only outside						
Obj	273	297	329	347	299	302
Veh	14	15.375	15	15.625	13	13
In	0	0	0	0	0	0
Out	0.98	0.965	0.8	0.69	0.72	0.78

Table 6.4: Binary tree network, instance 4

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	385	390	385	390	385	390
Veh	18	18.125	18	18.125	18	18.125
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	297	302	299	321	280	288
Veh	15	15.625	15	15.875	15	15
In	0.12	0.22	0.56	0.405	0.74	0.708
Out	0.78	0.5825	0.06	0.20	0.08	0.06
Only inside						
Obj	317	329	317	329	297	299
Veh	16	16.5	16	16.5	15	15.25
In	0.4	0.5275	0.4	0.5275	0.7	0.65
Out	0	0	0	0	0	0
Only outside						
Obj	268	287	311	330	287	300
Veh	13	14.25	14	15.25	13	13.4
In	0	0	0	0	0	0
Out	0.68	0.7025	0.58	0.5375	0.66	0.672

Table 6.5: Binary tree network, instance 5

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	374	379	374	379	374	379
Veh	17	17.125	17	17.125	17	17.125
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	250	265	292	301	269	277
Veh	13	13.625	13	13.375	12	12.4
In	0	0.0025	0.1	0.0425	0	0.008
Out	0.96	0.82	0.58	0.6225	0.7	0.676
Only inside						
Obj	304	326	304	326	294	304
Veh	16	16.25	16	16.25	15	15.6
In	0.58	0.41	0.58	0.41	0.64	0.544
Out	0	0	0	0	0	0
Only outside						
Obj	246	254	288	301	268	273
Veh	13	12.875	13	13.625	12	12.2
In	0	0	0	0	0	0
Out	0.78	0.88	0.66	0.6575	0.8	0.732

Table 6.6: Semi grid network, instance 1

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	297	304	297	304	297	304
Veh	15	15.375	15	15.375	15	15.375
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	189	196	225	242	207	217
Veh	11	11.125	11	11.875	9	10.4
In	0	0.0125	0	0.025	0	0.044
Out	0.86	0.88	0.74	0.595	0.84	0.712
Only inside						
Obj	253	260	253	260	233	238
Veh	14	14.125	14	14.125	12	12.8
In	0.38	0.3525	0.38	0.3525	0.32	0.312
Out	0	0	0	0	0	0
Only outside						
Obj	171	192	205	232	206	213
Veh	10	11.125	10	11.5	10	10.2
In	0	0	0	0	0	0
Out	0.9	0.91	0.54	0.6325	0.7	0.732

Table 6.7: Semi grid network, instance 2

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	272	283	272	283	272	283
Veh	14	14.5	14	14.5	14	14.5
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	166	178	198	219	191	201
Veh	10	10.625	10	11	9	9.8
In	0	0	0.12	0.085	0	0.048
Out	0.94	1	0.7	0.625	0.72	0.716
Only inside						
Obj	228	237	228	237	207	216
Veh	13	13.75	13	13.75	12	12.4
In	0.32	0.42	0.32	0.42	0.42	0.46
Out	0	0	0	0	0	0
Only outside						
Obj	169	180	216	226	203	207
Veh	10	10.75	11	11.375	10	10
In	0	0	0	0	0	0
Out	1.04	1.0175	0.62	0.755	0.8	0.8

Table 6.8: Semi grid network, instance 3

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	274	282	274	282	274	282
Veh	14	14.375	14	14.375	14	14.375
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	188	198	224	235	209	214
Veh	11	11.5	11	12.25	12	11.8
In	0	0.01	0.04	0.175	0.5	0.304
Out	0.9	0.7625	0.68	0.3825	0.16	0.264
Only inside						
Obj	217	237	217	237	217	222
Veh	12	13.125	12	13.125	12	12.6
In	0.36	0.365	0.36	0.365	0.26	0.424
Out	0	0	0	0	0	0
Only outside						
Obj	187	193	225	233	210	214
Veh	11	11.375	11	11.625	10	10.6
In	0	0	0	0	0	0
Out	0.8	0.8525	0.62	0.565	0.6	0.608

Table 6.9: Semi grid network, instance 4

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	335	337	335	337	335	337
Veh	17	16.875	17	16.875	17	16.875
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	181	194	223	245	223	227
Veh	10	10.875	10	11.375	10	10.2
In	0	0.01	0	0.0525	0	0.064
Out	1.2	1.085	0.9	0.8275	0.88	0.872
Only inside						
Obj	246	272	246	272	246	251
Veh	14	14.875	14	14.875	13	13.8
In	0.72	0.555	0.72	0.555	0.5	0.62
Out	0	0	0	0	0	0
Only outside						
Obj	180	192	227	239	221	224
Veh	10	10.875	10	11	10	10
In	0	0	0	0	0	0
Out	1.18	1.0625	0.84	0.83	0.92	0.92

Table 6.10: Semi grid network, instance 5

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	320	333	320	333	320	333
Veh	16	16.375	16	16.375	16	16.375
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	184	201	235	254	211	218
Veh	10	11.375	12	12.875	11	12
In	0	0.0025	0.3	0.25	0.4	0.488
Out	0.92	0.99	0.16	0.41	0.44	0.292
Only inside						
Obj	231	251	231	251	216	228
Veh	13	13.625	13	13.625	12	12.4
In	0.6	0.54	0.6	0.54	0.62	0.552
Out	0	0	0	0	0	0
Only outside						
Obj	181	192	233	248	224	227
Veh	10	11	11	11.75	11	11
In	0	0	0	0	0	0
Out	1.06	0.99	0.6	0.7775	0.72	0.792

Table 6.11: Ternary tree network, instance 1

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	300	309	300	309	300	309
Veh	15	15.75	15	15.75	15	15.75
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	180	195	232	246	213	218
Veh	11	11.625	13	12.375	11	10.66
In	0	0.005	0.42	0.0725	0.26	0.193
Out	1.04	0.9275	0.38	0.615	0.48	0.606
Only inside						
Obj	268	280	268	280	243	250
Veh	15	15.125	15	15.125	13	13.4
In	0.5	0.33	0.5	0.33	0.36	0.392
Out	0	0	0	0	0	0
Only outside						
Obj	185	196	220	238	220	223
Veh	11	11.75	11	11.75	11	11
In	0	0	0	0	0	0
Out	0.96	0.96	0.72	0.71	0.72	0.74

Table 6.12: Ternary tree network, instance 2

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	299	308	299	308	299	308
Veh	15	15.625	15	15.625	15	15.625
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	183	188	231	239	218	223
Veh	11	11	11	11.375	10	10.6
In	0	0	0	0	0	0.02
Out	1.1	1.0275	0.8	0.77	0.88	0.796
Only inside						
Obj	283	290	283	290	256	265
Veh	15	15.5	15	15.5	14	14.2
In	0.32	0.265	0.32	0.265	0.4	0.344
Out	0	0	0	0	0	0
Only outside						
Obj	176	188	232	238	210	219
Veh	10	10.875	11	11.25	10	10.2
In	0	0	0	0	0	0
Out	1.02	0.9925	0.92	0.8	0.66	0.78

Table 6.13: Ternary tree network, instance 3

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	291	297	291	297	291	297
Veh	15	15	15	15	15	15
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	189	197	222	243	217	224
Veh	11	11.625	11	12.25	12	11.2
In	0	0.02	0	0.055	0.44	0.088
Out	0.72	0.825	0.62	0.5075	0.32	0.596
Only inside						
Obj	263	278	263	278	246	254
Veh	15	14.75	15	14.75	14	13.8
In	0.34	0.2225	0.34	0.2225	0.42	0.368
Out	0	0	0	0	0	0
Only outside4						
Obj	186	194	221	241	218	222
Veh	11	11.375	11	12	10	10.8
In	0	0	0	0	0	0
Out	0.72	0.895	0.6	0.6175	0.7	0.7

Table 6.14: Ternary tree network, instance 4

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	309	319	309	319	309	319
Veh	15	15.75	15	15.75	15	15.75
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	193	202	227	248	235	249
Veh	11	11.875	11	12.125	11	12.2
In	0	0	0.02	0.07	0	0.08
Out	1.04	0.9925	0.72	0.6875	0.78	0.66
Only inside						
Obj	267	277	267	277	267	277
Veh	15	15.125	15	15.125	15	15.2
In	0.52	0.4325	0.52	0.4325	0.52	0.428
Out	0	0	0	0	0	0
Only outside						
Obj	184	194	229	243	229	245
Veh	11	11.5	11	11.875	11	11.8
In	0	0	0	0	0	0
Out	1	1	0.82	0.7825	0.82	0.784

Table 6.15: Ternary tree network, instance 5

Setting	Cost 0		Cost 1		Cost 1, long *	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	313	321	313	321		
Veh	16	16.125	16	16.125		
In	0	0	0	0		
Out	0	0	0	0		
Regular						
Obj	202	210	222	250		
Veh	12	12.375	13	13.875		
In	0	0.025	0.54	0.445		
Out	0.94	0.9225	0.08	0.205		
Only inside						
Obj	242	256	242	256		
Veh	14	14.125	14	14.125		
In	0.56	0.4925	0.56	0.4925		
Out	0	0	0	0		
Only outside						
Obj	188	204	240	252		
Veh	11	12	12	12.625		
In	0	0	0	0		
Out	1.02	0.93	0.66	0.645		

* java ran into a StackOverflowError while executing this instance

Table 6.16: Line network with branch, instance 1

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	421	428	421	428	421	428
Veh	17	17	17	17	17	17
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	233	255	217	266	228	240
Veh	12	13.25	11	13.5	11	12.4
In	0.74	0.6325	0.82	0.6525	0.74	0.796
Out	0.18	0.19	0	0.0775	0	0.004
Only inside						
Obj	266	289	266	289	240	247
Veh	14	14.875	14	14.875	13	13.75
In	0.8	0.735	0.8	0.735	0.76	0.825
Out	0	0	0	0	0	0
Only outside						
Obj	220	239	270	286	256	264
Veh	10	11.875	11	12.25	11	11
In	0	0	0	0	0	0
Out	0.76	1.0125	0.6	0.675	0.76	0.72

Table 6.17: Line network with branch, instance 2

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	371	378	371	378	371	378
Veh	16	15.5	16	15.5	16	15.5
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	213	252	258	274	239	247
Veh	10	12.75	13	13.375	13	13
In	0	0.32	0.6	0.5675	0.74	0.68
Out	0.9	0.5575	0.06	0.1275	0.02	0.064
Only inside						
Obj	254	276	254	276	248	251
Veh	15	14.375	15	14.375	12	12.8
In	0.84	0.7025	0.84	0.7025	0.48	0.664
Out	0	0	0	0	0	0
Only outside						
Obj	210	232	269	281	246	252
Veh	10	11.375	11	11.75	10	10.2
In	0	0	0	0	0	0
Out	0.98	0.98	0.78	0.7575	0.66	0.7

Table 6.18: Line network with branch, instance 3

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	292	299	292	299	292	299
Veh	13	13	13	13	13	13
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	228	243	232	248	220	226
Veh	12	13.125	13	12.625	11	12
In	0.08	0.305	0.54	0.41	0.54	0.544
Out	0.62	0.355	0	0.0375	0.08	0.016
Only inside						
Obj	234	253	234	253	214	233
Veh	12	12.625	12	12.625	11	11.5
In	0.62	0.49	0.62	0.49	0.6	0.535
Out	0	0	0	0	0	0
Only outside						
Obj	235	241	248	259	232	239
Veh	12	12.125	11	11.5	10	10.6
In	0	0	0	0	0	0
Out	0.76	0.6525	0.38	0.4175	0.5	0.516

Table 6.19: Line network with branch, instance 4

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	368	407	368	407	368	407
Veh	15	16.75	15	16.75	15	16.75
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	228	259	239	271	240	250
Veh	12	13	11	13.5	12	12.8
In	0.64	0.585	0.54	0.6375	0.74	0.744
Out	0.04	0.1675	0.18	0.0925	0.06	0.064
Only inside						
Obj	269	285	269	285	254	254
Veh	13	14.375	13	14.375	12	13
In	0.62	0.72	0.62	0.72	0.76	0.85
Out	0	0	0	0	0	0
Only outside						
Obj	228	260	271	285	252	261
Veh	11	12.625	11	12.25	10	10.6
In	0	0	0	0	0	0
Out	0.88	0.925	0.48	0.5625	0.56	0.592

Table 6.20: Line network with branch, instance 5

Setting	Cost 0		Cost 1		Cost 1, long	
	Best	Average	Best	Average	Best	Average
No transfers						
Obj	354	368	354	368	354	368
Veh	15	15.625	15	15.625	15	15.625
In	0	0	0	0	0	0
Out	0	0	0	0	0	0
Regular						
Obj	256	284	275	286	236	256
Veh	13	15.25	14	15.5	11	13.75
In	0.64	0.5775	0.72	0.68	0.6	0.685
Out	0.12	0.1675	0.06	0.035	0.04	0.015
Only inside						
Obj	249	277	249	277	253	256
Veh	12	14	12	14	13	13.6
In	0.68	0.64	0.68	0.64	0.7	0.756
Out	0	0	0	0	0	0
Only outside						
Obj	251	262	275	287	259	264
Veh	13	13.5	12	12.5	11	11.2
In	0	0	0	0	0	0
Out	0.94	0.88	0.58	0.545	0.52	0.556