Universiteit Leiden
The Netherlands

Bachelor Data Science and Artificial Intelligence

Optimization for Food Delivery Under Uncertainty

Kay van Noordenne

First & second supervisors:
Dr. Yingjie Fan & Wei Liu

RESEARCH PROPOSAL BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/07/2026

Abstract

Food delivery apps are becoming increasingly popular, since they enables customers to receive meals prepared by restaurants, easily at their door. A fundamental part of how these platforms work, is the planning of routes and assigning the right drivers to deliver the meals. This results in a Dynamic Multi-Vehicle Stochastic Food Delivery Route Planning Problem (DMS-FDRPP). The field of vehicle routing problems is heavily researched. Since these route planning problems are considered NP-hard, natural computing algorithms prove to be very efficient in these environments, but most research lacks the possibility of stochasticity and its impact. Therefore, this research compares six different meta-heuristics (Greedy Randomized Adaptive Search Procedure (GRASP), Simulated Annealing (SA), Genetic Algorithm (GA), Artificial Bee Colony (ABC), Tabu-Search (TS) and Adaptive Large Neighborhood Search (ALNS)) in their ability to minimize costs as well as how they handle batching in environments of varying sizes and different levels of stochasticity. It is concluded that higher levels of stochasticity increase the customer waiting times, while decreasing the driving distances. The results show that GA overall performs the best, with being the top-performing or close to it in each comparison. ALNS shows to achieve the lowest costs, but requires significantly more evaluations and runtime. GRASP shows to have large scaling issues of costs with larger environments and higher stochasticity, but it is reaching the highest batching values, followed by GA and ALNS.

Contents

1 Introduction

1.1 Context

Currently, food delivery is getting increasingly more popular, meals can be ordered through a Food Delivery App (FDA), these apps then make sure the order is sent to the restaurant you picked, and a driver will be scheduled to pick the food up when it is ready and deliver it to the customer as fast as possible. The effort required by customers to receive quality meals is decreased massively by these services. The global market value of online food delivery in 2026 is estimated to be $326.4 billion USD, with an average yearly growth of 10.3% since 2020 and an expected yearly growth of 13.8% until 2033. In 2025 food deliveries from restaurants to customers represented 65% of this market [Per26]. This market is spread globally with an estimated 2.1 billion users in 2024 [Del26], this is approximately 26% of the world population. This shows that it is an immense market which has grown steadily over the last years and is expected to expand even faster in the coming years. Therefore it is a highly actual and important subject.

The delivery of these orders requires quite some logistics, when having a set of orders that need to be delivered, and a set of drivers, the FDA needs to assign the orders to the right drivers, and plan the sequence in which location have to be visited for the optimal route. This is what is considered the Food Delivery Route Planning Problem (FDRPP) [WWW+21]. While this already seems challenging, the real-world is not deterministic, there are a lot of stochastic elements that can delay deliveries. In addition, this is not a problem where all orders are known beforehand, orders can be placed at anytime, with only a small window for dispatching a driver to the restaurant. This research will focus on this exact problem, the Dynamic Multi-Vehicle Stochastic Food Delivery Route Planning Problem (DMS-FDRPP).

The field of Natural Computing focuses on solving highly complex problems with relatively simple methods inspired by nature. The field of Vehicle Routing Problems (VRPs), such as this DMS-FDRPP, but also other Pickup and Delivery Problems (PDPs) or regular routing problems like the Traveling Salesman Problem (TSP) all are considered NP-hard problems, meaning that the problems expand exponentially in size, resulting in the fact that exhaustive search is not an option. Therefore Natural Computing algorithms prove to be highly efficient in solving these types of problems.

1.2 Research questions

The existing research shows that FDRPP is an important topic based on optimizing customer satisfaction and cost reduction. There is extensive research in the field of deliveries, but most only use a low level of stochasticity. In addition, most research focuses on developing new methods, instead of comparative research. Lastly, there is little known about the degrees of batching in urgent PDP. Therefore, this paper will research the FDRPP with more extensive stochastic variables while still allowing and monitoring batching. With these main goals, the following questions will be answered:

- **RQ1:** How do different natural computing algorithms compare in optimization of route

planning for food delivery in dynamic stochastic environments based on customer waiting times, driving distances, number of evaluations and run-time?

- **RQ2:** What is the impact of stochasticity on batching in food delivery routing in dynamic stochastic environments using different natural computing algorithms measured by the actively used capacity of vehicles?

1.3 Thesis overview

This chapter contains the introduction to the topic, the background of the problem and the research questions. Section 2 discusses related work and the gaps in the literature. Section 3 states the problem, including its mathematical outline. Section 3 contains the different algorithms that will be compared and their methodologies, including the different mutations these rely on. Section 5 discusses how the experiments are set-up and which will be run. Section 6 shows the results from the experiments and discusses insights. Section 7 concludes the study.
This study was done as a bachelor thesis at LIACS, under supervision of Dr. Yingjie Fan and Wei Liu.

2 Related Work

The field of vehicle routing problems (VRPs) is an important part of logistics and has received a lot of attention by researchers. There are numerous different versions and extensions to the classic VRP, each with its own specific constrains. The focus of this research is specifically the FDRPP which has become increasingly relevant since the rise of food delivery apps (FDAs), but there has been little research on this exact topic. The FDRPP as defined in this research is similar to the methods used by Wang et al. in 2021 [WWW+21]. There has been a lot of research on similar topics closely related to the FDRPP, where each has it's own slightly different characteristics. The DMS-FDRPP is a version of pickup and delivery problem (PDP), with multiple vehicles, time constrains, urgency and in a highly stochastic and dynamic environment.

There are a lot of different versions of PDPs, a constraint applicable on the FDRPP and that returns in research, are restricted time-windows, also named PDPTW this limits the possible routes by setting a certain time window in which the location has to be visited. This constraint is widely used in research since it is applicable to many real-world scenarios, therefore it is applied to a number of different environments. Chami et al. [CMMF17] proposed an algorithm for the Selective PDP with Time Windows and Paired Demands (SPDPTWPD), in which strict time windows are set for each location, forcing vehicles to arrive before the time window closed, or having to wait when arriving too early. Ghilas et al. [GDW16] used it as an extra constraint while focusing on adapting schedules and travel lines in the distribution network. Furthermore Cherkesly et al. [CDL15, CDIL16], applied it in environments with capacity constraints and LIFO loading.

Besides time-windows another factor is urgency, which is rarely seen in studies, but Ferrucci & Bock [FB14] did a study in a dynamic PDP with real-time control (DPDPRC) which incorporates the fact that the deliveries have an urgency, meaning that the route planning has to be fast, and forcing the vehicles to take limited detours for other deliveries. Another related problem is the

on-demand food delivery (ODFD) has a lot in common with the FDRPP, this problem focuses on dynamic urgent food delivery to customers from a single depot. Mogyorósi [Mog25] did comparative research of different Natural Computing algorithms on this problem.

All these problems are versions of the more common Vehicle Routing Problem (VRP), which is widely studied by researchers like Wu & Hifi [WH20] and Chen et al.[CCW+18], which researched versions of the VRP with specific time windows in which certain locations have to be reached, which is named the VRP with time windows (VRPTW). Wu & Hifi focused on a robust version of the problem, where the stochasticity is indicated by a set of possible scenarios. While the research of Chen et al. is more aimed at optimization in a dynamic environment.

2.1 Pickup Locations

The existing research on problems related to the FDRPP can be separated by their the distribution of their respective pickup or starting locations. Most research on the regular VRP focuses on single starting points, since they do not have to visit pickup locations. For most research aimed at food delivery this means that it is aimed at a single restaurant where the delivery drivers come back to. In other cases the reality is simplified and all restaurants are merged into a single location referred to as a cloud-kitchen [BDJ25]. But most research focuses on other kinds of businesses with routing problems, such as package delivery or house visits, which have a single starting location working as a depot [CCW+18, Mog25, WH20]. These different approaches generally work the same, since each approach has a single location from which a route is generated and to which the driver has to return. There are also studies which incorporate different depots, Tiwari & Sharma [TS23] and Rios & Xavier [OX25] performed studies on environments that acts like a hybrid, it uses multiple depots from which the vehicles depart, their methods take into account the different locations, and the algorithms tries to decide which depot is closer and more efficient to leave from. Kachitvichyanukul et al. [KSK15] did a study on a similar environment, with the addition that some deliveries or pickups can be at a depot and each vehicle does have to return to their starting depot.

PDP environments also take pickup-locations into account, which have to be visited before the delivery locations. This results in different constraints and behavior of route planning algorithms. Wang [WWW+21] did research on the FDRPP with this environment. Yu et al. [YLM+24] studied an environment that incorporates both deliveries and pickups from a depot and multiple stores at fixed locations, while also delivering goods to varying customer locations. Wang et al. [WMZS15] added a possibility to the PDP that a single location can be both used as a pickup and delivery location, but each having their own time window. Naccache et al. [NCC18] proposed an interesting addition to the classic PDP, they did a study where multiple pickups can be required for a single delivery. Similarly, in the environment Yamashita et al. [YdMR19] used, a single transport could not only require multiple pickups but also multiple deliveries, all based on the capacity of the vehicles and the locations. There are also multiple studies that incorporate transshipment, to split long journeys into multiple smaller trips [XW23, DAG18, QB12, NMAGAL16], this creates extra locations that need to be taken into account for possible pickups and deliveries.

2.2 Batching

Batching can allow vehicles to carry multiple goods for different locations at the same time, to reduce travel distance, but the time between departure and delivery for each good can be increased, resulting in a trade-off between time and costs. Most research on VRP uses batching and can easily do so if the vehicles depart from a single location, since only the delivery locations have to be grouped [Mog25, BDJ25, TS23]. For PDP and FDRPP it is harder as observed by Wang [WWW+21], since each delivery has two locations that have to be visited, and in FDRPP each customer expects their delivery as soon as possible. The customer does not want to wait extensively for the driver to pickup other packages. In large transport, vehicles mostly have an additional constraint of LIFO loading, meaning that the last object loaded to the vehicle must be delivered first, before allowing access to the next stored good. This creates extra difficulties for the route planning and is research by Pereira et al. [PMU22] and Cherkesly et al. [CDIL16] among others. In long-distance transports there is also the possibility of transshipment [XW23, DAG18], where vehicles pickup the packages at a location and transfer these to a transshipment location from which a batch can be send to another transshipment location before distribution to delivery locations. The solution of Ghilas et al. [GDW16] follows a similar setup by using public transport to transport packages between hubs.

2.3 Stochasticity

Stochastic environments fundamentally change how planned routes perform and the type of stochastic variance has a great impact on how routes should be altered. The real-world is highly stochastic, even-though most research focuses on deterministic environments with only randomized orders. Random orders are the easiest way to add variation to the simulations and can easily be used as benchmarks for new methods. Some examples are Wang et al. [WWW+21], Tiwari & Sharma [TS23] and Naccache et al. [NCC18].

But there has also been some research on stochastic environments, a common stochastic variable is travel time approximation [Mog25, UGMT20, BDJ25]. This simulates traffic and slightly varies how long a route takes, based on a Gaussian distribution. Even more used, is stochastic request generation, [UGMT20, OX25], which randomly create new customer orders and time windows. Ferrucci & Bock [FB14] applied, as one of the few, more stochastic variables such as vehicle disturbances. Wu & Hifi [WH20] used a different approach, they applied robust variability, this is scenario based uncertainty, which means that they modeled a set of possible discrete scenarios, which are randomly picked with equal probability, instead of using random variables with varying probability.

As one of the few studies in this field, Huang [Hua15] compared the proposed Tabu-Search algorithm on both stochastic and dynamic environments. This comparison showed that as expected the algorithm delivers sightly less optimal results in a stochastic environment caused by the uncertainty, and showing how important it is to take this stochasticity into account in simulations and research, since this stochasticity is more similar to the real-world.

2.4 Methods

Research in the fields of VRP and PDP are highly aimed at proposing new or existing methods for solving new problems. This creates a wide range of algorithms that are applied and can be compared in this field.

One of the most popular methods is Adaptive Large Neighborhood Search (ALNS), it uses destroy and repair steps, to remove and rebuild large sections of the solution, the specific methods used to destroy and rebuild are assigned finesses. This method is applied in a wide range of environments, including a Dynamic VRP [CCW+18], a Muli-PDP with Time Windows [NCC18] and Dynamic PDP with Transshipments and LIFO-constraints [XW23]. In these studies it proved to be a high performing method. In comparison to Danloup et al. [DAG18] which compared a regular Large Neighborhood Search (LNS) with a Genetic Algorithm (GA) in a PDP environment with transshipments, where it showed that GA actually outperforms LNS in this setup.

Another widely used approach is Tabu-Search (TS), is uses a tabu-list with a set of previously visited solutions or applied moves, of a fixed length, which prevents the algorithm to be repeated. It is applied in a wide range of environments, but each showed promising results. Tiwari & Sharma [TS23] applied Tabu-Search in a VRP with capacity constraints. Their research showed that although it is significantly slower than some other search methods, it does have more optimal results significantly outperforming other local-search heuristics. Huang [Hua15] performed a separation algorithm to decrease the search space before applying Tabu-Search to increase its efficiency. Ferruci & Bock [FB14] stores fingerprints of solutions, which significantly reduces the amount of memory needed to store the highly complex previous states states. Erdoğan et al. [EBLV12] performed comparisons between regular Tabu-Search, Iterative Local Search (ILS) and Iterative Tabu-Search (ITS) in a TSP with pickups and deliveries. Which showed that TS significantly outperformed ILS and ITS. Lastly, Rios & Xavier [OX25] proposed a TS algorithm specifically for stochastic VRP.

A different algorithm is Simulated Annealing (SA), this algorithm is widely used in the field of natural computing and serves as a basis for a number of new methods. Afifi et al. [ADM13] showed that SA is able to quickly find optimal solutions in a VRP environment with time windows. Mogyorósi [Mog25] compared multiple natural computing algorithms on a dynamic VRP showing that SA has decent, although not outstanding performance, but it does only need a low amount of function calls. Wang et al. [WMZS15] proposed a version of SA that proved to be performing similarly to GA. According to a study done by Yu et al. [YLM+24] an efficient SA algorithm is highly effective at finding global or near-global optima, while also needing a low amount of compute.

The fourth type is Genetic Algorithms (GAs), which modifies solutions based on simulating a population. This method is minimally used in research on PDP, but does show promising results. Chami et al. [CMMF17] showed that it has great performance and can find the optimal solution in most situations. Danloup et al. [DAG18] proposed an algorithm based on GA which gained very promising results. In addition, Mogyorósi [Mog25] did a comparative study which showed that GA is one of the best performing methods when compared to other natural computing algorithms.

Artificial Bee Colonies (ABCs) are another type of algorithms with promising results, but rarely

used. Zhan et al. [ZSSC21] proposed a type of ABC which greatly outperformed the greedy benchmark. In addition, Mogyorósi [Mog25] found that, compared to a set of other natural computing algorithms, it was the best performing one.

One of the most common approaches, especially for benchmarks, are greedy algorithms. In VRPs and PDPs the greedy randomized adaptive search procedure (GRASP) is mostly used. By for example Zhan et al. [ZSSC21] as benchmark for their modified ABC algorithm, and by Qu & Bard [QB12] in combination with ALNS for PDPs with transshipment.

Lastly, there are also a number of other algorithms proposed for these problems. Wang et al. [WWW+21] for example proposed a XGBoost-enhanced method. Cherkesly et al. and Wolfinger et al. [CDIL16, WSG21] both proposed branch-and-cut algorithms. While Kachitvichyanukul et al. [KSK15] used a Particle Swarm Optimization algorithm.

2.5 Gaps in literature

Even though there has been quite an amount of research in the fields of VRPs and specifically PDPs, there are still some gaps to be found. First of all, research in PDP and related fields are mostly focused on practical implementations, which result in a wide variety of different variables, constrains and algorithms. Most focus on proposing new algorithms for specific environments, but there is less attention for comparative research. When suggesting a new method, it is generally only compared to a single other algorithm. This reduces the ability to truly compare methods and find which work best. Furthermore, the FDRPP problem itself has only been mentioned a handful of times, while the basics are comparable to other PDPs, this environments does have some unique challenges. For example urgency which is barely used in research. In addition, although there has been quite some research on stochastic environments, the stochasticity in these models do not reach nearly the same level as is found in the real-world. Lastly, most studies allow for batching, since it can significantly improve performance, but in some environments this can be a bit harder to do based on the type of rewards the environment gives. But batching is mostly just seen as something that is allowed or not, and not observed in a way of how much batching is actually done and what affects it.

3 Problem Description

3.1 Problem

The problem that this research is aimed at is the Dynamic Multi-vehicle Stochastic Food Delivery Route Planning Problem (DMS-FDRPP). This means that it focuses on food deliveries that are dynamically generated in stochastic environments with multiple delivery drivers, who in this environment use bikes as their mode of transportation.

To be able to deliver all food orders, the routes have to be generated and assigned to drivers. These orders consist of pickup and delivery locations as well as a time that the food is expected to be

prepared. Which means that the driver first has to visit the pickup location before being able to deliver it at the delivery location. This problem also allows for batching, since that is possible in real world applications, therefore each driver has a set maximum capacity of 5 orders. This is a flexible batching feature, meaning that the drivers are flexible in mixing pickups and deliveries and do not require the driver to empty their inventory before starting to pickup (a set of) new orders. Order can stay in the inventory as long as needed, but longer waiting times for customers do result in higher costs. This batching can be useful since the pickup and delivery locations are not generated in the same way. To simulate real cities, the pickup locations (restaurants) are generated in the center region of the map, to simulate a city center with the highest density of restaurants. While the delivery locations are generated over the entire map, within and outside of this center region.

At the beginning of each simulation the drivers will start at the center node, and will also try to return here if they do not have a route assigned. When a driver has a route assigned it will drive towards the next location on their route, once arrived it requires 2 minutes of service time, to load or unload the order form their bike and get ready for the next part of their trip. If a driver arrives at a pickup location before the food is ready, it will stay there waiting until the food is ready. Once a pickup or delivery is completed, the location is removed from their route and the inventory of the bike is updated. Since the orders are dynamically placed, the routes have to be re-optimized in an interval and the methods that will be compared, will each have to assign all orders to the different delivery vehicles, and plan their routes, such that the total costs will be minimized. If there are no new order to be added to the routes, the optimization will still happen to try and find more optimal routes and adapt to stochastic events that might have occurred since the last optimization. To ensure fairness, during re-optimizations the locations and the current inventories of the drivers can not be altered, which also means that the delivery locations of the orders in the inventories, have to remain in the route of the driver, but where they are placed in the route can of course be optimized.

Since real life is not deterministic, the drivers in this problem will also encounter a set of different stochastic events. First of all, traffic delays, these delays represent congestion and traffic lights etc., when a driver encounters a traffic delay it will have to slow down slightly, which causes the planned routes to take longer. In addition, there are restaurant delays, since even though restaurants can plan for a specific pickup time, sometimes they are a bit slower or faster, but the drivers will only know the exact time the orders are ready, when they are actually at the location of the restaurant. To be clear, these pickup times do not indicate a strict time the order has to be picked-up, it only indicates at what time the food is ready to be picked up. Lastly, drivers can experience a vehicle breakdown, this breakdown simulates events like a chain that has fallen off, resulting in the fact that drivers have to walk alongside their bikes (with a reduced speed) until the next location on their route, at which they will require some extra time to repair their bike after which they can continue at their regular speed. All these stochastic events happen dynamically, thus the optimization algorithms can not plan for them in advance, but they may have to change routes after the fact, by assigning future pickups of a delayed driver to other drivers.

3.2 The model

This section provides a more concrete model of the problem that is described above. The map is simulated by a grid, with each location being on a node of the grid, and the drivers travel along this grid, which means that for all distances, it will use the manhattan distance. The way the routes of drivers will be represented is by a sequence with each item being a location, either a pickup or delivery location, and the first location of the route being the current location of the driver. Keeping in mind that for each order the delivery location has to be placed later than the pickup location. The inventory will be a list of all orders that are currently in the drivers backpack, thus orders that have been picked up, but not yet delivered. The environment has some constraints, first of all, the pickup location of an order must be earlier in a route than its delivery location, unless the item is already in the inventory of the driver, then only the delivery location has to be present in the route. This also means that an order can only be assigned to a single driver, thus its locations can not be split over different drivers. Which includes the fact that if a vehicle carries an order, then the order cannot be re-assigned to another driver. As mentioned before there is also a capacity constraint set at a maximum of 5 packages at the same time.

Wang et al. [WWW+21] provided a mathematical outline for single vehicle FDRPP, this forms a great basis, which is changed to fit this multi-vehicle FDRPP. With the variables described in Table 1. The main objective function for the models is Equation 1, which tries to minimize the total costs, or fitness, which is calculated by a weighted sum of the customer waiting times and the total driving distances. The customer waiting time WT indicate the total time the customers have to wait for their food, calculated by the difference in expected delivery time ETA_i and the expected time the food will be prepared EPT_i. The driving distance DD indicates the total distance driven by all the drivers, calculated by calculating the manhattan distance, Equation 2, between each location in a driver's route. The optimal weights α and β are to be determined based on Pareto-optimal solutions. The importance of different weights is because the driving distances are greater than the customer waiting times, because a driver can travel multiple nodes per minute. Therefore if both values are set the same, the driving distance then has a bigger impact on the fitness, purely based on the fact that it has larger values, even though in reality the main objective of the route planning should be to have low customer costs, since longer waiting times, result in lower satisfaction.

Symbol	Explanation
O	set of all active orders
i	index of order
V	set of all drivers
d	index of driver
P	set of all pickup locations
D	set of all delivery locations
L	set of all locations
R_d	ordered sequence of locations for driver d
n_d	number of locations is R_d
$id_{l,d}$	index of location l in R_d of driver d
cl_d	current location of driver d
Q	max capacity per driver
u_d	number of orders currently being carried by driver d
c_d	current inventory of driver d
s_k	service time at location k
ETA_i	estimated delivery time of order i
EPT_i	estimated earliest pickup time of order i
APT_i	actual earliest pickup time of order i
TC	total costs
WT	customer waiting time
DD	driving distance
TDR	traffic delay rate
VBR	vehicle breakdown rate
RDR	restaurant delay rate
α, β	weight factors

Table 1: Overview of problem variables

$$\min TC = \alpha \sum_{i \in O} (ETA_i - EPT_i) + \beta \sum_{d \in D} \sum_{j=0}^{n_d - 1} dist(L_{R_{d,j}}, L_{R_{d,j+1}}) \tag{1}$$

$$dist(m, n) = |x_m - x_n| + |y_m - y_n| \tag{2}$$

These variables allow for the constraints to be formulated. Equations 3, 4 represent the constraints that the delivery time must be after the pickup times. This is enforced by Equation 5, which ensures that the delivery locations are placed later in the routes than the pickup locations. Equation 6 enforces the fact that all orders that are assigned to a driver have their pickup and delivery location in the route of that driver, unless the order is already in the inventory of the driver, in that case only the delivery location has to be in the route. Lastly, Equation 7 ensures that the size of the inventory of each drive does not exceed the maximum capacity.

$$ETA_i > EPT_i, \forall i \in O \tag{3}$$

$$ETA_i > APT_i, \forall i \in O \tag{4}$$

Variable	Values
Travel delay rate	0, 0.1, 0.2
Vehicle breakdown rate	0, 0.002, 0.005
Restaurant delay rate	0, 0.1, 0.3

Table 2: Overview of variables used to control stochasticity and their possible values

$$id_{P_i,d} < id_{D_i,d}, \forall i \in O, \forall d \in D \iff D_i \in R_d \land P_i \in R_d \tag{5}$$

$$\forall i \in O, \exists! d \in V \text{ such that } D_i \in R_d \land (Pi \in R_d \lor i \in c_d) \tag{6}$$

$$0 \leq u_d \leq Q, \forall d \in V \tag{7}$$

The stochasticity of the environments is controlled by a set of variables, which are shown in Table 2 with the possible values they can have. The travel delay rate values indicate the chance, for each driver independently in each timestep of the simulation, that the driver will be slightly delayed when driving. The vehicle breakdown rate indicate the chance for each driver at each timestep that a breakdown of their vehicle occurs. The restaurant delay rate indicates using Equation 8 how much the preparation is delayed or expedited.

$$PreparationChange = \max(10, \mathcal{N}(0, 15 * RDR)) \tag{8}$$

4 Methodology

4.1 Methods

The methods that will be implemented are Greedy Randomized Searh Procedure (GRASP), Simulated Annealing (SA), Genetic Algorithm (GA), Artificial Bee Colony (ABC), Tabu-Search (TS) and Adaptive Large Neighborhood Search (ALNS). This section will discuss these methods and their unique characteristics.

4.1.1 Greedy Randomized Search Procedure

The first methods that will be implemented is a greedy algorithm called GRASP, this implementation is inspired by Laguna et al. [LMMG+25]. This method works by running a set of iterations, in each iteration the entire solution is cleaned, by removing all orders, that are not picked up yet, from the routes. Then it finds all possible insertions of these unassigned orders and calculates their additional costs. When all possibilities are collected, only the insertions that fall in the acceptance range, based on the minimal and maximal extra costs, will be added to the Restricted Candidate List (RCL). From this RCL the method will randomly pick an insertion and apply it to the state. This will then be repeated until there are no unassigned orders left. This results in a new complete solution. To make it more competitive to the other algorithms, it will also include a local search heuristic, which is applied after the greedy section. This combination of creating a random greedy solution and local search will be repeated until all iterations are completed, while keeping track and updating the best solution that is found. The this method works as shown in Algorithm 1.

Algorithm 1 GRASP

```
function GRASP( )
    BestSolution ← None
    for iter in MaxIterations do
        Solution ← RandomizedGreedySolution()
        LocalSolution ← Mutation(Solution)
        BestSolution ← UpdateBest(LocalSolution, BestSolution)
    end for
    return BestSolution
end function

function RandomizedGreedySolution(State)
    Solution, U ← CleanState(State)
    while U ≠ ∅ do
        Options ← []
        for i ∈ U, d ∈ V do
            for all possible insert of P_i, D_i at p, q in R_d do
                cost ← AdditionalCosts(R_d, i, p, q)
                Options.add({order : i, driver : d, pos : (p, q), cost : cost})
            end for
        end for
        minCost ← min(Options.cost)
        maxCost ← max(Options.cost)
        threshold ← minCost + 0.1 · (maxCost − minCost)
        RCL ← {option ∈ Options|option.cost ≤ threshold}
        chosenChange ← Random(RCL)
        Solution, U ← UpdateSolution(chosenChange)
    end while
    return Solution
end function
```

4.1.2 Simulated Annealing

Simulated Annealing is one of the simpler metaheuristics that will be implemented, Algorithm 2 shows how it works. It keeps track of a single solution and applies a random local-search mutation, as described in Section 4.2.2. This mutated solution is then compared to the original solution, and is immediately accepted if it has an improved fitness. If the mutated solution is worse than the original solution there is still a change that it is accepted. This probability is calculated by Equation 9, where Δ indicates the relative decrease in fitness, and T is regulated by a cooling schedule. The cooling schedule that will be used in this research is an exponential scale. Means the temperature will be decreased by multiplying the it with a set α after each set of iterations. This means that when a set of iterations is run, the temperature will decrease and therefore the probability of accepting a worse solution also decreases. Which results in the fact that the algorithm starts with a high chance of exploration and gradually settles into exploitation of the reached optimum. This will continue until a minimum temperature is reached after which the best found solution will be executed.

$$p = \exp(\frac{-\Delta}{T}) \tag{9}$$

4.1.3 Genetic Algorithm

Genetic Algorithms aim to improve solutions by simulating a set of solutions, the implementation of this method is shown in Algorithm 3. This approach is modeled after the natural genetics and

Algorithm 2 SA

```
function SA(State, T, k, alpha)
    Solution ← State
    BestSolution ← State
    T ← T
    while T > minT do
        for iter in range(k) do
            mutatedSolution ← Mutate(Solution)
            Delta ← DeltaCosts(mutatedSolution, Solution)
            if Delta < 0 then
                Solution ← mutatedSolution
                BestSolution ← UpdateBest(LocalSolution, BestSolution)
            else if Random(0, 1) < exp(−Delta/T) then
                Solution ← mutatedSolution
            end if
        end for
        T ← T · alpha
    end while
    return BestSolution
end function
```

evolution in population. This is done by generating a random set of solutions, called a population. Within this population good performing individuals are selected to create offspring, by applying the crossover operator, as described in Section 4.2.3. These children are then mutated using local-search to increase diversity within the gene-pool. Once the size of the population of offspring has reached its limit, the parent population is replaced by the new generation, and the cycle starts over until the maximum allowed generations is reached.

Algorithm 3 GA

```
function GA(Psize, maxGen)
    Population ← InitializePopulation(Psize)
    BestSolution ← None
    for gen ← 1 to maxGen do
        newPopulation ← []
        while size(newPopulation) < Psize do
            parent1, parent2 ← Tournament(Population)
            child1, child2 ← Crossover(parent1, parent2)
            child1 ← Mutation(child1)
            BestSolution ← UpdateBest(child1, BestSolution)
            child2 ← Mutation(child2)
            BestSolution ← UpdateBest(child2, BestSolution)
            newPopulation.add(child1, child2)
        end while
        Population ← newPopulation
    end for
    return BestSolution
end function
```

4.1.4 Artificial Bee Colony

Artificial Bee Colony (ABC) optimization is based on the dynamics within bee colonies. This algorithm uses three types of bees, each with its own particular goal. The population of solutions are seen as a set of food-sources. Each iteration these food-sources are passed through these three bee types sequentially, starting with employed bees. These bees provide local-search by applying a mutation to each food-source, and replacing the original source if it has an improved fitness,

these focus on exploitation the current solution space. Secondly, the food-sources are passed to the onlooker bees. These bees randomly select food-sources weighted by their fitness, giving sources with a great fitness a higher probability to be picked, once again small mutations are applied to these food-sources, and replace the original sources if they are found to be improved. Lastly, it reaches the scout bees, during the previous phases the number of consecutive non-improving steps is tracked, and for each sources if it reaches a set limit of consecutive non-improvements, it is discarded and replaced by a new random solution to ensure exploration. During the entire process, the current best found solution is stored and updated if in any bee phase an improving source is found, the solution which is found to be the best at the end of all iterations, will be executed in the environment. The implementation of this method is shown in Algorithm 4, which is based on the work of Zhan et al. [ZSSC21].

Algorithm 4 ABC

```
function ABC(Psize, maxIterations, Limit)
    FoodSources ← InitializePopulation(Psize)
    BestSource ← None
    for iter ← 1 to maxIterations do                                  ▷ Employed Bees
        for source ∈ FoodSources do
            source, BestSource, l_source ← MutateAndUpdate(source, BestSource, l_source)
        end for
        for p ← 1 to Psize do                                         ▷ Onlooker Bees
            source ← RandomSourceByFitness(FoodSources)
            source, BestSource, l_source ← MutateAndUpdate(source, BestSource, l_source)
        end for
        for source ∈ FoodSources do                                   ▷ Scout Bees
            if l_source ≥ Limit then
                source ← RandomSolution()
                l_source ← 0
                BestSource ← UpdateBest(source, BestSource)
            end if
        end for
    end for
    return BestSolution
end function

function MutateAndUpdate(source, BestSource, l_source)
    newSource ← Mutation(source)
    if Fitness(newSource) < Fitness(source) then
        source ← newSource
        l_source ← 0
        BestSource ← UpdateBest(source, BestSource)
    else
        l_source ← l_source + 1
    end if
    return source, BestSource, l_source
end function
```

4.1.5 Tabu-Search

Tabu-Search (TS) is a powerful local-search heuristic, its implementation is shown in Algorithm 5. TS keeps track of a single solution over a number of iterations. An important distinctions between Tabu-Search and other methods is the fact that it keeps a list of the last n performed moves, the tabu-list, and prevents the method of returning to a previous solution by reversing a move in this list, this prevents cyclic behavior and gives the model more opportunities to improve. Each iteration the algorithm will generate a neighborhood of candidates based on the current

solution by applying different possible mutations. This list of candidates is then compared against the tabu-list, removing all moves that counteract the a move in this list. This results in a set of possible and allowed new solutions, the one with the best fitness is then selected to become the new solution. The move towards this new solution is then also added to the tabu-list and the oldest entry is removed to keep the list at a fixed length. During the iterations each new solution is compared to the overall best found solution, and updates this if it has an improved fitness, when all iterations are completed this best found solution will be returned and applied in the environment.

Algorithm 5 TS

```
function TS(State, maxIterations, Tsize)
    BestSolution ← None
    TabuList ← []
    Solution ← State
    for iter ← 1 to maxIterations do
        neighborhood ← GenerateNeighborhood(Solution, Tsize)
        validMoves ← []
        for candidate ∈ neighborhood do
            if candidate ∉ TabuList then
                validMoves.add(candidate)
            end if
        end for
        bestMove ← min(Fitness(validMoves))
        Solution ← ApplyMove(Solution, bestMove)
        BestSolution ← UpdateBest(Solution, BestSolution)
        TabuList ← UpdateTabuList(bestMove)
    end for
    return BestSolution
end function
```

4.1.6 Adaptive Large Neighborhood Search

The last algorithm that will be implemented is ALNS, in comparison to the other methods, this relies on large neighborhood search instead of local-search. This means that this method does not rely on small local-search mutation, but it uses larger destroy and repair mutations. These work exactly one might expect, the destroy operator, destroys a part of the solution, by removing certain orders from a set of drivers and placing these in a separate list. The repair operator will repair the solution by assigning the orders in this list to drivers. The possible destroy and repair mutations are explained in Sections 4.2.4 and 4.2.5. For each iteration the current solution is destroyed and repaired, and accepted as the new solution with the same mechanics as SA uses, where improving steps are always allowed, but bad steps are also allowed with some probability. The exact destroy and repair operators used by the algorithm are based on a set of weights, with each weight updated after each iteration based on the performance of those operators. This ensures that the algorithm learns which operators generate high results, and thus assigning higher probabilities to them. By gradually updating these weights and implementing probabilistic acceptance, the focus of the method slowly moves from exploration to exploitation. The implementation of this method is shown in Algorithm 6. The main advantages of this large neighborhood approach in comparison to a local search method, is that by allowing large changes in solutions, it can better adapt to sudden impact full changes in the environment, such as vehicle breakdowns.

Algorithm 6 ALNS

```
function ALNS(State, maxIterations)
    BestSolution ← None
    Solution ← State
    destroyWeights ← Uniform(DestroyOperators)
    repairWeights ← Uniform(RepairOperators)
    for iter ← 1 to maxIterations do
        destroy ← SelectOperator(destroyWeights)
        repair ← SelectOperator(repairWeights)
        brokenSolution ← ApplyDestoy(destroy, Solution)
        newSolution ← ApplyRepair(repair, BrokenSolution)
        if Accepting(newSolution, Solution) then
            Solution ← newSolution
            BestSolution ← UpdateBest(Solution, BestSolution)
        end if
        destroyWeights, repairWeights ← UpdateWeigths(destroyWeights, repairWeights)
    end for
    return BestSolution
end function
```

4.2 Mutation operators

Each method that will be evaluated needs to be able to affect the routes and solutions. To be able to generate, change and combine these routes and solutions there are a set of mutation operators that can be applied. This section will explain how all the different mutations work in this environment. Each natural computing algorithm works differently, therefore they do not all use the same mutations. First basic insertion is discussed, which is primarily used by GRASP, but also before running the other methods to insert the newly placed orders into the current state. Next the local-search mutations are discussed as will be used by most of the algorithms. Afterwards the crossover mutation is described which is only used by GA and lastly the destroy and repair mutations are discussed which are used by ALNS.

4.2.1 Insertion

To be able to optimize routes, the methods require all orders to be present in the solution. Therefore new orders have to be inserted into the current state or solution. This requires a pair of insertion indexes for the pickup and delivery location in a route that complies with the environment constraints. This is done by selecting a place to insert the pickup location that does not violate the capacity constraint, and from here finding a spot for the delivery location, in the same route, and after the pickup location.

4.2.2 Local-search mutations

To mutate existing solutions in the search space the algorithms can perform the following mutations. These mutations can be done to further explore solutions and trying to improve them without changing large sections. Only in re-ordering are all orders are eligible, while the other methods cannot mutate deliveries for orders that are already in the inventory of a driver.

Re-ordering: In re-ordering the algorithm will try to swap two locations within a route, if it is allowed within the constraints. This mutation can be used to find different sequences within a route

that might be beneficial by for example moving pickups closer together to reduce travel distances.

Relocation: With relocation, an order that has not been pickup yet, will be moved from one driver to another. This mutation can be beneficial when a certain driver has more orders assigned while others are idle or have less orders assigned. This way the order assignment can be redistributed.

Swapping: This method picks two orders from different drivers and swaps them. By replacing the pickup locations and delivery locations in the route, with the locations of the other order. By doing this better distributions can be found in order assignments.

2-opt*: 2-opt* is a powerful mutation that swaps the ends of two routes. It first identifies all drivers that have orders in their route that have not been picked-up yet, thus qualifying them to be swapped. From the list of possible drivers are then 2 randomly selected. For both of these drivers a random index is picked from their route, which is used as a cutoff for the tail. Next the complete orders inside this tail are identified. An order is completely in a tail if both the pickup and the delivery location are in the tail. Then all incomplete orders in the tail, are removed from the tail, so that the environment constraints will not be violated. When the two tails are created, they are swapped between the drivers.

4.2.3 Crossover Mutation

Crossover simulates genes during reproduction. In this case the genes are represented by drivers and their routes. It is provided with 2 parent solutions and generates 2 child solutions, with each having partly the genes of parent 1 and partly the genes of parent 2. First of all, the number of swapped drivers is randomly chosen, based on this amount, the actual drivers that will be switched are randomly picked. Then the crossover happens by generating two children, child 1 has all the genes of parent 1 except for the chosen drivers, which it inherits from parent 2, and child 2 has the exact opposite. Since it is possible for orders to get lost or assigned twice because of this mutation, the resulting child solutions have to be repaired to adhere to the environment constraints. This repair identifies all double assigned orders and randomly removes them from one of the routes. In addition, it identifies all unassigned orders and inserts them using greedy insertion.

4.2.4 Destroy mutations

Destroy mutations are used in ALNS to be able to explore large neighborhoods, by first applying one of the destroy mutations mentioned below, which removes a part of the solution, to later be repaired by a repair mutation described in Section 4.2.5.

Random removal: The first destroy operation, is random removal, this method randomly selects a set of orders and removes their locations from the planned routes.

Worst removal: Worst removal assesses for each order, what the impact is on the costs of the driver, by comparing the fitness of a route with and without this order. After evaluating each order,

it removes the ones with the highest additional costs.

Shaw removal: Shaw removal focuses on removing orders that are similar to each other such that they could be repaired together. This is done by randomly selecting an order, and for each remaining order, calculating their relatedness, by comparing the pickup-locations, delivery locations, and EPTs, using Equation 10. The orders with the highest relation scores are then removed from the solution.

$$relation = \alpha(dist(P_i, P_j) + dist(D_i, D_j)) + \beta \cdot |EPT_i - EPT_j| \tag{10}$$

Route removal: This operation randomly selects a driver and removes its entire route, except for the delivery locations associated with the orders already present in the driver's inventory.

4.2.5 Repair mutations

In ALNS after the destroy mutation is executed and the model has a partial solution and a list of unassigned orders, as described in Section 4.1.6. One of the repair mutations, described below, is applied to reach a new complete solution.

Greedy insertion: This method iterates over each unassigned order, for this order all possible insertions are considered and evaluated, the insertion with the lowest additional costs is then selected to insert the order.

Noise insertion: Noise insertions uses greedy insertion but adds slight noise to each possible insertion using Equation 11, in order to add slight variability. This makes it more robust, since insertions with minimally less optimal costs also get a chance to be picked.

$$DeltaCosts = \mathcal{U}(-1, 5) \tag{11}$$

Regret-2 insertion: This method first calculates the additional costs for each insertion, based on these values it selects the best and second-best insertions for each order. Then the regret value is calculated with Equation 12. The order with the highest regret is then inserted first using the best insertion, to avoid large additional costs.

$$regret = best - secondbest \tag{12}$$

Regret-3 insertion: Regret-3 insertions, works similarly to regret-2 insertion, but in addition it also takes the third-best insertion into account. Which results in Equation 13.

$$regret = (best - secondbest) + (best - thirdbest) \tag{13}$$

5 Experiment Setup

The experiments are run using an entirely custom made python script, which can be found in GitHub [vN26]. Since existing libraries like PyVRP [WLK24], do not allow for the specific variables and mechanics used in this research. The code simulates an actual environment by iterating over a

set of timesteps, updating the environment and running the optimization methods when required. Each timestep simulates a minute. At each timestep the environments is updated by checking arrivals, updating states of drivers and their inventories and routes, while also checking for newly visible orders. In addition all active drivers are moved towards their next destination. Every 5 timesteps the re-optimization is run to incorporate the new orders into the routes and react to possible delays or breakdowns that may have occurred since the last optimization. During these optimizations, for each solution will be check if it follows the environment constraints discussed in Section 3.2, this is done using Algorithm 7.

Algorithm 7 Constraint validation

```
function ConstrValidation(drivers, orders, maxCap)
    for d ∈ drivers do
        if not CapacityValidation(d, maxCap) then
            return False
        end if
    end for
    for i ∈ orders do
        if i not assigned to a driver then
            return False
        end if
        d ← GetDriverByOrder(i)
        R_d ← GetRoute(d)
        P_i, D_i ← GetLocations(i)
        if D_i ∉ R_d then
            return False
        else if i ∈ c_d ∧ P_i ∈ R_d then
            return False
        else if i ∉ c_d ∧ P_i ∉ R_d then
            return False
        else if GetIndex(P_i, R_d) > GetIndex(D_i, R_d) then
            return False
        end if
    end for
    return True
end function
```

To achieve a fair comparison of the algorithms they are all tested in a set of 11 different environments, consisting of 3 different sizes as shown in Table 3, with each 3 different stochasticity levels and a high pressure environment, except for the large environment as it requires to much compute for the scale of this study, as shown in Table 4. These combinations of variables are designed to simulate different levels of stochasticity and order pressure. Which allow to observe the adaptability to stochasticity of each method and to observe the batching behavior of these methods with different amounts of pressure. The different sizes of the environments simulate different sizes of cities, with 1 block representing 100 m, thus for example a 50 x 50 grid represents a city of 5 km by 5 km.

For each environment it runs 30 repetitions to retrieve fair generalizable data. At the start of each repetition the orders, delays and breakdowns are generated, such that each method will be evaluated on the exact same environments. Figure 1 shows examples of the locations of all orders in a repetition, with the pickup locations clearly located in the center region of the map, simulating a city center with its restaurants, while the delivery locations are spread out over the entire map. As can be seen, larger environment sizes have more drivers and orders, thus the locations are spread more densely. During the initialization of these orders, they are not only assigned their locations, but also the timestep at which the order will be placed, and thus from which it will be visible to

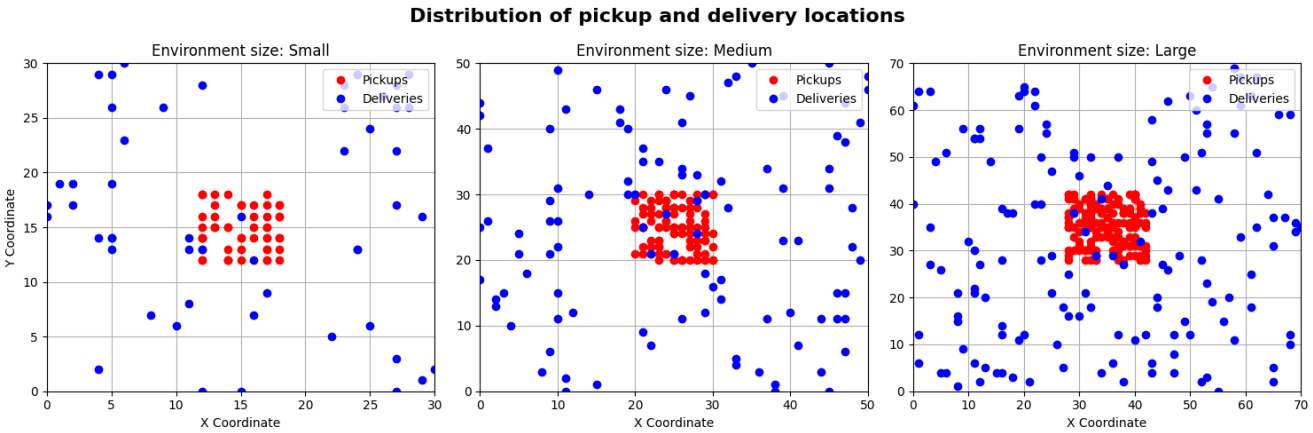

Figure 1: Sample of order location distributions of different environment sizes

the optimization algorithm, as well as the expected prepared time (EPT) of the food and the actual prepared time (APT) which is determined by Equation 14 with RDR representing the Restaurant Delay Rate, this formula adds noise to the EPT with a maximum difference of 10 minutes. When initializing the travel delays, each driver is assigned a list of timesteps based on the travel delay rate, which reduce the speed of a driver at that specific timestep by 1, thus possibly reducing the regular speed of 3 (18 km/h) to a speed of 2 (12 km/h). The initialization of breakdowns happens in the same way and is thus based on the breakdown rate. When a breakdown occurs, it simulates an event that forces the driver to continue on foot, thus reducing the speed of that driver to 1 block per timestep (6 km/h), until it has reached the next location, where it will have to wait an additional 5 minutes, which simulates a quick repair.

$$APT_i = EPT_i + \max(10, \mathcal{N}(0, 15 * RDR)) \tag{14}$$

Each simulation runs for a minimum of 240 timesteps (4 hours) representing a regular work shift during a lunch or dinner rush, during this time-frame the first hour it generates 20% if the total orders to simulate orders slightly before the rush, then in hours 2 and 3 it will generate 35% of the total orders each hour, to simulate the rush, and in the next 30 minutes it generates the last 10% of the orders to simulate the rush slowing down, afterwards, there are 30 minutes without new orders, to provide the simulation time to deliver remaining orders. If there are still active orders after the 240 timesteps, the simulation will continue until all orders are completed, in order to fairly compare the total costs of all orders.

Before the regular simulations can be run, the weights of α and β have to be determined in order to use Equation 1 as a fitness function. These values can be determined using a Pareto-front, based on the objectives of customer waiting times and driving distance. This front is a set of Pareto-optimal solutions, which are solutions that are not dominated by any other possible solution. A solution dominates another if it is better in at least one objective and not worse in the other objective. To determine the right weights, a simulation of GA will be run on a medium sized environment with a medium level of stochasticity. Each re-optimization generates a new front, therefore the fronts of the optimizations from t=120 to t=180 will be aggregated in order to get

Environment Type	Environment size	N drivers
Small	30x30	5
Regular	50x50	10
Large	70x70	15

Table 3: Characteristics of different environment sizes

a clear visualization of the fronts at the peak of the simulation. An example of the solutions and Pareto-front of an re-optimization is shown in Figure 2. For each front, the knee-point is determined, this is the point on the Pareto-front where a small improvement in one variable results in an unacceptably high increase in costs of the other variable. These points are used to calculate the slope and ratio between the variables. Resulting in the ideal weights for this re-optimization.

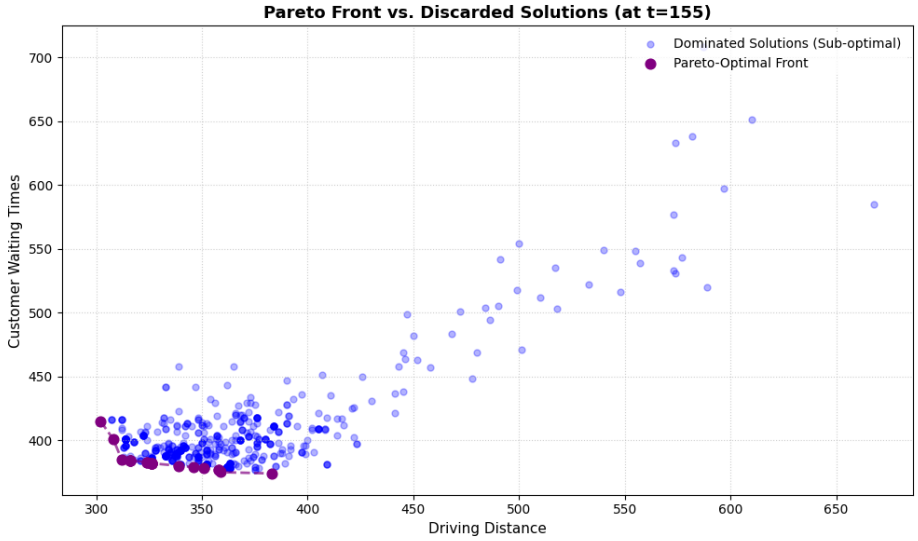

Figure 2: Comparison of evaluated solutions and Pareto-front by a Genetic Algorithm in a medium sized environment with a medium level of stochasticity at timestep 155

During the simulation, logs are collected to be able to extract data from afterwards and to compare the different algorithms for analysis. Each timestep the following data is logged: the drivers locations, routes, inventory sizes and statuses, as well as the statuses of all orders and the current fitness, driving distances and customer waiting times. In addition, at the end of each run it also logs the total runtime and the number of environment evaluations that have been run.

6 Results

In this section, the results of the experiments will be discussed, with the first section focusing determining the values of α and β using Pareto-fronts. The second section is aimed at answering RQ1, by comparing the fitness, costs, run times and evaluations of the different methods, in the

Environment Type	Travel delay	Vehicle breakdown	Rest. delay	Order to driver ratio
Regular	0	0	0	10
Medium Stochastic	0.1	0.002	0.1	10
High Stochastic	0.2	0.005	0.3	10
High Pressure	0.1	0.002	0.1	15

Table 4: Characteristics of different environment types

different environments and levels of stochasticity. The third section will focus on the batching that occurred during the simulations, as to answer RQ2. The graphs and table used to analyze the results are extracted from the logs, and their descriptive characteristics can be found in Appendices A, B, C, D, E & F.

6.1 Pareto-front

Applying a Pareto-front analysis on GA in a medium sized environments with a medium level of stochasticity, from t=120 to t=180, with the objectives Driving Distance (DD) and Waiting Time (WT), results in the data shown in Table 5. This table shows for each selected timestep the knee-point indicated by their Driving Distances and Waiting Times, as well as the size of their shifts and the determined slope or ratio. These last values are used to derive the α and β values, by using $\alpha + \beta = 1$. Which results in in the average weights $\alpha = 0.61$ and $\beta = 0.39$. While having distance sensitive states (e.g. t=160 with $\alpha = 0.14$), but also customer-sensitive states (e.g. t=130 with $\alpha = 0.90$) to prevent exponential wait-times in peak timesteps.
Customer satisfaction, and thus customer waiting times are crucial for a FDA to stay operational, while reduction of driving distances only lowers costs for the FDA slightly, but is not the main focus. This means that even in critical cases the optimization has to be able to ensure low waiting times. Therefore this study will use $\alpha = 0.90$ and $\beta = 0.10$ in all future calculations of fitness. Since these are the most customer-sensitive values encountered in the analysis and thus ensure a low customer waiting times, while still allowing for optimization of driving distances.

6.2 Route optimization analysis

6.2.1 Fitness

First of all, the different algorithms are compared by their fitness in different levels of stochasticity. Figure 3 shows the performance of all six algorithms on a small city with different levels of stochasticity. Interestingly, when there is no or a medium level of stochasticity in a small environment, all methods are able to perform similarly. Only when experiencing a high level of stochasticity GA is able to distinguish itself by achieving a slightly better performance and SA and TS fall slightly behind. As expected the models all have increasing fitness when applying a higher level of stochasticity, since the delays and breakdowns force drivers to travel slower, increasing the time it takes for the drivers to reach their destinations, which increases customer waiting times.

Timestep	Knee-point (DD,WT)	ΔDD	ΔWT	Ratio($\left\lvert\frac{\Delta WT}{\Delta DD}\right\rvert$)	Derived α	Derived β
120	$(283, 155) \rightarrow (316, 145)$	33	-10	0.30	0.77	0.23
125	$(317, 174) \rightarrow (326, 171)$	9	-3	0.33	0.75	0.25
130	$(315, 220) \rightarrow (324, 219)$	9	-1	0.11	0.90	0.10
135	$(336, 224) \rightarrow (340, 220)$	4	-4	1.00	0.50	0.50
140	$(216, 196) \rightarrow (245, 190)$	29	-6	0.21	0.83	0.17
145	$(195, 205) \rightarrow (197, 203)$	2	-2	1.00	0.50	0.50
150	$(230, 169) \rightarrow (232, 166)$	2	-3	1.50	0.40	0.60
155	$(184, 140) \rightarrow (191, 138)$	7	-2	0.29	0.78	0.22
160	$(288, 170) \rightarrow (290, 158)$	2	-12	6.00	0.14	0.86
165	$(301, 192) \rightarrow (347, 187)$	46	-5	0.11	0.90	0.10
170	$(278, 209) \rightarrow (286, 189)$	8	-20	2.5	0.29	0.71
175	$(354, 185) \rightarrow (356, 179)$	2	-6	3.00	0.25	0.75
180	$(267, 213) \rightarrow (276, 212)$	9	-1	0.11	0.90	0.10

Table 5: Results of weight calculation based on knee-points in pareto-fronts at different timesteps

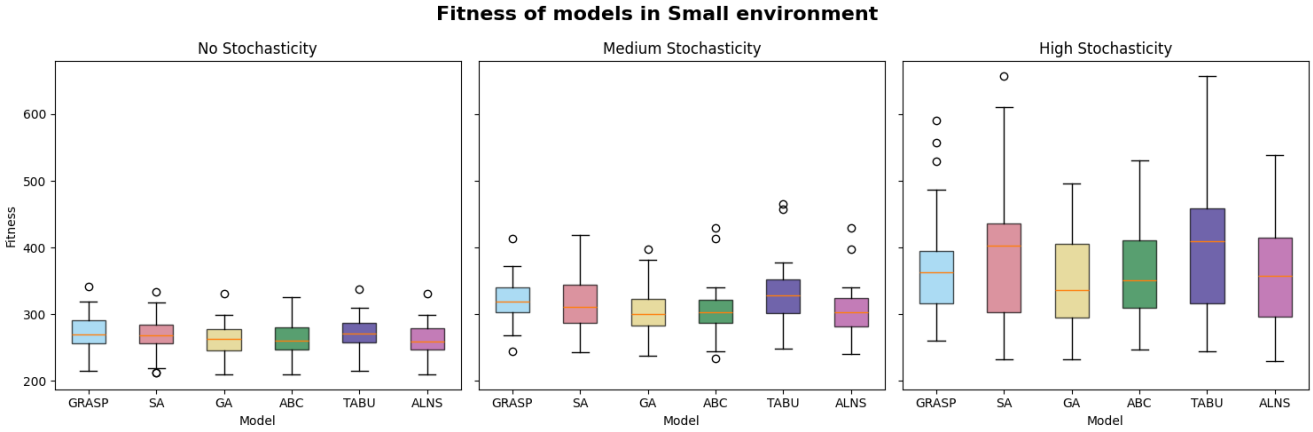

Figure 3: Comparison of multiple methods in small environments with different levels of stochasticity, displayed by their fitness results over 30 runs

Figure 4 shows the results of the simulations on the medium sized environment. Interestingly, in this environment there is a distinction between the methods, it shows that GA and ALNS are the best performing methods and GRASP the worst. When comparing the different stochasticity levels it can be seen that no matter the level of stochasticity, the models perform similarly in comparison to each other, with GA and ALNS in the lead, and GRASP slightly behind the remaining methods.

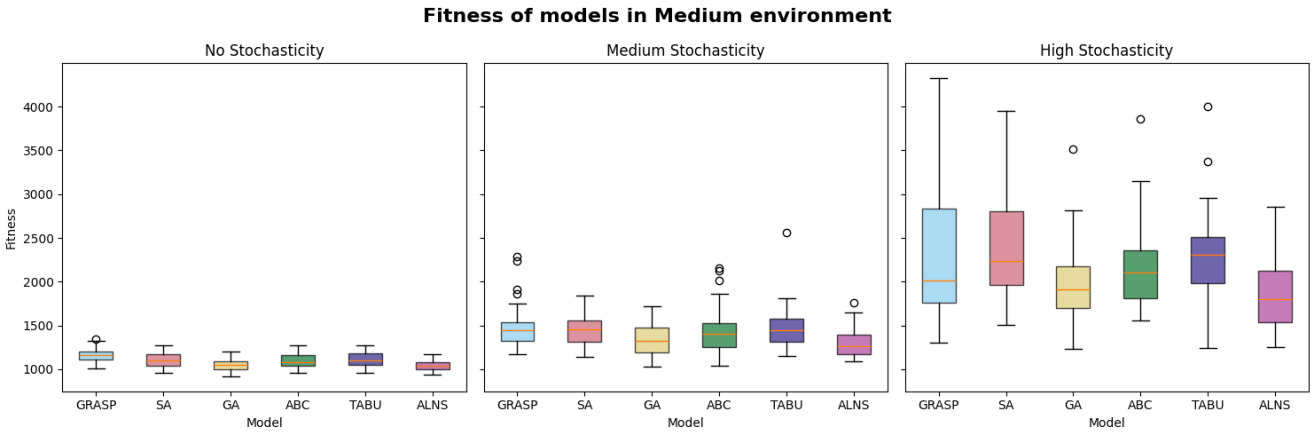

Figure 4: Comparison of multiple methods in medium environments with different levels of stochasticity, displayed by their fitness results over 30 runs

The results of the large environment as shown in Figure 5, visualize the trend that in larger environments, the performance of the different models are distinguished further, since again the same ranking is followed with GA and ALNS as top performers, behind them are SA, ABC and TS, which still perform similarly. But GRASP shows to have a harder time optimizing these large problems, as its fitness is increasingly behind in comparison to the other methods. Which is to be expected, since a larger problem size introduces more nuances and possible solutions, making it increasingly harder for GRASP to find the optimal solutions.

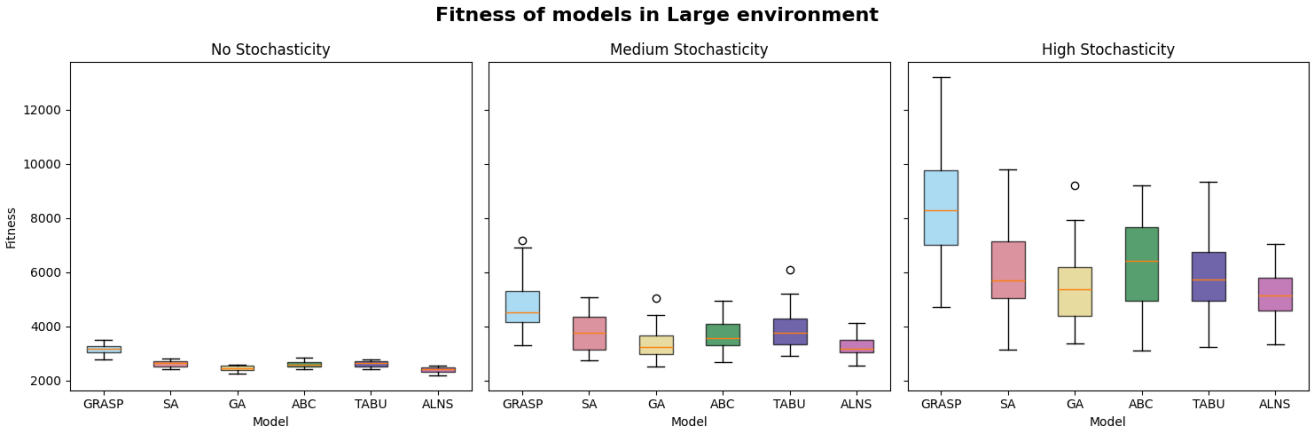

Figure 5: Comparison of multiple methods in large environments with different levels of stochasticity, displayed by their fitness results over 30 runs

To compare the ability of the methods, to handle environments with a higher order pressure. They are also evaluated on a small and medium environment with medium stochasticity, but more orders per driver. These results are shown in Figure 6. While the medium environment shows quite similar results in the ranking, but it does show a significant difference in the distance between the models, since GRASP performs significantly worse than the other methods, while GA and ALNS significantly

perform better, resulting in a better defined difference between the models. This change is to be expected, since higher pressure means that there are more orders at the same time, which increases the possible solutions and restaurant delays, causing GRASP to achieve suboptimal solutions. But even more interestingly, a high pressure in the small environment results in completely different performances then that have been observed before, since in this case GRASP outperforms SA and TS, but ABC does still manage to outperform GRASP and has fitness's closer to GA and ALNS.

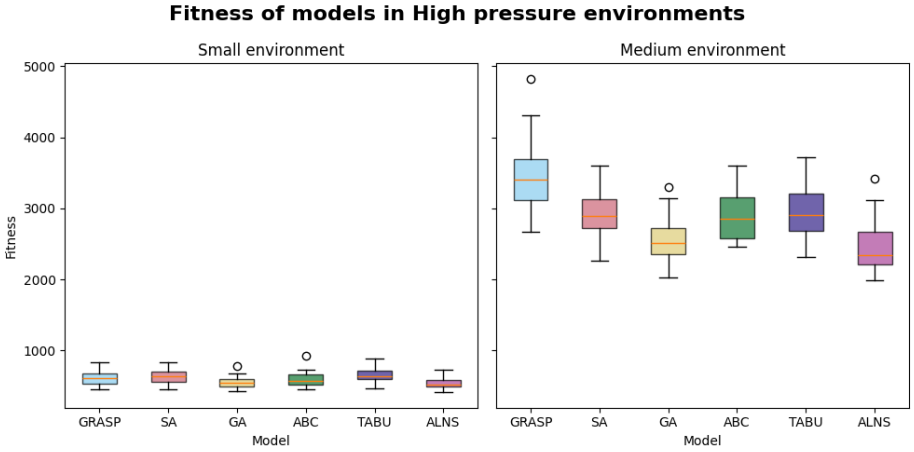

Figure 6: Comparison of multiple methods in high pressure environments with different sizes, displayed by their fitness results over 30 runs

To compare the performance of these methods and how they respond to stochasticity individually in a bit more detail, Table 6.2.1 shows the median fitness of each model in the different environments. As shown before, GA and ALNS both have a great performance, while ALNS is the best performer all medium and large environments, GA outperforms ALNS in the smaller environments. But in all environments their results lie close together. This table shows even more precisely the difference in methods, showing that in small environments there are only slight differences with TS performing the worst, but in larger environments it shows that ALNS and GA are clearly the highest performing methods, with SA, ABC and TS performing similarly, but in large high stochastic environments ABC falls behind a bit. GRASP has surprising results, in the small and medium sizes, it has competitive results compared to the other methods, but in the large environments, it performs significantly worse than all other methods. The table also shows the relative changes between the performance in stochastic environments and the performance in the deterministic environment. This clearly shows that higher stochasticity levels result in worse fitness levels. What is also interesting to see is that the relative change in fitness, caused by stochasticity increase, increases in larger environments, with the largest difference in a small environment being 51.4% by TS, while TS reaches 110.2% in medium environments and GRASP even reaches 162.2% in large environments.

Size	Small					Medium					Large				
Stoch.	No	Med.		High		No	Med.		High		No	Med.		High	
GRASP	270.0	318.5	18.0%	363.5	34.6%	1165.5	1450.0	24.4%	2018.0	**73.1%**	3160.0	4508.0	42.7%	8287.0	162.2%
SA	269.0	311.5	15.8%	402.5	49.6%	1099.5	1451.0	32.0%	2237.5	103.5%	2634.0	3773.0	43.2%	5702.5	116.5%
GA	262.5	**300.0**	**14.3%**	**336.5**	**28.2%**	1047.0	1323.0	26.4%	1914.0	82.8%	2442.5	3247.0	32.9%	5351.5	119.1%
ABC	261.0	302.5	15.9%	351.5	34.7%	1079.5	1402.0	29.9%	2109.0	95.4%	2584.0	3549.0	37.3%	6419.5	148.4%
TS	270.5	328.5	21.4%	409.5	51.4%	1099.5	1442.0	31.2%	2311.5	110.2%	2640.0	3748.0	42.0%	5712.5	116.4%
ALNS	**259.5**	302.5	16.6%	358.0	38.0%	**1038.5**	**1266.5**	**22.0%**	**1805.0**	73.8%	**2404.0**	**3178.5**	**32.2%**	**5151.0**	**114.3%**

Table 6: Median fitness values of models in different environments with the percentile increase at stochastic environments from no stochasticity, over 30 runs

These fitness values are calculated based on the customer waiting times and driving distances as described in Section 3.2 using the weights determined in Section 6.1. Encouraging customer satisfaction over driving costs. So, as to get a better understanding of the actual distribution of these costs, Table 6.2.1, gives all median values for the customer waiting times in each setup and Table 6.2.3 shows the median driving distances. These results give us interesting insights into the behavior of the algorithms. Even though the overall fitness increases with stochasticity, the actual driven distance decreases, with the largest difference being 23.0% by GA in a highly stochastic large environment. These changes also scale with environment size, with small environments reaching a decrease of max 7.5%, the medium environments max 16.5%. To compensate for this, the customer waiting times increase significantly more under stochasticity, with the small environments experiencing an increase of at least 74.8%, the medium environments 123.2% minimally, and the large environments even having 174.5% at minimum.

Size	Small					Medium					Large				
Stoch.	No	Med.		High		No	Med.		High		No	Med.		High	
GRASP	126.0	175.5	39.3%	228.0	81.0%	800.5	1129.0	41.0%	1787.0	**123.2%**	2610.5	4177.0	60.0%	8455.0	223.9%
SA	123.5	168.0	36.0%	269.5	118.2%	733.0	1138.5	55.3%	2081.0	183.9%	2038.5	3399.0	66.7%	5644.5	176.9%
GA	117.0	**154.0**	**31.6%**	**204.5**	**74.8%**	674.0	1008.5	49.6%	**1714.5**	154.4%	1855.0	2848.0	53.5%	5295.0	185.4%
ABC	**116.0**	164.0	41.4%	214.5	84.9%	708.0	1071.0	51.3%	1933.5	173.1%	1997.5	3150.0	57.7%	6464.0	223.6%
TS	123.5	191.5	55.1%	282.5	128.7%	723.0	1126.5	55.8%	2139.5	195.9%	2073.0	3394.0	63.7%	5691.0	**174.5%**
ALNS	**116.0**	158.5	36.6%	226.0	94.8%	**668.0**	**935.5**	**40.0%**	1609.5	140.9%	**1825.0**	**2772.5**	**51.9%**	**5077.5**	178.2%

Table 7: Median customer waiting times of models in different environments with the percentile increase at stochastic environments from no stochasticity, over 30 runs

Size	Small					Medium					Large				
Stoch.	No	Med.		High		No	Med.		High		No	Med.		High	
GRASP	1605.0	1573.0	-2.0%	1497.0	-6.7%	4520.5	4303.5	-4.8%	3858.0	-14.7%	8034.0	7452.5	-7.2%	6766.0	-15.8%
SA	1619.0	1562.0	**-3.5%**	1510.0	-6.7%	4509.0	4265.0	-5.4%	3776.0	-16.3%	7794.5	7059.5	-9.4%	6132.5	-21.3%
GA	**1585.0**	1552.0	-2.1%	1488.0	-6.1%	4410.0	**4162.0**	**-5.6%**	3732.0	-15.4%	7675.0	6922.5	**-9.8%**	5912.0	**-23.0%**
ABC	1598.0	1544.5	-3.3%	**1478.0**	**-7.5%**	4453.0	4236.0	-4.9%	3762.0	-15.5%	7749.0	7067.5	-8.8%	6096.0	-21.3%
TS	1621.0	1580.0	-2.5%	1509.0	-6.9%	4513.5	4263.0	**-5.6%**	3795.0	-15.9%	7794.5	7073.0	-9.3%	6122.0	-21.5%
ALNS	1594.0	**1543.0**	-3.2%	1479.5	-7.2%	**4395.0**	4202.5	-4.4%	**3672.0**	**-16.5%**	**7611.5**	**6884.0**	-9.6%	**5866.0**	-22.9%

Table 8: Median driving distances of models in different environments with the percentile increase at stochastic environments from no stochasticity, over 30 runs

When comparing the different environment sizes, it shows an interesting detail, in the deterministic small environments the average driving distance is 1603.7 and the average customer waiting time is

120.0 resulting in a waiting time to distance ratio of 0.07. While for the deterministic medium sized environment this ratio is 0.16, and in the large environment this even increases to 0.27. Showing that not only stochasticity changes the distribution of costs, but also the size itself shifts more of the costs towards the customers.

In the small environments, ABC shows to be affected the most by stochasticity by reaching a relative changes in driving distance of -7.5%. While the relative change in waiting times are average, compared to the other methods, therefore the final fitness was not outstanding. More interestingly, in the small high stochastic environment, GA shows to reach the lowest customer waiting times in addition to the lowest change in waiting times, which causes it to have the least decrease in driving distances, and overall this leads to the best fitness as shown before. When looking at the medium and large environments is is clear that ALNS has the lowest customer waiting times and driving distances in most of the environments, with only being outperformed by GA twice, but these were not significant enough to be able to outperform ALNS in the overall fitness scores, since a lower score in one type of costs resulted in a higher score in the other type of costs. In the large environments, it can also be seen that GA is able to score the best relative changes in driving distance (-9.8% and -23.0%), in both stages of stochasticity, indicating that it is great at adapting to stochasticity, but the actual values are unfortunately outperformed.

Thus, according to the fitness levels, the best performing model is ALNS closely followed by GA, with ALNS performing the best medium and large environments, while GA performs the best in small environments. GA outperforms ALNS in a few environments when comparing waiting times and driving distances, and has the best relative changes in driving distance in the large environment. While GRASP shows to be the worst in scaling, since it is able to perform well in small/medium environments, but in the large environment, it performs significantly the worst. SA, ABC and TS all perform comparably in the middle, with no significant differences, except for the fact that TS starts with the worst values in the smaller environments, but catches up in larger environments and is able to reach the best relative changes in a few occasions when comparing customer waiting times and driving distances.

6.2.2 Average run times

To compare applicability in real world scenarios, it is good to compare the run time of the algorithms, since in cases where computational speed is important, it can be preferred to choose a faster algorithm that produces slightly worse results. Therefore, Figure 7 compares the average run times for each re-optimization of the different algorithms in four different environments. As can be seen in the small environments, GRASP and SA only require around 0.05 seconds or less. GRASP evaluates all possible insertions, therefore the small size causes it to find a solution fast, for SA also holds that it can find an optimum fast because of the small sample space. GA requires a bit more time, since it uses a large population size instead of a single solution. ALNS takes slightly longer with run times around 0.2 seconds, this is a result of the fact that it tries to apply large changes to the solution, while there is only a small sample space, making it less optimal and needing more time to converge. Lastly we have ABC and TS which both have run times around 3.5 seconds, for TS it is to be expected since it generates neighborhoods each iteration instead of just applying a

single mutation, and for ABC it is also not surprising since it uses a large population like GA, but also parses these through three different stages. Comparing the different levels of stochasticity in the small environment shows not many significant differences since GRASP requires a much larger scale that makes the other methods hard to differentiate, except for the fact that GRASP and ALNS both have higher run times than the other methods, showing that they scale badly with problem size. In addition it shows that GRASP requires significantly higher run-times in highly stochastic environments, which can be caused by the fact that the stochasticity means that it takes longer to deliver order, which is supported by the results in, Table 6.2.1, therefore more orders will be active at the same time, increasing the search space for GRASP and ALNS.

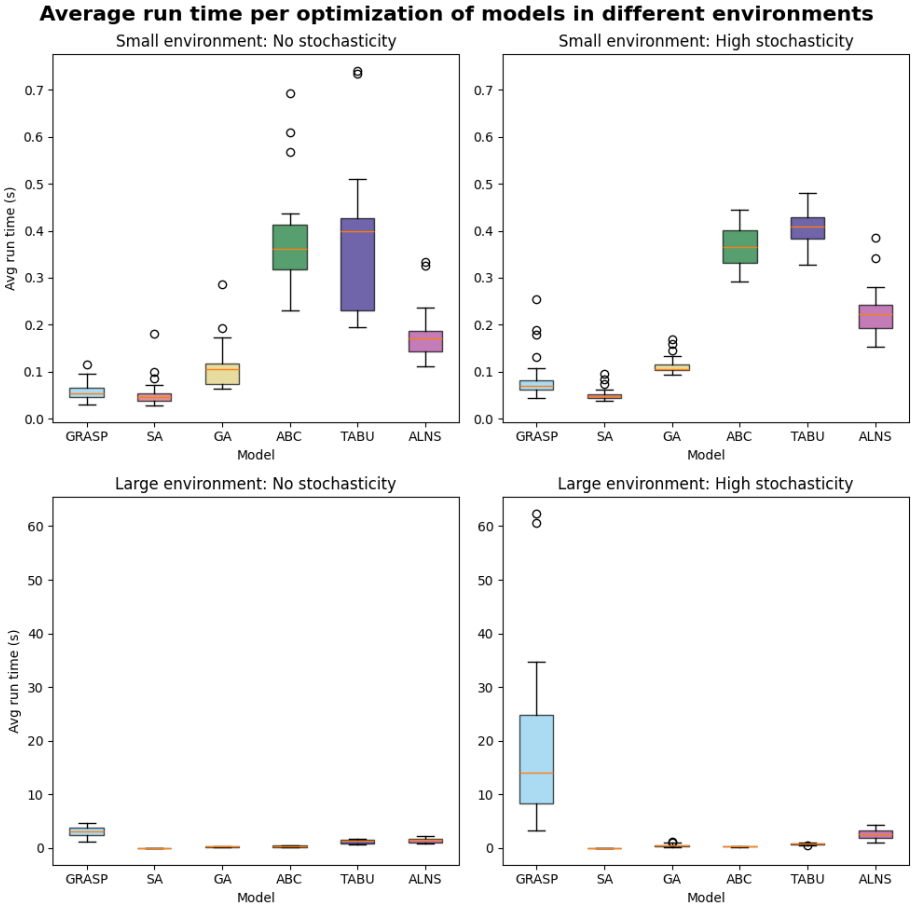

Figure 7: Comparison of multiple methods in different environments, displayed by their average run times per optimization results over 30 runs

Size	Small					Medium					Large				
Stoch.	No	Med.		High		No	Med.		High		No	Med.		High	
GRASP	0.053	0.069	28.8%	0.069	30.0%	0.459	0.490	6.6%	1.211	163.6%	3.087	5.941	92.4%	14.043	354.9%
SA	**0.045**	**0.053**	17.9%	**0.048**	5.9%	**0.048**	**0.030**	-38.9%	**0.047**	-3.1%	**0.049**	**0.040**	-17.8%	**0.038**	-23.5%
GA	0.105	0.144	37.4%	0.106	**0.8%**	0.161	0.126	-21.8%	0.211	31.4%	0.315	0.349	11.1%	0.433	37.7%
ABC	0.361	0.440	21.7%	0.366	1.3%	0.381	0.221	**-42.2%**	0.363	-4.7%	0.371	0.386	4.1%	0.276	-25.5%
TS	0.400	0.435	**8.7%**	0.409	2.2%	0.874	0.613	-29.8%	0.657	**-24.9%**	1.350	1.062	**-21.3%**	0.782	**-42.1%**
ALNS	0.170	0.203	19.1%	0.223	31.4%	0.698	0.530	-24.1%	1.108	58.7%	1.580	2.163	36.9%	2.638	67.0%

Table 9: Median average run-times of a re-optimization by models in different environments with the percentile increase at stochastic environments from no stochasticity, over 30 runs

The median average run-times are shown in Table 6.2.2, which shows a better overview of the values of each methods in the environments, especially since the large environment now do not suffer from the scale, making it possible to analyze their results. As can be seen SA has the absolute best run-times, with having the lowest values in every single environment type. In addition, all values no matter the environment are similar. Another suprising fact overall is that the differences in run-time between different levels of stochasticity seem to be random, expect for the performance of GRASP in the medium and large environments, and the performance of ALNS in the large environments. These methods show to be suffering from increasingly higher run-times in larger problem sizes, with ALNS and GRASP being the only methods to have median values higher than 2 seconds, while SA has a steady run-time of around 0.04 seconds. Suprisingly besides SA, ABC seems to be the only other method where the runtime does not increase because of larger environments.

These results show that according to computing time, in real life scenarios it it preferred to use SA, GA or ABC. Since SA is incredibly fast, and like ABC does not increase in larger environments. GA shows to have run-times lower than ABC in most environments, but it does scale with environment size. In real world applications, most environments are larger environments, which for GRASP and ALNS are immediate issues. Since TS generates a limited neighborhood, and not explores all possible solutions, it is slower than SA, GA or ABC, but it is significantly faster than ALNS or GRASP, especially in larger environments.

6.2.3 Number of evaluations

To gather the expected finesses of routes during an optimization, it simulates the route of a driver and calculates the distance it will need to drive to finish the route, and calculates the planned customer waiting time. Each time during an optimization when it simulates the route of a single driver, it is counted as an evaluation. Since these types of calculation can take a lot of compute when applied in complex environments. Therefore, Figure 8 shows a comparison of the average number of evaluations each method uses during an re-optimization in different environments. The first observation that can be made, is the fact that higher stochasticity results in an increase of evaluations for only, two of the five methods. As shown, GRASP and ALNS have a significant increase in evaluations when dealing with higher stochasticity. This is caused by the fact that higher stochasticity means, that deliveries take longer because of the delays, which results in the fact that more orders are active at the same time, which in turn results in a larger solution spaces, since there are more orders that need to be planned. GRASP evaluates all possible insertions before applying a change, which results in more evaluations in larger solution spaces caused by stochasticity. In

addition, ALNS applies mutations, some of which also evaluate all possible insertions, such as regret-repair or worst-removal. This results in the fact that both methods are highly impacted by stochasticity, but since ALNS only uses it sparingly, it shows a less significant change. SA, GA, ABC and TS on the other hand are not impacted by stochasticity.

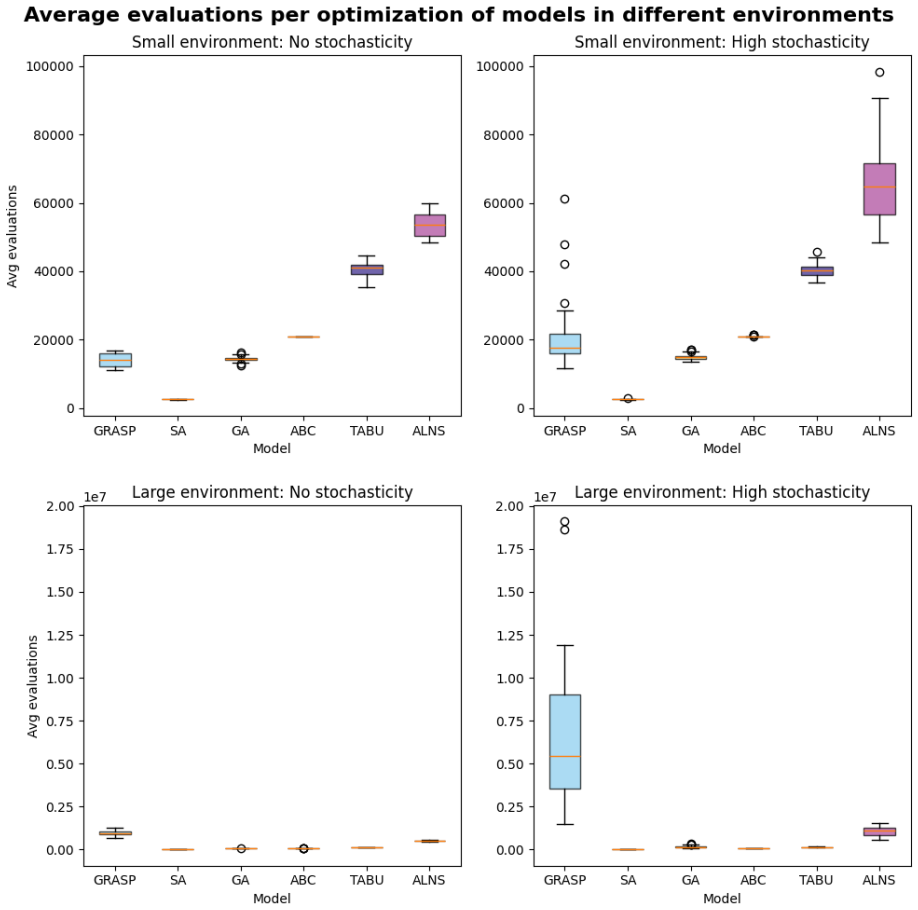

Figure 8: Comparison of multiple methods in different environments, displayed by their average number of evaluations per optimization results over 30 runs

Size	Small					Medium					Large				
Stoch.	No	Med.		High		No	Med.		High		No	Med.		High	
GRASP	14168	15518	9.5%	17742	25.2%	138341	189904	37.3%	346805	150.7%	961536	2094702	117.8%	5434621	465.2%
SA	**2589**	**2600**	0.4%	**2580**	-0.4%	**5167**	**5204**	0.7%	**5336**	3.3%	**8021**	**8204**	2.3%	**8497**	5.9%
GA	14440	14648	1.4%	14843	2.8%	34830	36986	6.2%	43188	24.0%	65626	83643	27.5%	149417	127.7%
ABC	21007	20990	-0.1%	20979	-0.1%	41321	41297	**-0.1%**	41391	**0.2%**	61630	62489	**1.4%**	62625	1.6%
TS	40954	40205	**-1.8%**	40345	**-1.5%**	84336	85206	1.0%	85614	1.5%	135358	137327	1.5%	130274	**-3.8%**
ALNS	53675	56358	5.0%	64890	20.9%	211443	247041	16.8%	346543	63.9%	496100	654991	32.0%	1075096	116.7%

Table 10: Median average number of evaluations in a re-optimization by models in different environments with the percentile increase at stochastic environments from no stochasticity, over 30 runs

When comparing the models is can be seen that SA is significantly the best model, this is caused

by the fact that it optimizes a single solution with local search mutations, which needs only a small number of evaluations. GA and ABC in turn show a higher number of evaluations, which is caused by the fact that instead of a single solution, it keeps a population, which results in the fact that it also needs significantly more evaluations to generate good performing generations. Next is TS, this method keeps track of just a single solution, just as SA, but it still requires more evaluations, this is caused by the fact that SA generates a single new solution each time, while TS generates a neighborhood by applying different mutations, this leads to the fact that all solutions in the neighborhood need to be evaluated, resulting in a high number of evaluations, even more than the population based algorithms. As said before in Section 6.2.2 ALNS, relies on mutations than require evaluating al lot of possibilities, this combined with SA results in the fact that it performs the worst of all methods in a small environment, and in larger environments it is only outdone by GRASP. For GRASP holds that in small environments, there is a small solution space, meaning that searching all possible insertions, does not require much evaluations, leading to the fact, that it actually outperforms local search methods. But when applied in a large environment, the other methods, do not have to search the whole solution space to find optima, while GRASP still has to evaluate all possibilities, resulting in an exponential increase of evaluations.

6.3 Batching analysis

To be able to compare the batching of the different models, they will be compared by the Active Batching Utilization (ABU) metric, this represents the average driver inventory size for all active drivers. To show why this is useful, Figure 9, shows this metric in comparison to total batching utilization, which calculates the overall average driver inventory size. What is apparent in this figure is that the ABU metric has higher values, this is because it does not take empty idling drivers into account which lower the batching values. But more importantly is the fact that while total batching does not show significant variations between the different methods, except for GRASP, the active metric is able to make distinctions between the models.

The ABU results of the larger environments are shown in Figure 10. First of all it shows that higher stochasticity leads to higher batching values, this is because of the fact that the delays slow down traveling, resulting in the fact that drivers spend more time in the center region where there is a higher chance of an adjacent order being present or placed. GRASP shows to have the highest batching value in each environment, this is because GRASP iterates over all possible insertions for order, and only applies the best, resulting in placing similar orders with each other and thus identifying batches more often. SA and TS prove to perform similarly and have the lowest batching scores in comparison to the other methods. GA and ALNS also perform similarly, but having slightly higher batching values, compared to to the other methods excepts for GRASP. This is mainly because ALNS has greedy mutations, that allow it to identify batches easier. ABC's performance lands between SA/TS and GA/ALNS, showing steady results, but not really leaning towards high or low batching. All models show to be impacted similarly by the different levels of stochasticity, with only increasing ABU but not varying results between each other, this is likely since the optimizations can not predict the stochasticity, therefore they assume a deterministic future and are only impacted by the stochasticity resulting in a different initial state during the next re-optimization, but this does not have a large impact on the planned batching behavior.

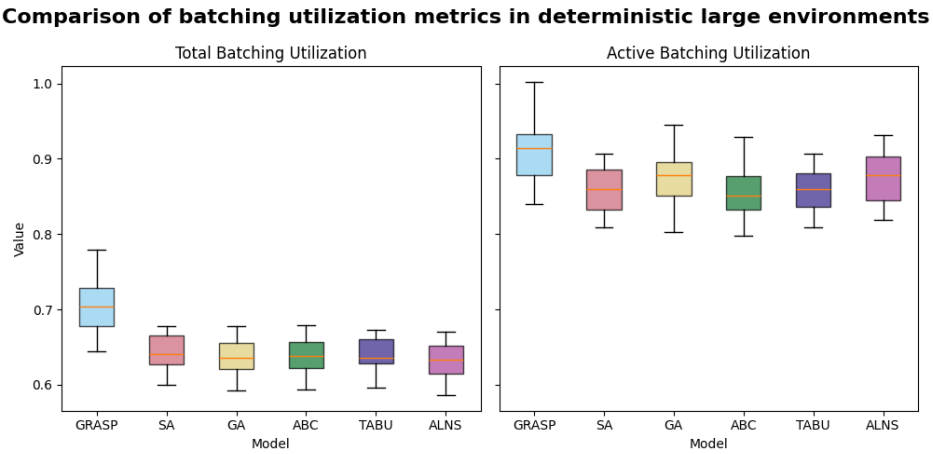

Figure 9: Comparison between active batching utilization and total batching utilization values of multiple methods in large deterministic environments, over 30 runs

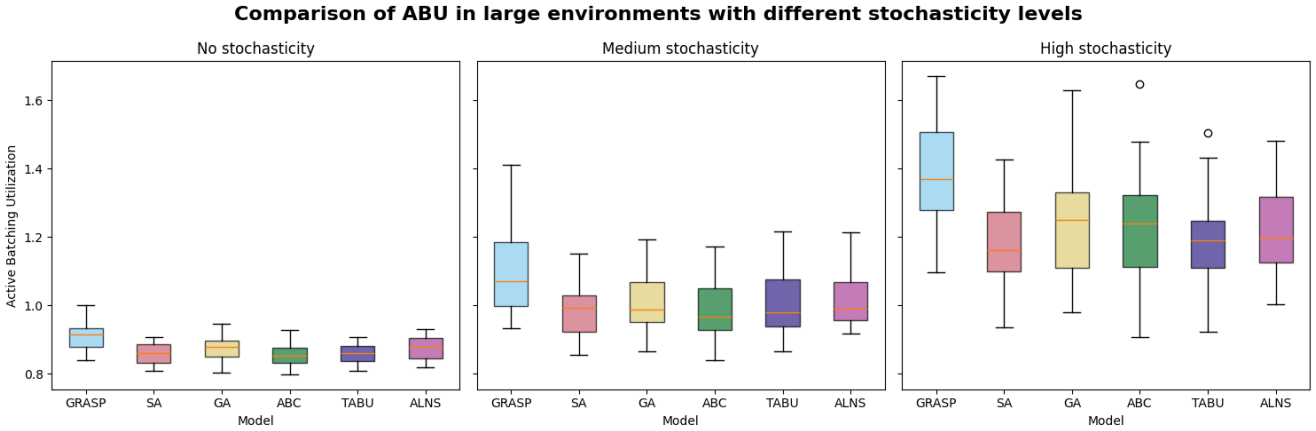

Figure 10: Comparison of multiple methods in large environments with different levels of stochasticity, displayed by their average active batching utilization, over 30 runs

Size	Small					Medium					Large				
Stoch.	No	Med.		High		No	Med.		High		No	Med.		High	
GRASP	**0.50**	0.54	7.4%	0.56	11.8%	**0.68**	0.74	8.7%	**0.90**	**32.2%**	**0.91**	**1.07**	**17.2%**	**1.37**	**50.0%**
SA	**0.50**	0.53	6.9%	0.55	9.7%	0.67	0.73	8.9%	0.86	27.6%	0.86	0.99	15.6%	1.16	35.1%
GA	**0.50**	0.54	6.6%	0.56	10.8%	**0.68**	**0.76**	12.3%	0.88	29.8%	0.88	0.99	12.5%	1.25	42.5%
ABC	**0.50**	0.54	8.7%	0.56	11.9%	0.67	0.73	10.1%	0.84	26.9%	0.85	0.97	13.5%	1.24	45.6%
TS	**0.50**	0.53	6.6%	0.55	10.8%	0.67	0.72	7.5%	0.85	27.3%	0.86	0.98	13.9%	1.19	38.5%
ALNS	**0.50**	**0.55**	**9.0%**	**0.57**	**13.0%**	**0.68**	0.75	**10.5%**	0.87	28.5%	0.88	0.99	12.8%	1.20	36.2%

Table 11: Median Active Batching Utilization (ABU) of models in different environments with the percentile increase at stochastic environments from no stochasticity, over 30 runs

Table 6.3, gives an even better overview of the ABU values of the different methods. It shows that this behavior continues in smaller environments, with only ALNS taking the lead in the small

environments. The different environments sizes also seem to have an increasing impact on the batching behavior, with larger environments having higher baselines, but also having larger relative increases. When analyzing the ABU results in Figure 10 next to the fitness results in Figure 5, we can see if these results appear to have any impact on the performance of the models. Comparing these figures gives some interesting insights, first of all, while GRASP presents the highest batching, it can not be seen in the performance of the model, since it it performs the worst there. This is likely since GRASP is iteratively adding locations to routes, thus if it finds orders similar to already existing deliveries, it insert them there, with less regard for the other locations already planned. Thus if some of the first orders seem similar they are placed together, while later orders can then inserted in between, pushing less similar orders backwards. This results in increased customer waiting times, while they could have been moved to another driver where they can de delivered earlier. For the other methods, it seems to hold, that higher batching values result in better performance, since GA and ALNS both have high batching values while also having the lowest costs. While SA, ABC and TS all show slightly less batching, and also more costs. This can be explained by a higher batching value results in lower driving distances, since it has to drive certain stretches less often, but it likely does increase customer waiting times since the driver visits other locations before delivering. When cross-referencing these results, this correlation seems to imply a relation between batching and the performance of the models, but the nature of this relation can not be proved to be causal in this experiment.

7 Conclusion

7.1 Summary of findings

An analysis of Pareto-fronts showed that in order to optimize customer waiting times (WT) and driving distances (DD), with a preference for customers, the fitness function used in the study should rely on the weights $\alpha = 0.10$ and $\beta = 0.90$, resulting in the function: $Fitness = 0.1 \cdot DD + 0.90 \cdot WT$.

A comparison of the overall fitness's reached by the different methods resulted in the fact that ALNS is the overall best performing method closely followed by GA. With GA being able to outperform ALNS in either customer waiting time or driving distance in a few environments, but this also results in higher costs for the other type, thus resulting in worse fitness results than ALNS. GRASP was identified as to be handling scaling the worst. SA, ABC and TS performed similarly with moderate values, but TS is able to reach the best relative changes resulting from stochasticity in a few environments when comparing waiting times and driven distances.

Comparing the customer waiting times and driving distances showed that higher levels of stochasticity result in shorter driving distances, (maximum decrease of 23.0%) and longer customer waiting times (at least 174.5% increase in large highly stochastic environments). Which drastically shifts the distribution of costs from driving distance to customer waiting time. In addition, it showed that larger environments by default already shift the distribution towards the customers. Resulting in a large disadvantage for customers in larger environments with higher levels of stochasticity.

Based on the run-times, it showed that SA is the absolute best, with GA and ABC following a bit behind. SA and ABC are both are not influenced by stochasticity or size, while GA does have this problem. GRASP and ALNS are highly influenced by stochasticity and environment size, needing exponentially more time. TS performs moderately.

A comparison of the number of evaluations showed that SA significantly needs the least amount of evaluations, followed by the population based algorithms GA and ABC, followed by TS since it generates entire neighborhoods. The methods that rely (partly) on greedy methodology, show to be less efficient. In a small environment, GRASP has comparable results with GA, while ALNS requires the most evaluations, even more than TS. But in large environments, GRASP takes the lead, it requires significantly more evaluations than all other methods, with stochasticity even increasing this difference. ALNS is slightly less influenced my environment size and stochasticity, but it still increases significantly more than the other methods.

The active batching utilization results, show that GRASP has the highest batching value in the environments, while the other methods all have significantly lower batching, but do all lie fairly close to each other, with stochasticity even reducing the variability between methods. But still GA and ALNS do show to have slightly higher batching values than the other methods. When comparing batching and fitness levels, it shows that except for GRASP, batching does look like to have a beneficial impact.

7.2 Answers to the Research Questions

7.2.1 RQ1: Route optimization algorithms comparison

The first research question focused on how the different natural computing algorithms compare in stochastic dynamic environments. The experiments showed that, the Genetic Algorithm (GA) overall performed the best by performing among the best methods in each comparison. When comparing the overall fitness each method reached, Adaptive Large Neighborhood Search (ALNS) also showed to be a high performing algorithm. While Greedy Randomized Adaptive Search Procedure (GRASP) performed the worst in each environment, and the other three algorithms (Simulated Annealing (SA), Artificial Bee Colony (ABC) and Tabu-Search (TS)) performed similar to each other but slightly worst than GA or ALNS. ALNS showed to large ly be the best performing algorithm when comparing customer waiting times (WT) and driving distances (DD) separately, but with GA having the best WT values in small environments and the best relative changes of DD in large environments. But all algorithms show to decrease DD with higher levels of stochasticity and thus increasing WT. A comparison of the run-times showed that SA is the absolute fastest in all environments, with GA and ABC following closely. GRASP and ALNS both perform increasingly worse in larger environments or environments with a higher stochasticity. This is also demonstrated by the results of the average number of evaluations each methods required. Overall, GRASP shows to require a lot of runtime and evaluations, but result in high costs, SA performs mediocre on fitness, but requires significantly the least amount of runtime and evaluations, GA performs great in all environments specifically under high stochasticity while also requiring a small amount of runtime and evaluations, ABC has mediocre fitness, and similar results in runtime and evaluations to GA in larger environments, Tabu-Search performs similarly to ABC with only requiring some more

evaluations, lastly ALNS performs great fitness wise especially under deterministic and medium stochastic environments but requires a large number of evaluations and runtime.

7.2.2 RQ2: Impact of stochasticity on batching

The results show that in environments with higher stochasticity, batching is increased. Likely due to slower deliveries, resulting in higher amounts of active orders, increasing the chance of finding good batches. GRASP consistently shows to have significantly the highest batching value. The other methods (SA, GA, ABC, TS and ALNS) show to have similar results, with GA and ALNS having a slightly higher value, indicating that they are slightly better at batching.

7.3 Limitations

Some limitations of this research were first of all not applying hyper-parameter optimization for each model, this can lead to suboptimal performances for the algorithms. In addition. the idea was to have a high pressure environment for each environment size, to better compare the impact of higher order pressure, but since running simulations on large high pressure environments on the same device would have taken around 60 hours, it was decided not to include it. Lastly, the number of repetitions was set at 30 runs to keep the run-times manageable, but to gather better generalized results, these runs could be increased.

7.4 Future research

Future work could apply hyper-parameter optimization to each method in order to gain the best results for each method. In addition, it is also advisable to explore the stochastic variables further by applying a systematic sensitivity analysis on the variables used to indicate stochasticity, such that the impact of each separate variable can be discovered. Lastly, future work may use real life data to simulate the environments, such as map-data to generate real life routes and locations and traffic data to accurately simulate delays.

References

[ADM13] Sohaib Afifi, Duc-Cuong Dang, and Aziz Moukrim. A simulated annealing algorithm for the vehicle routing problem with time windows and synchronization constraints. In Giuseppe Nicosia and Panos Pardalos, editors, *Learning and Intelligent Optimization*, pages 259–265, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BDJ25] Yang Bo, Milind Dawande, and Ganesh Janakiraman. Food-delivery platforms: A near-optimal policy for capacity sizing, order batching, and spatial routing. *Triennial Symposium on Transportation Analysis conferen*, 2025.

[CCW+18] Shifeng Chen, Rong Chen, Gai-Ge Wang, Jian Gao, and Arun Kumar Sangaiah. An adaptive large neighborhood search heuristic for dynamic vehicle routing problems. *Computers Electrical Engineering*, 67:596–607, 2018.

[CDIL16] Marilène Cherkesly, Guy Desaulniers, Stefan Irnich, and Gilbert Laporte. Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and multiple stacks. *European Journal of Operational Research*, 250(3):782–793, 2016.

[CDL15] Marilène Cherkesly, Guy Desaulniers, and Gilbert Laporte. A population-based metaheuristic for the pickup and delivery problem with time windows and lifo loading. *Computers Operations Research*, 62:23–35, 2015.

[CMMF17] Z. Al Chami, H. Manier, M.-A. Manier, and C. Fitouri. A hybrid genetic algorithm to solve a multi-objective pickup and delivery problem. *IFAC-PapersOnLine*, 50(1):14656–14661, 2017. 20th IFAC World Congress.

[DAG18] N. Danloup, H. Allaoui, and G. Goncalves. A comparison of two meta-heuristics for the pickup and delivery problem with transshipment. *Computers Operations Research*, 100:155–171, 2018.

[Del26] Deliverect. The state of the food delivery industry worldwide. https://www.deliverect.com/en-nl/blog/trending/the-state-of-the-food-delivery-industry-worldwide, 2026. Accessed: 2026-06-08.

[EBLV12] Güneş Erdog˘an, Maria Battarra, Gilbert Laporte, and Daniele Vigo. Metaheuristics for the traveling salesman problem with pickups, deliveries and handling costs. *Computers Operations Research*, 39(5):1074–1086, 2012.

[FB14] Francesco Ferrucci and Stefan Bock. Real-time control of express pickup and delivery processes in a dynamic environment. *Transportation Research Part B: Methodological*, 63:1–14, 2014.

[GDW16] Veaceslav Ghilas, Emrah Demir, and Tom Van Woensel. The pickup and delivery problem with time windows and scheduled lines. *INFOR: Information Systems and Operational Research*, 54(2):147–167, 2016.

[Hua15] Shan-Huen Huang. Solving the multi-compartment capacitated location routing problem with pickup–delivery routes and stochastic demands. *Computers Industrial Engineering*, 87:104–113, 2015.

[KSK15] Voratas Kachitvichyanukul, Pandhapon Sombuntham, and Siwaporn Kunnapapdeelert. Two solution representations for solving multi-depot vehicle routing problem with multiple pickup and delivery requests via pso. *Computers Industrial Engineering*, 89:125–136, 2015. Maritime logistics and transportation intelligence.

[LMMG+25] Manuel Laguna, Rafael Martí, Anna Martínez-Gavara, Sergio Pérez-Peló, and Mauricio G.C. Resende. Greedy randomized adaptive search procedures with path relinking. an analytical review of designs and implementations. *European Journal of Operational Research*, 327(3):717–734, 2025.

[Mog25] Á. Mogyorósi. Dynamic multi-objective optimization for real-time grocery delivery routing using natural computing algorithms. Bachelor thesis data science and artificial intelligence, Leiden University, 2025.

[NCC18] Salma Naccache, Jean-François Côté, and Leandro C. Coelho. The multi-pickup and delivery problem with time windows. *European Journal of Operational Research*, 269(1):353–362, 2018.

[NMAGAL16] F. Neves-Moreira, P. Amorim, L. Guimarães, and B. Almada-Lobo. A long-haul freight transportation problem: Synchronizing resources to deliver requests passing through multiple transshipment locations. *European Journal of Operational Research*, 248(2):487–506, 2016.

[OX25] Brenner Humberto Ojeda Rios and Eduardo Candido Xavier. Metaheuristic approaches for the stochastic capacitated multi-depot vehicle routing problem with pickup and delivery. *Expert Systems with Applications*, 290:128258, 2025.

[Per26] Persistence Market Research. Online food delivery services market: Global industry analysis and forecast. https://www.persistencemarketresearch.com/market-research/online-food-delivery-services-market.asp, 2026. Accessed: 2026-06-08.

[PMU22] Armando Honorio Pereira, Geraldo Robson Mateus, and Sebastián Alberto Urrutia. Valid inequalities and branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *European Journal of Operational Research*, 300(1):207–220, 2022.

[QB12] Yuan Qu and Jonathan F. Bard. A grasp with adaptive large neighborhood search for pickup and delivery problems with transshipment. *Computers Operations Research*, 39(10):2439–2456, 2012.

[TS23] Krishna Veer Tiwari and Satyendra Kumar Sharma. An optimization model for vehicle routing problem in last-mile delivery. *Expert Systems with Applications*, 222:119789, 2023.

[UGMT20] Marlin W. Ulmer, Justin C. Goodson, Dirk C. Mattfeld, and Barrett W. Thomas. On modeling stochastic dynamic vehicle routing problems. *EURO Journal on Transportation and Logistics*, 9(2):100008, 2020.

[vN26] Kay van Noordenne. Code base for bachelor thesis titled Optimization for Food Delivery under Uncertainty in DMS-FDRPP , 2026.

[WH20] Lei Wu and Mhand Hifi. Discrete scenario-based optimization for the robust vehicle routing problem: The case of time windows under delay uncertainty. *Computers Industrial Engineering*, 145:106491, 2020.

[WLK24] Niels A. Wouda, Leon Lan, and Wouter Kool. PyVRP: a high-performance VRP solver package. *INFORMS Journal on Computing*, 36(4):943–955, 2024.

[WMZS15] Chao Wang, Dong Mu, Fu Zhao, and John W. Sutherland. A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup–delivery and time windows. *Computers Industrial Engineering*, 83:111–122, 2015.

[WSG21] David Wolfinger and Juan-José Salazar-González. The pickup and delivery problem with split loads and transshipments: A branch-and-cut solution approach. *European Journal of Operational Research*, 289(2):470–484, 2021.

[WWW+21] Xing Wang, Ling Wang, Shengyao Wang, Jing fang Chen, and Chuge Wu. An xgboost-enhanced fast constructive algorithm for food delivery route planning problem. *Computers Industrial Engineering*, 152:107029, 2021.

[XW23] Xiaofeng Xu and Zhifei Wei. Dynamic pickup and delivery problem with transshipments and lifo constraints. *Computers Industrial Engineering*, 175:108835, 2023.

[YdMR19] Denise Yamashita, Bruno Jensen Virginio da Silva, Reinaldo Morabito, and Paulo César Ribas. A multi-start heuristic for the ship routing and scheduling of an oil company. *Computers Industrial Engineering*, 136:464–476, 2019.

[YLM+24] Vincent F. Yu, Ching-Hsuan Lin, Renan S. Maglasang, Shih-Wei Lin, and Kuan-Fu Chen. An efficient simulated annealing algorithm for the vehicle routing problem in omnichannel distribution. *Mathematics*, 12(23), 2024.

[ZSSC21] Xingbin Zhan, W.Y. Szeto, C.S. Shui, and Xiqun (Michael) Chen. A modified artificial bee colony algorithm for the dynamic ride-hailing sharing problem. *Transportation Research Part E: Logistics and Transportation Review*, 150:102124, 2021.

A Fitness results

Table 12: Descriptive values of fitness scores aggregated over 30 runs

Size	Stoch.	Model	Min	1st Q	Median	3rd Q	Max	Range	Mean	Std.
Small	No	GRASP	215	256.2	270.0	290.5	342	127	272.4	28.8
Small	No	SA	212	257.0	269.0	284.8	334	122	269.0	27.8
Small	No	GA	210	246.2	262.5	278.0	331	121	263.0	25.7
Small	No	ABC	210	247.2	261.0	280.0	326	116	263.6	25.9
Small	No	TABU	215	257.2	270.5	286.5	338	123	271.0	26.7
Small	No	ALNS	210	247.0	259.5	278.8	331	121	262.6	25.7
Small	Medium	GRASP	244	302.8	318.5	340.8	414	170	318.5	33.1
Small	Medium	SA	243	287.8	311.5	344.8	419	176	317.2	42.5
Small	Medium	GA	238	283.2	300.0	323.5	398	160	303.4	34.4
Small	Medium	ABC	234	287.0	302.5	321.5	430	196	306.3	40.3
Small	Medium	TABU	249	302.0	328.5	352.0	466	217	331.2	47.0
Small	Medium	ALNS	241	281.2	302.5	325.0	430	189	304.6	39.4
Small	High	GRASP	261	317.0	363.5	395.5	590	329	376.2	82.6
Small	High	SA	232	303.2	402.5	436.0	657	425	394.1	100.7
Small	High	GA	233	294.8	336.5	405.0	496	263	351.5	69.9
Small	High	ABC	247	310.2	351.5	410.2	531	284	360.9	76.2
Small	High	TABU	245	316.2	409.5	458.8	657	412	403.5	101.1
Small	High	ALNS	230	296.0	358.0	414.2	539	309	359.7	72.9
Medium	No	GRASP	1012	1108.2	1165.5	1198.5	1344	332	1158.1	77.8
Medium	No	SA	959	1044.0	1099.5	1175.0	1272	313	1104.7	77.8
Medium	No	GA	919	1002.8	1047.0	1095.2	1200	281	1047.3	62.3
Medium	No	ABC	958	1042.5	1079.5	1158.0	1272	314	1091.6	74.6
Medium	No	TABU	958	1051.8	1099.5	1177.0	1273	315	1107.4	77.2
Medium	No	ALNS	942	997.2	1038.5	1079.8	1167	225	1039.4	57.0
Medium	Medium	GRASP	1167	1321.5	1450.0	1537.0	2285	1118	1501.0	269.7
Medium	Medium	SA	1140	1309.8	1451.0	1561.0	1846	706	1460.4	193.6
Medium	Medium	GA	1032	1195.2	1323.0	1480.8	1718	686	1343.1	178.7
Medium	Medium	ABC	1044	1252.0	1402.0	1531.2	2152	1108	1437.3	282.7
Medium	Medium	TABU	1147	1310.5	1442.0	1579.8	2566	1419	1478.8	264.5
Medium	Medium	ALNS	1095	1173.5	1266.5	1398.2	1760	665	1301.7	166.3
Medium	High	GRASP	1300	1762.8	2018.0	2836.2	4324	3024	2339.3	753.9
Medium	High	SA	1502	1966.2	2237.5	2808.8	3952	2450	2379.6	583.9
Medium	High	GA	1237	1696.2	1914.0	2176.5	3518	2281	1964.0	473.6
Medium	High	ABC	1557	1814.8	2109.0	2360.0	3863	2306	2191.0	507.4
Medium	High	TABU	1238	1981.5	2311.5	2511.8	4001	2763	2317.7	546.7
Medium	High	ALNS	1255	1536.8	1805.0	2120.2	2858	1603	1856.1	391.6
Large	No	GRASP	2791	3042.2	3160.0	3259.2	3499	708	3157.8	169.6
Large	No	SA	2408	2512.2	2634.0	2721.0	2818	410	2617.7	118.1
Large	No	GA	2255	2374.0	2442.5	2536.8	2581	326	2439.6	97.9
Large	No	ABC	2424	2497.2	2584.0	2677.5	2837	413	2596.6	107.2
Large	No	TABU	2407	2506.2	2640.0	2705.5	2780	373	2619.0	114.1
Large	No	ALNS	2188	2330.2	2404.0	2494.8	2550	362	2406.6	97.9

Continued on next page

Table 12 – Continued from previous page

Size	Stoch.	Model	Min	1st Q	Median	3rd Q	Max	Range	Mean	Std.
Large	Medium	GRASP	3309	4142.5	4508.0	5304.2	7167	3858	4849.4	982.0
Large	Medium	SA	2750	3128.0	3773.0	4348.0	5058	2308	3781.3	673.1
Large	Medium	GA	2524	2966.2	3247.0	3651.5	5040	2516	3362.9	542.0
Large	Medium	ABC	2667	3312.0	3549.0	4103.0	4942	2275	3724.1	565.3
Large	Medium	TABU	2904	3328.5	3748.0	4270.8	6088	3184	3890.7	720.7
Large	Medium	ALNS	2549	3049.8	3178.5	3488.2	4135	1586	3281.7	372.6
Large	High	GRASP	4721	7006.8	8287.0	9766.5	13197	8476	8353.5	1905.4
Large	High	SA	3143	5036.2	5702.5	7120.0	9804	6661	6021.9	1461.3
Large	High	GA	3360	4382.2	5351.5	6188.2	9201	5841	5547.4	1443.8
Large	High	ABC	3118	4940.2	6419.5	7645.5	9217	6099	6378.9	1614.6
Large	High	TABU	3221	4944.8	5712.5	6728.8	9321	6100	6021.6	1469.7
Large	High	ALNS	3348	4572.0	5151.0	5776.0	7041	3693	5138.2	878.2

B Customer Waiting times

Table 13: Descriptive values of customer waiting times aggregated over 30 runs

Size	Stoch.	Model	Min	1st Q	Median	3rd Q	Max	Range	Mean	Std.
Small	No	GRASP	68	105.5	126.0	139.8	180	112	124.3	27.5
Small	No	SA	67	105.2	123.5	134.5	176	109	119.6	26.0
Small	No	GA	67	102.2	117.0	129.8	171	104	115.1	23.6
Small	No	ABC	67	99.8	116.0	130.5	168	101	115.4	24.0
Small	No	TABU	69	106.5	123.5	135.0	178	109	121.2	25.0
Small	No	ALNS	67	103.0	116.0	128.2	171	104	114.7	23.4
Small	Medium	GRASP	110	158.2	175.5	201.8	292	182	179.1	37.1
Small	Medium	SA	106	150.2	168.0	202.2	295	189	178.2	46.5
Small	Medium	GA	108	138.0	154.0	187.2	274	166	164.9	38.5
Small	Medium	ABC	103	143.2	164.0	186.8	313	210	168.8	45.8
Small	Medium	TABU	105	163.2	191.5	210.5	340	235	192.0	50.3
Small	Medium	ALNS	103	136.5	158.5	187.8	313	210	166.6	44.3
Small	High	GRASP	112	189.2	228.0	266.0	507	395	251.0	94.0
Small	High	SA	109	169.2	269.5	315.0	571	462	269.9	112.9
Small	High	GA	117	166.2	204.5	283.0	394	277	225.0	76.7
Small	High	ABC	120	178.8	214.5	295.0	428	308	235.5	85.0
Small	High	TABU	104	183.2	282.5	338.5	572	468	279.4	113.6
Small	High	ALNS	112	167.0	226.0	292.5	450	338	235.1	81.1
Medium	No	GRASP	644	731.0	800.5	827.0	965	321	784.4	81.3
Medium	No	SA	591	654.2	733.0	791.0	889	298	728.0	80.6
Medium	No	GA	562	626.0	674.0	723.0	817	255	676.3	61.4
Medium	No	ABC	595	655.5	708.0	781.8	891	296	717.3	75.6
Medium	No	TABU	591	657.0	723.0	804.0	890	299	731.0	80.9
Medium	No	ALNS	583	626.5	668.0	709.5	781	198	669.7	55.8
Medium	Medium	GRASP	814	980.2	1129.0	1239.5	2110	1296	1194.8	316.2
Medium	Medium	SA	800	967.5	1138.5	1255.2	1607	807	1152.5	224.8

Continued on next page

Table 13 – Continued from previous page

Size	Stoch.	Model	Min	1st Q	Median	3rd Q	Max	Range	Mean	Std.
Medium	Medium	GA	681	843.0	1008.5	1184.0	1500	819	1032.1	214.8
Medium	Medium	ABC	700	924.8	1071.0	1256.2	1981	1281	1129.8	328.1
Medium	Medium	TABU	807	979.8	1126.5	1291.0	2433	1626	1172.9	305.4
Medium	Medium	ALNS	734	834.0	935.5	1091.0	1519	785	984.5	194.0
Medium	High	GRASP	982	1521.5	1787.0	2748.5	4423	3441	2172.6	860.6
Medium	High	SA	1179	1747.5	2081.0	2720.0	3999	2820	2229.9	671.4
Medium	High	GA	897	1457.0	1714.5	2025.0	3575	2678	1776.0	554.5
Medium	High	ABC	1265	1567.8	1933.5	2210.2	3923	2658	2018.1	583.9
Medium	High	TABU	875	1753.8	2139.5	2378.2	4091	3216	2160.0	635.7
Medium	High	ALNS	966	1290.2	1609.5	1951.5	2826	1860	1655.6	460.4
Large	No	GRASP	2212	2487.2	2610.5	2737.5	2968	756	2615.5	182.1
Large	No	SA	1830	1917.2	2038.5	2164.5	2248	418	2043.7	127.8
Large	No	GA	1675	1780.8	1855.0	1961.5	2006	331	1861.3	105.1
Large	No	ABC	1824	1919.2	1997.5	2126.0	2267	443	2021.4	116.1
Large	No	TABU	1826	1916.5	2073.0	2134.8	2221	395	2045.4	123.5
Large	No	ALNS	1604	1742.0	1825.0	1926.8	1992	388	1827.8	106.3
Large	Medium	GRASP	2808	3756.8	4177.0	5118.0	7215	4407	4557.6	1118.1
Large	Medium	SA	2213	2611.8	3399.0	4109.8	4876	2663	3415.8	788.8
Large	Medium	GA	1973	2496.0	2848.0	3310.2	4898	2925	2968.3	639.8
Large	Medium	ABC	2112	2892.2	3150.0	3801.8	4796	2684	3356.7	659.2
Large	Medium	TABU	2374	2880.5	3394.0	3993.5	6079	3705	3545.1	844.3
Large	Medium	ALNS	2053	2587.0	2772.5	3118.0	3866	1813	2880.2	443.2
Large	High	GRASP	4536	6984.5	8455.0	10128.2	13903	9367	8531.0	2127.1
Large	High	SA	2676	4912.0	5644.5	7282.5	10317	7641	6009.2	1662.6
Large	High	GA	2946	4182.8	5295.0	6279.5	9692	6746	5515.0	1653.3
Large	High	ABC	2623	4811.8	6464.0	7853.0	9629	7006	6412.6	1835.8
Large	High	TABU	2746	4803.8	5691.0	6808.8	9740	6994	6006.3	1671.6
Large	High	ALNS	2954	4400.0	5077.5	5763.5	7228	4274	5052.5	1012.7

C Driving Distances

Table 14: Descriptive values of driving distances aggregated over 30 runs

Size	Stoch.	Model	Min	1st Q	Median	3rd Q	Max	Range	Mean	Std.
Small	No	GRASP	1490	1554.0	1605.0	1635.0	1796	306	1604.1	70.8
Small	No	SA	1490	1564.5	1619.0	1653.0	1770	280	1612.5	73.5
Small	No	GA	1478	1540.5	1585.0	1638.5	1768	290	1593.6	74.7
Small	No	ABC	1458	1554.0	1598.0	1633.0	1754	296	1597.7	71.8
Small	No	TABU	1506	1579.0	1621.0	1645.0	1778	272	1619.2	68.2
Small	No	ALNS	1462	1540.5	1594.0	1631.0	1768	306	1593.3	74.1
Small	Medium	GRASP	1446	1520.5	1573.0	1635.0	1670	224	1571.4	64.7
Small	Medium	SA	1414	1533.5	1562.0	1616.5	1701	287	1568.2	67.1
Small	Medium	GA	1412	1511.0	1552.0	1602.0	1640	228	1549.9	62.0
Small	Medium	ABC	1414	1499.0	1544.5	1597.5	1638	224	1544.1	61.2

Continued on next page

Table 14 – Continued from previous page

Size	Stoch.	Model	Min	1st Q	Median	3rd Q	Max	Range	Mean	Std.
Small	Medium	TABU	1456	1530.5	1580.0	1623.8	1732	276	1582.8	69.3
Small	Medium	ALNS	1408	1511.0	1543.0	1601.0	1638	230	1546.4	60.7
Small	High	GRASP	1302	1449.5	1497.0	1587.8	1704	402	1503.9	100.4
Small	High	SA	1316	1460.0	1510.0	1568.0	1686	370	1512.2	96.0
Small	High	GA	1280	1429.5	1488.0	1558.5	1634	354	1490.0	90.2
Small	High	ABC	1289	1428.8	1478.0	1568.0	1656	367	1489.0	91.7
Small	High	TABU	1310	1443.8	1509.0	1591.0	1726	416	1520.5	102.4
Small	High	ALNS	1294	1427.8	1479.5	1565.5	1646	352	1481.1	96.3
Medium	No	GRASP	4288	4476.0	4520.5	4581.0	4769	481	4521.1	114.9
Medium	No	SA	4256	4422.8	4509.0	4569.2	4735	479	4494.6	122.9
Medium	No	GA	4128	4323.5	4410.0	4442.2	4650	522	4386.3	118.3
Medium	No	ABC	4224	4393.2	4453.0	4540.8	4706	482	4459.0	119.0
Medium	No	TABU	4256	4404.8	4513.5	4577.5	4741	485	4495.3	126.5
Medium	No	ALNS	4140	4290.2	4395.0	4438.8	4640	500	4367.4	118.8
Medium	Medium	GRASP	3810	4098.2	4303.5	4373.8	4554	744	4256.3	194.7
Medium	Medium	SA	3930	4070.0	4265.0	4377.8	4521	591	4231.3	166.3
Medium	Medium	GA	3680	3990.8	4162.0	4259.0	4587	907	4141.8	213.9
Medium	Medium	ABC	3689	4127.0	4236.0	4330.2	4625	936	4204.8	196.0
Medium	Medium	TABU	3763	4208.8	4263.0	4304.5	4483	720	4231.4	145.5
Medium	Medium	ALNS	3838	4009.8	4202.5	4274.0	4430	592	4156.7	165.1
Medium	High	GRASP	3207	3661.0	3858.0	4034.5	4364	1157	3840.6	259.1
Medium	High	SA	3127	3558.8	3776.0	3924.2	4410	1283	3727.0	277.6
Medium	High	GA	3003	3488.8	3732.0	3849.5	4299	1296	3655.7	284.3
Medium	High	ABC	3327	3565.5	3762.0	3926.8	4219	892	3748.5	250.9
Medium	High	TABU	3064	3590.0	3795.0	3927.5	4509	1445	3736.7	302.7
Medium	High	ALNS	3119	3570.0	3672.0	3885.2	4064	945	3659.7	263.8
Large	No	GRASP	7737	7958.2	8034.0	8113.2	8356	619	8038.6	150.9
Large	No	SA	7500	7661.8	7794.5	7888.5	8094	594	7783.1	147.7
Large	No	GA	7415	7520.0	7675.0	7748.8	7883	468	7644.9	134.2
Large	No	ABC	7522	7674.0	7749.0	7891.0	8033	511	7773.6	134.2
Large	No	TABU	7502	7648.2	7794.5	7879.2	8113	611	7781.6	157.2
Large	No	ALNS	7340	7543.2	7611.5	7720.8	7872	532	7616.0	133.2
Large	Medium	GRASP	6580	7250.8	7452.5	7797.2	8172	1592	7475.1	395.5
Large	Medium	SA	6218	6758.0	7059.5	7399.2	7989	1771	7070.6	462.4
Large	Medium	GA	6221	6634.2	6922.5	7231.8	7729	1508	6914.8	416.0
Large	Medium	ABC	6260	6766.0	7067.5	7190.8	7665	1405	7030.3	360.5
Large	Medium	TABU	6168	6637.2	7073.0	7353.5	7884	1716	7001.4	475.5
Large	Medium	ALNS	6048	6700.2	6884.0	7136.2	7490	1442	6895.3	334.5
Large	High	GRASP	6161	6566.0	6766.0	6952.8	7407	1246	6755.5	301.9
Large	High	SA	5188	5944.0	6132.5	6372.5	7344	2156	6135.4	458.5
Large	High	GA	4781	5531.0	5912.0	6168.8	7082	2301	5840.0	508.4
Large	High	ABC	5193	5691.8	6096.0	6274.2	7572	2379	6075.6	486.9
Large	High	TABU	5334	5863.5	6122.0	6446.5	7496	2162	6159.8	460.1
Large	High	ALNS	5025	5573.5	5866.0	6149.2	6899	1874	5909.4	429.9

D Runtimes

Table 15: Descriptive values of average runtimes per optimization aggregated over 30 runs

Size	Stoch.	Model	Min	1st Q	Median	3rd Q	Max	Range	Mean	Std.
Small	No	GRASP	0.030	0.045	0.053	0.066	0.115	0.085	0.059	0.020
Small	No	SA	0.028	0.038	0.045	0.054	0.181	0.153	0.052	0.029
Small	No	GA	0.063	0.073	0.105	0.117	0.287	0.224	0.111	0.047
Small	No	ABC	0.231	0.317	0.361	0.414	0.693	0.462	0.370	0.105
Small	No	TABU	0.194	0.231	0.400	0.428	0.740	0.546	0.371	0.137
Small	No	ALNS	0.112	0.143	0.170	0.186	0.333	0.221	0.175	0.050
Small	Medium	GRASP	0.024	0.064	0.069	0.078	0.111	0.086	0.070	0.021
Small	Medium	SA	0.020	0.048	0.053	0.068	0.107	0.087	0.057	0.020
Small	Medium	GA	0.044	0.112	0.144	0.157	0.178	0.134	0.130	0.035
Small	Medium	ABC	0.165	0.379	0.440	0.481	0.576	0.411	0.422	0.100
Small	Medium	TABU	0.194	0.405	0.435	0.454	0.500	0.305	0.413	0.075
Small	Medium	ALNS	0.083	0.185	0.203	0.231	0.287	0.204	0.203	0.046
Small	High	GRASP	0.044	0.061	0.069	0.082	0.253	0.209	0.086	0.045
Small	High	SA	0.037	0.044	0.048	0.053	0.096	0.059	0.051	0.013
Small	High	GA	0.093	0.103	0.106	0.116	0.168	0.075	0.113	0.017
Small	High	ABC	0.291	0.332	0.366	0.401	0.445	0.154	0.366	0.044
Small	High	TABU	0.328	0.382	0.409	0.429	0.481	0.154	0.405	0.036
Small	High	ALNS	0.153	0.192	0.223	0.243	0.385	0.232	0.224	0.048
Medium	No	GRASP	0.203	0.258	0.459	0.608	0.822	0.620	0.462	0.183
Medium	No	SA	0.021	0.024	0.048	0.058	0.074	0.053	0.044	0.018
Medium	No	GA	0.077	0.088	0.161	0.197	0.244	0.167	0.155	0.055
Medium	No	ABC	0.171	0.207	0.381	0.468	0.596	0.425	0.367	0.139
Medium	No	TABU	0.344	0.387	0.874	0.921	1.114	0.771	0.737	0.254
Medium	No	ALNS	0.342	0.381	0.698	0.841	1.327	0.985	0.670	0.249
Medium	Medium	GRASP	0.248	0.354	0.490	0.776	2.095	1.847	0.606	0.393
Medium	Medium	SA	0.017	0.024	0.030	0.051	0.086	0.069	0.038	0.017
Medium	Medium	GA	0.077	0.087	0.126	0.192	0.256	0.179	0.141	0.056
Medium	Medium	ABC	0.156	0.184	0.221	0.415	0.577	0.421	0.303	0.130
Medium	Medium	TABU	0.259	0.353	0.613	1.024	1.869	1.611	0.709	0.394
Medium	Medium	ALNS	0.367	0.448	0.530	0.896	1.396	1.029	0.681	0.282
Medium	High	GRASP	0.304	0.840	1.211	2.111	6.565	6.261	1.696	1.343
Medium	High	SA	0.018	0.027	0.047	0.056	0.069	0.051	0.043	0.016
Medium	High	GA	0.078	0.133	0.211	0.261	0.385	0.308	0.209	0.087
Medium	High	ABC	0.153	0.193	0.363	0.449	0.608	0.455	0.343	0.133
Medium	High	TABU	0.337	0.508	0.657	1.064	1.316	0.979	0.755	0.304
Medium	High	ALNS	0.422	0.724	1.108	1.442	2.708	2.286	1.180	0.548
Large	No	GRASP	1.282	2.396	3.087	3.771	4.602	3.319	3.040	0.827
Large	No	SA	0.022	0.029	0.049	0.056	0.072	0.049	0.046	0.015
Large	No	GA	0.164	0.180	0.315	0.338	0.368	0.204	0.284	0.076
Large	No	ABC	0.171	0.234	0.371	0.436	0.546	0.375	0.348	0.108
Large	No	TABU	0.618	0.777	1.350	1.626	1.650	1.032	1.235	0.396
Large	No	ALNS	0.882	1.091	1.580	1.794	2.222	1.340	1.501	0.382

Continued on next page

Table 15 – Continued from previous page

Size	Stoch.	Model	Min	1st Q	Median	3rd Q	Max	Range	Mean	Std.
Large	Medium	GRASP	2.036	4.242	5.941	11.969	40.181	38.146	9.805	9.241
Large	Medium	SA	0.023	0.033	0.040	0.068	0.124	0.101	0.053	0.026
Large	Medium	GA	0.172	0.233	0.349	0.462	1.101	0.929	0.397	0.228
Large	Medium	ABC	0.191	0.277	0.386	0.540	0.912	0.721	0.416	0.182
Large	Medium	TABU	0.491	0.748	1.062	1.744	2.889	2.397	1.221	0.594
Large	Medium	ALNS	0.905	1.357	2.163	3.100	6.741	5.836	2.521	1.545
Large	High	GRASP	3.215	8.317	14.043	24.883	62.298	59.083	19.080	14.450
Large	High	SA	0.029	0.034	0.038	0.042	0.053	0.024	0.038	0.006
Large	High	GA	0.203	0.320	0.433	0.572	1.237	1.034	0.483	0.250
Large	High	ABC	0.210	0.250	0.276	0.298	0.368	0.158	0.280	0.036
Large	High	TABU	0.506	0.729	0.782	0.870	1.018	0.512	0.780	0.124
Large	High	ALNS	1.080	1.928	2.638	3.217	4.340	3.260	2.647	0.832

E Evaluations

Table 16: Descriptive values of average number of evaluations per optimization aggregated over 30 runs

Size	Stoch.	Model	Min	1st Q	Median	3rd Q	Max	Range	Mean	Std.
Small	No	GRASP	11072	12280	14168	15890	16883	5811	14127	2000
Small	No	SA	2516	2558	2589	2634	2709	194	2594	49
Small	No	GA	12414	14055	14440	14737	16296	3882	14414	806
Small	No	ABC	20964	20988	21007	21016	21031	68	21004	17
Small	No	TABU	35486	39152	40954	41801	44534	9047	40433	2225
Small	No	ALNS	48531	50205	53675	56688	59920	11388	53893	3518
Small	Medium	GRASP	12057	14337	15518	16778	25815	13759	16123	3015
Small	Medium	SA	2515	2565	2600	2638	2732	217	2607	55
Small	Medium	GA	12726	14119	14648	14959	17316	4591	14565	880
Small	Medium	ABC	20925	20981	20990	21012	21189	264	20997	42
Small	Medium	TABU	34104	38400	40205	42076	45465	11361	40062	2479
Small	Medium	ALNS	49056	54019	56358	60861	72321	23266	58189	6383
Small	High	GRASP	11594	15987	17742	21626	61306	49712	21668	10691
Small	High	SA	2517	2565	2580	2625	2833	316	2599	64
Small	High	GA	13635	14418	14843	15253	16991	3356	14971	916
Small	High	ABC	20869	20956	20979	20997	21369	500	21006	119
Small	High	TABU	36843	38862	40345	41355	45820	8976	40431	2157
Small	High	ALNS	48334	56712	64890	71508	98304	49970	65756	10985
Medium	No	GRASP	105050	124031	138341	146203	186520	81470	137343	19240
Medium	No	SA	5102	5142	5167	5221	5365	263	5186	60
Medium	No	GA	32440	34004	34830	36290	38149	5709	35045	1491
Medium	No	ABC	41268	41295	41321	41350	41374	106	41322	30
Medium	No	TABU	79311	82546	84336	86080	90044	10733	84466	2573
Medium	No	ALNS	186855	205214	211443	216222	248829	61974	211562	12787
Medium	Medium	GRASP	126937	143707	189904	231198	478023	351087	207363	90908
Medium	Medium	SA	5141	5191	5204	5265	5340	199	5224	58

Continued on next page

Table 16 – Continued from previous page

Size	Stoch.	Model	Min	1st Q	Median	3rd Q	Max	Range	Mean	Std.
Medium	Medium	GA	31271	35045	36986	38402	48907	17636	37467	3963
Medium	Medium	ABC	41171	41229	41297	41323	42139	968	41334	202
Medium	Medium	TABU	73614	83403	85206	87349	93262	19648	84754	3863
Medium	Medium	ALNS	202604	228182	247041	264752	323727	121122	250421	30380
Medium	High	GRASP	167337	249156	346805	656168	1461684	1294347	508459	340244
Medium	High	SA	5194	5261	5336	5412	5608	414	5349	106
Medium	High	GA	32656	40533	43188	49547	73180	40524	47099	10657
Medium	High	ABC	41094	41220	41391	41620	41971	877	41437	240
Medium	High	TABU	76276	84131	85614	86892	91746	15470	85256	3672
Medium	High	ALNS	230663	312283	346543	394434	623774	393110	362901	87832
Large	No	GRASP	669639	864191	961536	1022454	1250030	580391	951146	127409
Large	No	SA	7922	7962	8021	8087	8158	235	8029	72
Large	No	GA	61715	63880	65626	66584	71975	10260	65718	2196
Large	No	ABC	61502	61603	61630	61925	62739	1237	61824	379
Large	No	TABU	129945	133418	135358	137212	139311	9366	135222	2531
Large	No	ALNS	447962	477772	496100	528347	572114	124152	504961	36256
Large	Medium	GRASP	874599	1234892	2094702	2916988	7083198	6208598	2463771	1550642
Large	Medium	SA	7990	8080	8204	8377	8547	557	8225	167
Large	Medium	GA	61498	71467	83643	97176	160306	98809	89041	24055
Large	Medium	ABC	61675	62087	62489	62770	63189	1514	62457	420
Large	Medium	TABU	113185	130111	137327	141849	150580	37394	135838	9379
Large	Medium	ALNS	465193	578747	654991	720096	1145046	679853	675940	151556
Large	High	GRASP	1493342	3551879	5434621	9046544	19095695	17602354	6852542	4331658
Large	High	SA	7975	8297	8497	8625	9030	1055	8487	237
Large	High	GA	75027	111420	149417	177359	349640	274613	158560	68139
Large	High	ABC	61930	62303	62625	62881	63333	1403	62609	411
Large	High	TABU	114402	124624	130274	141119	155099	40696	132282	11163
Large	High	ALNS	540514	841339	1075096	1254269	1548798	1008285	1072654	262002

F Active Batching Utilization

Table 17: Descriptive values of active batching utilization aggregated over 30 runs

Size	Stoch.	Model	Min	1st Q	Median	3rd Q	Max	Range	Mean	Std.
Small	No	GRASP	0.45	0.49	0.50	0.52	0.56	0.11	0.50	0.03
Small	No	SA	0.45	0.48	0.50	0.51	0.55	0.09	0.50	0.02
Small	No	GA	0.46	0.49	0.50	0.52	0.55	0.10	0.50	0.03
Small	No	ABC	0.46	0.49	0.50	0.51	0.55	0.10	0.50	0.03
Small	No	TABU	0.46	0.49	0.50	0.51	0.55	0.09	0.50	0.02
Small	No	ALNS	0.46	0.49	0.50	0.52	0.55	0.10	0.50	0.02
Small	Medium	GRASP	0.47	0.53	0.54	0.57	0.59	0.12	0.54	0.03
Small	Medium	SA	0.48	0.52	0.53	0.56	0.58	0.10	0.54	0.02
Small	Medium	GA	0.50	0.52	0.54	0.56	0.61	0.11	0.54	0.03
Small	Medium	ABC	0.49	0.53	0.54	0.56	0.62	0.13	0.55	0.03

Continued on next page

Table 17 – Continued from previous page

Size	Stoch.	Model	Min	1st Q	Median	3rd Q	Max	Range	Mean	Std.
Small	Medium	TABU	0.49	0.52	0.53	0.55	0.57	0.08	0.53	0.02
Small	Medium	ALNS	0.49	0.53	0.55	0.57	0.62	0.13	0.55	0.03
Small	High	GRASP	0.48	0.54	0.56	0.58	0.78	0.30	0.57	0.06
Small	High	SA	0.48	0.52	0.55	0.60	0.78	0.30	0.56	0.06
Small	High	GA	0.51	0.54	0.56	0.60	0.80	0.29	0.58	0.06
Small	High	ABC	0.50	0.54	0.56	0.60	0.71	0.21	0.58	0.05
Small	High	TABU	0.49	0.51	0.55	0.58	0.77	0.28	0.56	0.06
Small	High	ALNS	0.50	0.54	0.57	0.60	0.78	0.28	0.58	0.06
Medium	No	GRASP	0.63	0.65	0.68	0.70	0.74	0.11	0.68	0.03
Medium	No	SA	0.61	0.65	0.67	0.69	0.72	0.11	0.67	0.03
Medium	No	GA	0.63	0.67	0.68	0.70	0.73	0.10	0.68	0.02
Medium	No	ABC	0.62	0.65	0.67	0.69	0.71	0.09	0.67	0.02
Medium	No	TABU	0.61	0.65	0.67	0.69	0.72	0.11	0.67	0.03
Medium	No	ALNS	0.63	0.67	0.68	0.70	0.73	0.10	0.68	0.02
Medium	Medium	GRASP	0.67	0.71	0.74	0.79	1.00	0.33	0.76	0.08
Medium	Medium	SA	0.66	0.70	0.73	0.76	0.85	0.19	0.73	0.05
Medium	Medium	GA	0.65	0.72	0.76	0.80	0.86	0.21	0.76	0.05
Medium	Medium	ABC	0.62	0.69	0.73	0.77	0.90	0.28	0.74	0.06
Medium	Medium	TABU	0.66	0.70	0.72	0.77	0.90	0.23	0.74	0.05
Medium	Medium	ALNS	0.67	0.72	0.75	0.78	0.86	0.19	0.75	0.05
Medium	High	GRASP	0.71	0.81	0.90	1.00	1.36	0.65	0.93	0.14
Medium	High	SA	0.69	0.80	0.86	0.93	1.16	0.47	0.88	0.11
Medium	High	GA	0.71	0.83	0.88	0.95	1.16	0.46	0.89	0.10
Medium	High	ABC	0.71	0.81	0.84	0.88	1.08	0.37	0.86	0.08
Medium	High	TABU	0.66	0.81	0.85	0.94	1.15	0.48	0.87	0.10
Medium	High	ALNS	0.74	0.80	0.87	0.94	1.16	0.42	0.88	0.10
Large	No	GRASP	0.84	0.88	0.91	0.93	1.00	0.16	0.91	0.04
Large	No	SA	0.81	0.83	0.86	0.89	0.91	0.10	0.86	0.03
Large	No	GA	0.80	0.85	0.88	0.90	0.95	0.14	0.87	0.03
Large	No	ABC	0.80	0.83	0.85	0.88	0.93	0.13	0.86	0.03
Large	No	TABU	0.81	0.84	0.86	0.88	0.91	0.10	0.86	0.03
Large	No	ALNS	0.82	0.85	0.88	0.90	0.93	0.11	0.88	0.03
Large	Medium	GRASP	0.93	1.00	1.07	1.18	1.41	0.48	1.10	0.12
Large	Medium	SA	0.86	0.92	0.99	1.03	1.15	0.30	0.99	0.08
Large	Medium	GA	0.87	0.95	0.99	1.07	1.19	0.33	1.01	0.08
Large	Medium	ABC	0.84	0.93	0.97	1.05	1.17	0.33	0.99	0.08
Large	Medium	TABU	0.87	0.94	0.98	1.08	1.21	0.35	1.01	0.09
Large	Medium	ALNS	0.92	0.96	0.99	1.07	1.21	0.30	1.01	0.07
Large	High	GRASP	1.10	1.28	1.37	1.51	1.67	0.57	1.38	0.15
Large	High	SA	0.94	1.10	1.16	1.27	1.43	0.49	1.18	0.13
Large	High	GA	0.98	1.11	1.25	1.33	1.63	0.65	1.24	0.16
Large	High	ABC	0.91	1.11	1.24	1.32	1.65	0.74	1.23	0.16
Large	High	TABU	0.92	1.11	1.19	1.25	1.50	0.58	1.19	0.13
Large	High	ALNS	1.00	1.13	1.20	1.32	1.48	0.48	1.21	0.12