Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

Dynamic Stochastic Vehicle Routing Problem

Bc. Petr Eichler

Supervisor: Štěpán Kopřiva, MSc.

Study Programme: Open Informatics

Field of Study: Artificial Intelligence

May 2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Eichler** Jméno: **Petr** Osobní číslo: **420129**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Stochastický dynamický vehicle routing problem

Název diplomové práce anglicky:

Stochastic Dynamic Vehicle Routing Problem

Pokyny pro vypracování:

1. Study the algorithms and modelling approaches for dynamic vehicle routing problems ? problems where the routing is computed dynamically based on the current operational situation. Example of such a problem may be real-time food delivery service.
2. Construct a model capable of representing the dynamic routing problem and reasoning on the model. Make sure that the model c1ontains stochastic nature of the newly created orders.
3. Design an algorithm which computes assignment of a new order to the existing routes (vehicle and placement to the specific place on the route) or creates a new route. The algorithm should use stochastic knowledge about domain to anticipate future orders.
4. Implement the algorithm designed in 3) using suitable framework.
5. Evaluate the performance of the algorithm on data from a real/synthetic data set.

Seznam doporučené literatury:

1. S. Vonolfen and M. Affenzeller, Distribution of waiting time for dynamic pickup and delivery problems, Annals of Operations Research, vol. 236, no. 2, pp. 359-382, 2016.
2. R. Bent and P. V. Hentenryck, Scenario-based planning for partially dynamic vehicle routing with stochastic customers, Operations Research, vol. 52, no. 6, pp. 977-987,2004.
3. P. Van Hentenryck, R. Bent, and E. Upfal, Online stochastic optimization under time constraints, Annals of Operations Research, vol. 177, no. 1, pp. 151-183, 2010.
4. H. Lei, G. Laporte, and B. Guo, The capacitated vehicle routing problem with stochastic demands and time windows, Computers and Operations Research, vol. 38, no. 12, pp. 1775 -1783, 2011

Jméno a pracoviště vedoucí(ho) diplomové práce:

Štěpán Kopřiva, MSc., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **13.02.2018** Termín odevzdání diplomové práce: **25.05.2018**

Platnost zadání diplomové práce: **30.09.2019**

Štěpán Kopřiva, MSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Acknowledgements

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated. I would also like to thank to my supervisor Štěpán Kopřiva, MSc. for his valuable expertise and guidance. Finally, I must express my sincere gratitude to my family and friends for their help and encouragement throughout my years of study.

Author statement

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 24th May, 2018 .....

signature

Abstrakt

Dynamický stochastický vehicle routing problém (DSVRP) si v posledních letech získal mnoho pozornosti. Jedná se o klasické VRP, ve kterém však všichni zákazníci nejsou známi předem, ale jsou postupně odhalováni během vykonávání plánu. Plánovač má také od začátku k dispozici stochastické údaje o požadavcích jednotlivých zákazníků, které může využít k předvídání budoucích událostí. Protože u mnoha problémů z reálného světa, jako např. rozvoz potravin, je nutné zákazníkům dovážet určité zboží, zformulovali jsme novou DSVRP variantu, ve které jsou všichni zákazníci zásobováni z některého z dostupných skladů.

V této práci uvádíme matematickou definici statické a dynamické verze vehicle routing problému se sklady. K řešení tohoto problému navrhujeme dvě vyčkávací heuristiky a optimalizační strategii, která využívá soubor scénářů obsahujících jak známé, tak možné budoucí požadavky zákazníků. Z výsledků testování našeho algoritmu na množině syntetických testovacích příkladů vyplývá, že náš postup ve většině případů dokáže překonat nejlepší známé metody.

Klíčová slova: Dynamický stochastický vehicle routing problem, mnoho scénářů, vyčkávací heuristika, VRP se sklady

Abstract

The Dynamic Stochastic Vehicle Routing Problem (DSVRP) has received increased attention in recent years. It considers a routing problem where not all customers are known in advance but are dynamically revealed during the execution of the plan. Stochastic knowledge about the dynamic customer requests is available to the solver and can be used to anticipate possible future events. Because many real-world scenarios, such as grocery delivery services, need to bring some goods to their customers, we formulate a novel DSVRP variant where each customer is supplied from one of the available warehouses.

In this work, we present a mathematical definition of the static and dynamic version of the vehicle routing problem with warehouses. To solve this problem, we propose two novel waiting heuristics together with optimization strategy utilizing a pool of scenarios including both known and possible future requests. The computational results obtained on a set of synthetic test scenarios show that our approach in most cases surpasses existing state-of-the-art methods.

Keywords: Dynamic stochastic vehicle routing problem, multiple scenarios, waiting heuristics, VRP with warehouses

Contents

Chapter 1

Introduction

The vehicle routing problem (VRP) is one of the most popular and most challenging problems in combinatorial optimization. The main objective is to visit a set of customers with a fleet of vehicles while minimizing the overall travel costs. The problem was firstly formulated by Dantzig and Ramser [1] in 1959 and since then, several variants have been proposed to incorporate additional business constraints.

The traditional formulations expect that once the scheduling process is started, all its inputs are fixed and will not change during the execution of the plan. This approach has become insufficient in recent years when the development in positioning services and communication technologies allowed logistics companies to track and manage their fleets in real time. Dynamic stochastic VRP (DSVRP) introduces new opportunities to reduce operational costs and improve customer service. It is able to adapt the solution dynamically by incorporating real-time information and considering possible future events. The inherent difficulty and wide applicability of these types of problems motivate the development of the scheduler presented in this work.

1.1 Goals of the Thesis

The overall goal of this thesis is to produce a robust and efficient scheduler that could be used to solve real-world DSVRP instances. To achieve this goal, we have to accomplish the following sub-goals:

1. Study algorithms and modeling approaches for dynamic vehicle routing problems – problems where the routing is computed dynamically based on the current operational situation.
2. Construct a model capable of representing the dynamic routing problem and design an algorithm which assigns a new order to the existing routes. The algorithm should use stochastic knowledge about the domain to anticipate future orders.
3. Implement the algorithm designed in 2 using a suitable framework and evaluate the performance of the algorithm on data from a synthetic data set.

1.2 Contribution

We propose and formally define a novel DSVRP variant which allows to model many real-time logistical problems such as grocery delivery, food delivery or any other service where the customers make orders. Our formulation ensures that the vehicles are regularly replenished in some near warehouse before they are allowed to serve the customers. The selection of warehouse is done automatically based on the current location of the vehicle and its customers in a way that minimizes the overall travel costs.

To optimize this problem, we introduce the *Stochastic Customer Satisfaction* solver which iteratively selects the most promising plan from a pool of solutions containing both existing and stochastic future requests. The selection procedure chooses the most robust solution which is applicable to the biggest number of possible future scenarios. We also present two novel waiting strategies which distribute the idle time in a vehicle route. These heuristics use stochastic knowledge about the problem to select the most promising waiting locations. Implementation of our algorithm is tested on a collection of test cases and compared with other state-of-the-art solvers. This comparison suggests that our approach is able to surpass the other methods in most of the instances.

1.3 Thesis Organization

The rest of the thesis has the following structure:

- **Chapter 2** describes the existing approaches used to solve non-deterministic vehicle routing problems. Three problem categories are discussed: (i) dynamic VRP, (ii) stochastic VRP and (iii) dynamic stochastic VRP. This chapter addresses the first goal of the thesis.
- **Chapter 3** provides a mathematical formalization of the static vehicle routing problem with warehouses using integer linear program. It also presents a multistage stochastic program used to formalize the dynamic stochastic variant of this problem.
- **Chapter 4** introduces TASP framework which is used to solve the static version of the VRP with warehouses.
- **Chapter 5** presents our *Stochastic Customer Satisfaction* (SCS) solver. It explains how the SCS solver exploits the stochastic information about the problem and how it uses waiting heuristics to distribute the idle time. This chapter addresses the second goal of the thesis.
- **Chapter 6** describes a set of synthetic benchmark instances, provides a comparison of our approach with other state-of-the-art methods and analyzes the obtained results. This chapter addresses the last goal of the thesis.
- **Chapter 7** summarizes our work and results. It also presents possible future extensions of our algorithm.

Chapter 2

State of the Art

In this chapter, we describe existing approaches used to solve vehicle routing problems (VRP) with delayed information availability or some level of uncertainty. In classical static and deterministic VRP, all problem inputs are known before the scheduling starts and do not change during the plan execution [2]. Unfortunately, real-world applications often contain some level of uncertainty or have to deal with a dynamic environment where the initial plan might become infeasible.

Section 2.1 is focused on approaches where not all information about the problem is known in advance. When some new information is revealed, the initial plan is recalculated to address the changes in the data. Typically, the update of the plan should be as small as possible, to avoid communication overhead between the vehicles and the central dispatcher.

In stochastic VRP (see Section 2.2), one or more parameters are only known as random variables with a known probability distribution. The final plan is created before the true values are obtained and is not changed afterward. When a vehicle is not able to follow the planned route (for example because its capacity is exceeded or it is unable to meet the next time window), the vehicle uses one of the predefined backup actions such as return to depot or omission of the next customer.

Finally, section 2.3 describes approaches which exploit the stochastic knowledge about the revealed data to predict future changes and information updates. This means that the routes are re-planned to not only satisfy the existing requests but also to allow easier handling of expected future events.

2.1 Dynamic Vehicle Routing Problem

In dynamic VRP (DVRP), also known as real-time or online VRP, the information available to the planner may change during the plan execution. These changes may involve arrival of new customers [3], increased travel times caused by traffic jams [4] or changes in customer's requirements such as demand, opening hours or service time [5]. After each information update, the planner recomputes the current plan with the new data set. Even though the

reoptimization procedure can be based on algorithms developed for classical static VRP, it often uses a different objective function. Attributes such as the number of changed routes, response time, or ability to handle future changes are often considered [6].

Because the DVRP planner has no information about the future events, exact methods provide a plan which is optimal only for the current state, with no guarantee about its optimality once all the data are obtained. Therefore, the majority of DVRP algorithms use heuristic methods which are able to quickly update a solution to reflect changes in the environment [7].

2.1.1 Dynamic Programming

One of the first authors who tried to solve DVRP problem was Psarafis et al. [8] in 1980. He demonstrated differences between the static and dynamic solution for dial-a-ride problem with a single vehicle. The dynamic version was solved using dynamic programming approach which recomputed the solution after every new customer request. Special priority rules were developed to preclude indefinite waiting of customers. The main drawback of this approach is its poor scaling for larger instances, where the problem very soon becomes intractable.

Ou and Sun [9] designed a dynamic programming algorithm based on chaos optimization algorithm (COA) to solve DVRP with real traffic information. Their solution consists of two modules. The first one is a route calculation module (RCM), which is used to search the optimal schedule in the feasible solution space. It uses chaotic variables which are able to non-repeatedly search all the states in a certain area. This module is called by the dynamic programming module (DPM), which evaluates the real-time traffic information and uses the RCM to adapt the plan when a critical value in the network is exceeded.

2.1.2 Biologically Inspired Metaheuristics

Many authors utilized biologically inspired algorithms which are able to adapt to unforeseen changes in the task environment. Elhassania et al. [10] used a genetic algorithm (GA) to iteratively schedule all customers which appeared in the last time step. Insertion heuristics were used during generation of the initial population to improve the quality of the obtained results. After that, a traditional GA approach was applied. The algorithm maintains a population of solutions through a fixed number of iterations. At each iteration, a combination of two genetic operators is used to produce the next generation. Crossover operator combines properties of two or more parents and mutation operator randomly changes part of the solution.

Benyahia and Potvin [11] used genetic programming (GP) to approximate the decision process of a professional dispatcher for a courier service company. GP extends the GA paradigm to nonlinear structures. Each program is represented as a tree structure constructed from the predefined functions, variables, and constants. Mutation and crossover operators then manipulate with the whole subtrees, instead of individual solution bits in the classical GA.

The evaluation function compares decisions produced by the created program with decisions from professional dispatchers to find a program which most accurately mimics the behavior of a real dispatcher.

Another popular biologically inspired approach used to solve DVRP instances is ant colony optimization (ACO), which uses simple agents interacting locally with each other. The ants walk around the graph representing the problem instance, laying down a pheromone trail which other ants follow. In each step, an ant probabilistically selects one edge to follow based on its pheromone intensity. When an ant finds a new solution, the edges which belong to this solution receive additional pheromone proportional to its quality. The final route then consists of edges with the highest pheromone intensity.

One of the main advantages of the DVRP ACO algorithms is the possibility to transfer pheromone traces after each information update. It allows to maintain characteristics of good solutions, which leads to significant improvements in the response times. Gambardella et al. [5] described ACO algorithm AntRoute, which is able to take into account additional constraints such as vehicle accessibility restrictions or time windows. The presented algorithm was used to schedule routes for the largest Swiss supermarket chain. A similar approach was also used by Rizzoli et al. [12] for freight distribution and Montemanni et al. [3] to serve customers of a fuel distribution company.

2.1.3 Tabu Search

Adaptive memory parallel tabu search was used for a courier service application by Gendreau at al. [13]. The algorithm maintains an adaptive memory which stores the routes of the best solutions visited during the search. Each new solution is then constructed by combining routes taken from this memory. The parallelized search is done by partitioning the current solution into smaller subproblems which are optimized independently. When a new customer request arrives, the algorithm tries to insert it into each solution in the memory to decide whether it should be accepted or rejected. The same approach was also used by Attanasio et al. [14] for dial-a-ride problem and Ichoua et al. [4] for vehicle dispatching with time-dependent travel times.

Algorithm for patient transportation problem in large hospitals was presented by Beaudry at. al [15]. The authors had to include many hospital-specific constraints, which complicated the search for a feasible solution. They proposed a two-phased heuristic procedure to handle all the necessary features and constraints. In the first phase, a simple feasible solution is generated. This solution is then improved in the second phase with a tabu search algorithm. The authors showed that this approach reduced both the waiting times for patients and the number of used vehicles.

2.2 Stochastic Vehicle Routing Problem

Stochastic VRP (SVRP) is used in many real-world scenarios, where not all data are known with certainty before the scheduling starts. For example, real customer demands might be revealed only when the vehicles are on their route. These changes might be resolved dynamically (see section 2.1), which typically means that some central dispatcher manually informs all the affected vehicles about the new route plan. Unfortunately, this approach is not always possible due to missing infrastructure or human resources which would allow to reschedule all the vehicles in the fleet after each information update.

In such a case, drivers are typically instructed to follow one or more simple rules of handling situations where it is no longer possible to follow the original route plan. For example, detour to the depot is used when the capacity of the vehicle is exceeded. Another possible action is to skip the next customer if it is no longer possible to meet the opening hours. To minimize the number of needed recourse actions, SVRP algorithms utilize stochastic information about uncertain parameters to produce robust plans which do not deviate much from the actual route execution.

2.2.1 Stochastic Parameter Estimation

The stochastic parameters either follow some known probabilistic distribution or are sampled from historical data. Ehmke et al. [16] described data mining procedures used to obtain time-dependent travel times and other planning constraints from a large set of individual travels. The data are extracted from approximately 230 million records obtained in the area of Stuttgart, Germany. To create a time-dependent travel matrix, the authors had to firstly cluster the records into homogeneous groups according to their relative variation of daily speeds. Each cluster is then represented by 24 speed reduction factors per weekday, depicting urban traffic patterns such as traffic jams.

Ferrucci [17] used a sophisticated offline procedure to obtain stochastic knowledge about expected future customers from historical data. This information is transformed into dummy customers, which are then planned together with the real requests. The dummy customers are mainly used to guide vehicles to request-likely areas and are replaced with real requests during the plan execution. To derive reliable stochastic knowledge, the whole state space is segmented into a spatiotemporal grid. Adjacent segments are then merged into clusters to allow reliable forecasts about future requests. Special clustering rules are derived to obtain clusters with desired properties. The potential customers are then sampled using Poisson distribution with a different rate parameter λ for each cluster. During the scheduling, the rate parameters are updated based on the real customer demands.

2.2.2 Mathematical Programming

The multi-compartment SVRP with stochastic demands is solved by Mendoza et al. [18]. In this problem, customers require several products which have to be loaded in independent

vehicle depots. This problem was modeled as a stochastic program with recourse, which minimizes the total travel cost together with expected costs caused by detours to the depot when a capacity of some vehicle is exceeded. This program was solved with a memetic algorithm. It works as a classical GA but uses local search procedure together with crossover and mutation operators to find the next generation of solutions.

In the previous paragraph, we mentioned stochastic program with recourse as one possible way how to mathematically model SVRP. Another approach often used in literature is a chance-constrained program. It ensures that the probability of meeting each stochastic constraint is above the given confidence level. As a result, the chance-constrained program produces robust solutions with a low number of recourse actions. This formalization was used for example by Beraldi et al. [19], Chen et al. [20] or Errico et al. [21]

Many authors tried to solve these mathematical programs using exact algorithms. Some of the used approaches are a branch-and-cut algorithm (Beraldi et al. [19], Chen et al. [20]), branch-and-price algorithm (Gauvin et al. [22], Christiansen et al. [23]) or integer L-shaped method (Cote et al. [24], Chang [25]). Unfortunately, all these methods are poorly scalable and cannot be used on bigger instances with more than 50-80 customers. For this reason, the majority of the authors proposed also some kind of approximation or heuristic to be able to solve bigger problems.

2.2.3 Local Search Metaheuristics

Simulated annealing (SA) is a popular probabilistic technique where a temperature variable is used to control the search process. When the temperature is high, solutions with higher cost are more likely to be accepted. As the temperature decreases, the acceptance criterion becomes stricter and the optimizing procedure can focus solely on improving solutions.

Ahmadi-Javid and Seddighi [26] used SA to solve SVRP in a supply-chain network with stochastic production capacity and randomly disrupted vehicles. Their algorithm starts with a random initial solution which is then updated in two phases. In the location phase, allocation of customers to individual producers is updated. Then, in the routing phase, 2-OPT algorithm is used to improve the routes with fixed allocation of customers. Both phases use SA to find an improving solution.

Lei et al. [27] described how to use record-to-record travel heuristic to solve capacitated VRP with stochastic demands and time windows (CVRPSDTW). The main optimization algorithm accepts each solution which is at most δ percent worse than the best-found solution. To intensify the search space exploration, adaptive large neighborhood search (ALNS) heuristic [28] was used to produce new solutions. At each iteration, ALNS selects one of several removal heuristics to remove 10%–20% of the customers from the current plan. After that, an insertion heuristic is used to reinsert all the removed customers back into the plan.

2.3 Dynamic Stochastic Vehicle Routing Problem

As the name suggests, dynamic stochastic VRP (DSVRP) combines approaches described in section 2.1 and 2.2. The schedulers firstly generate a robust initial plan which serves the customers known in advance and is also able to deal with anticipated changes caused by the stochastic events. When a new information is revealed, the schedulers update their stochastic model and recompute the current plan based on this new model. Because the solvers use all available information for all the decisions, it allows to create more accurate plans with lower overall costs.

2.3.1 Markov Decision Process

DSVRP with stochastic customers was modeled as a Markov decision process (MDP) by Thomas [29], [30]. The goal was to maximize the expected number of customers served by a single vehicle. For a given order of customers, this approach allowed to fully characterize the optimal policy for route construction with one dynamic customer. Because the state space grows exponentially with the number of customers, MDPs can be used to solve only very small instances. To overcome this limitation, the author also proposed a set of real-time heuristics which determine waiting positions for vehicles.

Kim et al. [31] proposed a Markov decision process to solve DSVRP with nonstationary stochastic travel times under traffic congestion. To avoid the exponential growth of the problem size, the authors also introduced dynamic programming approach which uses an approximation of the travel cost function based on a Monte Carlo simulation with possible scenarios. The performance of the proposed approach was verified on a delivery network in Singapore. When compared with a static solution which ignores the effects of traffic congestion, a 7% improvement in total travel time was achieved.

2.3.2 Waiting Heuristics

Waiting heuristics can be used in DSVRP instances where the vehicles have some idle time between requests. Each vehicle can either wait at the location of the last customer, head toward the next position, or select a strategic location with a higher probability of future requests. The selected waiting strategy influences position of the vehicles when a new request arrives, which can significantly increase the acceptance probability of this request.

The importance of a proper waiting strategy was demonstrated by Ichoua et. al.[32]. They proposed a threshold-based vehicle waiting heuristic that exploits probabilistic knowledge about future customer requests. The authors divided the service area into zones defined by geographic location and time period. For each zone, the probability of customer arrival is calculated. The vehicle waits at its current location if its next destination is far enough, there are not too many other vehicles in the neighborhood and the current area has a high probability of new request arrival. The presented heuristic was compared with a simple

deterministic strategy where the vehicle always waits at its current location until the next customer can be served. The experiments show that the proposed approach reduces overall travel time by 2%–10%.

Vonolfen and Affenzeller [33] analyzed waiting strategies for pickup and delivery problem with time windows. The authors compared heuristics which utilize historical data with general heuristics which do not incorporate any knowledge about the problem structure. Based on a set of test instances, heuristic using intensity measure provided the most robust performance over the whole benchmark set. This heuristic works similarly as the one developed by Ichoua et. al.[32] but needs only historical request data instead of a stochastic model. The intensity of some location is calculated as a normalized average transition time to all requests in the historical request set. Positions with higher intensity are then preferred as waiting locations.

2.3.3 Multi Scenario Approach

Bent and Hentenryck [34] developed Multiple Scenario Approach (MSA) which continuously generates a pool of routing plans from known and potential future requests. The selection function is used to choose one plan in each decision step (vehicle departure, request arrival). After that, the plan pool is updated to ensure that all the solutions are coherent with the selected plan. Consensus selection strategy chooses a plan most similar to other plans in the pool. This strategy produces significantly better results than the selection of the plan with the smallest travel cost (greedy strategy), especially on instances with many dynamic requests.

Because the selection strategy is crucial for algorithms based on MSA, Hentenryck et al. [35] compared three selection algorithms differing in the way of selecting the next customer. Expectation algorithm evaluates all available requests against all sampled scenarios at each decision step and selects the customer with the lowest overall cost. The main drawback of this approach is its time complexity because for n customers and m scenarios it is necessary to solve $n \cdot m$ VRP instances. Consensus algorithm solves each sampled scenario only once with all available requests together. The request which is selected as the next customer receives all the credit for the given scenario and all the other requests receive no credit. The main limitation of this algorithm is its elitism. It ignores requests which are never the best for any scenario but are most robust overall.

Regret algorithm is designed to overcome this issue. It solves the same problem as the consensus algorithm but uses a fast heuristic to approximate how much worse are the other available requests. To be more precise, let's say that customer c was selected as the first customer on the route of vehicle v. The regret function tries to swap another available request r with customer c on v and awards request r based on the quality of the newly obtained result. When the scenario becomes infeasible, the request r receives no credit. Note that this is only an approximation of the real insertion cost because the order of all the other customers is fixed. Computational experiments show that the regret and consensus algorithms produce

comparable results for problem instances with a low degree of dynamism. However, the regret algorithm obtains much better results on highly dynamic instances where the number of missed customers is reduced by 20%–50%.

This approach was further improved in Guillain et al [36]. The authors introduced a new decision rule called Global Stochastic Assessment (GSA) and described a heuristic approach which efficiently approximates this rule. Given a set of scenarios, GSA produces only one plan that best suits all the scenarios from the pool. This solution is designed to be as flexible as possible. On the contrary, MSA solves each scenario separately and then uses some selection strategy to choose one plan which is specialized for its associated scenario. Several versions of this algorithm were tested with different waiting and relocation strategies. The computational results suggest that this approach is very well suited for highly dynamic problems with many late customer requests, where GSA outperforms MSA.

Chapter 3

Problem Definition

We define a special variant of the classical dynamic stochastic VRP – dynamic stochastic vehicle routing problem with warehouses (DSVRPW). It is a generalization of pickup and delivery problem (PDP), where each request is characterized by a pickup and delivery location. Each load has to be then transported between these two locations by a single vehicle without any transshipment at other location. In our formulation, it is not necessary to specify the pickup location for each request in advance. The most suitable supply warehouse is then selected automatically from a set of existing warehouses. This formulation much better matches many real-world scenarios such as grocery delivery services, where the vehicle must firstly pick up the ordered groceries in one of the available warehouses before it is delivered to the customer.

In Section 3.1, we define a static version of the DSVRPW. In this variant, all the customers and their demands are known in advance. The main task is to assign the customers to vehicles and find which supply warehouse should be selected for each customer to minimize the total cost. In Section 3.2, we describe how the static version of the problem must be changed when some of the customers are not known in advance but are introduced dynamically during the search. This means that we are no longer minimizing the total cost for known customers, but the total expected cost including potential customers with known stochastic properties.

3.1 Static Problem

In the static version of the DSVRPW (VRPW), all the problem entities together with all their properties are known to the scheduler before the scheduling starts. This means that no additional information about the given problem is revealed during the plan execution. We firstly define all the entities occurring in the VRPW in Subsection 3.1.1. For each entity, a brief description of its properties is presented. In the next part (see Subsection 3.1.2), a proper mathematical model of the VRPW is presented, describing both the objectives and constraints of this problem.

3.1.1 Problem Notation

Customers

Each problem contains a set of customers C. This set consists of two disjoint sets, $C^{\mathcal{W}}$ and $C^{\mathcal{D}}$, that satisfy $C = C^{\mathcal{W}} \cup C^{\mathcal{D}}$, $C^{\mathcal{W}} \cap C^{\mathcal{D}} = \emptyset$. Customers from the set $C^{\mathcal{W}}$ need to have their goods firstly picked up in a warehouse before they can be served. On the other hand, the set $C^{\mathcal{D}}$ contains customers which do not accept goods from warehouses and their needs must be satisfied directly from the vehicle depot. Each customer $c \in C$ has also defined penalty u_c when the corresponding request is not served and set $\mathcal{V}_c$ containing vehicles which are allowed to visit the customer.

While it might seem that the difference between these two types of customers is quite negligible, they produce significantly different results in practice. If we work with customers supplied from the depot, the vehicle must load goods for all the served customers before it starts its route. This means that its capacity strongly limits the maximal number of served customers. Conversely, one vehicle is able to visit a virtually unlimited number of customers supplied from the warehouse because it is possible to periodically replenish the goods in warehouses.

Warehouses

The set $\overline{\mathcal{W}}$ contains locations of all existing warehouses in the problem. Each warehouse has an unlimited amount of goods and can be visited repeatedly. The number of visits is not known in advance. This makes the VRPW problem more complicated than classical VRP instances because the scheduling algorithm has to simultaneously assign customers to vehicles and plan how often and which warehouses to visit.

To be able to create a mathematical model of VRPW, we need to distinguish individual warehouse visits (for example to track arrival times). Because the number of warehouse assignments is unknown, it is impossible to create decision variables for each assignment. To overcome this issue, we propose to construct set $\mathcal{W}$ containing all relevant potential warehouse assignments (RPWA). The number of RPWA for each warehouse location $\overline{w} \in \overline{\mathcal{W}}$ is equal to the number of adjacent customers and vehicle depots. Because each customer and depot can be visited only once, we can uniquely identify each RPWA by looking at its predecessor in vehicle plan. The transformation process from the original problem into a problem with RPWA is depicted in Figure 3.1.

Note that the RPWA does not contain all possible warehouse assignments because their number is unlimited. It excludes plans with two or more consecutive warehouse visits. The transformation process from the original plan into a plan with RPWA is depicted in Figure 3.2. Fortunately, these scenarios can be easily neglected because consecutive warehouse visits only increase plan cost without any benefits. This means that the optimal solution with RPWA will have always the same cost as the optimal solution of the original problem. For this reason, whenever we will talk about warehouses, we will assume the RPWA from the set $\mathcal{W}$, not the original warehouse locations from the set $\overline{\mathcal{W}}$.

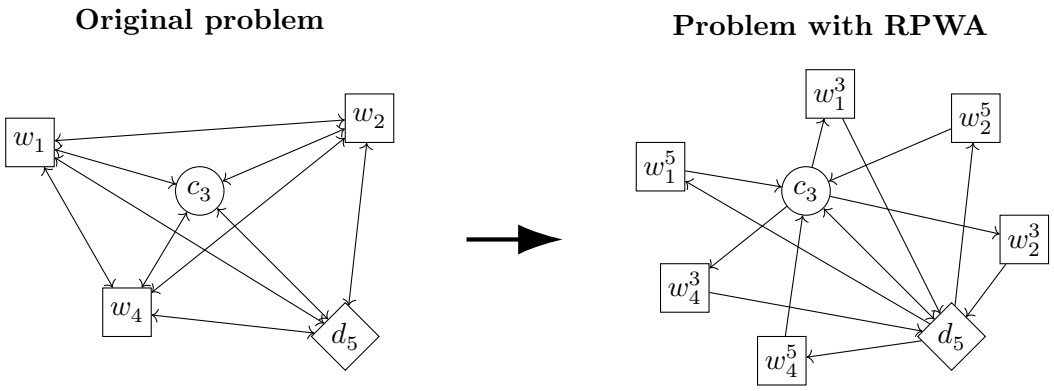


Figure 3.1: Transformation process from the original problem into a problem with RPWA. In the original problem, we have one customer c_3, one depot d_5, and three warehouses w_1, w_2 and w_4. As we can see, all the nodes are connected with oriented edges. In the transformed problem, each customer and depot has its own set of warehouses which can be visited from the given node. This relation is illustrated with the superscript number referring to the associated node. In our case, a vehicle starting from the position of the customer c_3 can visit warehouses w_1^3, w_2^3, and w_4^3 and from the depot d_5 can visit warehouses w_1^5, w_2^5, and w_4^5. Notice that the newly created warehouses are no longer connected with edges, which means that it is not possible to directly travel between warehouses.

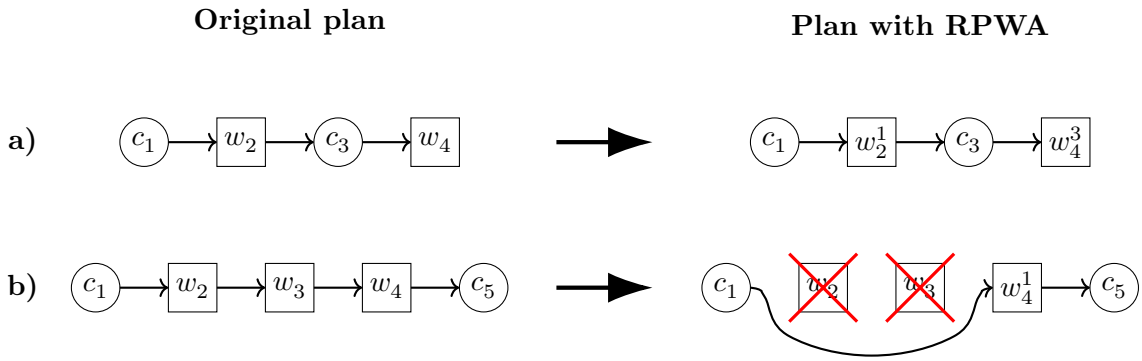


Figure 3.2: Transformation process from the original plan into a plan with RPWA. In scenario **a)**, the transformed plan is almost identical with its predecessor. The only difference is that the original warehouses are replaced with the RPWA. Scenario **b)** illustrates the case with three consecutive warehouse visits. Because with RPWA it is no longer possible to directly travel between warehouses, the first two assignments are removed from the plan and the vehicle travels from the customer c_1 straight to the warehouse w_4^1.

Waypoints

The set of waypoints, defined as $\mathcal{P} = \mathcal{C} \cup \mathcal{W}$, consists of customers and warehouses. Each waypoint has defined its opening time o_p, closing time c_p and visit duration v_p. All the vehicles must arrive before the closing time c_p, otherwise the visit cannot be done. When a vehicle arrives before the opening time o_p, it is required to wait until o_p to start the service.

Vehicles

Let $\mathcal{V}$ be a set of vehicles. Each vehicle $v \in \mathcal{V}$ starts its route in its start depot y_v and cannot depart before the earliest start time e_v. The route then ends in its end depot z_v

and the vehicle must arrive before the latest end time l_v. Fixed cost f_v is charged for each used vehicle. All the vehicles have also defined maximal capacity w_v which cannot be exceeded.

Nodes and Edges

VRPW instance is defined on a complete directed graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$. Each node $n \in \mathcal{N}$ represents either a customer, warehouse or vehicle depot, $\mathcal{N} = \mathcal{C} \cup \mathcal{W} \cup \mathcal{D}$. All the nodes are connected with oriented edges $\mathcal{E} = \mathcal{N} \times \mathcal{N}$. For each edge $(i, j) \in \mathcal{E}$, we define travel time $r_{i,j}$ and road distance $n_{i,j}$. When a pair of nodes is not directly connected in the real world, we set $r_{i,j} = n_{i,j} = \infty$.

We summarized all the entities used in the problem definition in Table 3.1 and all the properties for each entity in Table 3.2. Note that the proposed notation is based on the notation introduced in Eichler [37].

Set	Content
$\mathcal{C}$	All customers
$\mathcal{C}^{\mathcal{W}}$	All customers which need a supply warehouse
$\mathcal{C}^{\mathcal{D}}$	All customers which are supplied from a depot
$\overline{\mathcal{W}}$	All warehouse locations
$\mathcal{W}$	All relevant potential warehouse assignments
$\mathcal{P}$	All waypoints (customers and warehouses)
$\mathcal{D}$	All vehicle depots
$\mathcal{N}$	All nodes (waypoints and vehicle depots)
$\mathcal{V}$	All vehicles
$\mathcal{V}_c$	All allowed vehicles for customer c
$\mathcal{E}$	All edges

Table 3.1: Summary of all the entities used in the problem definition

Name	Meaning	Name	Meaning
o_p	Opening time of waypoint p	e_v	Earliest start time of vehicle v
c_p	Closing time of waypoint p	l_v	Latest end time of vehicle v
v_p	Visit duration of waypoint p	w_v	Capacity of vehicle v
d_c	Demand of customer c	y_v	Start depot of vehicle v
u_c	Penalty for unassigned customer c	z_v	End depot of vehicle v
$r_{i,j}$	Drive time between nodes i and j	f_v	Fixed cost for using vehicle v
$n_{i,j}$	Distance between nodes i and j	x	Large scalar

Table 3.2: Summary of all the properties for each entity used in the problem definition

3.1.2 Mathematical Model

The decision variables of the model are specified in Table 3.3.

Name	Meaning
$\alpha_{i,j,v}$	1, if vehicle v visits node j immediately after node i; 0, otherwise
$\beta_{i,j,v}$	The amount of depot goods in vehicle v between nodes i and j
$\gamma_{i,j,v}$	The amount of warehouse goods in vehicle v between nodes i and j
ω_i	The arrival time to node i

Table 3.3: Decision variables used in the mathematical model of the problem

The objectives and constraints of the VRPW can be formulated as follows:

$$\min \sum_{v\in\mathcal{V}}\sum_{i\in\mathcal{N}}\left(\alpha_{y_v,i,v}\cdot f_v+\sum_{j\in\mathcal{N}} n_{i,j}\cdot\alpha_{i,j,v}\right)+\sum_{c\in C} u_c\cdot\left(1-\sum_{v\in\mathcal{V}}\sum_{i\in\mathcal{N}}\alpha_{c,i,v}\right) \tag{3.1}$$

The first sum counts the penalties for used vehicles and the cost for a total traveled distance, the second sum adds penalties for unassigned customers.

$$\text{s.t. } \alpha_{i,j,v}\in\{0,1\} \qquad \forall i,j\in\mathcal{N},\ \forall v\in\mathcal{V} \tag{3.2}$$

$$\beta_{i,j,v}\geq 0 \qquad \forall i,j\in\mathcal{N},\ \forall v\in\mathcal{V} \tag{3.3}$$

$$\gamma_{i,j,v}\geq 0 \qquad \forall i,j\in\mathcal{N},\ \forall v\in\mathcal{V} \tag{3.4}$$

$$\omega_i\geq 0 \qquad \forall i\in\mathcal{N} \tag{3.5}$$

The first four constraints are the integrality constraints. They specify allowed value ranges for all the decision variables.

$$\sum_{v\in\mathcal{V}}\sum_{j\in\mathcal{N}}\alpha_{i,j,v}\leq 1 \qquad \forall i\in\mathcal{N} \tag{3.6}$$

$$\sum_{j\in\mathcal{N}}\alpha_{c,j,v}=0 \qquad \forall c\in\mathcal{C},\ \forall v\in\mathcal{V}\setminus\mathcal{V}_c \tag{3.7}$$

$$\sum_{i\in\mathcal{N}}\alpha_{i,p,v}-\sum_{j\in\mathcal{N}}\alpha_{p,j,v}=0 \qquad \forall p\in\mathcal{P},\ \forall v\in\mathcal{V} \tag{3.8}$$

Constraints 3.6–3.8 are known as degree constraints. They ensure that each node is visited at most once (Constraint 3.6), each customer is visited only by allowed vehicles (Constraint 3.7) and that each vehicle always leaves all visited waypoints (Constraint 3.8).

$$\sum_{i\in\mathcal{N}}\alpha_{y_v,i,v}\leq 1 \qquad \forall v\in\mathcal{V} \tag{3.9}$$

$$\sum_{i\in\mathcal{N}}\alpha_{i,z_v,v}-\sum_{j\in\mathcal{N}}\alpha_{y_v,j,v}=0 \qquad \forall v\in\mathcal{V} \tag{3.10}$$

$$\sum_{i \in \mathcal{N}} \alpha_{z_v,i,v} + \sum_{j \in \mathcal{N}} \alpha_{j,y_v,v} = 0 \qquad \forall v \in \mathcal{V} \tag{3.11}$$

Constraint 3.9 determines that each vehicle is used at most once. Constraints 3.10 and 3.11 together ensure that each vehicle starts in start depot y_v and ends in end depot z_v.

$$\omega_i + v_i + r_{i,j} - x \cdot (1 - \alpha_{i,j,v}) \leq \omega_j \qquad \forall i, j \in \mathcal{N},\ \forall v \in \mathcal{V} \tag{3.12}$$

Constraint 3.12 sets the earliest arrival time to node j with respect to the previous visits. The large scalar x ensures that this constraint is always satisfied when $\alpha_{i,j,v}$ is 0. Note that for depots, the visit duration v_i is set to 0.

$$\omega_{y_v} \geq e_v \qquad \forall v \in \mathcal{V} \tag{3.13}$$

$$\omega_{z_v} \leq l_v \qquad \forall v \in \mathcal{V} \tag{3.14}$$

$$o_p \leq \omega_p \leq c_p \qquad \forall p \in \mathcal{P} \tag{3.15}$$

The next three constraints imply that each vehicle respects its earliest start time, its latest end time and that all waypoints are visited within their time window.

$$\beta_{i,j,v} \leq x \cdot \alpha_{i,j,v} \qquad \forall i, j \in \mathcal{N},\ \forall v \in \mathcal{V} \tag{3.16}$$

$$\gamma_{i,j,v} \leq x \cdot \alpha_{i,j,v} \qquad \forall i, j \in \mathcal{N},\ \forall v \in \mathcal{V} \tag{3.17}$$

$$\beta_{i,j,v} + \gamma_{i,j,v} \leq w_v \qquad \forall i, j \in \mathcal{N},\ \forall v \in \mathcal{V} \tag{3.18}$$

These three constraints ensure that the vehicles may transport goods only on routes in their plan (Constraints 3.16 and 3.17) and that the amount of goods in a vehicle must not exceed its capacity (Constraint 3.18).

$$\sum_{j \in \mathcal{N}} \beta_{y_v,j,v} = \sum_{c \in \mathcal{C}^{\mathcal{D}}} \sum_{j \in \mathcal{N}} \alpha_{c,j,v} \cdot d_c \qquad \forall v \in \mathcal{V} \tag{3.19}$$

$$\sum_{j \in \mathcal{N}} \gamma_{y_v,j,v} = 0 \qquad \forall v \in \mathcal{V} \tag{3.20}$$

Constraints 3.19 and 3.20 state that each vehicle at the start of its route contains only goods for customers who do not need supply warehouse.

$$\sum_{v \in \mathcal{V}} \sum_{i \in \mathcal{N}} \beta_{i,c,v} - \sum_{v \in \mathcal{V}} \sum_{j \in \mathcal{N}} \beta_{c,j,v} = d_c \cdot \sum_{v \in \mathcal{V}} \sum_{i \in \mathcal{N}} \alpha_{c,i,v} \qquad \forall c \in \mathcal{C}^{\mathcal{D}} \tag{3.21}$$

$$\sum_{v \in \mathcal{V}} \sum_{i \in \mathcal{N}} \gamma_{i,c,v} - \sum_{v \in \mathcal{V}} \sum_{j \in \mathcal{N}} \gamma_{c,j,v} = d_c \cdot \sum_{v \in \mathcal{V}} \sum_{i \in \mathcal{N}} \alpha_{c,i,v} \qquad \forall c \in \mathcal{C}^{\mathcal{W}},\ \forall v \in \mathcal{V} \tag{3.22}$$

These two constraints are known as flow conservation constraints. They ensure that all the customers will receive their goods either from the depot (Constraint 3.21) or from the warehouse (Constraint 3.22). Note that the demand d_c is multiplied by a sum of α to exclude unassigned customers. When some customer is unassigned, the sum is zero which means that no goods are delivered to the customer. Otherwise, the sum is 1 (see Constraint 3.6).

$$\sum_{i \in \mathcal{N}} \beta_{i,p,v} = \sum_{j \in \mathcal{N}} \beta_{p,j,v} \qquad \forall p \in \mathcal{P} \setminus \mathcal{C}^{\mathcal{D}}, \ \forall v \in \mathcal{V} \tag{3.23}$$

$$\sum_{i \in \mathcal{N}} \gamma_{i,c,v} = \sum_{j \in \mathcal{N}} \gamma_{c,j,v} \qquad \forall c \in \mathcal{C}^{\mathcal{D}}, \ \forall v \in \mathcal{V} \tag{3.24}$$

Constraint 3.23 states that only customers supplied from depot are allowed to change the amount of depot goods in a vehicle. This means that the amount of depot goods in a vehicle cannot be replenished in a warehouse. Constraint 3.24 then specifies that customers supplied from depot cannot be served with goods loaded in a warehouse. Note that we do not need to introduce constraints describing how are vehicles replenished in warehouses because this is done automatically in the optimal solution.

3.2 Dynamic Problem

In the dynamic formulation, some of the customers are not known in advance but are gradually revealed during the plan execution. For this reason, the solution must be periodically recomputed to include the newly revealed customers into the plan. We assume that the distribution of incoming requests is known in advance. If no such distribution exists, it can be approximated from historical data. For more information about stochastic parameter approximation in DSVRP context, see for example Ehmke et al. [16] or Ferruci [17].

In DSVRPW, we define a set of all customer requests $\mathcal{R} \subseteq C \times [0, T]$, where the planning horizon $[0, T]$, $T \in \mathbb{N}$ contains all integer values i such that $0 \leq i \leq T$. The deterministic customers c^t are known in time $t = 0$, whereas the dynamic customer requests are revealed in time $t \in [1, T]$. For each time $t \in [1, T]$ and customer c, we have a random variable $X_{t,c}$ which is defined as

$$X_{t,c} = \begin{cases} 1, & \text{if customer } c \text{ is revealed in time } t, \\ 0, & \text{otherwise.} \end{cases} \tag{3.25}$$

The realization of the random variable $X_{t,c}$ is noted as $x_{t,c}$. The probability that customer c is revealed in time t is then denoted as $p(X_{t,c} = 1)$. The vector of random variables for all customers in time t is defined as X_t and its realization is x_t.

Because the solution is continuously updated, we define s^t as a currently valid solution in time t. This solution must satisfy all the VRPW constraints defined in Subsection 3.1.2. Whenever a new customer request arrives, the currently valid solution s^t is updated to serve the new customer. Because the scheduling is done in real-time, all the vehicle movements

and decisions made before the time t must remain unchanged after the update.

Special attention must be paid to the warehouse assignments. Whenever a warehouse is visited at time $t_1 < t$, all the goods loaded into the vehicle must be delivered to customers selected in time t_1. The customer selection cannot be changed in the future because all the packages are unique (e.g. different food). This means that all the customers served from this warehouse cannot be serviced by another vehicle, even when the visit occurs in the future. Because the list of served customers must be known in time t_1, it is also forbidden to load some additional goods for potential customers which might materialize in the future. Therefore, when a new customer request is revealed, the vehicle must firstly visit a warehouse to load goods for this customer.

When dealing with dynamic problems, it is no longer enough to represent the solution as an ordered sequence of visited locations with corresponding arrival times. The problem occurs when there is an idle time between two visits. In static problems, it does not matter where the vehicle waits because the total travel time and distance are always the same. But in the dynamic context, we need to know the exact position of each vehicle in time t. For this reason, each assignment must contain two values - arrival time ω_i and departure time ψ_i. After the departure, the vehicle travels directly to the next location where it waits before the next assignment can start.

We can now formulate the multistage stochastic program describing how to derive a new currently valid solution s^t from s^{t-1} in time t:

$$s^t = \underset{s^t \in S(x_t)}{\operatorname{argmin}} \mathbb{E}_{X_{t+1}} \left(\min_{s^{t+1} \in S(x_{t+1})} \mathbb{E}_{X_{t+2}} \left(\dots \min_{s^{T-1} \in S(x_{T-1})} \mathbb{E}_{X_T} \left(\min_{s^T \in S(x_T)} \mathrm{f}(s^T) \right) \right) \right) \tag{3.26}$$

The function f calculates the solution cost using the mathematical model presented in Subsection 3.1.2. We are looking for a sequence of solutions s^i, $i \in [t, T]$ which minimizes the total expected cost at the end of the planning horizon. The sets $S(x_i)$, $i \in [t, T]$ represent all possible solutions for the realization of the random vector x_i.

$$\text{s.t. } s^i[0..i-1] = s^{i-1}[0..i-1] \qquad \forall i \in [t, T] \tag{3.27}$$

Constraint 3.27 ensures that each solution created in time i is identical with its predecessor up to time $i - 1$. This ensures that all the vehicle movements made in the past remain unchanged.

Chapter 4

Technical Background – TASP

In this chapter, we describe the key building blocks of the *Task and Asset Scheduling Platform* (TASP). This framework, developed by Blindspot Solutions [38], is designed to solve a large variety of NP-complete scheduling problems. It is a modular, efficient planning engine written in Kotlin. The scheduling module uses optimization heuristics derived from the algorithms presented in Eichler [37]. It is based on the *Adaptive Large Neighborhood Search* principle proposed by Pisinger and Ropke [28].

Firstly, the overall architecture of the TASP framework is presented in Section 4.1. In this part, we introduce individual components and explain how they interact with each other. In Section 4.2, we describe the properties of the most important entity in TASP – Plan. It is an immutable collection of assignments which allows fast querying and ensures consistency of the stored data. Then, we discuss Evaluator module responsible for efficient constraint evaluation in Section 4.3. The Evaluator is able to decide which parts of the Plan were changed and evaluate only those parts. This technique leads to significantly lower calculation times. In the last part (see Section 4.4), we present the main optimization heuristics used to solve the scheduling problems.

4.1 Overall Architecture

TASP consists of three main components – Plan, Evaluator and Scheduler. All these modules are designed to be highly flexible and allow the user to easily extend their functionality. They communicate through a defined API and can be freely replaced with a different implementation that is more suitable for the given task. The overall design of the TASP framework showing the interaction between the individual modules can be seen in Figure 4.1.

Before the TASP can be used to solve some planning problem, it is necessary to firstly specify the Data Model. It describes which classes are planning entities and how they are mapped to the real-world problem. Its correct definition is crucial in TASP because it can heavily influence the performance of the whole framework. It might be also very difficult to adapt

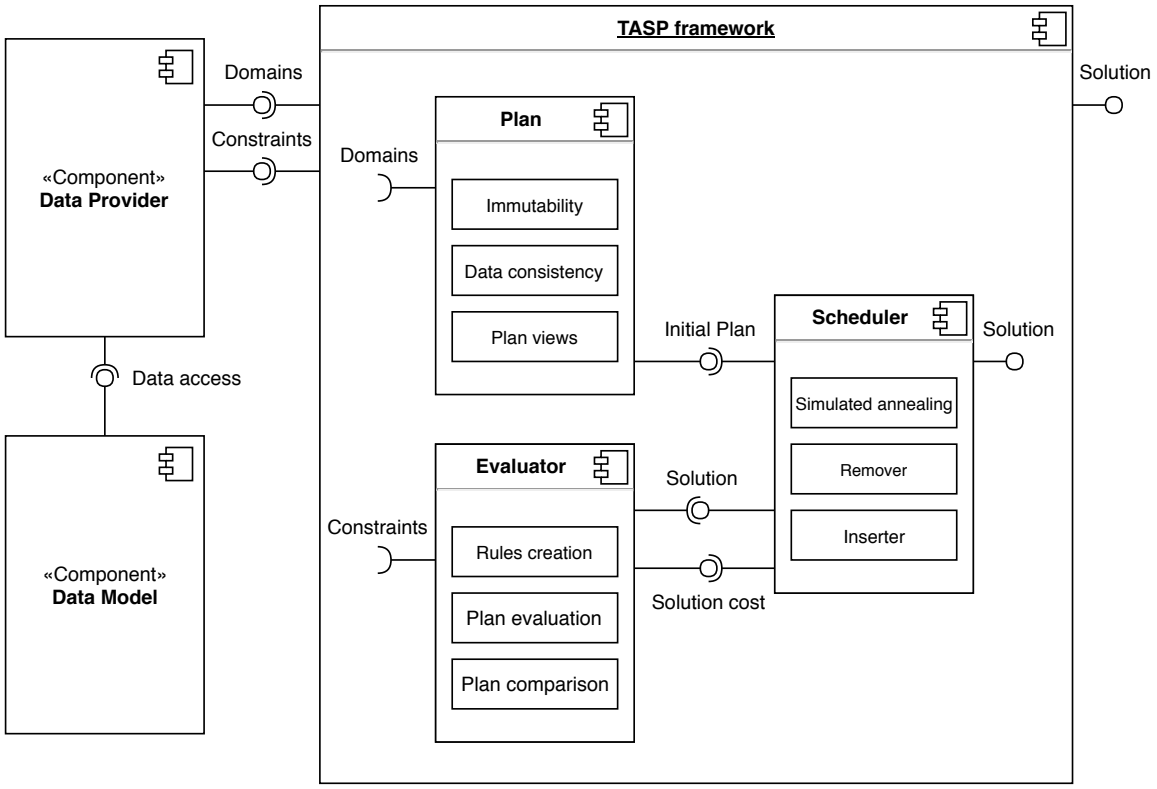


Figure 4.1: The overall architecture of the TASP framework.

the code base to future changes in the business requirements if the Data Model is selected inappropriately.

At the beginning of the scheduling process, Data Provider is used to submit the problem definition to TASP. It consists of domains for individual entities from the Data Model and constraints describing the real-world business rules. The domains are used to create an initial instance of Plan which will be then iteratively improved in Scheduler. The constraints are stored in Evaluator where they are used to evaluate the quality of the produced solutions.

The Scheduler then uses an iterative optimization strategy controlled by simulated annealing metaheuristic to optimize the initial Plan. In each iteration, part of the current solution is disrupted by Remover and optimized by Inserter. At the end of each iteration, Evaluator is used to determine whether the newly obtained Plan is good enough for further exploration or whether it should be discarded. After a predefined number of iterations, the best-found solution is returned as a result of the given problem.

4.2 Plan

The Plan is the most important entity in TASP. It is a collection of assignments with advanced querying capabilities which allow to perform most of the operations in constant time. In TASP, the assignment can be an arbitrary immutable object which groups planning entities from the Data Model together. Each assignment is also associated with some time or other comparable property to allow ordering of the stored data. In VRP, assignment consists of the vehicle, customer, and arrival time. These three entities fully describe one vehicle visit on its route.

Majority of the methods in other modules accept Plan as one of its input arguments and work with its content. For this reason, it is important to understand its basic properties and functionality. In the following paragraphs, we explain how Plan works internally and present some usage examples from the VRP domain.

4.2.1 Immutability

Whole TASP framework is inspired by the functional programming paradigm and tries to minimize undesirable side effects as much as possible. For this reason, Plan is implemented as an immutable data structure. It supports two basic operations allowing changes in the stored data – insertion and removal of assignments. Each time one of these functions is invoked, a new instance of Plan is created and returned as a result.

Internally, all the data are stored in immutable persistent collections with constant modification time. When updated, the new collection shares unchanged sub-structures with the original instead of making a full clone. In TASP, we use a type-safe version of Clojure's persistent collections named Paguro [39]. As a result, Plan has a very similar performance as its mutable variant with all the advantages resulting from its immutability.

To have Plan as a truly immutable object, it is necessary to ensure that all the entities from the domain model will be immutable as well. This is usually easily accomplished because the problem definition is known in advance and isn't changed during the scheduling. Because the assignment object is also immutable, it is forbidden to change its properties once created. This means that to modify for example arrival time, one must firstly remove the old assignment and replace it with a new one.

4.2.2 Data Consistency

When a new, empty instance of Plan is created, it is possible to specify rules which keep the stored data consistent. Firstly, the domain for each entity is provided. In VRP, this means that all the existing customers and vehicles from the problem definition are submitted. Plan then automatically controls each inserted assignment and rejects those which contain unknown entities. This helps in situations when the problem definition is gradually updated and not all parts of the codebase have the newest data.

Plan also tracks a level of allocation for specified entities. In a classical VRP, a vehicle is fully allocated when its remaining capacity is lower than the lowest demand and a customer is usually fully allocated when it is visited by some vehicle. Plan then rejects those assignments which contain some fully allocated items (e.g. customer which is already visited by some other vehicle). This functionality not only helps to keep the stored data consistent but also improves the performance of the scheduler by excluding non-perspective potential assignments.

4.2.3 Plan Views

Plan provides many functions through which the stored data are accessible in constant time. It applies to all the data described in previous paragraphs, such as stored assignments, the domain for each entity, or a level of allocation for some specified item. For entities which are marked as *indexed*, some additional queries which further simplifies data access are available. In VRP with indexed vehicles, it is possible to request a route plan for one vehicle, arrival times to all its assigned customers, or query which customer is visited in the specified time. Because all these additional queries are also done in constant time, it can significantly improve the performance of the scheduler.

Internally, the inserted assignments and other data are stored in a special multidimensional data structure backed by persistent collections. It behaves as a multi-key map (sometimes known as a table) which allows to have more than one value for each combination of keys. The first key (row key) is typically some item such as vehicle or customer. The next key (column key) then represents time. Each pair of keys is then associated with corresponding assignments.

4.3 Evaluator

Evaluator is used to calculate the cost of a given Plan. Individual Plans are compared by their cost and the scheduler aims to find the Plan with the lowest possible cost. Plan cost is calculated as a sum of penalties for violated business rules and other solution fees such as travel fares. Evaluator is responsible for correct evaluation of the stored constraints. It is also able to decide which parts of the Plan were changed and evaluate only those parts.

4.3.1 Constraint and Rule

In the real-world problems, business constraints are very often changed and added during the development phase and after the first releases. For this reason, TASP is designed to make working with constraints as easy as possible for the developer. Each TASP constraint accepts Plan and up to two additional entities for which the cost should be calculated.

For example, checking that delivery happens within customer's time window accepts an assignment as an additional entity and checks whether its arrival time is allowed. Another

constraint enforcing the maximal number of night shifts per month (for problems with multi-day planning horizons) accepts driver and current month. The constraint then extracts schedule for the given driver and month from the Plan and calculates the number of night shifts.

As we can see from the second example, the additional entities do not have to be part of the assignment. In our case, assignment consists of customer, vehicle and arrival time which means that there is no field for driver nor current month. In such cases, the user must create converter which returns the corresponding entity from the given assignment. In our example, the information about the month would be extracted from arrival time and the name of the driver would be probably available as one of the fields in the vehicle object. Constraint and corresponding converters together form one Evaluator rule.

4.3.2 Plan Evaluation

The main idea of constraint evaluation is that each constraint will be invoked for each entity pair only once. Let us say we have assignments which contain the following driver and day: (Mark - January 6), (Mark - January 9), (Mark - February 12), (Lisa - February 7). Then the constraint enforcing maximal monthly working hours will be evaluated for pairs (Mark - January), (Mark - February), (Lisa - February). We can see that the evaluation is done only once for Mark in January, even though Mark has two assignments in this month. There is also no evaluation for Lisa in January because she worked only in February.

Input: Plan P, Evaluator rules $\mathcal{R}$
Output: Plan cost C
begin
1: $\mathcal{A} \leftarrow P$.assignments
2: $C \leftarrow 0$
3: **for all** $R \in \mathcal{R}$ **do**
4: $\quad \mathcal{E} \leftarrow \{\}$
5: $\quad$ **for all** $A \in \mathcal{A}$ **do**
6: $\quad\quad \mathcal{E} \leftarrow \mathcal{E} \cup R$.convertAssignment(A)
7: $\quad$ **end for**
8: $\quad$ **for all** $(E_1, E_2) \in \mathcal{E}$ **do**
9: $\quad\quad C \leftarrow C + R$.evaluateConstraint$(P, E_1, E_2)$
10: $\quad\quad$ **if** C.isInfeasible() **then**
11: $\quad\quad\quad$ **terminate**
12: $\quad$ **end for**
13: **end for**
end

Algorithm 4.1: Plan cost calculation in TASP

Plan cost calculation is described in Algorithm 4.1. Evaluator firstly transforms all the assignments in the Plan P into a set of unique entity pairs (line 6). Note that when some

constraint needs less than two entities (e.g. uses only customer), the redundant entity is replaced with a placeholder object with no specific meaning. In line 9, the rule R is invoked for each unique entity pair. The total cost of the Plan is calculated as a sum of costs over all rules evaluated over all entity pairs. Line 11 then ensures premature termination when the evaluated Plan P is infeasible.

4.3.3 Plan Comparison

So far, it might be unclear what is the benefit of additional entities in constraints. After all, each constraint could accept only Plan and the user would check individual items manually. This approach would require two additional for-loops and a few lines of code to extract existing entities from the Plan. On the other hand, the user would not have to create converters, so the total effort for the user would be almost the same.

The real benefit of additional entities in constraints is apparent when two Plans are compared with each other. In TASP, Plan comparison is used much more often than the Plan cost calculation presented in Subsection 4.3.2. Whenever we want to add a new assignment, we compare all possible variants with each other to select the most promising option. To optimize a medium-sized problem, we typically need to compare millions of Plans while the exact cost is calculated only for hundreds of them.

For this reason, it is critical to compare Plans as fast as possible. Evaluator is able to determine which entities must be evaluated and can omit those which would not change the cost difference between the Plans. Without additional entities in constraints, Evaluator would have to calculate the total cost for both Plans and compare them at the end. Because the Plans most often differ only in one assignment, the time complexity of constraint evaluation is usually reduced from linear to constant time when the identical parts of the Plans are ignored. The selection of entities which must be evaluated is resolved completely by the Evaluator and the user only submits individual rules without any additional work.

Algorithm 4.2 describes how two Plans P_1 and P_2 are compared by the Evaluator. At the beginning, all the assignments from both Plans are split into three sets $\mathcal{A}_c, \mathcal{A}_1, \mathcal{A}_2$ in lines 1-3. The set $\mathcal{A}_c$ contains assignments occurring in both Plans, while the sets $\mathcal{A}_1$ and $\mathcal{A}_2$ contain those assignments which are only in P_1 and P_2, respectively.

In line 4, sets with assignments created in the previous step are converted into corresponding entity pairs. This process is the same as the one described in lines 3-7 in Algorithm 4.1. After that, final sets $\overline{\mathcal{E}_1}$ and $\overline{\mathcal{E}_2}$ containing entity pairs which must be evaluated in constraints are created (see lines 5 and 6). Obviously, set $\overline{\mathcal{E}_1}$ contains entity pairs $\mathcal{E}_1$ created from assignments $\mathcal{A}_1$ which are only in the first Plan. It also contains entity pairs from the set $\mathcal{E}_2 \cap \mathcal{E}_c$, which consists of common pairs derived from sets $\mathcal{A}_2$ and $\mathcal{A}_c$.

The reason why the pairs from the set $\mathcal{E}_2 \cap \mathcal{E}_c$ must be included in the final set of pairs $\overline{\mathcal{E}_1}$ is best illustrated with an example. We will again work with constraint enforcing a maximal number of night shifts per month which accepts driver and current month. For simplicity, the driver is allowed to have only one night shift per month and each additional

Input: Plan P_1, Plan P_2, Evaluator rules $\mathcal{R}$
Output: Plan cost difference
begin
1: $\mathcal{A}_c \leftarrow$ findCommonAssignments(P_1.assignments, P_2.assignments)
2: $\mathcal{A}_1 \leftarrow P_1$.assignments $\setminus$ $\mathcal{A}_c$
3: $\mathcal{A}_2 \leftarrow P_2$.assignments $\setminus$ $\mathcal{A}_c$
4: $(\mathcal{E}_c, \mathcal{E}_1, \mathcal{E}_2) \leftarrow$ convertAssignmentsToEntityPairs$(\mathcal{R}, \mathcal{A}_c, \mathcal{A}_1, \mathcal{A}_2)$
5: $\overline{\mathcal{E}_1} \leftarrow \mathcal{E}_1 \cup (\mathcal{E}_2 \cap \mathcal{E}_c)$
6: $\overline{\mathcal{E}_2} \leftarrow \mathcal{E}_2 \cup (\mathcal{E}_1 \cap \mathcal{E}_c)$
7: $C_1 \leftarrow$ calculateCost$(\mathcal{R}, P_1, \overline{\mathcal{E}_1})$
8: $C_2 \leftarrow$ calculateCost$(\mathcal{R}, P_2, \overline{\mathcal{E}_2})$
9: **return** $C_1 - C_2$
end

Algorithm 4.2: Plan comparison in TASP

night shift is penalized with penalty 100. In both Plans, Mark has night shifts on January 6 and January 12, which means that both Plans are penalized with penalty 100. In the second Plan, Mark has also morning shift on January 9. The morning shift does not change the number of night shifts, so the penalty for both Plans remains 100.

Because the first Plan does not have any unique assignments for Mark in January, we can see that the set $\mathcal{E}_1$ derived from such assignments is empty. If the final set $\overline{\mathcal{E}_1}$ would contain only pairs from the set $\mathcal{E}_1$, our constraint would not be evaluated for the first Plan. This means that the penalty for the first Plan would be 0. On the other hand, the second Plan contains a unique morning shift on January 9. This assignment is then converted into pair (Mark, January) and evaluated in our constraint with penalty 100 caused by the two additional night shifts. Because this constraint is evaluated only for the second Plan, it seems that the Plan cost difference is $0 - 100 = -100$, which is incorrect.

As we know, both Plans should obtain penalty 100 caused by the common assignments and the unique morning shift in the first Plan should not cause any additional costs. For this reason, if some pair exists for unique assignments in the second Plan and also for common assignments, we have to evaluate this pair also for the first Plan and vice versa. This evaluation ensures that the common violations are counted for both Plans and does not influence the cost difference. Contrariwise, if some pair exists only for common assignments, it can be ignored because both Plans would obtain the same cost for this pair.

Finally, cost caused by the selected entity pairs is calculated for both Plans in lines 7 and 8. The resulting cost difference is then returned in line 9.

4.4 Scheduler

Because the TASP framework is designed to solve a wide range of large, NP-complete problems, it is practically impossible to create algorithms which would be able to solve these

problems optimally. For this reason, scheduling module in TASP is designed to combine many simple heuristics which compete to modify the current solution. An adaptive layer based on a roulette wheel selection stochastically decides which heuristics to use. The scheduling progress is then controlled by a local search metaheuristic which decides whether to accept some solution or not. Note that the algorithms presented in this section are derived from the optimization heuristics introduced in Eichler [37].

4.4.1 Local Search Algorithm

Input: Initial Plan P, Evaluator E, Remover R, Inserter I, Set of Listeners $\mathcal{L}$
Output: Best found Plan P^*
begin
1: schedulingStarted($\mathcal{L}, P$)
2: $P^* \leftarrow P$
3: **while** isEnoughTime() **do**
4: $P' \leftarrow R$.removeAssignments(P, E)
5: $P' \leftarrow I$.insertAssignments(P', E)
6: **if** acceptanceProbability(P', P) > random$(0, 1)$ **then**
7: solutionAccepted($\mathcal{L}, P'$)
8: $P \leftarrow P'$
9: **else**
10: solutionRejected($\mathcal{L}, P'$)
11: **if** $P'.Cost < P^*.Cost$ **then**
12: bestSolutionFound($\mathcal{L}, P'$)
13: $P^* \leftarrow P'$
14: **end while**
15: schedulingEnded($\mathcal{L}, P^*$)
end

Algorithm 4.3: Local search algorithm used for scheduling in TASP

The local search algorithm is described in Algorithm 4.3. It starts with the initial Plan, which is then iteratively improved. In each iteration, some assignments are removed from the Plan (see line 4) and reinserted afterward (see line 5). Because different parts of the Plan are recalculated in each iteration, this approach is able to optimize complex problems with many constraints.

In line 6, simulated annealing is used to decide whether to accept the newly created Plan or keep the original one. This decision is influenced by the temperature variable T, which is calculated as

$$T = T_0 \cdot e^{\frac{-t}{n}},$$

where $t \in (0, 1)$ is time progress, n is a normalization constant and T_0 is the initial temperature. This value is calculated such that a solution that is x percent worse than the best-found Plan is accepted with probability 0.5.

Given a solution with cost c, candidate Plan with cost c' is accepted with probability

$$p = \min\left(e^{\frac{c-c'}{T}}, 1\right)$$

In the beginning, when the temperature is high, the candidate Plans are more likely to be accepted even when they are worse than the current solution. This ensures that the scheduling algorithm will be eventually able to jump out of a local optimum. As the temperature is reduced, the acceptance criterion is stricter and the algorithm can focus solely on improving solutions.

Listeners from the set $\mathcal{L}$ are regularly informed about important decisions of the scheduler. Specifically, they are notified when the scheduling started (line 1), after the candidate Plan is accepted or rejected (lines 7 and 10), when a new best solution is found (line 12), and when the scheduling ended (line 15). The user can register an unlimited number of listeners to monitor the decisions of the scheduler. By default, two listeners are used. One of them is a logger which logs the progress of the scheduler and the second one updates score of Removers and Inserters based on their performance (see Subsection 4.4.4).

4.4.2 Remover

Input: Plan P, Metric M, Set of entities to remove $\mathcal{E}$
Output: Plan with removed assignments P'
begin
1: $R \leftarrow$ getRandomAssignment(P)
2: $N \leftarrow$ getAmountToRemove(P)
3: $P' \leftarrow P$
4: **while** $N > 0$ **do**
5: $\quad A \leftarrow$ getNearestAssignment(M, R, P')
6: $\quad \mathcal{A}_{\mathcal{E}} \leftarrow$ getSameEntityAssignments($P', \mathcal{E}, A$)
7: $\quad P' \leftarrow P'$. removeAssignments($\mathcal{A}_{\mathcal{E}}$)
8: $\quad N \leftarrow N-$ size($\mathcal{A}_{\mathcal{E}}$)
9: **end while**
end

Algorithm 4.4: Default Remover in TASP

The default Remover in TASP, described in Algorithm 4.4, selects which assignments to remove based on the given metric M. More precisely, it selects one assignment randomly (see line 1) and then iteratively finds the most similar assignment in the Plan (see line 5). The metric M specified by the user determines the similarity of assignments. It could be based on arrival time, geographical distance, demand, etc.

In line 2, we select the number of assignments which should be removed from the Plan. It is important to remember that simple insertion heuristics generally fail to produce satisfactory

results when the number of removed assignments is too large. Therefore, the Remover should not remove too many assignments from the Plan in order to keep the resulting set of unassigned items small enough for the insertion method. On the other hand, if the number of removed assignments is too small, the scheduler is unable to optimize bigger neighborhoods which reduces the overall quality of the Plan. The optimal amount of removed assignments is problem dependent and have to be selected by the user. Our computational results suggest that it is reasonable to use values in range [10,75].

Sometimes, it is necessary to remove more than one assignment per iteration. For example, we often want to remove all the assignments visited by one vehicle to reduce the number of used vehicles. Our algorithm allows the user to specify which entities in other assignments should be checked when some assignment is selected for removal. In line 6, all the assignments with the same entity values are selected. Note that when the set $\boldsymbol{\mathcal{E}}$ with entities is empty, no other additional assignments are selected and the Remover removes only one assignment in each iteration.

We demonstrate the functionality of our Remover on a special VRP variant where customers can be visited repeatedly. In our case, the metric M is a time difference, we want to remove 2 assignments, the selected entities for removal are customer and vehicle and the Plan contains assignments defined in Table 4.1

Assignment name	Customer	Vehicle	Arrival time
asg1	C_2	V_2	14:00
asg2	C_1	V_2	10:00
asg3	C_1	V_1	12:00
asg4	C_1	V_1	19:00

Table 4.1: Assignments in Plan used for demonstration of Remover functionality.

The first randomly selected assignment is asg2. In the first iteration, asg2 is also selected as the most similar assignment with time difference equal to 0. Because there are no other assignments with the same customer and vehicle, only this one assignment is removed from the Plan. In the next iteration, asg3 with time difference equal to 2 hours is selected for removal. Because both asg3 and asg4 contain the same customer C_1 and vehicle V_1, asg4 is also removed from the Plan, even though it has the highest time difference. After this iteration, the number of removed assignments is 3, which means that no other iterations are needed and we can return a Plan which contains only asg1.

4.4.3 Inserter

By default, TASP uses Inserter which combines eager and greedy insertion techniques. The complete procedure is described in Algorithm 4.5. For eager entities, the Inserter repeatedly selects the first randomly selected item which does not cause infeasible Plan. On the other hand, all the items which exist for greedy entities are checked with the given eager items and the Inserter selects the best combination.

Input: Plan P, Evaluator E, Plan generator G, Eager and greedy entities $\mathcal{E}_{\mathcal{E}}, \mathcal{E}_{\mathcal{G}}$
Output: Plan with inserted assignments P'
begin
1: $P' \leftarrow P$
2: **for all** $E_E \in P'$.getUnassignedItems($\mathcal{E}_{\mathcal{E}}$) **do**
3: $\quad \mathcal{I} \leftarrow \{\}$
4: $\quad$ **for all** $E_G \in P'$.getUnassignedItems($\mathcal{E}_{\mathcal{G}}$) **do**
5: $\quad\quad S \leftarrow$ createPartialAssignment(E_E, E_G)
6: $\quad\quad \mathcal{I} \leftarrow \mathcal{I} \cup G$.findPossibleInsertions($P', S$)
7: $\quad$ **end for**
8: $\quad P' \leftarrow$ getCheapestInsertion($\mathcal{I}, E$)
9: **end for**
end

Algorithm 4.5: Default Inserter in TASP

For example in VRP, the customers might be inserted eagerly and the vehicles greedily. Our Inserter then iteratively selects one customer at random and tries to assign it to all existing routes. After that, the Plan with the cheapest assignment is selected and used in the next iteration. When both the customers and vehicles are inserted greedily, all combinations of these two entities are repeatedly tested to select the best existing pair in each iteration. When both the customers and vehicles are inserted eagerly, first feasible assignment combining randomly selected customer and vehicle is inserted into the Plan, until there are no other feasible assignments (for example when all customers are assigned).

The partial assignment created in line 5 combines both the eager and greedy items into one object. This object is similar to a classical assignment but it is missing some fields without domain, such as arrival time. In line 6, Plan generator receives the partial assignment and produces a new Plan containing the given items. Typically, this method creates a new assignment with a proper arrival time but it is possible to modify the Plan more dramatically when necessary.

4.4.4 Roulette Wheel Selection

It is often beneficial to use more than one Inserter and Remover with different optimization strategies. This mix of approaches usually produces much better results than individual strategies alone. On the other hand, not all the methods are equally useful during the search. Because we want to prioritize the successful heuristics, we employ a roulette wheel selection often used in genetic algorithms. In our implementation, Inserters and Removers are selected independently, which means that we need two separate roulette wheels. Each registered method has its own score based on its success in previous iterations. Candidates with a higher score are then more likely to be selected, but there is always a non-zero chance for all the registered methods.

In each iteration, one Inserter and Remover is selected with probability

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j},$$

where f_i is the fitness of the given method and N is the number of registered methods. At the beginning, uniform weights $f_i = 1/N$ are used. At the end of each iteration, the selected methods are rewarded based on the quality of the produced Plan. After M iterations (usually 100-500), new fitness values are calculated from the obtained score. Given a method i, its score s_i and fitness value f_i, the updated fitness value f_i' can be calculated as

$$f_i' = \rho \frac{s_i}{n_i} + (1 - \rho) f_i,$$

where n_i is the number of times the method has been invoked and ρ is reaction factor which controls how quickly the fitness value reacts to changes in score.

Chapter 5

Algorithms

This chapter thoroughly describes our *Stochastic Customer Satisfaction* (SCS) solver which is able to solve the DSVRPW instances defined in Chapter 3. The solver updates the current Plan whenever a new customer request arrives. It works with a set of possible scenarios which include both the existing and potential future requests sampled from the stochastic information about the problem. The solution which is able to satisfy the biggest number of sampled stochastic customers is then selected as a travel plan for the whole fleet.

This means that we are not selecting a solution with the smallest cost, but the one which is able to cover the biggest number of possible future scenarios. Our strategy is inspired by the *Multiple Scenario Approach* (MSA) introduced by Bent and Hentenryck [34]. When the best solution is selected, the sampled future requests are removed and one of the implemented waiting heuristics is used to distribute the idle time between the remaining customers.

The following text is divided into two parts. Firstly, algorithms used to solve the static version of the problem are described in Section 5.1. In this part, we work with a solver which receives some initial Plan with a fixed set of customers and vehicles and returns an optimized solution after a given number of iterations. This means that the algorithm is unaware of the stochastic and dynamic properties of the original problem. In the following Section 5.2, we describe how the SCS solver must update the dynamic Plan to use it as an input for the static scheduler. We also explain how to decide which of the optimized static Plans to use as a solution for the original dynamic stochastic problem.

5.1 Static Algorithm

In this section, we describe how the general TASP framework (see Chapter 4) was adapted to solve the VRPW problem instances. Firstly, Subsection 5.1.1 is dedicated to techniques which efficiently enforce Plan consistency. The Removers used in our solver are then described in Subsection 5.1.2. Each of them makes use of a different removal neighborhood, targeting various properties of the VRPW Plans. Finally, Inserter whose responsibility is to insert the unassigned customers into the Plan is described in Subsection 5.1.3.

5.1.1 Plan Consistency

Plan in TASP is a collection of assignments with some additional capabilities. In our case, assignment consists of the waypoint (customer or warehouse), vehicle, arrival time, and departure time. As we already mentioned in Chapter 4, TASP has two options how to enforce business rules and consistency of the stored data. In our implementation, we use both ways to achieve the best possible performance. The vehicle capacity is enforced by the Plan itself, while all the other rules are implemented as constraints registered in Evaluator.

Evaluator Constraints

We implemented 4 constraints which are enforcing most of the rules described in Chapter 3. Constraint testing arrival time checks whether there is enough time to travel between two consecutive locations, whether the opening hours are satisfied, and whether the vehicle can be used to serve the given customer. Costs for unassigned customers and used vehicles are calculated simply as a sum of individual costs over corresponding entities. The last constraint calculating the cost for traveled distance iterates through all visited locations of the given vehicle and counts the total traveled distance. All the implemented constraints are summarized in Table 5.1.

Constraint	Tested entity
Arrival time constraint	Assignment
Cost for unassigned customers	Whole Plan
Cost for used vehicles	Vehicle
Cost for traveled distance	Vehicle

Table 5.1: Constraints registered in the Evaluator used to enforce consistency of VRPW Plans.

Vehicle Capacity

In a classical vehicle routing problem, it is rather easy to test whether a vehicle has enough remaining capacity to visit some additional customer. It is sufficient to subtract the demand of each served customer from the initial vehicle capacity and we know precisely how much space remains in the vehicle. Unfortunately, the situation is much more complicated in VRPW, where the vehicles can be replenished in warehouses and where we have to deal with two kinds of customers - those who must be supplied from a warehouse and those whose needs must be satisfied directly from the vehicle depot.

On top of that, our Inserter (see Subsection 5.1.3) needs to know whether a customer can be served in the specified time or whether it is necessary to firstly visit some nearby warehouse. Because we need this information as quickly as possible, it is insufficient to use a classical Evaluator constraint that would need time linearly proportional to the number of visited waypoints.

For this reason, we implemented this constraint directly as a part of the Plan. More precisely, we utilized one of the data consistency capabilities of the Plan (see Subsection 4.2.2) which allows us to monitor a level of allocation for a specified entity. In our case, we registered our

own callback function which is notified whenever some assignment is inserted or removed from the Plan. We then store the remaining vehicle capacity after each warehouse visit and which customers are visited between individual warehouses. This allows us to make decisions about customer insertion in a constant time because it is sufficient to check only the previous warehouse.

5.1.2 Removers

SCS uses 8 different Removers, all based on the universal Remover defined in Subsection 4.4.2. They differ in the metric used for assignment selection and in the set of entities which should be removed together in one iteration. Specifically, the *random vehicle Remover* and the *most expensive vehicle Remover* remove all the assignments associated with the selected vehicle, while the rest of the Removers remove only one assignment per iteration. Note that under the word *metric*, we understand the choosing criterion, not the topological measure.

Random Assignment Remover

Each customer for removal is selected randomly. More precisely, this Remover uses a metric which always returns a random value, ignoring the similarity between assignments. This brings some uncertainty into the search process and helps scheduler to overcome local optima.

Random Vehicle Remover

Remover which selects one vehicle randomly and removes all its customers and warehouses, making the vehicle unused. This helps to decrease the total number of vehicles in the solution. The metric returns random value for each assignment, which ensures randomized selection of assignments for removal. When some assignment is selected, the Remover finds all the assignments with the same vehicle and removes them from the Plan as well.

Distance-Oriented Remover

This Remover is used to remove entire geographical clusters of assignments. For some neighborhoods, the Inserter is prone to insert the removed assignments back into the same route. This Remover allows the Inserter to optimize the whole neighborhood in one step, making it easier to find a new, unexplored solution. It selects the first assignment randomly and measures the distance to all the other existing assignments. After that, the nearest ones are selected and removed from the Plan.

Time-Oriented Remover

Works similarly as the previous Remover, but selects customers based on the arrival time, not geographical distance. This ensures that all the vehicles will be available at the same time and thus the removed customers can be easily interchanged between vehicles.

Most Outlying Assignment Remover

Remover which is designed to remove the most remote visits in the Plan. These assignments are very often misplaced and should be reinserted in another vehicle. For each assignment i, the metric firstly finds preceding visit $i-1$ and following visit $i+1$ (either other assignment or vehicle depot). After that, it counts the remoteness of the assignment i as $r_i = d(i-1, i) + d(i, i+1)$, where the function $d(x, y)$ returns the distance between nodes x and y.

Most Idle Assignment Remover

This Remover is focused on assignments with the longest idle time. While waiting, the vehicle is stationary and does not do any useful work. For this reason, it is desirable to minimize the idle time as much as possible. The metric calculates the waiting time caused by assignment i as $w_i = \psi_i - \omega_i - v_i$, where v_i is visit duration, ω_i is arrival time and ψ_i is departure time.

Most Expensive Vehicle Remover

With this Remover, the most expensive vehicles are completely removed from the Plan. The expensiveness of vehicle v is calculated as $e_v = \frac{d_v + f_v}{n}$, where n is the number of served customers, d_v is traveled distance and f_v is fixed cost for using this vehicle. After that, all the assignments with the most expensive vehicles are iteratively removed from the Plan.

Least Used Warehouse Remover

Remover which tries to remove the least utilized warehouses together with all customers served by them. This minimizes the number of redundant warehouse visits and shortens the overall traveled distance. It uses a metric which counts how many customers is served by each warehouse. Assignments associated with least used warehouses are then removed from the Plan.

5.1.3 Inserter

Our Inserter works similarly as the general Inserter described in Subsection 4.4.3. It assigns customers eagerly and vehicles greedily, but their order is not random. Firstly, the unassigned customers are divided into two groups. The first group contains all the confirmed customers and we try to assign them preferentially. The second group consists of potential future customers which currently do not exist and thus should be assigned with lower priority. For more information about the potential customers and why they are used see Subsection 5.2.2. Note that this is the only place where the static version of the SCS solver has some knowledge about the dynamic problem. We also tried a variant where the customers were not divided, but the algorithm then converged slightly slower.

For each unassigned customer, we try to find an assignment with the smallest insertion cost. Because each customer can be served only by a subset of available vehicles, we firstly exclude all the forbidden ones. After that, we prefer the used vehicles to the ones with no served

customers. When the customer can be assigned to some used vehicle, it is typically not worth to check the unused ones because the additional penalty for a new vehicle is usually much bigger than the travel costs. Only when all the used vehicles are fully loaded, we try to assign the given customer to one of the unused vehicles.

For each vehicle, we try to assign the given customer to all positions in its route. When the vehicle has an insufficient remaining capacity at the given position, we also try to add an additional warehouse visit right before the customer is served. In this situation, the Inserter selects warehouse with the smallest additional travel cost. Of course, it is not always possible to find a suitable nearby warehouse or to satisfy the capacity requirements even when the additional warehouse visit is added. In such cases, this position in the vehicle route is skipped and we check the remaining ones.

Note that the warehouse visits are inserted into the Plan only when the vehicle capacity is exceeded and the given customer can be supplied from a warehouse. This means that our Inserter, without any additional changes, can be immediately used to solve a classical VRP. All we need to do is to specify that all customers should be served from a depot and the SCS solver will automatically produce solutions without warehouse visits.

5.2 Dynamic Algorithm

This section explains how the SCS solver exploits the stochastic knowledge about the customers to anticipate future events and thereby maximize the total number of served customers. Subsection 5.2.1 is dedicated to the overall planning algorithm which handles new customer request arrivals. This method transforms the global dynamic Plan into a static version which can be then used as an input for the static solver covered in Section 5.1. In the next part (see Subsection 5.2.2), we describe how is the new Plan selected from a pool of scenarios which include both known and anticipated future requests. Finally, waiting strategies which distribute the idle time between individual assignments in a vehicle route are described in Subsection 5.2.3. Unlike in a static case, in dynamic problems, the choice of a waiting strategy has a significant impact on the obtained results.

5.2.1 Global Planner

The properties of the global planner are described in Algorithm 5.6. Its main task is to continuously adapt the Plan to accommodate newly arrived customers. Because the algorithm which adapts the Plan uses results obtained from the static solver, it is necessary to firstly modify the global dynamic Plan into a more suitable form. The main problem with the global Plan is that it contains both past and future events. Since the static solver always reshuffles all the given data, the results obtained directly from the global Plan would contain changes in historical actions, which are by its nature immutable. The whole transformation process from the global Plan into its static form is illustrated in Figure 5.1.

Input: Current Plan P, Customer request R, Time T
Output: Updated Plan P^*
begin
1: $\mathcal{V} \leftarrow \{\}$, $\mathcal{A} \leftarrow \{\}$
2: **for all** $V \in P$.vehicles **do**
3: $D_V \leftarrow$ findCurrentLocation(V, P, T)
4: $E_V \leftarrow$ findEarliestStartTime(V, P, T)
5: $\mathcal{V} \leftarrow \mathcal{V} \cup$ updateVehicle(V, D_V, E_V)
6: **end for**
7: **for all** $A \in P$.assignments **do**
8: **if** isFinished(A, T) **then continue**
9: **if** isCustomer$(A$.waypoint$)$ **and** isSupplyLoaded(A, P, T) **then**
10: restrictAllowedVehicles$(A$.customer$)$
11: setSupplyTypeToDepot$(A$.customer$)$
12: $\mathcal{A} \leftarrow \mathcal{A} \cup A$
13: **end for**
14: $P_S \leftarrow$ createStaticPlan$(\mathcal{V}, \mathcal{A}, R)$
15: $P_S^* \leftarrow$ optimizeStaticPlan(P_S, T)
16: $P^* \leftarrow$ updateChangedAssignments(P, P_S^*)
end

Algorithm 5.6: Handling of a new customer request arrival in SCS

Vehicle Modification

Lines 2-6 show how the global planner updates the individual vehicles for the static version of the Plan. The main goal is to change the properties of the vehicles in such a way that they would no longer accept assignments placed in the past. In line 3, the vehicle depot (its start position) is updated to its current location. Note that when the vehicle travels between two nodes, the algorithm selects the location of the target node because we do not allow the vehicles to stop while they are traveling between locations.

The start time of the vehicle is then changed in line 4. There are three possible options based on the current state of the vehicle. Firstly, when the vehicle travels between two locations, we specify its earliest start time to be equal to its arrival time. Secondly, when the vehicle has some work at the given time (i.e. serving some customer or loading goods from a warehouse), the vehicle can start after the job is finished. Lastly, the start time is changed to the current time when the vehicle is waiting at some location. In line 5, we create a new vehicle with updated depot and earliest start time.

Assignment Modification

In the next part of the algorithm (lines 7-13), we specify which assignments the global planner keeps in the static Plan and how they are updated. In line 8, we omit the assignments that already started. Note that this also excludes the assignments which are currently in progress because once some assignment started, it cannot be stopped. The next step is applicable

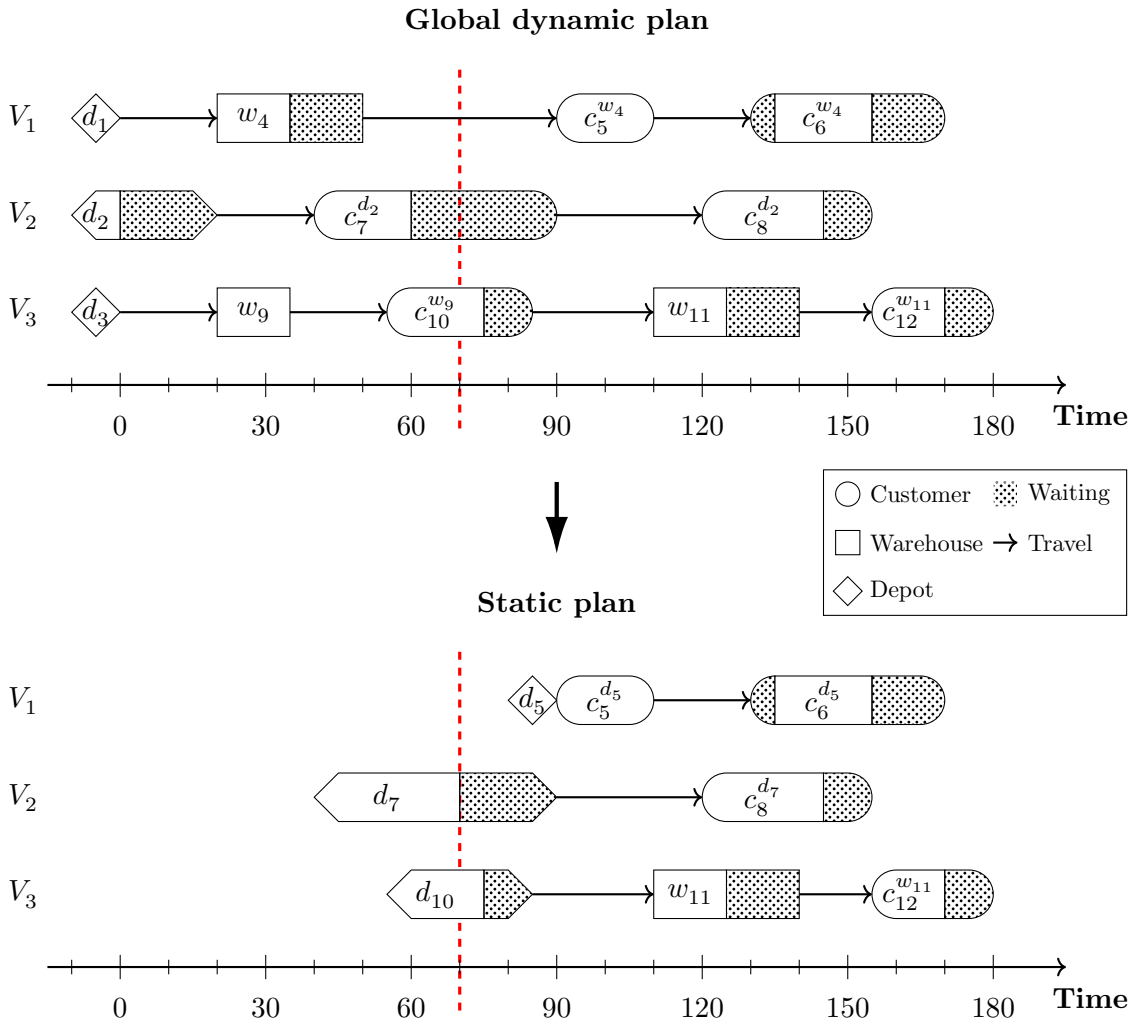


Figure 5.1: Transformation process from the original, global dynamic Plan into a version which can be used as an input for the static solver. The transformation is shown on a route schedule for three vehicles V_1, V_2 and V_3. Diamond (or hexagonal) nodes correspond with depots, circular (or rounded) nodes are customers and rectangular nodes act as warehouses. Parts of the nodes filled with the dotted pattern represent waiting, while the white areas stand for the actual job. The arrows connecting the nodes can be interpreted as travel between two locations. The lower index numbers occurring at node labels define individual locations and the upper indices associated with customers identify node from which is the customer supplied. The red dotted line represents the current time $T = 70$.

As we can see, all three vehicles have different start depot in the static version. Vehicle V_1 starts from the location of the next visited customer c_5 and the earliest start time is changed to $T = 90$. The other two vehicles have their depot moved to their current location. Because in the original Plan, the vehicle V_2 is waiting, its earliest start time is equal to the current time. The other vehicle V_3 has to firstly finish serving the customer, so its earliest start time is changed to $T = 75$. We can also see that both customers served by the vehicle V_1 are now supplied from the depot d_5 instead of the warehouse w_4. Their set of allowed vehicles is also restricted to a single vehicle V_1. These two changes are necessary because the supplies for both customers are already loaded in the vehicle, which means that they cannot be served by another vehicle.

only for customer assignments. If the assignment contains warehouse, it can be inserted into the static Plan without any additional changes.

If the assignment contains customer, we check in line 9 whether the supplies requested by the customer are already loaded in the vehicle or not. When the supplies are present, we must ensure that the customer will be served only by this vehicle because otherwise the loaded supplies would be unused and thrown out. We have to also somehow notify the solver that the vehicle is not empty at the beginning of its route. Note that it is not sufficient to decrease the capacity of the vehicle by the number of loaded goods because once these customers are supplied, the vehicle will have again its original capacity.

The first requirement is addressed in line 10. The set of allowed vehicles for the given customer is restricted to contain only the vehicle transporting the loaded supplies. This step ensures that the customer will not be served by any other vehicle. In line 11, we specify that the customer should be supplied directly from the depot. This effectively solves the second requirement because whenever the customer is served by this vehicle, the solver ensures that the goods are loaded in the depot, which is exactly what we want. Some problems might arise when the Plan returned by the static solver would keep the customer unassigned. Fortunately, this is not a problem in SCS because our algorithm ensures that once a customer request is accepted, it will be always serviced.

When all the vehicles and assignments are updated, we can construct a new Plan that can be used as an input for the static solver in line 14. This Plan is then optimized by methods described in Subsection 5.2.2. Finally, the global Plan is updated based on the optimized static solution in line 16. The whole fleet then follows this Plan until it is updated again for some newly arrived customer request.

5.2.2 Optimization Strategy

Algorithm 5.7 describes how the SCS solver updates the given Plan to accommodate the newly arrived customer. While the obvious option would be to directly use the static solver and return a Plan with the lowest overall cost, this approach would not produce satisfactory results. Because such strategy ignores potential future customer requests, the obtained results tend to be tightly scheduled with very few options for possible future updates. For this reason, our algorithm uses multiple scenarios containing both the real and sampled stochastic customers to determine the most promising strategy for the fleet. This ensures that the produced Plans are better prepared for expected future changes.

Plan Creation

Lines 2-9 describe how the SCS solver finds solutions for individual scenarios. In line 3, we sample a set of potential future customers based on the stochastic knowledge about the problem. The number of samples is calculated from the total expected problem size with respect to the current time. Note that in our implementation, we assume that the probability distribution for each customer is provided as part of the problem definition. If

Input: Current Plan P, Number of scenarios N, Time T
Output: Selected Plan P^*
begin
1: $\mathcal{S} \leftarrow \{\}$, $\mathcal{P} \leftarrow \{\}$
2: **for** $i = 1$ to N **do**
3: $S \leftarrow$ sampleStochasticCustomers(T)
4: $\mathcal{S} \leftarrow \mathcal{S} \cup S$
5: $P' \leftarrow$ scheduleScenario(P, S)
6: **if** size(P'.unassignedCustomers) > 0 **then continue**
7: $P' \leftarrow$ removeSamples(P')
8: $\mathcal{P} \leftarrow \mathcal{P} \cup P'$
9: **end for**
10: **if** isEmpty($\mathcal{S}$) **then return** P
11: $P^* \leftarrow \{\}$, $U^* \leftarrow \infty$
12: **for all** $P' \in \mathcal{P}$ **do**
13: $U \leftarrow 0$
14: **for all** $S \in \mathcal{S}$ **do**
15: $\overline{P} \leftarrow$ insertSampledCustomers(P', S)
16: $U \leftarrow U +$ size(S) - size($\overline{P}$.samples)
17: **end for**
18: **if** $U < U^*$ **then**
19: $U^* \leftarrow U$
20: $P^* \leftarrow P'$
21: **end for**
22: specifyWaitingLocations(P^*)
end

Algorithm 5.7: Plan selection from multiple scenarios in SCS

no such information exists, it is always possible to approximate these distributions from historical data.

The scenario is then scheduled in line 5. The scheduling is done by the static solver introduced in Section 5.1. Because the SCS algorithm guarantees that all the accepted customer requests will be always served, we reject those solutions that contain some unassigned real customers. This means that only the sampled requests are allowed to be unassigned. If there is no scenario where all the confirmed customers are served, the original Plan is returned in line 10. In such a case, the customer request is rejected as unschedulable and we keep the current Plan without changes.

If all the existing customers are successfully scheduled, the added samples are removed from the obtained solution in line 7. This process also removes redundant warehouse visits which were only needed to supply the sampled customers. The algorithm simply checks whether the previous warehouse is able to accommodate all the customers supplied by the examined warehouse and if so, the redundant warehouse is removed. The resulting Plan then consists only from known customers with enough free space for possible future updates.

Plan Selection

When we have solutions for all the sampled scenarios, we need to select one of them as a result. This is done in lines 12-21. Our algorithm tries to select the most universal Plan that is applicable to the biggest number of possible scenarios. This helps us to better respond to future changes because the selected Plan is not too focused only on one scenario. The applicability of a Plan for other scenarios is calculated in line 15. The Inserter described in Subsection 5.1.3 is used to insert the individual samples into the Plan. This method does not change the order of existing assignments, it only updates their arrival times. The whole process is demonstrated in Figure 5.2.

Once all the samples from one scenario are inserted into the Plan, we count how many of them remained unassigned in line 16. Note that because the Inserter inserts the customers iteratively, the number of unassigned samples is affected by their order. Unfortunately, all the deterministic variants of this algorithm would require much longer computational time. On top of that, this issue becomes less significant with increasing number of scenarios where the outliers are suppressed by the remaining results. For this reason, we think that our Inserter ensures a good balance between efficiency and quality of the obtained results.

In line 20, the Plan with the lowest number of unassigned stochastic customers is selected. Before this result can be returned, we have to specify waiting locations for all the vehicles in the Plan. This is done in line 22, where one of the methods described in Subsection 5.2.3 is used. They all change the arrival and departure times of individual assignments to distribute the idle time between visits. This influences the current location of the vehicles when the next dynamic customer request is introduced.

5.2.3 Waiting Heuristics

In static problems, solvers typically do not specify exact arrival and departure times but only calculate when each assignment should start. Because the definition of a static problem remains unchanged throughout the execution of the whole Plan, it does not matter where the vehicle waits between assignments as long as it is able to start the next visit at the specified time. Contrary, each vehicle in dynamic context has to know whether it should wait at the current location or depart to another node because it influences its position when the Plan is recomputed.

A good waiting strategy can significantly help at minimizing travel times needed to serve future customers and hence decrease the overall number of declined requests. In the following text, we present two existing heuristics, namely Drive first and Wait first, which follow a simple deterministic rule. We also introduce two novel heuristics named Scenario waiting and Relocation waiting. These new methods exploit information from scenarios introduced in Subsection 5.2.2 to dynamically calculate where and for how long should each vehicle wait.

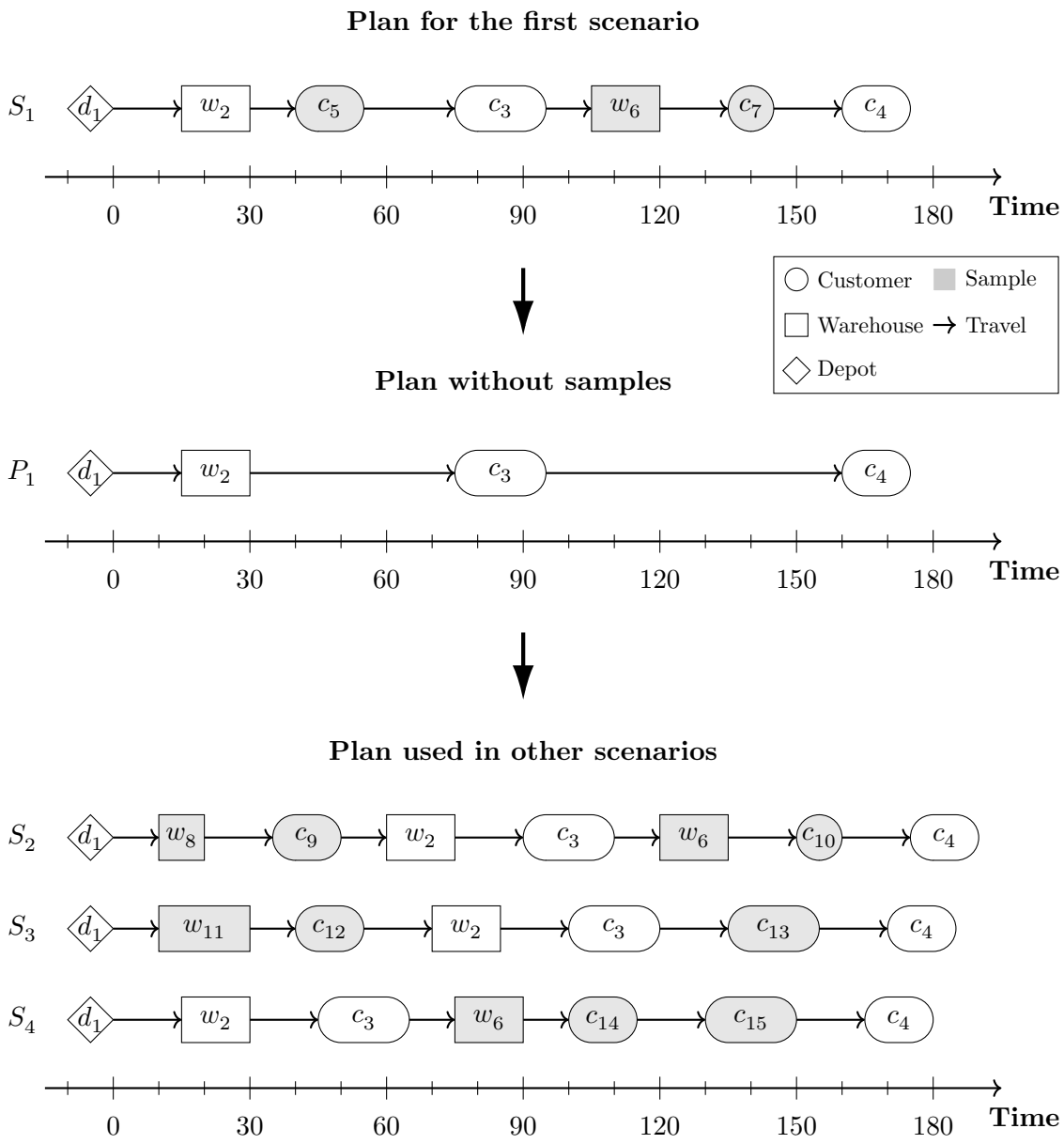


Figure 5.2: The adaptation of one Plan for other scenarios. Diamond nodes correspond with depots, circular (or rounded) nodes are customers and rectangular nodes act as warehouses. White nodes represent real, confirmed customers while the gray nodes symbolize samples or additional warehouse visits needed to serve the samples. The arrows connecting the nodes can be interpreted as travel between two locations. The lower index numbers occurring at node labels define individual locations.

We are presenting the adaptation of Plan S_1 obtained by solving one scenario with a static solver. This Plan is then transformed into Plan P_1 by removing all the samples, keeping only the existing customers and warehouses needed to serve them. In the next step, we try to insert the samples from other scenarios into the given Plan while counting how many of them cannot be assigned. Note that the order of known customers remains unchanged throughout the whole process. This results in Plans S_2, S_3 and S_4. The same procedure is then repeated with Plans for all the other scenarios and we select the one which is able to serve the biggest number of sampled customers.

Drive First

Input: Current Plan P
Output: Plan with defined waiting times P^*
begin
1: $P^* \leftarrow \{\}$
2: **for all** $V \in P$.vehicles **do**
3: **for all** $A \in P$.assignmentsForVehicle(V) **do**
4: $T \leftarrow P$.previousVisit(V, A)
5: A.arrival $\leftarrow T$.departure + travelTime(T.location, A.location)
6: P^*.insert(A)
7: **end for**
8: **end for**
end

Algorithm 5.8: Drive first heuristic

Algorithm 5.8 describes our implementation of Drive first heuristic. Its functionality is quite straightforward – we simply iterate over all vehicles and all their assignments and specify that the vehicle should always leave the current location when the job is finished. In line 4, we find a previous visit (either some other assignment or vehicle depot) for the given assignment. The arrival time is then calculated based on departure from the previous position and travel time between those two nodes in line 5. Note that we do not change the time of the actual job, only the time when the vehicle arrives at each location. Figure 5.3 demonstrates how this heuristic works on a Plan for one vehicle.

Wait First

Input: Current Plan P
Output: Plan with defined waiting times P^*
begin
1: $P^* \leftarrow \{\}$
2: **for all** $V \in P$.vehicles **do**
3: **for all** $A \in P$.assignmentsForVehicle(V) **do**
4: $T \leftarrow P$.nextVisit(V, A)
5: A.departure $\leftarrow T$.arrival - travelTime(A.location, T.location)
6: P^*.insert(A)
7: **end for**
8: **end for**
end

Algorithm 5.9: Wait first heuristic

Wait first heuristic works very similarly as the Drive first heuristic explained in the previous paragraph. The only difference is that the vehicle should stay at the current location as long

as possible. Its implementation is described in Algorithm 5.9. The main differences from the Drive first heuristic can be seen in lines 4 and 5. Firstly, we select the following visit instead of the previous. Departure time is then calculated as a difference between arrival time to the next visit and travel time between the selected locations. All the other properties of the Wait first heuristic are identical with the Drive first heuristic. Its functionality is again illustrated in Figure 5.3.

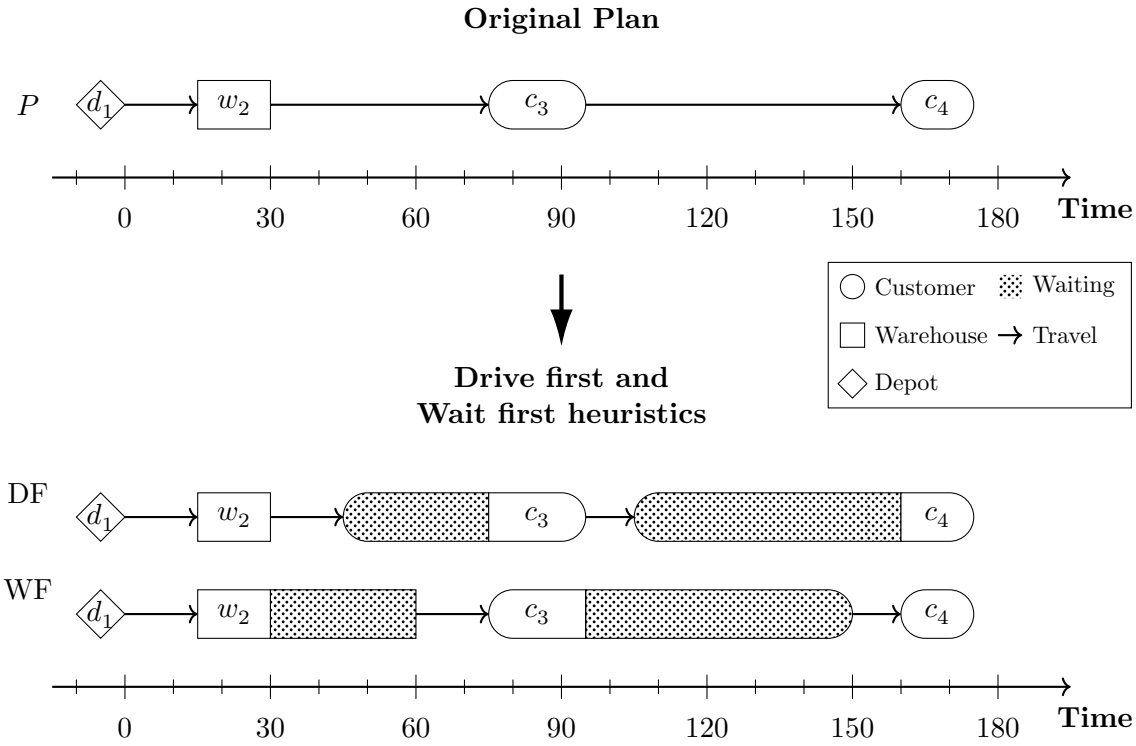


Figure 5.3: Representation of the Drive first (DF) and Wait first (WF) waiting heuristics. Diamond nodes correspond with depots, circular (or rounded) nodes are customers and rectangular nodes act as warehouses. Parts of the nodes filled with dotted pattern represent waiting, while the white areas stand for the actual job. The arrows connecting the nodes can be interpreted as travel between two locations. The lower index numbers occurring at node labels define individual locations.

Both the heuristics do not change the time when the actual job starts in the original Plan (P), they only specify locations where the vehicles will be waiting. In the DF Plan, the vehicle leaves each visited location as soon as possible and waits at the next node before the actual job can start. Contrary, the WF Plan tries to maximally delay the departures from the current location. This means that the vehicle arrives at the next node right when the assignment should start, eliminating waiting after arrival.

Scenario Waiting

Both the heuristics described so far do not incorporate any knowledge about future customer requests into their reasoning. They strictly apply the same rule over and over again, no matter how promising individual locations are. The Scenario waiting heuristic is designed to overcome this issue by utilizing information from scenarios created in Subsection 5.2.2. More precisely, this heuristic works with adaptations of the selected Plan for all existing scenarios (see line 15 in Algorithm 5.7).

Input: Current Plan P, Set of scenarios $\mathcal{S}$ applied to Plan P
Output: Plan with defined waiting times P^*
begin
1: $P^* \leftarrow \{\}$
2: **for all** $V \in P$.vehicles **do**
3: **for all** $A \in P$.assignmentsForVehicle(V) **do**
4: $D \leftarrow \{\}$
5: **for all** $S \in \mathcal{S}$ **do**
6: $A' \leftarrow$ findCorrespondingAssignment(S, A)
7: $D \leftarrow D \cup A'$.departure
8: **end for**
9: A.departure $\leftarrow$ average(D)
10: $T \leftarrow P^*$.previousVisit(V, A)
11: A.arrival $\leftarrow T$.departure + travelTime(T.location, A.location)
12: P^*.insert(A)
13: **end for**
14: **end for**
end

Algorithm 5.10: Scenario waiting heuristic

The whole process is described in Algorithm 5.10. We again iterate over all vehicles and their assignments to specify where and how long should the vehicles wait. In lines 5-8, we calculate the departure time for the given assignment. This is done by finding an average departure time over all scenarios. The arrival time is then calculated in line 11 as the earliest arrival time from the previous updated visit. The whole procedure is demonstrated in Figure 5.4

Because the order of confirmed customers is the same in all scenarios, the average departure time for one assignment is certainly higher than the average departure time for the previous visit and lower than for the next visit. The minimal time difference between two consecutive assignments is in all scenarios always at least equal to the travel time between these two locations. This means that the Plan created by this heuristic will be always feasible with enough time to travel between assignments.

If some assignments are together in a cluster, it is very likely that in each scenario, they all will be visited at a similar time. This does not mean that the departure times will be similar in all the scenarios, in fact, the cluster can be easily visited at the beginning of a route in one scenario and at the end of a route in another one. However, the time difference between the first and the last assignment from one cluster will be small in all scenarios because it would be inefficient to visit the same cluster more than once. This means that in the final Plan returned by this heuristic, all the assignments from a cluster will be visited right after each other without additional waiting and the vehicle will wait only when the whole cluster is completely served.

On the other hand, if some cluster has only a few confirmed customers and a high probability

that other requests will arrive in the future, the time difference between the confirmed assignments will be higher. The reason is that the sampled potential customers will be inserted between the confirmed requests because they are from the same cluster. This results in a bigger average time difference between the first and the last assignment in a cluster, which means that the vehicle will spend a longer time in areas with higher probability of future customer requests.

Coincidentally, both these properties are identified as crucial by authors of a state-of-the-art waiting heuristic presented in Vonolfen and Affenzeller [33]. We thoroughly describe their approach in Subsection 2.3.2. Their algorithm must firstly discretize the service area into spatiotemporal zones and then determine values of four meta-parameters which are problem-dependent. This contrasts with our heuristic with very similar properties which does not need any parameters and only uses scenarios created as a by-product of our optimization strategy.

Relocation Waiting

Input: Current Plan P, Set of scenarios $\mathcal{S}$ applied to Plan P
Output: Plan with defined waiting times P^*
begin
1: $\mathcal{A} \leftarrow \{\}$
2: **for all** $V \in P$.vehicles **do**
3: **for all** $A \in P$.assignmentsForVehicle(V) **do**
4: $\mathcal{N} \leftarrow$ nextAssignment($\mathcal{S}, V, A$)
5: $A' \leftarrow$ getMostFrequentAssignment($\mathcal{N}$)
6: **if** $\mathcal{N}$.numberOfOccurences(A') $<$ size($\mathcal{S}$) / 2 **then continue**
7: $T_N \leftarrow P$.nextVisit(V, A)
8: **if** $A' == T_N$ **then continue**
9: $\mathcal{A}$.add(A')
10: **end for**
11: **end for**
12: P.insertAssignments($\mathcal{A}$)
13: $P^* \leftarrow$ scenarioWaitingHeuristic($P, \mathcal{S}$)
end

Algorithm 5.11: Relocation waiting heuristic

Relocation waiting heuristic is based on the Insertion waiting but further extends the amount of information obtained from the given scenarios. As the name suggests, it allows the vehicle to relocate into a promising new location which is not present in the original Plan. This is especially useful with warehouses, where the vehicle can wait next to a strategically placed warehouse. This improves the time needed to serve a newly arrived customer because it is not necessary to travel to the nearest warehouse for supplies. We illustrate how this heuristic works in Figure 5.4.

Algorithm 5.11 describes the Relocation waiting heuristic in more detail. In line 4, the

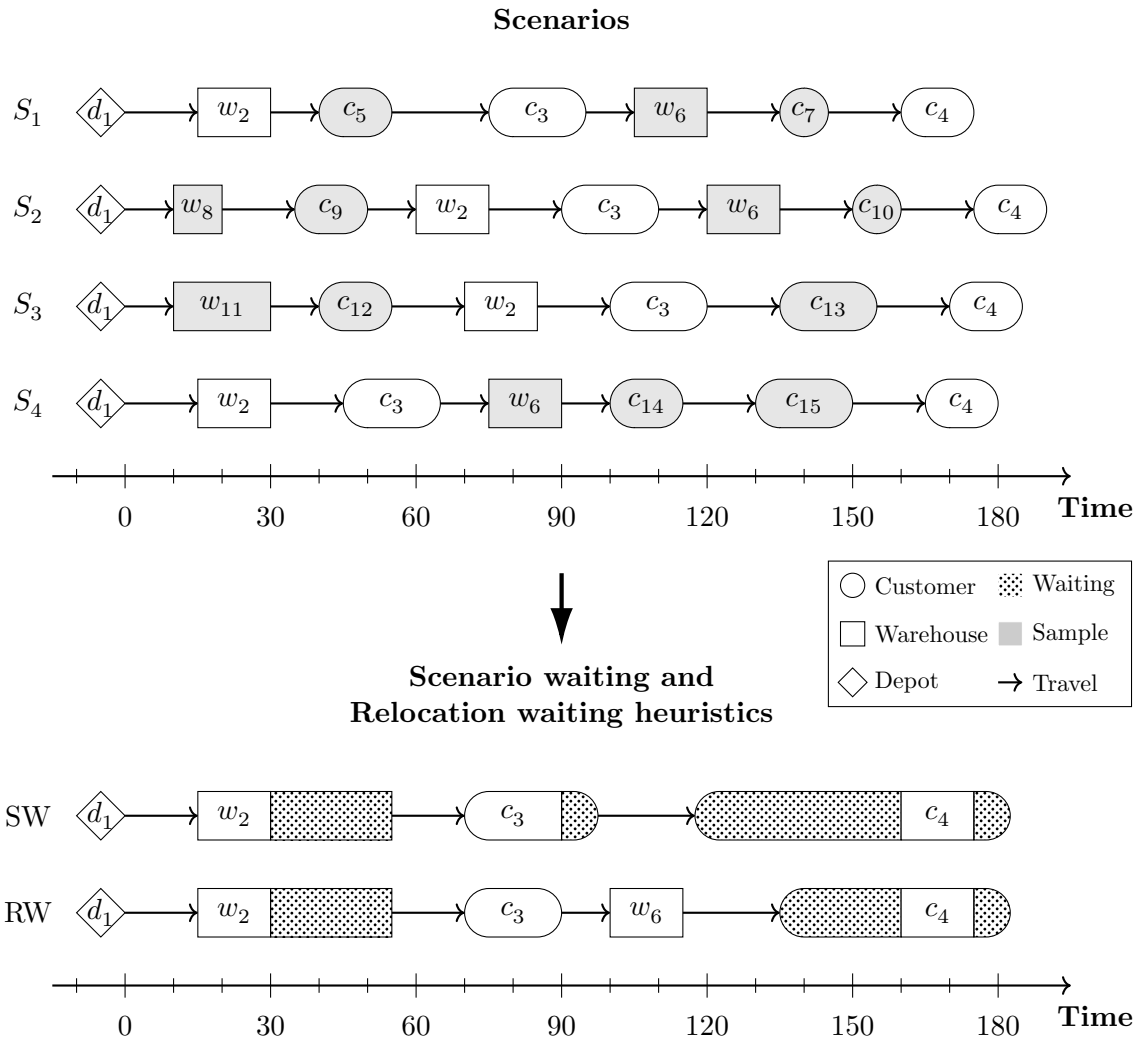


Figure 5.4: Representation of the Scenario waiting (SW) and Relocation waiting (RW) heuristics. Diamond nodes correspond with depots, circular (or rounded) nodes are customers and rectangular nodes act as warehouses. The white nodes, representing the real and confirmed entities, are divided into two parts - the area filled with the dotted pattern illustrates waiting, while the white parts stand for the actual job. The gray nodes symbolize samples or additional warehouse visits needed to serve the samples. The arrows connecting the nodes can be interpreted as travel between two locations. The lower index numbers occurring at node labels define individual locations.

Both the heuristics do not work directly with the original Plan, but with its adaptation to different scenarios S_1–S_4. In the SW Plan, the departure times from each location are calculated as an average departure time in all the scenarios. For warehouse w_2, the departures are $T = 30, 75, 85, 30$, which result in average departure $T = 55$. The arrival times are then easily calculated from the travel times between locations and the actual jobs start immediately after the time window is opened.

The RW Plan is constructed very similarly as the SW Plan, but it is possible to add additional waiting locations. More precisely, if some pair of nodes is visited in more than half of the scenarios, the sampled assignment is kept in the resulting Plan. In our case, this happens with nodes $c_3 \rightarrow w_6$ occurring in scenarios S_1, S_2 and S_4. The waiting times for nodes neighboring with the newly added assignment then can be reduced to allow the additional visit. This can be seen in node c_3, where the vehicle leaves immediately after the customer is visited and in node c_4, where the vehicle arrives later.

algorithm finds assignments from all the scenarios which are visited right after the current assignment. The assignment occurring in the biggest number of scenarios is then selected in line 5. In lines 6-8, we test if it is worth to add the selected assignment as an additional visit into our Plan. We firstly check whether it is visited in the majority of scenarios. This condition should eliminate assignments which are useful only for a fraction of scenarios and otherwise might be disadvantageous. The condition in line 8 then excludes assignments which are already present in the original Plan.

All assignments that satisfy both conditions are then inserted into the Plan in line 12. This insertion is different for customers and warehouses. Since all the warehouses are known before the algorithm starts, it is possible to add the assignment directly as it is. On the other hand, the sampled customers officially do not exist, so we cannot insert them into the Plan. This issue is resolved by introducing dummy customers whose visit duration is 0, which effectively means that the vehicle only waits at the given location. Because these waiting locations are only useful for the global planner to determine the location of each vehicle (see Subsection 5.2.1), the dummy customers are removed from the Plan during the transformation into its static form.

Finally, the arrival and departure times are calculated in line 13. This method works almost identically as the Insertion waiting heuristic presented in the previous section. The only difference is that it is no longer true that the time difference between two assignments will be always at least equal to the travel time between them. Because the sampled assignments do not occur in all scenarios, there might be not enough time to travel between the nodes when the average departure times are followed. If it is possible, we try to reduce the waiting times of the previous and next assignment. Only when there is still not enough time for the visit, the additional waiting location is simply not inserted into the Plan.

Chapter 6

Experiments

In this chapter, we explore how the SCS solvers with different waiting heuristics handle various types of synthetic problem instances. In Section 6.1, we describe properties of the DSVRP and DSVRPW benchmarks used in our experiments. All these instances were derived from the classical static VRP test cases and adapted to the dynamic context. In Section 6.2, we compare the performance of our algorithms with the state-of-the-art solvers presented in Bent and Hentenryck [34] and Guillain et al [36] on the DSVRP benchmark instances. We also present our results obtained on the DSVRPW test cases. In the last section, we analyze the produced solutions and compare differences between the algorithms. We examine which factors influence rejection of customer requests and how the warehouse assignments affect the obtained schedules. For more information, see Section 6.3.

6.1 Testing Data

Throughout the whole chapter, we work with dynamic problems derived from the classical Solomon's static VRP benchmark instances [40]. All 56 problems consist of 100 customers and one depot with 25 vehicles. The instances can be divided into 6 sets based on the geographical distribution and vehicle capacity. Bent and Hentenryck [34] adapted these test cases for the dynamic context by transforming each customer into a region from which the dynamic requests are sampled. The expected number of customers for each instance is again 100, which means that the expected number of samples from each region is 1. This ensures that the adapted instances will have a very similar structure as the original test cases.

Bent and Hentenryck [34] created in total 60 DSVRP instances [41]. The problems are divided into 4 classes of 15 test cases characterized by the number of dynamic customers. The differences between individual classes are explained in Subsection 6.1.2. Each class then consists of 3 problem types with diverse time windows. We also created additional 60 DSVRPW benchmark instances by randomly adding 10 warehouses into the original Bent's test cases. The complete set of properties specifying the instances used in our experiments is described in Subsection 6.1.1.

6.1.1 Static Properties

The locations of customers are derived from Solomon's RC1 class. These problems contain both the randomly sampled and clustered customers and have low vehicle capacity allowing to visit at most 5-10 customers. All the problem instances used in our experiments have the same locations of customers, warehouses, and vehicle depots. Their placement can be seen in Figure 6.1. Note that the warehouses are only available in the newly created DSVRPW test cases, not in the original Bent's benchmarks.

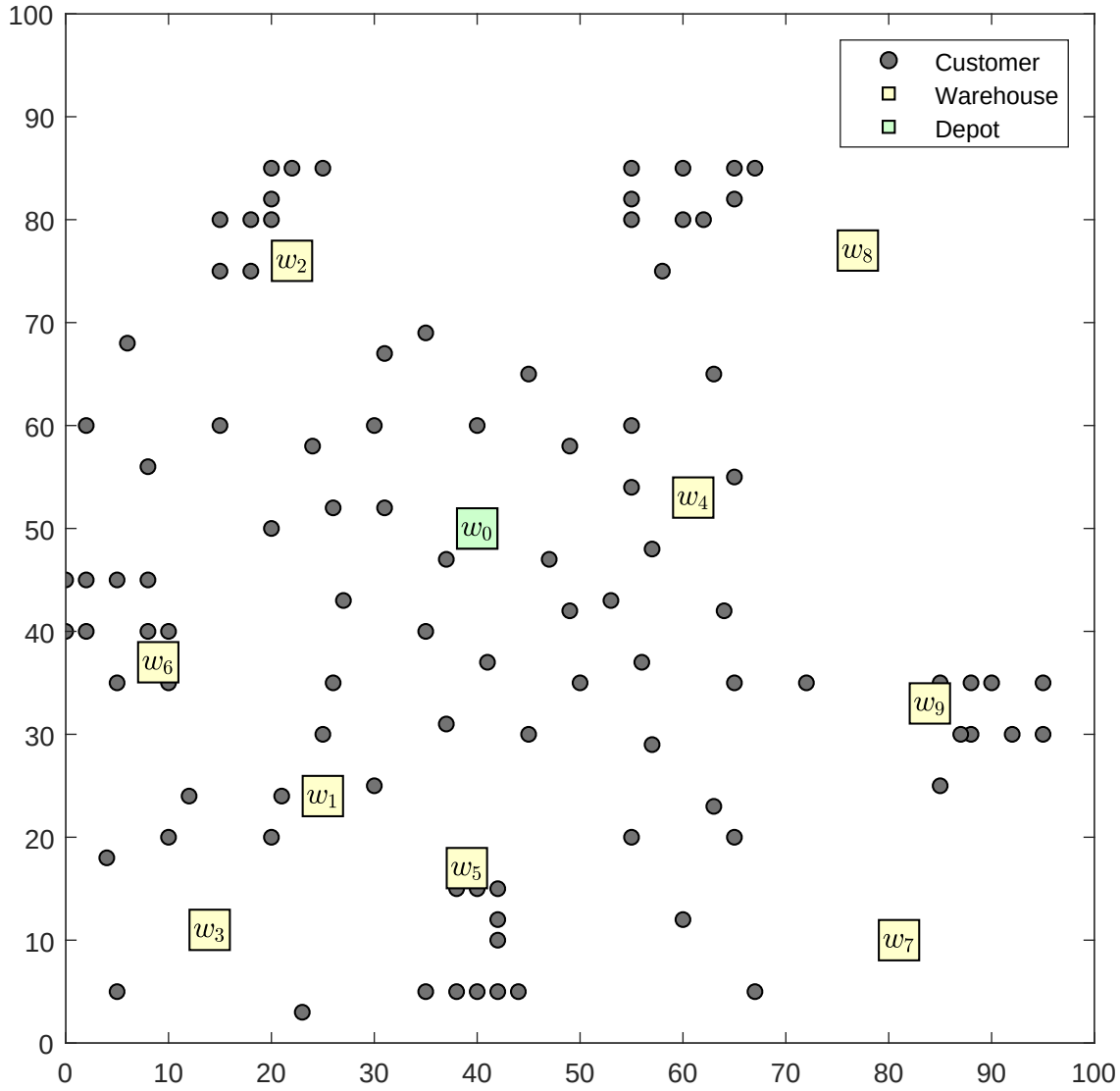
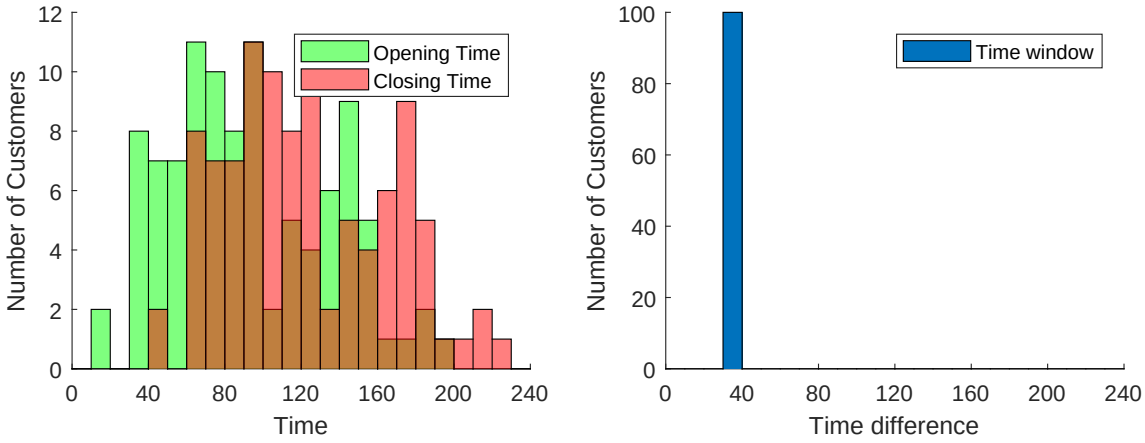
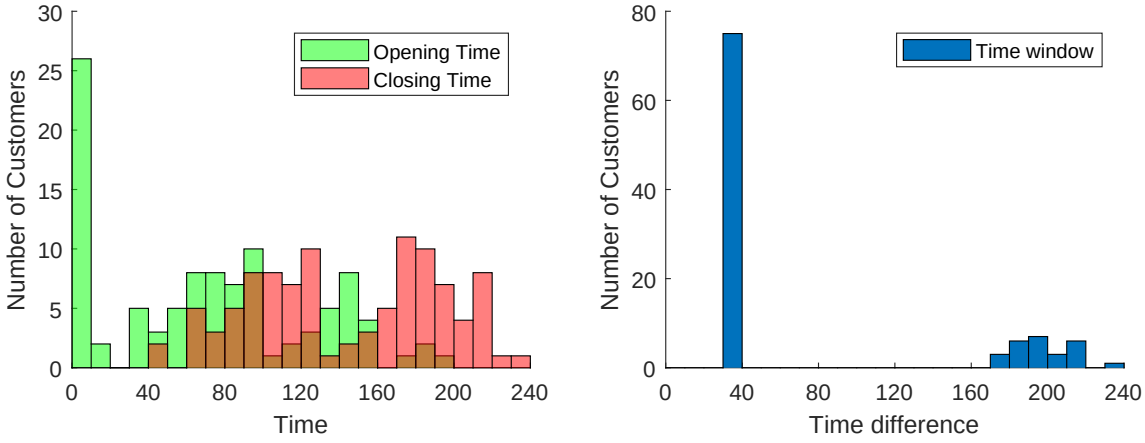


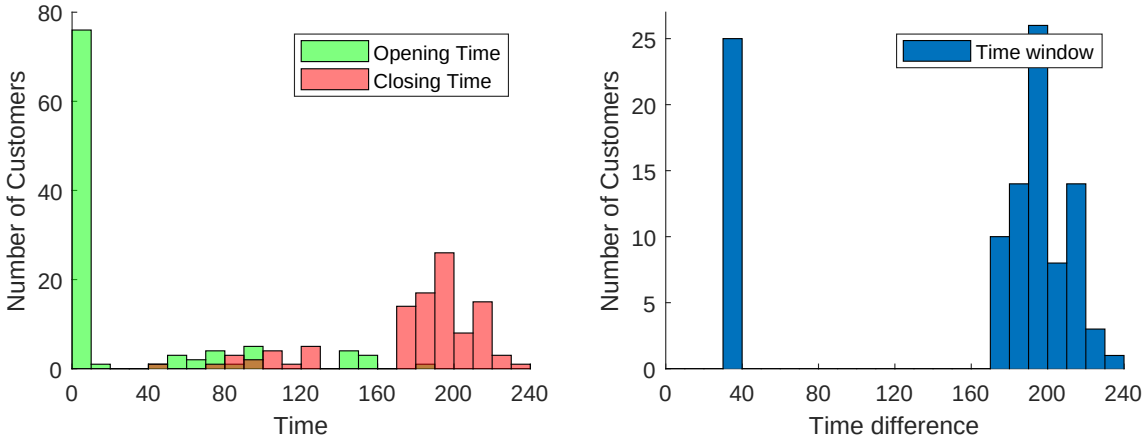
Figure 6.1: Locations of customers, warehouses, and vehicle depots for all problem instances used in our experiments. Note that the position $(40, 50)$ is shared both by the vehicle depot and warehouse w_0, which allows the vehicle to supply customers near the depot without additional travel to some distant warehouse.



(a) RC101 instances



(b) RC102 instances



(c) RC104 instances

Figure 6.2: (Left) Histograms of opening and closing times for different problem types. (Right) Histograms of time window lengths for different problem types. This value is calculated as a difference between the closing and opening time.

Bent's benchmarks consist of 3 problem types with diverse time windows. The RC101 instances have customers with the opening times regularly distributed throughout the scheduling horizon. The length of each time window in this instance is equal to 30. The RC102 instances have more than one-quarter of customers with the opening time equal to 0 and with a very long time window. The remaining customers have similar opening hours as the customers in the RC101. Finally, the RC104 instances have more than three-quarters of customers with a very long time window and the opening time equal to 0. Even though it is technically possible to specify time windows also for the warehouses, we decided that the warehouses in the newly created DSVRPW test cases will be always open. Histograms showing the distribution of time windows for all the problem types can be seen in Figure 6.2.

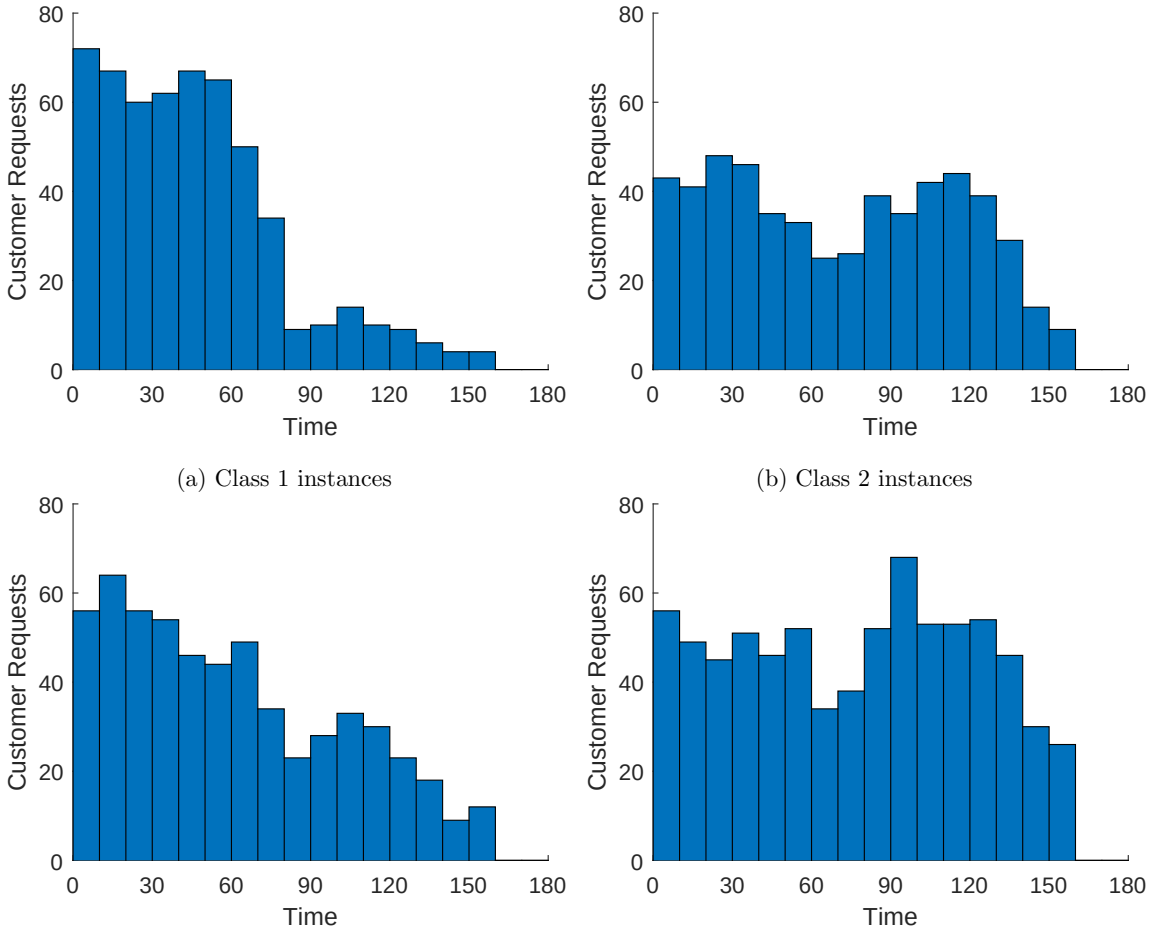
In the original Solomon's problems, all the instances were defined with 25 vehicles. This number was reduced in Bent's test cases to ensure that the solver will be forced to reject some of the dynamic customer requests. More precisely, the authors firstly solved the offline instances (i.e. all dynamic requests are known in advance) with a state-of-the-art static solver. Then the number of used vehicles was increased by 2 to compensate the higher difficulty of the online problem. This reduced the number of available vehicles from 25 to 12–17.

6.1.2 Request Arrivals

Bent's benchmarks are divided into 4 classes with different degree of dynamism (DOD), which specifies the ratio of requests revealed at time $t > 0$ over the total number of customers. The classes also differ in time when the dynamic requests are introduced. The scheduling horizon $H = 240$ is divided into four periods. Period 0 represents the customers known before the scheduling starts. All the dynamic customer requests are revealed during the first period in time $t \in [1, 80]$ or during the second period in time $t \in [81, 160]$. There are no requests revealed during the last period in time $t \in [161, 240]$ to guarantee that the vehicle is theoretically able to serve all customer requests and then return to its depot.

Class 1 instances have many known customers from period 0, many early requests from period 1 and only a few late requests from period 2. Problems from class 2 have again many known customers from period 0 and a similar number of requests from periods 1 and 2. Class 3 is a mix of classes 1 and 2. Finally, class 4 is similar to class 2 but the number of known customers from period 0 is lower, which results in a higher number of dynamic customer requests in periods 1 and 2. The average DOD of classes 1 and 2 is 39%, class 3 has 42% and class 4 has 54%. Figure 6.3 summarizes the arrival times for all the classes.

For each class and problem type, 5 different instances were generated using probabilities specified for each customer region and time period. If some customer request should appear in time period i, then the exact arrival time of the request is drawn uniformly from time interval $[(i-1) \cdot H/3, \min(t_{d,c} + v_c + t_{c,d}, i \cdot H/3 - 1)]$, where $t_{d,c}$ is a travel time from the vehicle depot to the customer from the request, v_c is a visit duration and $t_{c,d}$ is a travel time back to the depot. Note that all these probabilities are known to the scheduler.



(a) Class 1 instances

(b) Class 2 instances

(c) Class 3 instances

(d) Class 4 instances

Figure 6.3: Histograms showing the distribution of arrival times of customer requests for different classes of problems.

6.2 Computational Results

In this section, we examine how the proposed SCS solver with different waiting heuristics performs on the DSVRP and DSVRPW benchmark instances. In the case of the DSVRP test cases, the results obtained with the SCS solver are also compared with two state-of-the-art methods introduced in Bent and Hentenryck [34] and Guillain et al [36]. The primary objective of all the test cases is to minimize the number of rejected customer requests and the secondary objective is to minimize the number of used vehicles.

6.2.1 Algorithms

All our algorithms were implemented in Kotlin 1.2 and compiled into Java 8 compatible bytecode. All the solvers also used the same static scheduler defined in Section 5.1 whenever they needed to optimize some Plan. The number of iterations for one run of the static

scheduler was set to 1500. This value provides a good compromise between computational time (2–3 seconds) and quality of the obtained results. All computations were performed on the National Grid Infrastructure MetaCentrum which operates and manages distributed computing infrastructure within the Czech Republic. Average results over 5 runs are reported.

Stochastic Customer Satisfaction

We consider 4 variants of the SCS solver with different waiting strategies. The SCS^{DF} and SCS^{WF} represent simple variants with Drive first and Wait first heuristics. The SCS^{SW} stands for Scenario waiting strategy which utilizes information from the potential future scenarios. Finally, the SCS^{RW} symbolize Relocation waiting heuristic. It uses the same strategy as the SCS^{SW} but allows the vehicles to relocate into promising new locations. In all variants, we generated and solved 32 potential future scenarios for each newly arrived customer request. Because all the scenarios are independent, we used 8 threads to reduce the overall computation time. On average, the whole insertion procedure for one customer request then took less than 12 seconds (i.e. 96 seconds with one thread).

Greedy Algorithm

To the best of our knowledge, this is the first work which considers the DSVRPW problems. For this reason, we use the Greedy algorithm as a baseline approach for this type of problems. Whenever a new customer request is created, the Greedy algorithm recomputes the current Plan and finds a solution with the lowest cost. This means that the stochastic information about future requests is neglected and the next Plan is selected based on the number of used vehicles and the total travel cost. Because we do not consider different scenarios, only one run of the static scheduler is needed to insert the newly arrived customer request. This means that the Greedy Algorithm needs only 2–3 seconds for one customer request.

Multiple Scenario Approach

The MSA solver was introduced in Bent and Hentenryck [34]. At the beginning, their algorithm creates a pool of 50 plans for existing customers and possible future requests, where each plan is optimized for 30 seconds. Every time unit, MSA adds one new solution into the pool by using local search algorithm for 10 seconds and selects one distinguished plan from the pool. This plan then determines the movement of vehicles to guarantee service of accepted requests. After that, the plans incompatible with the selected one are removed from the pool. The authors tested two ranking functions used to select the distinguished plan. They showed that the MSA with Consensus function (MSA^{C}) gives better results. This function selects a plan most similar to other plans in the pool.

Global Stochastic Assessment

The GSA solver was introduced in Guillain et al [36]. Their algorithm works very similarly as the MSA solver. The biggest difference is that the GSA does not select the distinguished plan from a pool of solutions but directly creates only one solution that best suits a pool of scenarios. This means that the GSA solver does not need any ranking function to select

the distinguished plan. Several waiting strategies were proposed and it was shown that the GSA with Relocation-only waiting (GSA^{RO}) produces the best results. This strategy selects waiting locations (denoted as relocation requests) where the vehicle waits as long as possible. For all the other locations, the Drive first heuristic is applied. The authors used 60 minutes of offline computation to find an initial solution and then allowed 4 seconds of online computation per time unit.

Comparison of Computation Times

Because the MSA and GSA solvers use a different strategy than our SCS solver, it is not possible to directly compare the computation time needed for each approach. While the SCS solver changes the current Plan only when some new customer request is revealed, the MSA and GSA solvers need long offline time to schedule the known customers and then generate one new plan each time unit within the time horizon. For this reason, we are only able to compare the average computation time needed to solve the whole test case. On average, one instance has 44 dynamic customer requests and the planning horizon is 240 time units. The overall computation times needed for each method are summarized in Table 6.1. We can see that the Greedy algorithm is much faster than the other approaches which need almost the same time for an average instance.

Method name	Offline (s)	Step (s)	Steps	Overall (s)
SCS	96	96	44	4320
Greedy	3	3	44	135
MSA	1500	10	240	3900
GSA	3600	4	240	4560

Table 6.1: Average time needed to solve one test case with different methods. Column *offline* represents time in seconds needed to create an initial solution for known customers. The next two columns show how many seconds are reserved for one step of the algorithm and how many steps the algorithm makes to solve one average test case. The last column then presents the overall computation time in seconds needed for one test case. This value is calculated as: $overall\ time = offline\ time + step\ time * steps$.

6.2.2 Bent's Benchmarks

Tables 6.2 and 6.3 summarize the obtained results on Bent's DSVRP instances. We compare average solution quality produced by the MSA solver with consensus function (MSA^{C}), GSA solver with Relocation-only waiting strategy (GSA^{RO}), SCS solvers with Drive first (SCS^{DF}) and Wait first (SCS^{WF}) heuristics and SCS solver with Scenario waiting strategy (SCS^{SW}). The first number in each column shows how many customer requests were rejected and the second one represents the number of used vehicles. Because Guillain et al [36] presented only the average number of rejected customer requests, we cannot show how many vehicles were used in their solutions.

Note that we do not present results from the SCS solver with Relocation waiting strategy because the additional waiting locations were almost never used. This means that the obtained solutions were practically identical with the results produced by the SCS^{SW} solver. This is

Problem	MSA^{C}		GSA^{RO}		SCS^{DF}		SCS^{WF}		SCS^{SW}	
instance	Uns.	Veh.	Uns.	Veh.	Uns.	Veh.	Uns.	Veh.	Uns.	Veh.
C1-RC101-1	0.6	16.0	1.2	-	**0.2**	**15.8**	0.8	16.0	0.0	16.0
C1-RC101-2	2.6	16.0	1.6	-	0.4	16.0	0.4	16.0	**0.2**	**16.0**
C1-RC101-3	1.0	15.0	1.6	-	**0.0**	**15.0**	0.4	15.0	**0.0**	**15.0**
C1-RC101-4	0.2	17.0	1.1	-	**0.0**	**16.8**	0.0	17.0	**0.0**	**16.8**
C1-RC101-5	1.0	17.0	2.2	-	**0.0**	**17.0**	**0.0**	**17.0**	**0.0**	**17.0**
C1-RC102-1	2.4	14.0	1.5	-	0.2	14.0	0.6	14.0	**0.0**	**14.0**
C1-RC102-2	0.8	13.0	0.8	-	**0.0**	**13.0**	**0.0**	**13.0**	**0.0**	**13.0**
C1-RC102-3	0.8	15.0	0.8	-	**0.0**	**15.0**	**0.0**	**15.0**	**0.0**	**15.0**
C1-RC102-4	1.4	14.0	0.5	-	**0.0**	**14.0**	**0.0**	**14.0**	**0.0**	**14.0**
C1-RC102-5	0.6	15.0	0.1	-	**0.0**	**15.0**	**0.0**	**15.0**	**0.0**	**15.0**
C1-RC104-1	0.2	11.0	0.4	-	**0.0**	**11.0**	0.2	11.0	**0.0**	**11.0**
C1-RC104-2	**0.0**	**12.0**	**0.0**	-	**0.0**	**12.0**	0.2	12.0	**0.0**	**12.0**
C1-RC104-3	**0.0**	**13.0**	**0.0**	-	**0.0**	**13.0**	**0.0**	**13.0**	**0.0**	**13.0**
C1-RC104-4	0.2	12.0	0.2	-	**0.0**	**12.0**	**0.0**	**12.0**	**0.0**	**12.0**
C1-RC104-5	**0.0**	**11.0**	**0.0**	-	**0.0**	**11.0**	**0.0**	**11.0**	**0.0**	**11.0**
Class 1 Avg	0.79	14.07	0.80	-	0.05	14.04	0.17	14.07	**0.01**	**14.05**
C2-RC101-1	0.2	13.0	1.5	-	**0.0**	**13.0**	1.6	13.0	**0.0**	**13.0**
C2-RC101-2	1.4	14.0	2.1	-	**0.0**	**14.0**	**0.0**	**14.0**	**0.0**	**14.0**
C2-RC101-3	**0.0**	**17.0**	2.3	-	**0.0**	**17.0**	0.4	17.0	**0.0**	**17.0**
C2-RC101-4	0.8	17.0	2.7	-	0.2	17.0	0.6	17.0	**0.0**	**17.0**
C2-RC101-5	1.4	16.0	2.1	-	**0.0**	**16.0**	0.4	16.0	**0.0**	**16.0**
C2-RC102-1	0.4	15.0	0.4	-	**0.0**	**15.0**	0.2	15.0	**0.0**	**15.0**
C2-RC102-2	1.2	14.0	0.8	-	**0.0**	**14.0**	0.2	14.0	**0.0**	**14.0**
C2-RC102-3	2.0	14.0	1.0	-	**0.0**	**14.0**	1.0	14.0	**0.0**	**14.0**
C2-RC102-4	0.4	15.0	0.8	-	**0.0**	**15.0**	0.2	15.0	**0.0**	**15.0**
C2-RC102-5	2.8	14.0	1.3	-	**0.0**	**14.0**	**0.0**	**14.0**	**0.0**	**14.0**
C2-RC104-1	3.0	12.0	0.1	-	**0.0**	**12.0**	0.4	12.0	0.2	12.0
C2-RC104-2	2.6	12.0	0.6	-	**0.0**	**12.0**	0.4	12.0	0.2	12.0
C2-RC104-3	0.8	12.0	**0.0**	-	**0.0**	**12.0**	**0.0**	**12.0**	**0.0**	**12.0**
C2-RC104-4	0.6	13.0	**0.0**	-	**0.0**	**13.0**	0.6	13.0	**0.0**	**13.0**
C2-RC104-5	0.2	12.0	0.1	-	**0.0**	**12.0**	1.6	12.0	**0.0**	**12.0**
Class 2 Avg	1.19	14.00	1.05	-	**0.01**	**14.00**	0.51	14.00	0.03	14.00

Table 6.2: Solutions on classes 1 and 2 from the Bent's DSVRP benchmark instances [41]. Each column contains two numbers. The first one represents the average number of unassigned customer requests and the second one shows the average number of used vehicles. Bold results highlight the best result in each line. The first two columns contain the best average results obtained by Bent and Hentenryck [34] and by Guillain et al [36], respectively. The last three columns show average results obtained by our SCS solver over 5 runs. Each column represents different waiting heuristic - Drive first (DF), Wait first (WF) and Scenario waiting (SW).

Problem instance	MSA^{C}		GSA^{RO}		SCS^{DF}		SCS^{WF}		SCS^{SW}	
	Uns.	Veh.	Uns.	Veh.	Uns.	Veh.	Uns.	Veh.	Uns.	Veh.
C3-RC101-1	0.8	15.0	1.8	-	**0.0**	**15.0**	0.6	15.0	**0.0**	**15.0**
C3-RC101-2	1.4	16.0	1.2	-	**0.0**	**16.0**	1.0	16.0	**0.0**	**16.0**
C3-RC101-3	0.8	14.0	1.5	-	**0.0**	**13.8**	0.0	14.0	0.0	14.0
C3-RC101-4	1.0	17.0	1.4	-	**0.0**	**16.8**	**0.0**	**16.8**	0.0	17.0
C3-RC101-5	0.8	16.0	0.7	-	**0.0**	**16.0**	**0.0**	**16.0**	**0.0**	**16.0**
C3-RC102-1	1.6	15.0	1.0	-	**0.0**	**15.0**	0.4	15.0	**0.0**	**15.0**
C3-RC102-2	1.8	14.0	0.8	-	**0.0**	**14.0**	0.4	14.0	**0.0**	**14.0**
C3-RC102-3	0.8	13.0	0.5	-	**0.0**	**13.0**	0.6	13.0	**0.0**	**13.0**
C3-RC102-4	1.8	15.0	0.4	-	0.2	15.0	**0.0**	**15.0**	**0.0**	**15.0**
C3-RC102-5	1.6	15.0	1.1	-	0.0	15.0	0.2	15.0	**0.0**	**14.8**
C3-RC104-1	2.4	12.0	0.3	-	1.0	12.0	0.8	12.0	**0.2**	**12.0**
C3-RC104-2	0.2	12.0	0.1	-	**0.0**	**12.0**	0.2	12.0	**0.0**	**12.0**
C3-RC104-3	0.4	12.0	**0.0**	-	**0.0**	**12.0**	0.2	12.0	**0.0**	**12.0**
C3-RC104-4	0.2	12.0	**0.0**	-	**0.0**	**12.0**	1.4	12.0	**0.0**	**12.0**
C3-RC104-5	0.6	12.0	**0.0**	-	**0.0**	**12.0**	1.6	12.0	**0.0**	**12.0**
Class 3 Avg	1.08	14.00	0.72	-	0.08	13.97	0.49	13.99	**0.01**	**13.99**
C4-RC101-1	1.0	16.0	1.2	-	**0.0**	**16.0**	**0.0**	**16.0**	**0.0**	**16.0**
C4-RC101-2	3.6	15.0	1.6	-	**0.0**	**15.0**	**0.0**	**15.0**	**0.0**	**15.0**
C4-RC101-3	1.6	16.0	0.2	-	**0.0**	**16.0**	0.6	16.0	**0.0**	**16.0**
C4-RC101-4	1.4	17.0	1.1	-	**0.0**	**17.0**	0.4	17.0	**0.0**	**17.0**
C4-RC101-5	2.2	16.0	3.5	-	**0.0**	**16.0**	1.4	16.0	**0.0**	**16.0**
C4-RC102-1	0.4	15.0	0.1	-	**0.0**	**15.0**	0.2	15.0	**0.0**	**15.0**
C4-RC102-2	1.4	15.0	0.2	-	**0.0**	**15.0**	1.2	15.0	**0.0**	**15.0**
C4-RC102-3	1.4	15.0	0.5	-	**0.0**	**14.8**	1.8	15.0	0.0	15.0
C4-RC102-4	**0.0**	**14.0**	0.1	-	**0.0**	**14.0**	**0.0**	**14.0**	**0.0**	**14.0**
C4-RC102-5	0.6	15.0	1.5	-	**0.0**	**14.8**	0.2	15.0	0.0	15.0
C4-RC104-1	3.2	13.0	**0.7**	-	2.2	13.0	2.8	13.0	1.6	13.0
C4-RC104-2	3.4	14.0	**0.0**	-	0.6	14.0	1.4	14.0	0.4	14.0
C4-RC104-3	5.6	13.0	**0.5**	-	1.6	13.0	1.6	13.0	1.0	13.0
C4-RC104-4	2.4	12.0	**0.4**	-	0.8	12.0	0.8	12.0	1.0	12.0
C4-RC104-5	2.0	11.0	**0.7**	-	2.0	11.0	1.0	11.0	1.0	11.0
Class 4 Avg	2.01	14.47	0.82	-	0.48	14.44	0.89	14.47	**0.33**	**14.47**

Table 6.3: Solutions on classes 3 and 4 from the Bent's DSVRP benchmark instances [41]. Each column contains two numbers. The first one represents the average number of unassigned customer requests and the second one shows the average number of used vehicles. Bold results highlight the best result in each line. The first two columns contain the best average results obtained by Bent and Hentenryck [34] and by Guillain et al [36], respectively. The last three columns show average results obtained by our SCS solver over 5 runs. Each column represents different waiting heuristic - Drive first (DF), Wait first (WF) and Scenario waiting (SW).

expected behavior because the Relocation waiting strategy was designed to allow the vehicles to wait next to a strategically placed warehouse. Since there are no warehouses in DSVRP problems and the sampled set of possible future customer requests is often very different in each scenario, it is much more difficult to find an assignment which is visited in the majority of scenarios.

Our algorithms were able to produce the best results in 55/60 cases. The SCS^{SW} solver had the lowest number of rejected customer requests on classes 1, 3, 4 and the SCS^{DF} solver on class 2. Also, both the SCS^{DF} and SCS^{SW} solvers were able to find at least one solution without any rejected customer requests on all test cases. Overall, both algorithms produced very similar results in almost all instances. The SCS^{SW} has a slightly lower number of rejected customer requests while the SCS^{DF} was able to slightly better reduce the number of used vehicles. But it is difficult to decide which of the algorithms is better when the majority of results have no rejected requests.

The SCS^{WF} solver is slightly worse than the SCS^{SW} and SCS^{DF} in all problem classes. This corresponds with findings in Vonolfen and Affenzeller [33] where the Wait first heuristic also produced worse results than the Drive first heuristic on their set of test instances. Nevertheless, the average number of rejected customer requests is still lower than the number of rejections from the best state-of-the-art solvers in 3 out of 4 test classes.

Authors of the GSA^{RO} solver stated that their approach is especially useful on instances with a high number of late customer requests. Our results confirm this assertion. The GSA^{RO} solver was able to find the best solution on all C4-RC104 instances which contain the biggest number of dynamic requests. Finally, the MSA^{C} solver never found a solution with a lower number of rejected customer requests than our SCS^{SW} and SCS^{DF} solvers. When compared with the SCS^{WF} solver, the MSA^{C} solver was able to find a better solution only in 7 cases.

6.2.3 Benchmarks with Warehouses

Tables 6.4 and 6.5 summarize the obtained results on DSVRPW benchmark instances. We compare average solution quality produced by the Greedy algorithm and the SCS solvers with all types of waiting heuristics - Drive first (SCS^{DF}), Wait first (SCS^{WF}), Scenario waiting (SCS^{SW}) and Relocation waiting (SCS^{RW}). The format of the tables is the same as for the DSVRP instances. The first value in each column represents the average number of rejected customer requests and the second one shows how many vehicles were used.

From the overall results, we can see that the number of rejected customer requests is much higher when the customers must be supplied from warehouses. This results from an increased number of assignments. Whenever a new customer request is revealed, the vehicle must firstly load the required goods in some warehouse before the customer can be visited. Because the requests are revealed gradually, the vehicles can load goods only for the currently known customers. This leads to additional warehouse visits when some new customer is added to vehicle's route.

Problem	Greedy		SCS ^{DF}		SCS ^{WF}		SCS ^{SW}		SCS ^{RW}	
instance	Uns.	Veh.	Uns.	Veh.	Uns.	Veh.	Uns.	Veh.	Uns.	Veh.
C1-RC101-1	5.8	16.0	2.8	16.0	3.4	16.0	**2.4**	**16.0**	**2.4**	**16.0**
C1-RC101-2	4.8	16.0	4.0	16.0	2.6	16.0	2.6	16.0	**2.4**	**16.0**
C1-RC101-3	5.8	15.0	**3.0**	**15.0**	4.6	15.0	4.2	15.0	3.2	15.0
C1-RC101-4	3.2	17.0	3.2	17.0	3.6	17.0	**2.2**	**17.0**	2.4	17.0
C1-RC101-5	6.4	17.0	4.4	17.0	4.4	17.0	**4.0**	**17.0**	**4.0**	**17.0**
C1-RC102-1	6.6	14.0	3.8	14.0	4.6	14.0	**3.2**	**14.0**	3.4	14.0
C1-RC102-2	4.4	13.0	2.8	13.0	3.2	13.0	1.8	13.0	**1.6**	**13.0**
C1-RC102-3	6.4	15.0	3.2	15.0	4.2	15.0	3.6	15.0	**2.8**	**15.0**
C1-RC102-4	4.8	14.0	2.2	14.0	2.4	14.0	1.4	14.0	**1.2**	**14.0**
C1-RC102-5	3.2	15.0	1.8	15.0	1.6	15.0	1.4	15.0	**1.0**	**15.0**
C1-RC104-1	4.4	11.0	3.0	11.0	3.2	11.0	2.6	11.0	**2.4**	**11.0**
C1-RC104-2	5.2	12.0	1.8	12.0	1.8	12.0	2.0	12.0	**1.4**	**12.0**
C1-RC104-3	6.6	13.0	1.8	13.0	1.8	13.0	**0.8**	**13.0**	1.0	13.0
C1-RC104-4	4.2	12.0	1.0	12.0	**0.8**	**12.0**	1.0	12.0	1.0	12.0
C1-RC104-5	2.4	11.0	1.8	11.0	2.0	11.0	**1.6**	**11.0**	1.8	11.0
Class 1 Avg	4.95	14.07	2.71	14.07	2.95	14.07	2.32	14.07	**2.13**	**14.07**
C2-RC101-1	**3.2**	**13.0**	4.0	13.0	4.0	13.0	4.0	13.0	4.2	13.0
C2-RC101-2	5.2	14.0	2.4	14.0	3.2	14.0	2.4	14.0	**2.0**	**14.0**
C2-RC101-3	6.6	17.0	4.2	17.0	6.0	17.0	**3.4**	**17.0**	3.8	17.0
C2-RC101-4	8.2	17.0	4.4	17.0	6.0	17.0	6.2	17.0	**3.6**	**17.0**
C2-RC101-5	5.8	16.0	**4.4**	**16.0**	5.0	16.0	4.6	16.0	**4.4**	**16.0**
C2-RC102-1	6.6	15.0	4.8	15.0	4.8	15.0	**3.8**	**15.0**	**3.8**	**15.0**
C2-RC102-2	4.2	14.0	2.8	14.0	2.6	14.0	2.8	14.0	**2.4**	**14.0**
C2-RC102-3	6.8	14.0	3.2	14.0	2.8	14.0	3.0	14.0	**2.4**	**14.0**
C2-RC102-4	4.8	15.0	**3.2**	**15.0**	3.8	15.0	3.6	15.0	**3.2**	**15.0**
C2-RC102-5	10.0	14.0	**4.6**	**14.0**	5.6	14.0	4.8	14.0	4.8	14.0
C2-RC104-1	12.4	12.0	9.2	12.0	10.4	12.0	8.8	12.0	**8.6**	**12.0**
C2-RC104-2	14.2	12.0	10.0	12.0	10.0	12.0	10.0	12.0	**9.6**	**12.0**
C2-RC104-3	14.4	12.0	9.8	12.0	9.6	12.0	10.2	12.0	**9.0**	**12.0**
C2-RC104-4	9.0	13.0	3.8	13.0	4.6	13.0	4.0	13.0	**2.8**	**13.0**
C2-RC104-5	9.8	12.0	8.4	12.0	8.4	12.0	7.8	12.0	**7.6**	**12.0**
Class 2 Avg	8.08	14.00	5.28	14.00	5.79	14.00	5.29	14.00	**4.81**	**14.00**

Table 6.4: Solutions on classes 1 and 2 from the DSVRPW benchmark instances. Each column contains two numbers. The first one represents the average number of unassigned customer requests and the second one shows the average number of used vehicles. Bold results highlight the best result in each line. The first column contains average results obtained with the Greedy algorithm. The remaining columns show average results obtained by our SCS solver. Each column represents different waiting heuristic - Drive first (DF), Wait first (WF), Scenario waiting (SW) and Relocation waiting (RW).

Problem	Greedy		SCS ^{DF}		SCS ^{WF}		SCS ^{SW}		SCS ^{RW}	
instance	Uns.	Veh.	Uns.	Veh.	Uns.	Veh.	Uns.	Veh.	Uns.	Veh.
C3-RC101-1	6.0	15.0	3.6	15.0	3.8	15.0	**3.4**	**15.0**	3.6	15.0
C3-RC101-2	6.8	16.0	**3.8**	**16.0**	5.0	16.0	4.8	16.0	4.0	16.0
C3-RC101-3	4.4	14.0	4.4	14.0	4.0	14.0	4.2	14.0	**3.8**	**14.0**
C3-RC101-4	6.0	17.0	5.0	17.0	4.6	17.0	**4.0**	**17.0**	**4.0**	**17.0**
C3-RC101-5	3.4	16.0	**1.8**	**16.0**	2.6	16.0	2.4	16.0	2.2	16.0
C3-RC102-1	4.6	15.0	3.2	15.0	2.6	15.0	3.2	15.0	**2.4**	**15.0**
C3-RC102-2	4.4	14.0	3.6	14.0	3.0	14.0	**2.4**	**14.0**	2.8	14.0
C3-RC102-3	6.4	13.0	4.2	13.0	4.6	13.0	4.8	13.0	**3.8**	**13.0**
C3-RC102-4	7.0	15.0	5.2	15.0	5.2	15.0	5.4	15.0	**4.8**	**15.0**
C3-RC102-5	5.4	15.0	3.0	15.0	3.2	15.0	**2.2**	**15.0**	2.4	15.0
C3-RC104-1	15.0	12.0	**10.8**	**12.0**	12.4	12.0	12.4	12.0	11.2	12.0
C3-RC104-2	5.6	12.0	3.0	12.0	2.4	12.0	**2.0**	**12.0**	2.4	12.0
C3-RC104-3	6.4	12.0	**3.4**	**12.0**	4.0	12.0	3.6	12.0	4.0	12.0
C3-RC104-4	9.4	12.0	**7.0**	**12.0**	7.6	12.0	**7.0**	**12.0**	7.4	12.0
C3-RC104-5	10.2	12.0	**7.2**	**12.0**	9.2	12.0	7.6	12.0	7.6	12.0
C3ass 1 Avg	6.73	14.00	4.61	14.00	4.95	14.00	4.63	14.00	**4.43**	**14.00**
C4-RC101-1	2.8	16.0	2.4	16.0	2.0	16.0	2.4	16.0	**1.8**	**16.0**
C4-RC101-2	6.0	15.0	4.8	15.0	5.6	15.0	4.8	15.0	**4.6**	**15.0**
C4-RC101-3	8.2	16.0	4.8	16.0	5.2	16.0	4.6	16.0	**3.8**	**16.0**
C4-RC101-4	7.0	17.0	4.2	17.0	5.2	17.0	4.6	17.0	**3.8**	**17.0**
C4-RC101-5	5.2	16.0	4.4	16.0	4.8	16.0	4.0	16.0	**3.6**	**16.0**
C4-RC102-1	**1.6**	**15.0**	2.6	15.0	2.4	15.0	2.2	15.0	2.4	15.0
C4-RC102-2	8.4	15.0	7.2	15.0	**6.0**	**15.0**	**6.0**	**15.0**	6.4	15.0
C4-RC102-3	7.8	15.0	5.4	15.0	**3.6**	**15.0**	**3.6**	**15.0**	3.8	15.0
C4-RC102-4	7.8	14.0	3.8	14.0	4.4	14.0	3.8	14.0	**3.2**	**14.0**
C4-RC102-5	8.2	15.0	5.2	15.0	**4.4**	**15.0**	4.8	15.0	**4.4**	**15.0**
C4-RC104-1	24.2	13.0	**15.0**	**13.0**	17.2	13.0	15.2	13.0	**15.0**	**13.0**
C4-RC104-2	26.0	14.0	15.6	14.0	17.8	14.0	**15.0**	**14.0**	15.6	14.0
C4-RC104-3	23.6	13.0	13.4	13.0	14.6	13.0	**12.8**	**13.0**	**12.8**	**13.0**
C4-RC104-4	18.8	12.0	12.0	12.0	13.0	12.0	12.6	12.0	**11.6**	**12.0**
C4-RC104-5	18.6	11.0	13.0	11.0	15.0	11.0	**12.6**	**11.0**	12.8	11.0
Class 2 Avg	11.61	14.47	7.59	14.47	8.08	14.47	7.27	14.47	**7.04**	**14.47**

Table 6.5: Solutions on classes 3 and 4 from the DSVRPW benchmark instances. Each column contains two numbers. The first one represents the average number of unassigned customer requests and the second one shows the average number of used vehicles. Bold results highlight the best result in each line. The first column contains average results obtained with the Greedy algorithm. The remaining columns show average results obtained by our SCS solver. Each column represents different waiting heuristic - Drive first (DF), Wait first (WF), Scenario waiting (SW) and Relocation waiting (RW).

The SCS^{RW} solver had the lowest number of rejected customer requests on all test classes. In 37/60 cases, the algorithm was able to find the best result among all the solvers. We can clearly see that the relocation strategy allowed the solver to serve a higher number of requests. It is because the vehicle more often waits next to a strategically placed warehouse which means that new customer requests can be served faster.

The differences between the SCS^{DF}, SCS^{WF}, and SCS^{SW} solvers are very similar as in the DSVRP case. The best overall results are produced by the SCS^{SW} solver with 19/60 best solutions and with the second lowest number of rejected customer requests. The SCS^{DF} solver is close behind with 11/60 best solutions. Finally, the SCS^{WF} solver is substantially worse than the remaining two solvers. It was able to find just 4/60 best solutions and only the Greedy algorithm had a higher number of rejected requests on each test class.

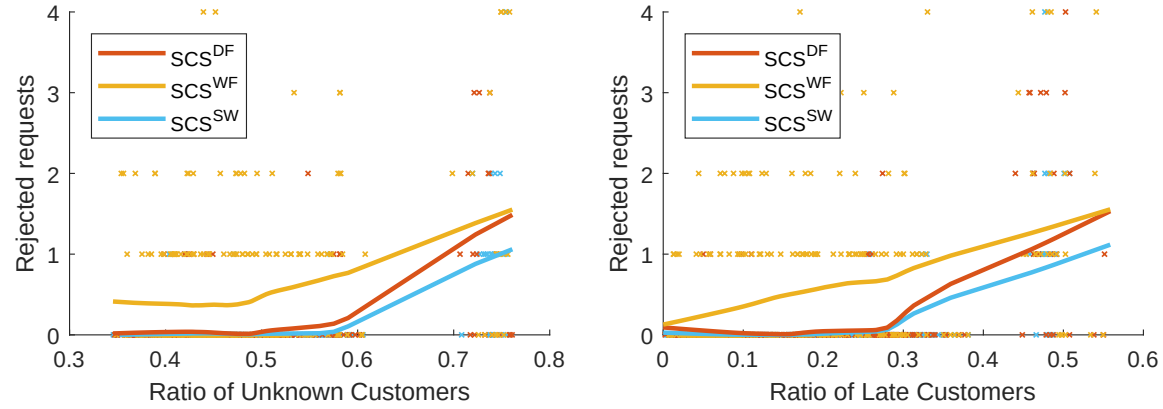
As we expected, the worst results were produced by the Greedy algorithm. Because it does not utilize the stochastic information about the problem, the produced Plans has much less options for possible future updates. This is best seen in C4-RC104 instances. As we already mentioned, these test cases have the highest number of late dynamic customer requests. While all the solvers have higher numbers of rejections for these instances, the Greedy algorithm rejects almost twice as much requests as the SCS^{RW} solver.

6.3 Search Analysis

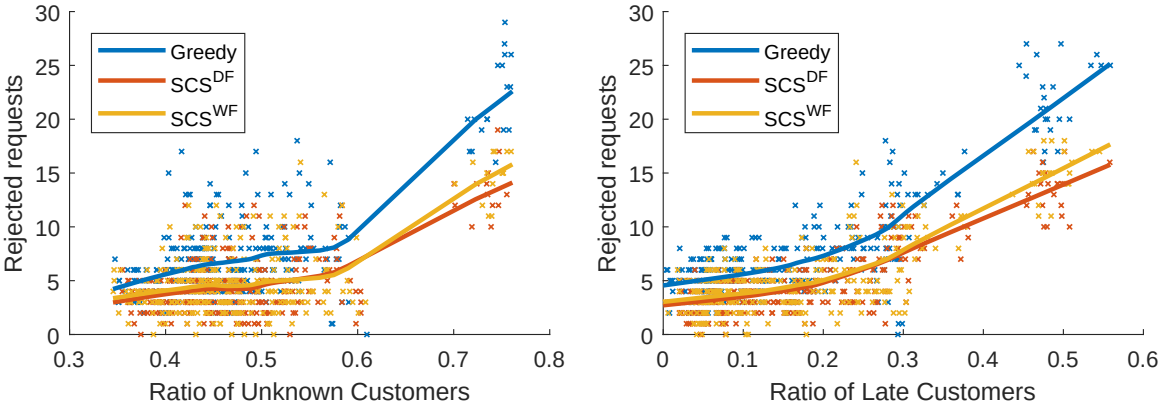
In this section, we study the differences between individual algorithms and analyze which factors influence the quality of the produced solutions. In Subsection 6.3.1, we try to determine which customer requests are more likely to be rejected by each algorithm. We compare arrival times, geographical locations and the overall number of dynamic requests. In the next part, we examine how the warehouse assignments affect the solution quality (see Subsection 6.3.2). We compare which warehouses cause the biggest vehicle detour and how the number of warehouse visits corresponds with the quality of the obtained results.

6.3.1 Rejected Customer Requests

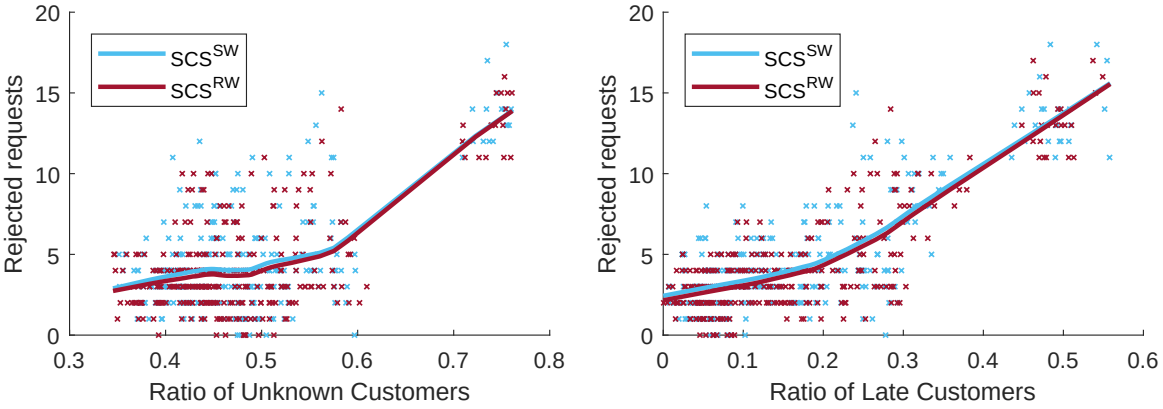
When we analyzed the solutions with the biggest number of rejected customer requests, we discovered that these instances have some common features. All algorithms had bigger problems on test cases with a higher degree of dynamism (DOD) and with many late customer requests. Both observations are not very surprising. With increasing DOD, the Plans contain less confirmed customers which means that the solvers have to deal with a higher level of uncertainty. With late customers, the algorithms have less options because there is often not enough time to reroute the vehicles and still visit all the confirmed customers. We also discovered that some remote regions have a very low number of rejections even though they are very far away from the vehicle depot. This means that the geographical distance is not a very reliable predictor of the probability of rejection.



(a) Bent's DSVRP benchmarks



(b) DSVRPW benchmarks



(c) DSVRPW benchmarks

Figure 6.4: Quality of solutions produced by different algorithms on both the DSVRP and DSVRPW benchmark instances. The points represent individual results and the lines are created by *locally weighted scatterplot smoothing* (LOWESS) method which uses weighted linear least squares model. (Left) The number of rejected customer requests with respect to the percentage of unknown dynamic customers (degree of dynamism). (Right) The number of rejected customer requests with respect to the percentage of late dynamic customers (appearance time is bigger than 80).

Analysis of Dynamism

On the left side of Figure 6.4, we compare the quality of the solutions as a function of the percentage of unknown customers. Each point represents the number of rejected customer requests (vertical axis) for one solution with the given DOD (horizontal axis). On the right side, we show how the number of rejections corresponds with the percentage of late dynamic customer requests. This means that we consider only the requests from period 2 (see Subsection 6.1.2) with arrival time $t \in [81, 160]$, not all dynamic customers as in the first case.

Subfigure 6.4a compares results obtained by the SCS solvers with Drive first, Wait first and Scenario waiting heuristics on DSVRP test cases. As we can see, both the SCS^{DF} and SCS^{SW} solvers have almost no problem in situations with a low number of dynamic requests. On the other hand, the SCS^{SW} solver is able to find significantly better solutions on highly dynamic instances with many late customer requests. We can also see that the SCS^{WF} solver lacks behind the SCS^{DF} and SCS^{SW} solvers mainly on instances with a low number of dynamic requests. As the DOD increases, the gap between these three solvers becomes smaller.

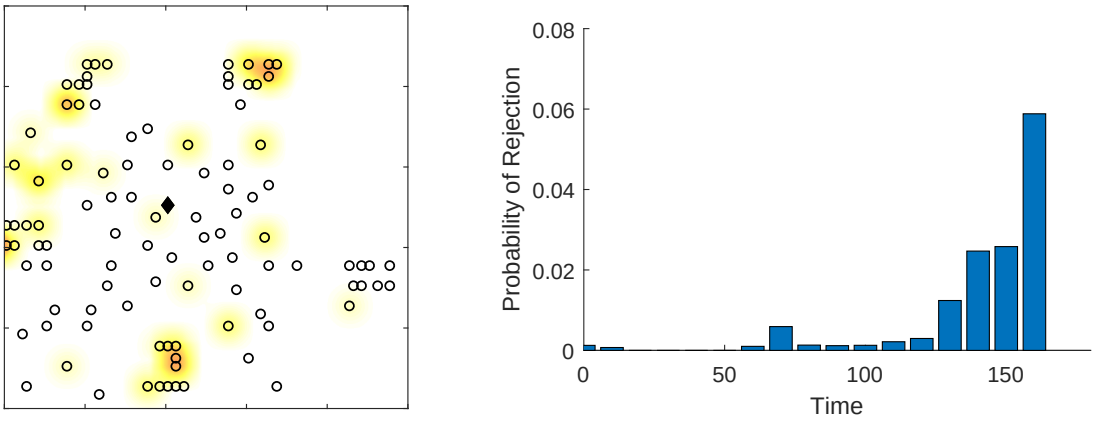
For clarity, we divided the solvers used on the DSVRPW instances into two groups. Subfigure 6.4b compares the Greedy algorithm with the SCS solvers using simple Drive first and Wait first heuristics. We can see that the Greedy algorithm is not much worse than the SCS^{DF} and SCS^{WF} solvers on instances with low DOD and a small number of late requests. In highly dynamic test cases, the benefits of stochastic information are more visible and the Greedy algorithm produces significantly less competitive results. When we compare the SCS^{DF} and SCS^{WF} solvers, we can see that the SCS^{WF} solver lacks in instances with many late customer requests. This is the exact opposite of the results we obtained on the DSVRP instances, where the SCS^{WF} solver struggled mainly on the test cases with low DOD.

Finally, Subfigure 6.4c compares the two best-performing solvers on the DSVRPW instances - the SCS with Scenario waiting heuristic and the SCS with Relocation waiting heuristic. Both algorithms produced almost identical results on all test cases. The SCS^{RW} solver is consistently slightly better than the SCS^{SW} solver but the difference is always really small. This result is not surprising because the Relocation waiting heuristic uses the Scenario waiting heuristic to calculate the final arrival and departure times. The additional waiting locations are inserted only rarely which explains why the produced solutions are so similar.

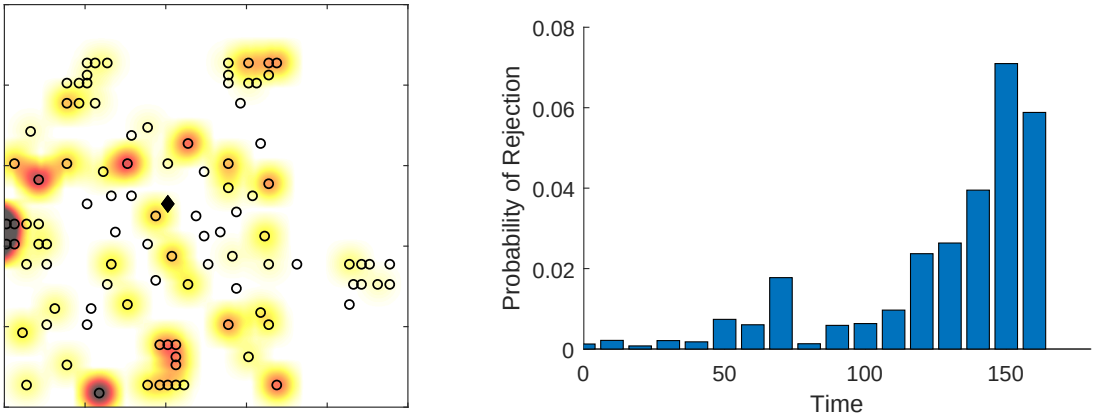
Analysis of Rejections

Figures 6.5 and 6.6 illustrate spatiotemporal properties of rejected customer requests. Each subfigure represents summary for one algorithm over all the DSVRP or DSVRPW test cases. On the left side, we present locations with the highest number of rejections in form of a heat-map (darker color represents a higher number of rejections). Histograms showing the probability of customer rejection in time can be seen on the right side. These figures help us to identify regions with weak accessibility and time periods where the vehicles are unable to serve additional customers.

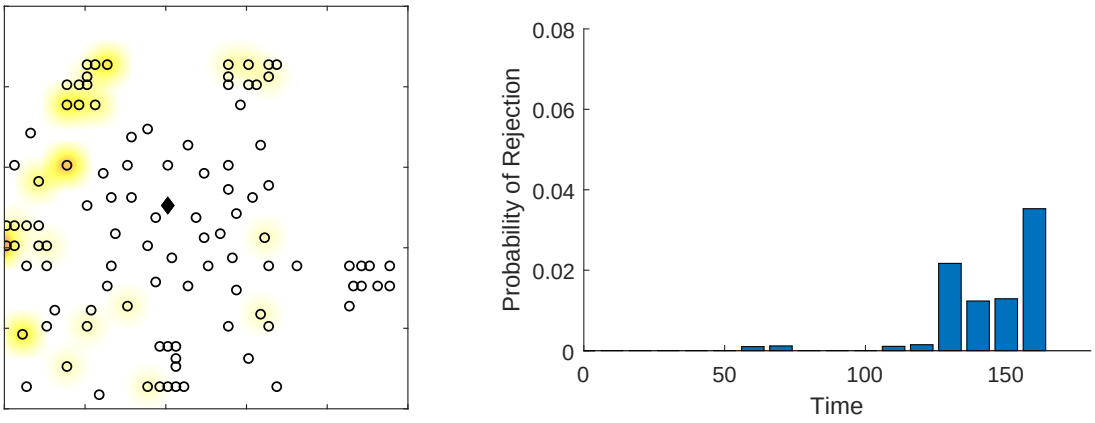
Figure 6.5 compares spatiotemporal properties of customer requests rejected by the SCS solvers with Drive first, Wait first and Scenario waiting heuristics on the DSVRP test cases.



(a) SCS with Drive first heuristic

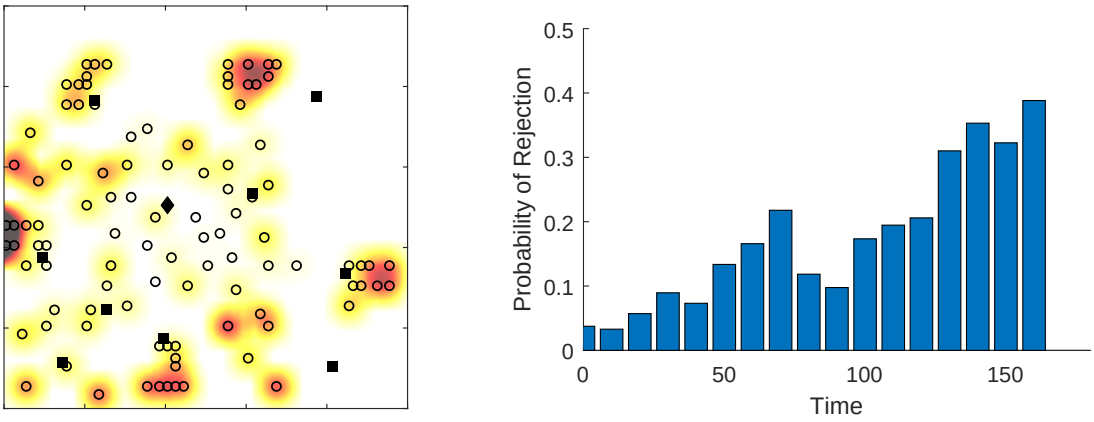


(b) SCS with Wait first heuristic

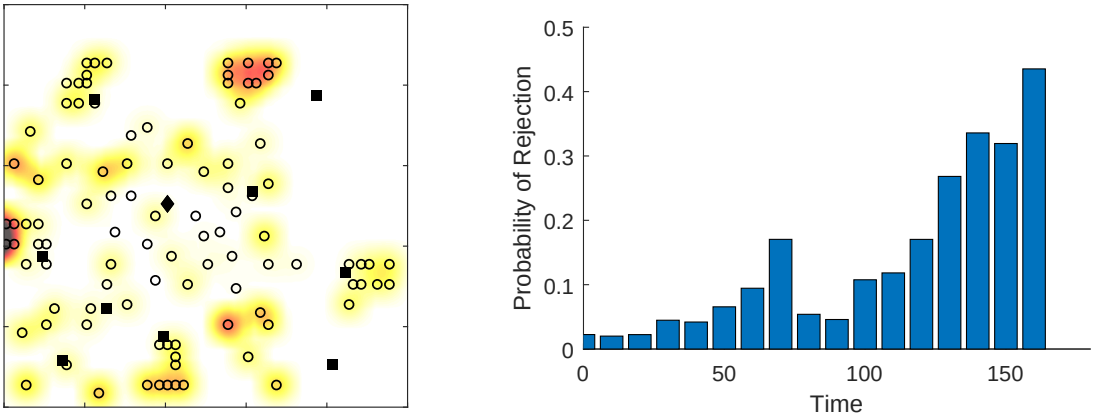


(c) SCS with Scenario waiting heuristic

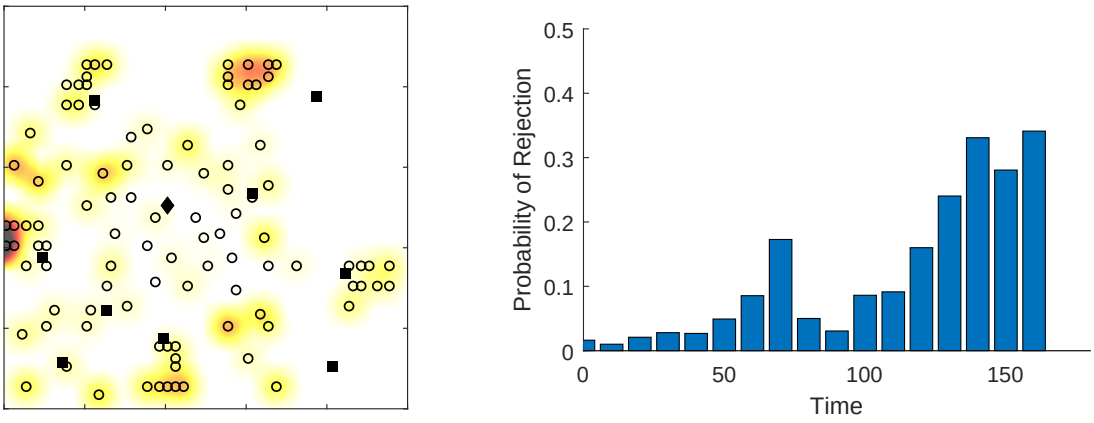
Figure 6.5: Results obtained on the Bent's DSVRP benchmark instances. (Left) Heat-map showing areas with the biggest number of rejected customer requests for different algorithms (darker color represents a higher number of rejections). (Right) Histogram showing the probability of customer request rejection in time for different algorithms.



(a) Greedy algorithm



(b) SCS with Wait first heuristic



(c) SCS with Relocation waiting heuristic on benchmarks with warehouses

Figure 6.6: Results obtained on the DSVRPW benchmark instances. (Left) Heat-map showing areas with the biggest number of rejected customer requests for different algorithms (darker color represents higher number of rejections). (Right) Histogram showing the probability of customer request rejection in time for different algorithms.

The SCS^{DF} solver had the biggest problems with the north-east and south regions and with very late customer requests. The SCS^{WF} solver rejected the biggest number of customers from the south and west regions. Quite surprisingly, it had also problems with customers near the vehicle depot, where other algorithms had almost no rejections. This algorithm also rejected much more early customer requests from the first period. Finally, the SCS^{SW} solver was able to serve customers from almost all regions and time periods. It had minor problems in the north-west area and with very late customer requests but the overall results do not show any signs of serious problems.

Figure 6.6 shows locations and time distributions of customer requests rejected by selected algorithms on the DSVRPW test cases. For conciseness, we do not show subfigures for the SCS^{DF} and SCS^{SW} solvers because they looked almost the same as the ones presented for the SCS^{WF} and SCS^{RW} solvers, respectively. As expected, the biggest number of problematic locations had the Greedy algorithm. Almost all peripheral regions have an increased amount of rejections. These customers were also rejected consistently throughout the whole planning horizon with peaks in time intervals $[50, 80]$ and $[110, 150]$. The SCS^{WF} and SCS^{RW} solvers produced very similar results. They both had the biggest problems with the north-east and west regions and with very late requests. We can also see that the SCS^{RW} solver rejected slightly less customer requests near the vehicle depot and in the south-east region in comparison with the SCS^{WF} solver.

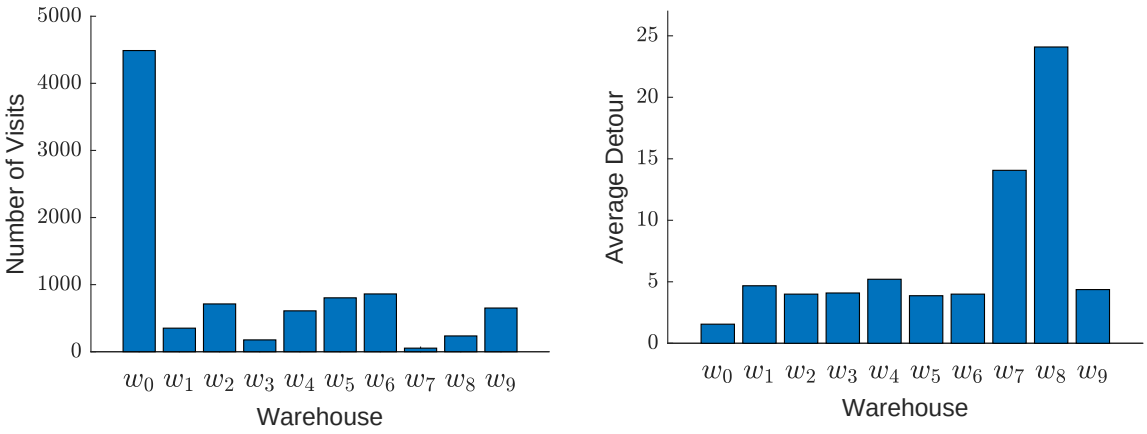
6.3.2 Warehouse Assignments

Because the introduction of warehouses strongly increased the number of rejected customer requests, we decided to analyze how often each warehouse was visited, how these visits prolonged the vehicle routes and whether there are some differences in warehouse visits between the algorithms. Note that the exact warehouse locations are depicted in Figure 6.1. Obviously, we studied only the results obtained on the DSVRPW test cases because the DSVRP instances do not allow warehouse visits. For clarity and conciseness, we also decided to omit results obtained for the SCS^{DF} and SCS^{SW} solvers because they are very similar as the ones presented for the SCS^{WF} and SCS^{RW} solvers, respectively.

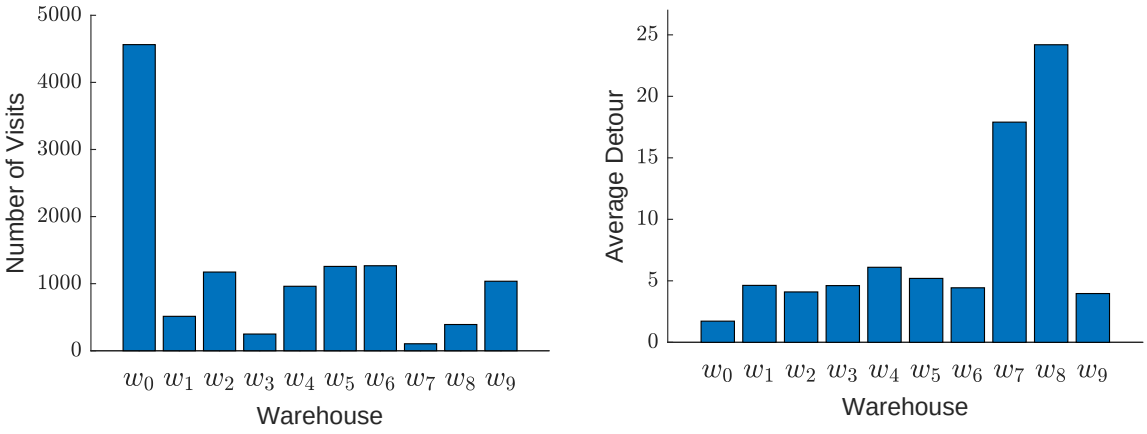
Analysis of Warehouse Locations

On the left side of Figure 6.7, we show the popularity of each warehouse. Unsurprisingly, the most popular warehouse is w_0 which has the same location as the vehicle depot. Almost all vehicles visit this warehouse before they supply their first customers because this visit does not prolong the route. Interestingly, the popularity of this warehouse is different for each algorithm. It is the most popular in solutions produced by the Greedy algorithm with 50.2 % of all warehouse visits (i.e. every second warehouse assignment contains the warehouse w_0). The SCS^{WF} solver used this warehouse in 39.6 % of visits and the SCS^{RW} solver only in 36.3 % of visits.

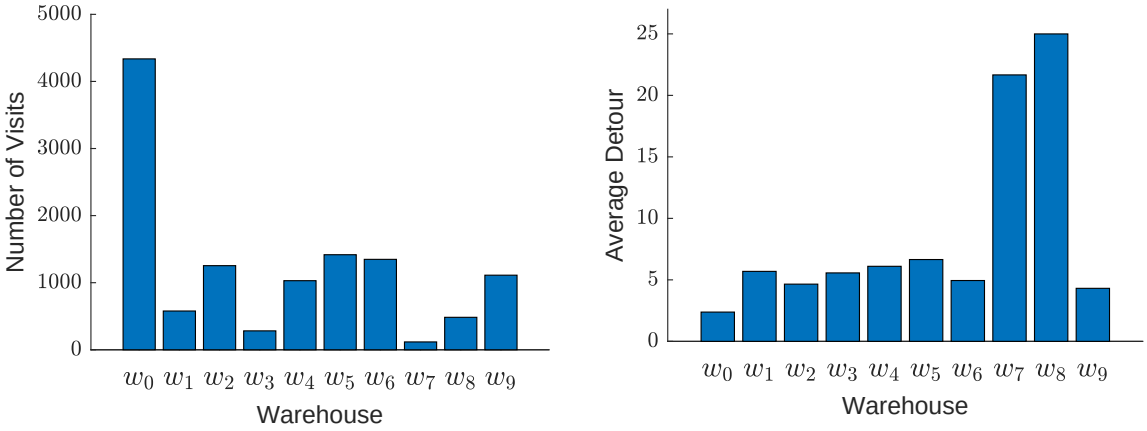
The warehouses w_2, w_5, w_6 and w_9 are also very often visited because they supply the north-west, south, west and east clusters, respectively. We are quite surprised that the



(a) Greedy algorithm



(b) SCS solver with wait first heuristic



(c) SCS solver with relocation waiting heuristic

Figure 6.7: (Left) The number of visits of each warehouse for different algorithms. (Right) The average detour caused by each warehouse for different algorithms. Detour symbolizes additional distance traveled by the vehicle to visit the warehouse.

warehouse w_8 has only a few visits even though it is the nearest warehouse to the north-east cluster. This region is probably supplied from the warehouse w_4 which is not too far away. The relative popularity of these warehouses is very similar for all the algorithms. On the other hand, they differ in the absolute number of visits. The Greedy algorithm needed the least number of warehouse assignments with 8940 visits over all solutions. The SCS $^{\text{WF}}$ solver visited 11514 warehouses and the SCS $^{\text{RW}}$ solver needed 11958 visits.

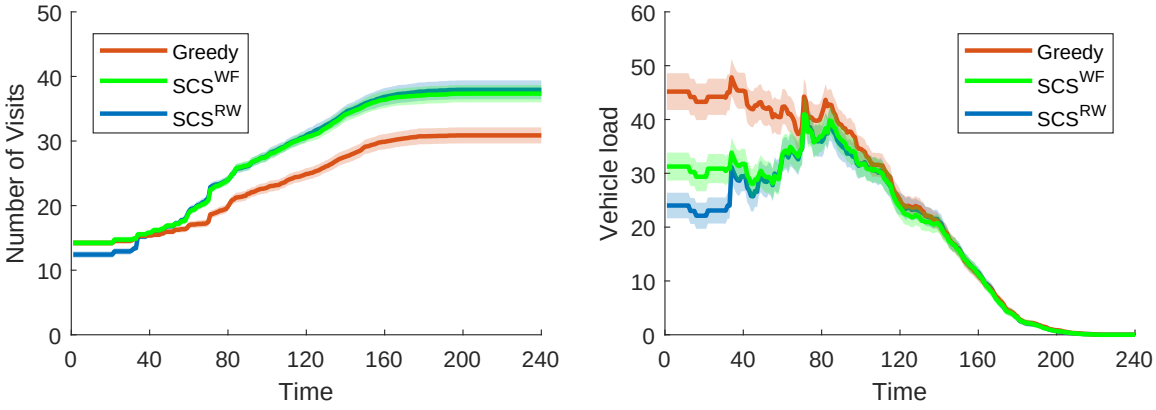
On the right side of Figure 6.7, we show the average detour caused by each warehouse. Detour is calculated from vehicle route as $s = d_{i-1,i} + d_{i,i+1} - d_{i-1,i+1}$, where $d_{i-1,i}$ is the distance from the previous node to the warehouse, $d_{i,i+1}$ is the distance from the warehouse to the next node and $d_{i-1,i+1}$ is the direct distance from the previous node to the next node. For each algorithm, the biggest detour is caused by the warehouse w_8. This warehouse is placed next to the north-east cluster which means that the vehicle must always abandon the region with customers, visit the warehouse and then travel back. The second longest detour is caused by the warehouse w_7. Interestingly, this distance is very different for each solver. The Greedy algorithm was able to order the customers in such a way that the detour is much smaller than for the SCS $^{\text{RW}}$ solver. On the other hand, this warehouse is visited so rarely that the caused detour is not very significant in the overall results.

Analysis of Warehouse Visits

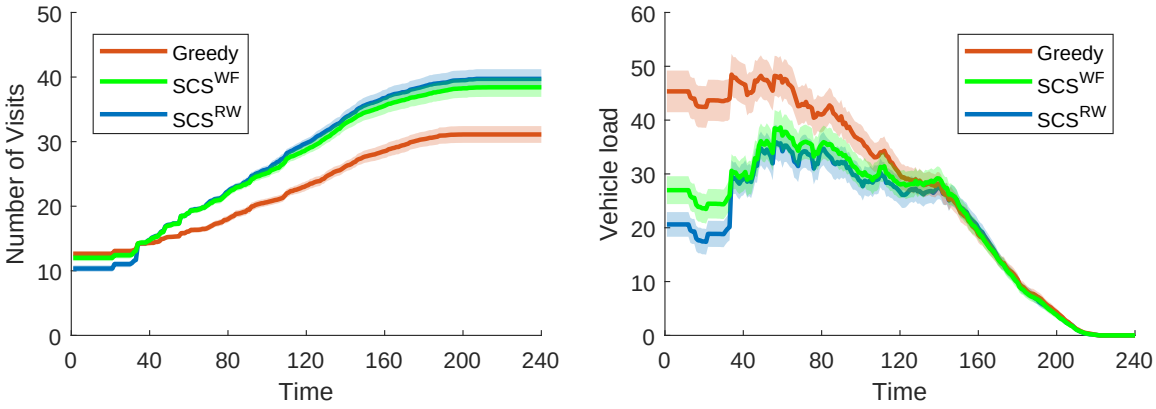
Figure 6.8 (left) shows an average number of visited warehouses over time on different problem types. It is clearly visible that the Greedy algorithm needed much less warehouse visits than the two remaining solvers. This difference is caused by two factors. Firstly, the other solvers rejected much less customer requests which means that they had to supply more customers and thus load the supplies more often. Secondly, the Greedy algorithm always created Plan only for the existing and confirmed customers. This allowed the static solver to create a more optimized Plan with a higher number of customers supplied from one warehouse. On the other hand, this also made the Plan less extensible when a new customer request arrived because the vehicle was not prepared for any additional assignments.

We can also see that the SCS $^{\text{RW}}$ solver visited less warehouses in time interval $t \in [0, 40]$ than the other two solvers. All these visits are made at the beginning of the scheduling horizon in the warehouse w_0 which has the same location as the vehicle depot. This difference is caused by the Relocation waiting heuristic which inserts promising new waiting locations into the Plan. The heuristic typically selects some warehouse location because then it is easier and faster to load supplies for a newly arrived customer. When this waiting location is the first visit in the vehicle route, it is not necessary to load supplies from the warehouse w_0 in time $t = 0$ because all customers can be served from the next waiting location with a warehouse.

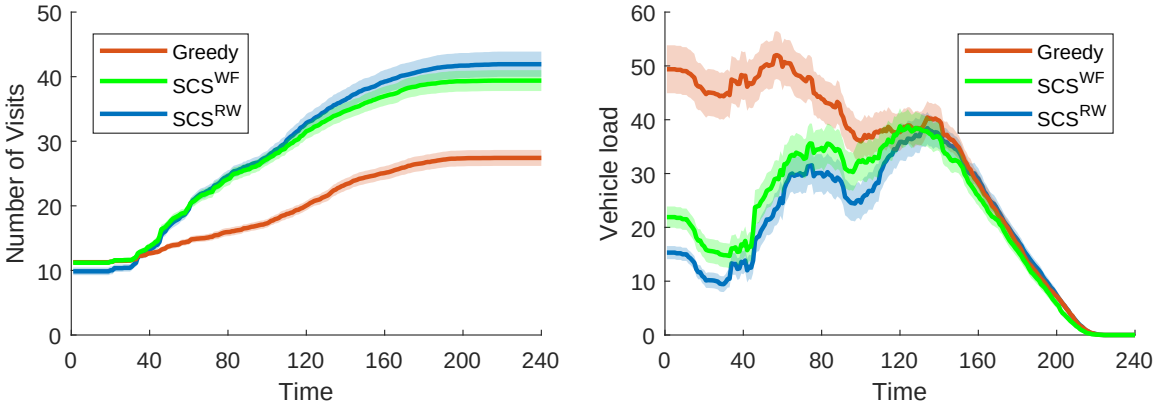
On the right side of Figure 6.8, we can see an average vehicle load over time. Because the solutions produced by the Greedy algorithm contain less warehouse visits, it is obvious that the vehicles must load higher amount of goods in these warehouses to be able to serve all the subsequent customers. We can also see that the average vehicle load in solutions produced



(a) RC101 instances



(b) RC102 instances



(c) RC104 instances

Figure 6.8: (Left) Number of visited warehouses over time on different types of problems. (Right) Vehicle load over time on different types of problems. Each line represents an average over all classes. The lightly colored boundary around each line illustrates the 99% confidence interval.

by the SCS$^{\text{RW}}$ solver is very low in time interval $t \in [0, 40]$. As we already stated, some vehicles skip the warehouse w_0 which means that they are empty in the first part of their route.

It is also interesting to compare the RC101 and RC102 problem types shown in Subfigures 6.8a and 6.8b with the type RC104 depicted in Subfigure 6.8c. The RC101 and RC102 problems are on average less dynamic than the RC104 instances. We can see that the difference between the Greedy algorithm and the remaining two SCS solvers is much smaller on the RC101 and RC102 test cases. The average vehicle load in solutions produced by the Greedy algorithm is higher on the RC104 problems than on the other types. On the other hand, both the SCS$^{\text{WF}}$ and SCS$^{\text{RW}}$ solvers produced solutions with a lower average vehicle load on the RC104 instances.

This probably explains why the Greedy algorithm had so many rejected requests on instances with high DOD. Each new customer can be served only after the selected vehicle loads the ordered goods in some warehouse. When the vehicle is already fully loaded and it has only a few warehouse visits planned in its route, it is very difficult to do such thing. The vehicle has to firstly visit at least some existing customers to deliver their order and make enough space for the new customer. In the next step, it is necessary to make an unplanned warehouse visit with a potentially high detour to load the ordered goods. Only then it is possible to visit the new customer. On the other hand, if the vehicle is half empty and it has many warehouse visits planned in its route plan, the solver can just insert the new customer at the best position in the route and the vehicle will simply load more goods during the next planned warehouse visit.

Chapter 7

Conclusion

All goals of the thesis were fulfilled. We reviewed the existing approaches used to solve different non-deterministic variants of the VRP in Chapter 2. Three types of the problem were discussed in more detail. In dynamic VRP, the whole instance is not known in advance and the scheduler must regularly recalculate the solution to address the changes in data. In stochastic VRP, some of the data are only available as random variables. The scheduler should utilize these data to produce a robust solution which will not deviate too much from the actual route execution. Finally, the dynamic stochastic VRP combines both techniques to produce more accurate solutions with lower overall costs.

Unfortunately, none of the examined approaches was applicable for problems where the customers make orders which must be firstly picked up in some warehouse before they are delivered. Because this is a crucial aspect of many real-world problems such as food delivery, we defined a novel DSVRP variant which considers warehouse visits in Chapter 3. The definition ensures that each order will be loaded in the most suitable warehouse before it is delivered to the customer. We defined the static version of this problem by an integer linear program and the dynamic version by a multistage stochastic program.

The main contribution of this thesis was presented in Chapter 5. We firstly explained how the TASP framework (described in Chapter 4) was adapted to solve the static version of the DSVRPW. In the next part, we introduced our SCS solver which is able to solve the original DSVRPW instances. It combines two approaches to produce very robust solutions applicable to many future scenarios. In the first step, the most universal plan is selected from a pool of solutions containing both existing and possible future customer requests. After that, we use one of the two novel waiting heuristics to distribute the idle time in a vehicle route. Both the heuristics utilize information about the possible future customer requests mentioned in the previous step to select the most promising waiting locations.

Finally, we tested our implementation of the SCS solver written in Kotlin in Chapter 6. We used a collection of DSVRP benchmark instances to compare our implementation with other state-of-the-art methods. In this comparison, the SCS solver found the best average solution in 55/60 test cases. We were also able to find at least one solution with no rejected

customer requests for all the test cases. As for the newly defined DSVRPW variant, there are obviously no published benchmarks available. For this reason, we created our own DSVRPW test cases and compared the SCS solver with a Greedy algorithm. The results achieved by our solver are better in 58/60 test cases. The difference is most noticeable in highly dynamic instances where the Greedy algorithm rejected almost twice as much customer requests. We also showed that the novel waiting heuristics produced better solutions when compared with the existing waiting strategies.

7.1 Future Work

Since our primary goal is to develop a scheduling framework which would be applicable for a wide spectrum of real-world dynamic routing problems, we identified many opportunities how to extend its scope. Firstly, our sampling procedure expects that the stochastic information about the problem is known in advance. Unfortunately, this is not the case in most of the real-world scenarios, where the behavior of customers is not easily predictable. For this reason, it would be useful to extend the sampling procedure with a new module which would be able to approximate the behavior of customers from historical data. This procedure would have to consider many factors such as a history of each customer, geographical location, time and day of the week, weather, season, etc., making it quite a difficult task.

It would be also possible to incorporate additional sources of dynamism in the planning. One of the options would be to consider fluctuations in traffic density. Especially in large cities, rush hours and traffic jams can dramatically influence the feasibility of the produced solutions. The solver should adapt the travel times based on the daytime and re-route the vehicles in case of unexpected car accidents or other temporary traffic restrictions. Similarly, it could be useful to include other possible sources of dynamism such as vehicle breakdowns, prolonged visit durations or canceled customer requests.

Finally, it might be useful to restrict the number of changes in the schedule when a new customer request is introduced. Currently, the solver is allowed to change the schedule arbitrarily as long as it does not affect the historical actions. This means that whenever a new customer request is accepted, all the vehicles in the fleet must update their route plans to reflect the changes in the plan. Without a proper control system, this might substantially increase the communication overhead between the central dispatcher and the vehicles and cause general dissatisfaction with the scheduling process. For this reason, the objective function might take into account the difference between the current and updated solution. This would force the solver to prefer solutions with less changes, making the plans more stable in time.

Bibliography

[1] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem", *Management Science*, vol. 6, Oct. 1959.

[2] P. Toth and D. Vigo, "1. an overview of vehicle routing problems", in *The Vehicle Routing Problem*. 2002, ch. 1, pp. 1–26.

[3] R. Montemanni, L. Maria Gambardella, A.-E. Rizzoli, and A. Donati, "Ant colony system for a dynamic vehicle routing problem", vol. 10, pp. 327–343, Dec. 2005.

[4] S. Ichoua, M. Gendreau, and J.-Y. Potvin, "Vehicle dispatching with time-dependent travel times", *European Journal of Operational Research*, vol. 144, no. 2, pp. 379 –396, 2003.

[5] L. Gambardella, A.-E. Rizzoli, F. Oliverio, N. Casagrande, A. V. Donati, R. Montemanni, and E. Lucibello, "Ant colony optimization for vehicle routing in advanced logistics systems", pp. 3–, Jan. 2003.

[6] U. Ritzinger, J. Puchinger, and R. Hartl, "A survey on dynamic and stochastic vehicle routing problems", vol. 54, pp. 215–231, Jan. 2016.

[7] V. Pillac, M. Gendreau, C. Gueret, and A. Medaglia, "A review of dynamic vehicle routing problems", *European Journal of Operational Research*, vol. 225, no. 1, pp. 1 –11, 2013.

[8] H. Psaraftis, "A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem", *Transportation Science*, vol. 14, no. 2, pp. 130–154, 1980.

[9] W. Ou and B. G. Sun, "A dynamic programming algorithm for vehicle routing problems", in *2010 International Conference on Computational and Information Sciences*, 2010, pp. 733–736.

[10] M. Elhassania, B. Jaouad, and E. A. Ahmed, "Solving the dynamic vehicle routing problem using genetic algorithms", pp. 62–69, 2014.

[11] I. Benyahia and J.-Y. Potvin, "Decision support for vehicle dispatching using genetic programming", *Trans. Sys. Man Cyber. Part A*, vol. 28, no. 3, pp. 306–314, May 1998.

[12] A.-E. Rizzoli, R. Montemanni, E. Lucibello, and L. Maria Gambardella, "Ant colony optimization for real-world vehicle routing problems", vol. 1, pp. 135–151, Nov. 2007.

[13] M. Gendreau, F. Guertin, J.-Y. Potvin, and E. Taillard, "Parallel tabu search for real-time vehicle routing and dispatching", *Transportation Science*, vol. 33, no. 4, pp. 381–390, Apr. 1999.

[14] A. Attanasio, J.-F. Cordeau, G. Ghiani, and G. Laporte, "Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem", *Parallel Computing*, vol. 30, no. 3, pp. 377 –387, 2004.

[15] A. Beaudry, G. Laporte, T. Melo, and S. Nickel, "Dynamic transportation of patients in hospitals", *OR Spectrum*, vol. 32, no. 1, pp. 77–107, 2010.

[16] J. F. Ehmke, A. Steinert, and D. C. Mattfeld, "Advanced routing for city logistics service providers based on time-dependent travel times", *Journal of Computational Science*, vol. 3, no. 4, pp. 193 –205, 2012, City Logistics.

[17] F. Ferrucci, "A new forecasting approach for generating stochastic knowledge from past request information and utilizing the stochastic knowledge", in *Pro-active Dynamic Vehicle Routing: Real-Time Control and Request-Forecasting Approaches to Improve Customer Service*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 167–183.

[18] J. Mendoza, B. Castanier, C. Gueret, A. Medaglia, and N. Velasco, "A memetic algorithm for the multi-compartment vehicle routing problem with stochastic demands", *Comput. Oper. Res.*, vol. 37, no. 11, pp. 1886–1898, Nov. 2010.

[19] P. Beraldi, M. E. Bruni, D. Laganà, and R. Musmanno, "The mixed capacitated general routing problem under uncertainty", *European Journal of Operational Research*, vol. 240, no. 2, pp. 382 –392, 2015.

[20] L. Chen, M. H. Hà, A. Langevin, and M. Gendreau, "Optimizing road network daily maintenance operations with stochastic service and travel times", *Transportation Research Part E: Logistics and Transportation Review*, vol. 64, no. Supplement C, pp. 88 –102, 2014.

[21] F. Errico, G. Desaulniers, M. Gendreau, W. Rei, and L.-M. Rousseau, "A priori optimization with recourse for the vehicle routing problem with hard time windows and stochastic service times", *European Journal of Operational Research*, vol. 249, no. 1, pp. 55 –66, 2016.

[22] C. Gauvin, G. Desaulniers, and M. Gendreau, "A branch-cut-and-price algorithm for the vehicle routing problem with stochastic demands", *Computers and Operations Research*, vol. 50, no. Supplement C, pp. 141 –153, 2014.

[23] C. H. Christiansen, J. Lysgaard, and S. Wohlk, "A branch-and-price algorithm for the capacitated arc routing problem with stochastic demands", *Operations Research Letters*, vol. 37, no. 6, pp. 392 –398, 2009.

[24] J.-F. Cote, J.-Y. Potvin, and M. Gendreau, *The Vehicle Routing Problem with Stochastic Two-dimensional Items*. CIRRELT, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, 2013.

[25] M.-S. Chang, "A vehicle routing problem with time windows and stochastic demands", *Journal of the Chinese Institute of Engineers*, vol. 28, no. 5, pp. 783–794, 2005.

[26] A. Ahmadi-Javid and A. H. Seddighi, "A location-routing problem with disruption risk", *Transportation Research Part E: Logistics and Transportation Review*, vol. 53, pp. 63 –82, 2013.

[27] H. Lei, G. Laporte, and B. Guo, "The capacitated vehicle routing problem with stochastic demands and time windows", *Computers and Operations Research*, vol. 38, no. 12, pp. 1775 –1783, 2011.

[28] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems", *Comput. Oper. Res.*, vol. 34, no. 8, pp. 2403–2435, 2007.

[29] B. Thomas and C. White, "Anticipatory route selection", *Transportation Science*, vol. 38, no. 4, pp. 473–487, 2004.

[30] B. Thomas, "Waiting strategies for anticipating service requests from known customer locations", *Transportation Science*, vol. 41, no. 3, pp. 319–331, 2007.

[31] G. Kim, Y. S. Ong, T. Cheong, and P. S. Tan, "Solving the dynamic vehicle routing problem under traffic congestion", *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 8, pp. 2367–2380, 2016.

[32] S. Ichoua, M. Gendreau, and J.-Y. Potvin, "Exploiting knowledge about future demands for real-time vehicle dispatching", *Transportation Science*, vol. 40, no. 2, pp. 211–225, 2006.

[33] S. Vonolfen and M. Affenzeller, "Distribution of waiting time for dynamic pickup and delivery problems", *Annals of Operations Research*, vol. 236, no. 2, pp. 359–382, 2016.

[34] R. Bent and P. V. Hentenryck, "Scenario-based planning for partially dynamic vehicle routing with stochastic customers", *Operations Research*, vol. 52, no. 6, pp. 977–987, 2004.

[35] P. Van Hentenryck, R. Bent, and E. Upfal, "Online stochastic optimization under time constraints", *Annals of Operations Research*, vol. 177, no. 1, pp. 151–183, 2010.

[36] M. Saint-Guillain, Y. Deville, and C. Solnon, "A multistage stochastic programming approach to the dynamic and stochastic vrptw", L. Michel, Ed., pp. 357–374, 2015.

[37] P. Eichler, "Vehicle routing problem with multiple time windows", Bachelor's Thesis, Czech Technical University in Prague, May 2016.

[38] Blindspot Solutions. [Online]. Available: <http://blindspot.ai/>.

[39] G. Peterson. (2018). Paguro collections, [Online]. Available: <https://github.com/GlenKPeterson/Paguro>.

[40] M. Solomon. (1987). Solomon benchmarks, [Online]. Available: <http://w.cba.neu.edu/~msolomon/problems.htm>.

[41] R. Bent and P. V. Hentenryck. (2004). Benchmarks for the dynamic and stochastic vehicle routing problem with time windows, [Online]. Available: <http://becool.info.ucl.ac.be/resources/benchmarks-dynamic-and-stochastic-vehicle-routing-problem-time-windows>.

Appendix A

Attached Files

The attached files contain the source code of the SCS solver, electronic version of the thesis, Bent's DSVRP and our DSVRPW benchmark instances, source files of this text, and our solutions on all test cases. Table A.1 describes the structure and content of the attached files.

File or folder name	Content
Benchmarks	Folder which contains DSVRP and DSVRPW benchmark instances
Solutions	Folder which contains our solutions on all test cases
SourceCode	Folder which contains the complete source code of the SCS solver
LatexCode	Folder which contains the LaTeX source files of the thesis
Thesis.pdf	Electronic version of the thesis
SCS_Solver.jar	Compiled code of the SCS solver

Table A.1: Structure and content of the attached files.

The SCS solver can be executed from the command line using command

```
java -jar SCS_Solver.jar -b BENCHMARKS -o OUTPUT
```

where `BENCHMARKS` specifies the input folder with benchmark instances and `OUTPUT` specifies the output folder. All the available options can be displayed with command

```
java -jar SCS_Solver.jar --help
```