ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

INGEGNERIA BIOMEDICA, ELETTRICA E DEI SISTEMI

Ciclo 36

Settore Concorsuale: 01/A6 – RICERCA OPERATIVA

Settore Scientifico Disciplinare MAT/09 – RICERCA OPERATIVA

MODELS AND ALGORITHMS FOR ROUTING AND SCHEDULING OPTIMIZATION PROBLEMS

Presentata da: ***Francesco Cavaliere***

Coordinatore Dottorato
Michele Monaci

Supervisore
Daniele Vigo

Co-supervisore
Michele Monaci

Esame finale anno 2024

Abstract

This thesis encompasses three distinct yet interrelated works, each contributing to the field of optimization.

In the first work, an effective heuristic algorithm tackles the Capacitated Vehicle Routing Problem, particularly addressing large-scale instances. The algorithm employs a combination of local search and restricted Set Partitioning problem optimization, leveraging Helsgaun's LKH-3 algorithm for the local search phase. Notably, this approach consistently enhances solutions available on the CVRPLIB website.

The second work delves into the extension of the FILO framework, initially designed for the Capacitated Vehicle Routing Problem. The objective is two-fold: to be competitive with state-of-the-art algorithms for simultaneous pickup and delivery problems, and to efficiently solve very large benchmark instances with numerous customers, all while maintaining linear scalability concerning problem size. A rigorous computational study validates the success in achieving both objectives.

The third work centers on PLATiNO, a Synthetic Aperture Radar Earth observation satellite. Efficient activity planning is essential to maximize the satellite's potential while adhering to platform constraints. A genetic algorithm, combined with repair procedures and local search operators, addresses the intricacies of this planning. Additionally, Mixed Integer Linear Programming formulations are utilized to provide precise estimations of optimal solution values. Extensive testing on real-world benchmark instances demonstrates the algorithm's proficiency in computing near-optimal solutions within practical time limits.

Acknowledgments

I would like to express my heartfelt gratitude to several individuals who have played a key role in my academic journey and personal growth during my doctoral studies.

First and foremost, I want to thank Luca Accorsi for his guidance and mentorship throughout my Ph.D. His support and friendship have played a crucial role in shaping my research and overall experience.

I am also grateful to my supervisors, Daniele Vigo and Michele Monaci, for the opportunities they provided and the guidance they offered, which enriched my academic and research endeavors.

I want to thank Silvia, Alan, Antonio, and Federico for their friendship and for sharing with me this three-year long journey.

My thanks go also to the entire group of researchers and professors of the OR group of the University of Bologna, DEI department, for creating a friendly and welcoming environment and engaging in insightful discussions.

Lastly, I appreciate the support of my parents, family, and friends during these transformative three years. Their encouragement and belief in me have been a constant source of strength.

I am sincerely thankful to each of you for your contributions and support, which have been essential to my academic growth and personal development.

Contents

1 Introduction **11**
1.1 Chapter 2: An integrated local-search/set-partitioning refinement heuristic for the Capacitated Vehicle Routing Problem . . . 12
1.2 Chapter 3: An Efficient Heuristic for Very Large-Scale Vehicle Routing Problems with Simultaneous Pickup and Delivery . . . 14
1.3 Chapter 4: Daily Planning of Acquisitions and Scheduling of Dynamic Downlinks for the PLATiNO Satellite . . . 15

2 An integrated local-search/set-partitioning refinement heuristic for the Capacitated Vehicle Routing Problem **17**
2.1 Introduction . . . 17
2.2 Previous work . . . 18
2.3 Algorithm Outline . . . 19
2.3.1 Phase 1: Lin, Kernighan and Helsgaun Heuristic . . . 20
2.3.2 Phase 2: Column Generation Filtering . . . 26
2.3.3 Phase 3: Restricted Set Partitioning Problem Optimization . . . 26
2.3.4 VRP Taxonomy . . . 27
2.4 Computational Results . . . 28
2.4.1 Original LKH vs New LKH . . . 29
2.4.2 Original LKH vs new LKH vs LS-CGH . . . 29
2.4.3 Statistical analysis of LS-CGH . . . 31
2.4.4 LS-CGH as a refinement tool for FILO . . . 31
2.4.5 CVRPLIB best-known solution improvements . . . 31
2.5 Conclusions . . . 33

3 An Efficient Heuristic for Very Large-Scale Vehicle Routing Problems with Simultaneous Pickup and Delivery **38**
3.1 Introduction . . . 38
3.2 Problem description . . . 40
3.3 Literature review . . . 40
3.4 Solution Approach . . . 43
3.4.1 An overview of FILO framework . . . 44
3.4.2 The FSPD framework . . . 46
3.5 Computational Results . . . 51
3.5.1 Implementation and Experimental Environment . . . 51
3.5.2 Parameter Tuning . . . 53
3.5.3 Testing on Instances from the Literature . . . 53
3.5.4 Testing on New Large-Scale Instances . . . 57
3.6 Algorithmic Components Analysis . . . 59
3.6.1 Segment Attributes Preprocessing . . . 59
3.6.2 Recreate Tuning . . . 62
3.6.3 Super-linear algorithmic phases management . . . 66
3.7 Conclusions . . . 68
3.8 Acknowledgments . . . 70
3.9 Appendix A: Complete results . . . 71
3.9.1 VRPSPD Instances . . . 71

3.9.2 VRPMPD Instances . . . 74

4 Daily Planning of Acquisitions and Scheduling of Dynamic Downlinks for the PLATiNO Satellite 81
4.1 Introduction . . . 81
4.2 Literature Review . . . 82
4.3 Problem Description . . . 83
4.4 Mathematical Formulation . . . 85
4.4.1 General scheduling constraints . . . 85
4.4.2 Memory management . . . 85
4.4.3 Relaxed Memory Management . . . 88
4.5 Genetic Algorithm Solution Approach . . . 89
4.5.1 Individual Representation . . . 90
4.5.2 Initial Population Generation . . . 90
4.5.3 Elitism . . . 90
4.5.4 Parent Selection . . . 91
4.5.5 Crossover and Elite Refinement . . . 91
4.5.6 Mutation . . . 92
4.5.7 Infeasibility Repair Procedures . . . 92
4.5.8 Local Search . . . 93
4.5.9 Indirect Dynamic Downlink Scheduling Algorithm . . . 94
4.6 Exact and Relaxed Formulation Approaches . . . 96
4.7 Math-Heuristic Competitors . . . 96
4.7.1 Relax and Repair Heuristic . . . 97
4.7.2 Strenghtened Discrete-Memory Relaxation . . . 97
4.7.3 Hybrid Genetic Algorithm - Relax and Repair Heuristic . . . 97
4.8 Computational Experiments . . . 97
4.8.1 Implementation and Experimental Environment . . . 98
4.8.2 Parameters Definition and Tuning . . . 98
4.8.3 Instances Description . . . 98
4.8.4 Upper Bound computation . . . 99
4.8.5 Heuristic Results . . . 102
4.9 Conclusions . . . 105

List of Figures

2.1 On the left, the average speedup of the "New" LKH with respect to the original version for 5 random seeds on the 57-X set. On the right the same chart including the 10 XXL instances. 29
2.2 Statistical analysis of percentage gap w.r.t. the best-known solution, on a representative subset of the 57-X and Belgium datasets. 33
2.3 Statistical analysis of percentage gap w.r.t. the best-known solution, on the ten instances of Table 2.2 where LS-CGH performed best in terms of relative gap. 34
2.4 Statistical analysis of percentage gap w.r.t. the best-known solution, on the ten instances of Table 2.2 where LS-CGH performed worst in terms of relative gap. 35

3.1 Illustration of segment concatenation associated with a swap move, in the case the depot is not contained in the segments to be swapped (a) or is contained in one of them (b). 49
3.2 Illustration of different insertion options. Although ν_1 and ν_2 are the closest customers to ρ, inserting ρ in this position would introduce a considerable detour. On the other hand, even if ν_3 and ν_4 are farther from ρ, the insertion cost of ρ between these two customers is smaller. 51
3.3 Illustration of the results on the first seven instances of Salhi and Nagy (1999) X (left) and Y (right) datasets. 56
3.4 Illustration of the results on the complete Salhi and Nagy (1999) X (left) and Y (right) datasets. 56
3.5 Computing time required by different FSPD versions as a function of problem size for X and XXL VRPSPD problem instances. 58
3.6 Typical evolution over time of the best solution found for XXL instances. The data refers to instance Flanders2X with seed 0. The different convergence speed of the algorithms is due to the simulated annealing criteria whose annealing schedule is based on the total number of iterations while the initial and final temperatures are kept fixed. 60
3.7 Plots of the average computing time with respect to instance size for the three REF preprocessing techniques 61
3.8 Plots of the average gaps sorted by size for the 4 recreate variants with recreate effort of 50 and 100. For each instance, the values obtained for the 5 H, Q, T, X, and Y are averaged together 64
3.9 Plots of the average computing time with respect to instance size for the 4 recreate variants with recreate effort of 50 and 100. For each instance, the values obtained for the 5 H, Q, T, X, and Y are averaged together 65
3.10 Plots of the average gaps sorted by size for the different levels of recreate effort and FILO's original recreate technique. 66
3.11 Plots of the average computing time with respect to instance size for the different levels of recreate effort and FILO's original recreate technique. 67
3.12 Speedup obtained substituting the C++ standard library sort algorithm, with Malte Skarupke's radix-sort implementation. 68

3.13 On the left, time comparison between preprocessing based on C++ standard library sort algorithm (blue) and Malte Skarupke's radix-sort implementation (red). For both series, the trend-line is also reported. On the left, the same plot at a different scale with the addition of the time needed by 100,000 coreopt iterations. The two intersection points represent a forecast of the size at which the preprocessing step will need the same amount of time as the actual refinement step. 69

4.1 Illustration of grouping intervals resulting from DTO and DLO overlapping. 87
4.2 High-level structure of the proposed genetic algorithm. 90
4.3 Memory saturation representation. Given the DLO l, the satellite memory saturation is computed before the start time of planned downlink activities (grayed areas) happening at times t_0, t_1 and t_2. 93
4.4 Comparison of average gaps obtained by the 5 ”heuristic“ algorithms. The median is shown at the left of each boxplot. 104
4.5 Memory profile computed with Algorithm IOM for a feasible plan (solid line) and a relaxed plan resulting from the continuous-memory model (dashed line). 104

List of Tables

2.1 Speedups of the modified LKH (newLKH) with respect to the original one. The size of the instances is computed as the number of customers plus the number of vehicles. Results are for the 57-X and XXL sets. (*) For the Flanders2 instance, the number of "trials" has been halved, since in the original algorithm $10,000$ "trials" would have been computationally too expensive. 30
2.2 Comparison between the solution obtained in 200 minutes runs by: the original LKH algorithm, Helsgaun's LKH with our changes and inserted in our scheme (newLKH), and our final LS-CGH algorithm (i.e., newLKH followed by RSP optimization). The best result for each instance is highlighted in boldface. 32
2.3 Best result for 10 runs of FILO with 10 million iterations for the largest 57 instances of the X data-set along with the improvement obtained after 200 minutes by our LS-CGH algorithm. For the XXL instances, SA was disabled due to their extremely large size. Entries in boldface highlight the cases where LS-CGH was able to improve the FILO solution. 36
2.4 Best result for 10 runs of FILO with 10 million iterations for the XXL dataset along with the improvement obtained after 200 minuted by our LS-CGH algorithm. 37
2.5 CVRPLIB best-known solution improvements by date. For the current best at the time of writing (March 2021), the following code identifies the authors of the algorithm: (1) Francesco Cavaliere, Emilio Bendotti, and Matteo Fischetti; (2) Eduardo Queiroga, Eduardo Uchoa, and Ruslan Sadykov; (3) Vinícius R. Máximo and Mariá C.V. Nascimento; (4) Thibaut Vidal; (5) Quoc Trung Dinh, Dinh Quy Ta, Duc Dong Do. 37

3.1 The CPU models used by algorithms from the literature and their single thread PassMark scores. 54
3.2 Results on the first seven instances of Salhi and Nagy (1999) CMTX and CMTY datasets. Average gaps marked by an asterisk are actually the best gap obtained along several runs. 55
3.3 Results on the complete Salhi and Nagy (1999) CMTX and CMTY datasets. Average gaps marked by an asterisk are actually the best gap obtained along several runs. 55
3.4 Results on the Dethloff (2001) and Montané and Galvão (2006) datasets. Average gaps marked by an asterisk are actually the best gap obtained along several runs. 57
3.5 Results on the first seven instances of Salhi and Nagy (1999) CMTH, CMTQ, and CMTT datasets. Average gaps marked by an asterisk are actually the best gap obtained along several runs. 57
3.6 Results on the complete Salhi and Nagy (1999) CMTH, CMTQ, and CMTT datasets. Average gaps marked by an asterisk are actually the best gap obtained along several runs. 57
3.7 Results on the new large-scale VRPSPD XX, XY instances. 59
3.8 Results on the new very large-scale VRPSPD XXLX, XXLY instances. 59
3.9 Results on the new large-scale VRPMPD XH, XQ, and XT instances. 59
3.10 Results on the new very large-scale VRPMPD XXLH, XXLT, XXLQ instances. 60
3.11 Average computing time (in seconds) for the three REF preprocessing techniques . . . 61
3.12 Average gaps for the 4 recreate variants with recreate effort of 50 and 100. For each instance, the values obtained for the 5 H, Q, T, X, and Y are averaged together 63

3.13 Average computing time (in seconds) for the 4 recreate variants with recreate effort of 50 and 100. For each instance, the values obtained for the 5 H, Q, T, X, and Y are averaged together . . . 63
3.14 Average gaps for the different levels of recreate effort and FILO's original recreate technique. . . . 64
3.15 Average computing time (in seconds) for the different levels of recreate effort and FILO's original recreate technique. . . . 65
3.16 Average gaps and computing time obtained using different fractions of the instance size as recreate-effort. . . . 66
3.17 Full results for dataset CMTX . . . 71
3.18 Full results for dataset CMTY . . . 71
3.19 Full results for dataset Dethloff . . . 72
3.20 Full results for dataset Montane . . . 72
3.21 Full results for dataset XX . . . 73
3.22 Full results for dataset XY . . . 74
3.23 Full results for dataset XXLX . . . 75
3.24 Full results for dataset XXLY . . . 75
3.25 Full results for dataset CMTH . . . 75
3.26 Full results for dataset CMTQ . . . 75
3.27 Full results for dataset CMTT . . . 76
3.28 Full results for dataset XH . . . 77
3.29 Full results for dataset XQ . . . 78
3.30 Full results for dataset XT . . . 79
3.31 Full results for dataset XXLH . . . 80
3.32 Full results for dataset XXLQ . . . 80
3.33 Full results for dataset XXLT . . . 80

4.1 Summary of medium instances and their key characteristics. Please note that M_{init} and M_{fin} are expressed as percentages of the total memory capacity, M_{tot}. . . . 100
4.2 Summary of large instances and their key characteristics. Please note that M_{init} and M_{fin} are expressed as percentages of the total memory capacity, M_{tot}. . . . 101
4.3 Relative percentage gaps computed by CPLEX on the medium dataset with two relaxed and two exact formulations, each within a one-hour time limit and executed with 32 threads. Bold indicates bounds matching the best-known solution (found by heuristics or during CPLEX optimization), and $*$ denotes cases where the time limit was reached. . . . 101
4.4 Relative percentage gaps achieved by CPLEX for the large dataset using the two relaxed formulations and two exact formulations. The optimization process was constrained to a one-hour time limit and executed with 32 threads. Bold values indicate bounds that match the best-known solutions, which were found either through heuristics or during CPLEX optimization. The symbol $*$ indicates that the optimization reached the time limit, while $\star$ signifies that there are instances with unbounded results excluded from the average. . . . 102
4.5 Average and best relative percentage gaps achieved by the two exact formulations and the three heuristics. These results were obtained within a time limit of 5 minutes and executed with a single thread. Values matching the best-known solutions are highlighted in bold. . . . 102
4.6 Average and best relative percentage gaps achieved by the two exact formulations and the three heuristics. These results were obtained within a time limit of 5 minutes and executed with a single thread. Values matching the best-known solutions are highlighted in bold. . . . 103

List of Algorithms

1 High-level pseudocode for the LS-CGH algorithm. 20
2 High-level pseudocode for the LKH algorithm. 21
3 Simplified representation of the LINKERNIGHAN function inside the LKH algorithm . . 22

4 High-level structure of FILO. 44
5 Structure of a function that, given a customer to insert back into a solution, finds a good insertion position. Note that we assume costs are symmetric. 52

6 High-level pseudocode for the IOM algorithm. 95

Chapter 1

Introduction

Combinatorial optimization is a discipline located at the intersection of mathematics, computer science, and practical problem-solving. It carries a profound historical legacy, with its roots extending into ancient times when individuals confronted challenges related to efficiency, resource management, and logistical matters. In the present day, it stands as a cornerstone of contemporary computational science, providing methodologies to tackle complex decision-making issues.

Fundamentally, combinatorial optimization is a discipline dedicated to discerning the best solution from a set of alternatives. It explores the intricate relationship between combinatorial choices and the objective functions that guide them. Its historical origins are evident in classic problems like the Traveling Salesperson Problem, which revolves around finding the most cost-effective route for a merchant to visit multiple cities, and the Knapsack Problem, which focuses on optimizing the value of carried items while adhering to capacity constraints.

As the world advanced, so did the field of combinatorial optimization. In the mid-20th century, renowned mathematicians and computer scientists, such as Dantzig, Bellman, and Ford, developed groundbreaking algorithms that formed the basis for solving combinatorial problems. These influential efforts led to the creation of methods like linear programming, network flows, and dynamic programming, which played a crucial role in various real-world applications.

The practical uses of combinatorial optimization are vast, as it applies to many different industries and fields. It helps improve manufacturing, manage resources effectively, and organize tasks efficiently.

This thesis explores two main optimization challenges, vehicle routing and satellite activity planning. Our goal is to contribute to the field of optimization by offering solutions and showcasing how they work in real-world situations. In the upcoming chapters, we will dive into each of these areas, unraveling their complexities and highlighting the progress made.

One of the main topics of this work regards the Vehicle Routing Problem (VRP) and, more specifically, how to handle different variants of this problem seamlessly. VRPs are a group of optimization problems that have gained significant attention due to their practical importance in transportation and logistics. The VRP family includes well-known variations such as the Capacitated Vehicle Routing Problem (CVRP), the Vehicle Routing Problem with Time Windows (VRPTW), the Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD), and others. These problems aim to find the best routes for a group of vehicles to serve a set of customers while considering constraints on vehicle capacity, time windows, and minimizing costs.

VRPs find applications in several areas, including:

- Field Service Management: VRP streamlines field service calls, which reduces travel time and costs.
- Healthcare Services: VRP plans ambulance routes and home healthcare visits more efficiently.
- Last-Mile Delivery: E-commerce platforms use VRP to optimize the final package delivery routes.

- Logistics and Supply Chain: VRP optimizes delivery routes, which reduces transportation costs.
- Public Transportation: VRP improves bus and train schedules, leading to better passenger service.
- Waste Management: VRP schedules waste collection routes, making it more cost-efficient.

The adaptability of VRPs to address real-world routing and scheduling challenges in these areas makes it an invaluable tool for enhancing operational efficiency, cost reduction, and service quality.

1.1 Chapter 2: An integrated local-search/set-partitioning refinement heuristic for the Capacitated Vehicle Routing Problem

In Chapter 2, we dive into the world of solving large-scale instances of the Capacitated Vehicle Routing Problem (CVRP). This particular challenge involves the efficient construction of routes for a group of delivery trucks to serve various customers while keeping within each truck's capacity limit. Our contribution lies in the design of an effective heuristic algorithm specifically tailored to tackle not only the challenges presented by this problem but potentially also some of its variants.

Let us now present a formal definition of the CVRP, which is the original VRP variant introduced by Dantzig and Ramser (1959).

Let $G = (V, E)$ be a weighted graph, where $V = \{0, 1, \ldots, n\}$ represents a set of $n + 1$ vertices (or nodes), and E is the set of edges connecting each pair of vertices. Vertex 0 corresponds to the depot, while the other vertices from 1 to n represent the customers. We denote $N = V \setminus \{0\}$ as the set of customer nodes.

The problem involves distributing goods from the depot to each customer. Each customer $i \in N$ is associated with a demand value $q_i \geq 0$, representing the quantity of goods requested.

Let $K = \{1, \ldots, |K|\}$ denote the fleet of vehicles, assumed to be homogeneous, meaning that each vehicle has the same maximum capacity Q. A vehicle serving a subset of customers $S \subseteq N$ starts at the depot, visits each customer in S once, and returns to the depot.

In the case where the travel costs are symmetric ($c_{ij} = c_{ji}$ for all $i, j \in V$), the graph $G = (V, E)$ is undirected. In situations where there exist vertices $i, j \in V$ with unequal travel costs depending on the direction ($c_{ij} \neq c_{ji}$), the problem is defined on a complete directed graph $G = (V, A)$, where $A = \{(i, j) \in V \times V : i \neq j\}$, and it is referred to as asymmetric.

A route for a single vehicle is represented as a sequence of vertices $p = (i_0, i_1, i_2, \ldots, i_s, i_{s+1})$, where $i_0 = i_{s+1} = 0$ (depot vertex), and the set $S = \{i_1, \ldots, i_s\} \subseteq N$, referred to as a cluster, includes all the customers visited within the route.

Each route is considered feasible if it satisfies to the following constraints:

- Each customer, denoted as i, is visited exactly once within the route.
- The total sum of customer demands does not exceed the maximum vehicle capacity, represented by Q.

A feasible solution for the CVRP entails a collection of $|K|$ routes, namely $p_1, \ldots, p_{|K|}$, meeting the following criteria:

- All routes, $p_1, \ldots, p_{|K|}$, adhere to the feasibility constraints.
- The set of clusters, namely $S_1, \ldots, S_{|K|}$, corresponding to these routes, collectively forms a partition of the customer set N.

A formal model for the symmetric Capacitated Vehicle Routing Problem (CVRP) is presented, as proposed by Laporte, Nobert, and Desrochers (1985):

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{1.1a}$$

$$\sum_{(i,j)\in\delta(j)} x_{ij} = 2, \quad \forall j \in N \tag{1.1b}$$

$$\sum_{(i,j)\in\delta(S)} x_{ij} \geq 2\gamma(S), \quad \forall S \subseteq N \tag{1.1c}$$

$$x_{ij} \in \{0,1,2\}, \quad \forall (i,j) \in \delta(0) \tag{1.1d}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E \setminus \delta(0) \tag{1.1e}$$

In this context, $\gamma(S)$ signifies the minimum number of vehicles needed to serve all customers within set S. Computing this value involves addressing the NP-Hard Bin Packing problem for the customer subset S. Alternatively, a feasible lower bound, denoted as $\overline{\gamma}(S)$, for $\gamma(S)$ can be determined using the formula:

$$\overline{\gamma}(S) := \left\lceil \sum_{i\in S} \frac{q_i}{Q} \right\rceil$$

The binary variables x_{ij} represent whether edge (i,j) has been selected:

$$x_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \text{ is part of the solution} \\ 0 & \text{Otherwise} \end{cases}$$

For a vertex subset $S \subset V$, the notation $\delta(S)$ denotes the set of edges with exactly one node in S. When edges are directed, $\delta^+(S)$ and $\delta^-(S)$ are defined as the sets of in-arc and out-arc edges of S respectively. Additionally, for simplicity, $\delta(i) := \delta(\{i\})$ for all $i \in V$.

Concerning the model formulation, the objective function (1.1a) involves minimizing the cost of all selected edges. Constraints (1.1b) ensure that each customer node has precisely two selected incident edges. Constraints (1.1c), known as Generalized Subtour Elimination Constraints (GSECs), handle Capacity Constraints, ensuring the feasibility of each route.

An alternative and common model for the CVRP (which is also valid for the VRPTW and VRPSPD), is the following Set-Partitioning (SP) formulation.

$$\min \sum_{p\in\Omega} c_p\theta_p \tag{1.2a}$$

$$\sum_{p\in\Omega} \theta_p = k \tag{1.2b}$$

$$\sum_{p\in\Omega_i} \theta_p = 1, \quad \forall i \in N \tag{1.2c}$$

$$\theta_p \in \{0,1\} \tag{1.2d}$$

where the new symbols are defined as follows:

- Ω: Set of all the feasible vehicle routes in G.
- c_p: Cost associated to each route $p \in \Omega$.
- Ω_i: subset of Ω routes that visit the customer $i \in N$.
- θ_p: binary value which is 1 if the route is in the optimal solution, 0 otherwise.

Constraints (1.2b) fix the number of routes equal to the number of vehicles, while (1.2c) are the Set-Partitioning constraints that ensure that each customer is visited exactly once. The simplicity of this formulation has an evident drawback: the number of θ_p variables, one for every possible

feasible route on the graph. However, this formulation has shown to be very effective for many VRP variants.

At the heart of our algorithm lies the SP formulation just described. This approach neatly separates considerations related to route feasibility and the composition of these routes into high-quality solutions. Importantly, the proposed approach is versatile and can be readily extended to a wide array of VRP variants with minimal adjustments.

Our approach takes inspiration from problem decomposition techniques, combining a local search method with a restricted SP problem optimization. The local search phase harnesses the power of Helsgaun's LKH-3 algorithm, augmented with various implementation improvements. Concurrently, the restricted SP formulation is solved using a commercial Integer Linear Programming solver.

Decomposition methods are a vital tool in the realm of optimization, particularly for addressing complex and computationally expensive problems. When dealing with large or hard optimization tasks, employing a monolithic approach that involves all variables and constraints from a given formulation, can be prohibitively costly. Decomposition methods offer an effective strategy by breaking down the original complex problem into simpler subproblems. This approach leverages the inherent structure and characteristics of the problem to facilitate more efficient optimization.

Examples of decomposition methods include Benders decomposition, Dantzig-Wolfe decomposition, and Lagrangian Decomposition (Wolsey and Nemhauser (1999)). These methods have proven valuable for addressing optimization problems with an extensive number of constraints or variables in their mathematical formulations.

The SP formulation's generality makes it a versatile tool for tackling various VRP variants through a technique that is inspired by the Dantzig-Wolfe decomposition, where additional constraints, primarily affecting the feasibility of the routes, are implicitly represented by the route set.

Despite advancements in mathematical programming decomposition algorithms, they often fall short when confronted with larger CVRP instances. Real-world scenarios frequently involve significantly larger problem sizes, necessitating efficient heuristic algorithms that can deliver high-quality solutions within reasonable computing times.

The primary objective of this chapter is to introduce a refinement heuristic that possesses the capability to enhance top-quality solutions. This algorithm is designed to complement, rather than replace, state-of-the-art heuristics, in keeping with the spirit of many refinement heuristics in the literature.

1.2 Chapter 3: An Efficient Heuristic for Very Large-Scale Vehicle Routing Problems with Simultaneous Pickup and Delivery

Chapter 3 takes a deep dive into city logistics, focusing on the complexities of handling simultaneous pickup and delivery requests, often seen in crowded urban areas. The booming e-commerce industry and the need to manage product returns make efficient logistics a top priority. In this chapter, we present an algorithm that builds upon the Fast Iterated Local Search framework (FILO), as introduced by Accorsi and Vigo (2021) for the CVRP. First, it aims to compete effectively with the best algorithms designed for solving the VRPSPD and its specific variant known as mixed pickup and delivery (VRPMPD). Second, it strives to efficiently solve new benchmark instances for these problems, even when dealing with a large number of customers. All of this is accomplished while ensuring that the algorithm's computing time (empirically) scales linearly with the size of the problem.

The VRPSPD can be defined on an undirected graph $G = (V, E)$. As for the CVRP, the vertex set V can be partitioned into two subsets: $V = \{0\} \cup N$, designating the depot (0) and the customers (N). Associated with each edge $(i, j) \in E$ is a cost c_{ij}. Each customer $i \in N$ has a delivery demand $d_i \geq 0$, and a pickup demand $p_i \geq 0$. Both depot demands, d_0 and p_0, are set to zero.

A fleet of homogeneous vehicles with a capacity Q is stationed at the depot and is available for customer service. The fleet may be unlimited or of a fixed size in different problem contexts from existing benchmark instances. A solution for VRPSPD comprises a set R of Hamiltonian circuits, referred to as "routes". These routes, start at the depot, visit specific customer subsets, and return to the depot.

A solution is considered feasible if:

- Each customer is visited by exactly one vehicle.
- Each vehicle follows at most one route.
- Each vehicle route starts and ends at the depot.
- For every route, the pickup demand at the depot does not exceed Q.
- Vehicle capacity constraints are maintained after each customer visit, ensuring that no overloading occurs.

VRPSPD's NP-hard nature stems from its generalization of the well-known CVRP. Various problem formulations exist, based on two- or three-index decision variables. For an in-depth exploration of this problem, readers may refer to the comprehensive survey by Koç, Laporte, and Tükenmez (2020), which outlines four potential formulations.

The VRPMPD is a slight variant of VRPSPD. In VRPMPD, each customer i either has a positive pickup quantity p_i or a positive delivery quantity q_i, but not both. This distinction leads to simple adaptations of algorithms employed for VRPSPD.

Our comprehensive computational study validates the successful fulfillment of both our goals. However, adapting algorithms developed for the traditional CVRP to address more complex routing constraints poses non-trivial challenges. Additional constraints introduced in variants like the one considered in this chapter can significantly complicate the feasibility evaluation process.

With the traditional CVRP, the feasibility of route solutions can often be evaluated using a small set of route attributes, such as demand-sum and distance-sum. However, when considering variants like the VRPSPD, inserting or removing a customer from a route may lead to local infeasibilities, requiring feasibility checks at a customer-level granularity.

To address this issue and establish a generic framework for handling complex routing constraints, research efforts have led to various techniques. One notable approach is the concept of Resource Extension Functions (REFs), originally proposed by Desaulniers et al. (1998). This framework offers a flexible and efficient approach for handling diverse types of constraints across various Vehicle Routing Problems (VRPs). The framework decouples the search strategy from the computation of a global state (like in Kindervater and Savelsbergh (1997)), making it highly adaptable to different VRP variants. A practical application of this framework is the segment REFs of Irnich (2008a), from which we took great inspiration to design the extension of FILO to the VRPSPD.

1.3 Chapter 4: Daily Planning of Acquisitions and Scheduling of Dynamic Downlinks for the PLATiNO Satellite

Chapter 4 marks a shift into the domain of satellite activity planning. PLATiNO, a Synthetic Aperture Radar Earth observation satellite launched in 2023, operates through carefully designed activity plans. These plans, generated routinely, encompass a spectrum of tasks, including acquisitions, maneuvers, and downlinks. The core objective of these plans is to maximize the satellite's profitability by accommodating as many requests as possible, all while staying within the boundaries of platform constraints. To address this intricate challenge, we introduce a genetic algorithm. This algorithm is coupled with repair procedures and local search operators, meticulously designed to promptly fix infeasible solutions and uncover high-quality local optima. Additionally, we harness the capabilities of Mixed Integer Linear Programming formulations to provide tight bounds on the optimal solution values and propose hybrid math-heuristics that work as competitors to the proposed genetic technique. Our approach undergoes rigorous testing on practical benchmark instances, derived from

real-world data kindly provided by Thales Alenia Space Italia. The results demonstrate the algorithm's proficiency in computing near-optimal solutions within realistic computational time.

The scheduling of satellite activities, whether in a single-satellite or multi-satellite environment, has garnered considerable attention in the literature. While in this chapter we will focus predominantly on single-satellite problems, it is important to acknowledge that satellite scheduling challenges span diverse industrial scenarios, each featuring specific satellite models, unique features, and constraints. Typically, real-world satellite scheduling problems demand heuristic approaches due to the constraint of producing high-quality solutions within limited computational time. The choice of the appropriate scheduling technique often hinges on the specific satellite model employed. For example, satellites like PLATiNO and SPOT5 have fixed acquisition times determined in advance, while Agile satellites offer greater flexibility in adjusting acquisition times within specified windows. The literature underscores the importance of capturing satellite memory constraints, as well as efficient scheduling of downlink activities. Although these aspects are often neglected or approximated in many studies, they play a critical role in optimizing satellite operations.

With the PLATiNO satellite, the scheduling of both acquisition and downlink activities assumes primary significance. In addition, the strategy must efficiently schedule the transmission of acquisitions to ensure memory occupation remains within capacity limits. This task is further complicated by the potential overlap of downlink operations with other scheduled tasks, creating a dynamic interdependence between task selection and the resulting downlink windows. Indeed, a fundamental aspect of satellite activity planning revolves around the choice of a downlink strategy. The efficient transmission of satellite acquisitions to Earth is crucial for the overall effectiveness of satellite operations. This strategy must consider key factors such as priority levels, aging methods, and acquisition categories, which significantly influence downlink activities.

Our algorithm is designed to allow a clear separation between the downlink strategy and the broader planning process, enabling specialization for distinct usage scenarios. In our work, we adopt a simplified yet practically significant strategy that maintains the core principles of these specializations and their defining characteristics.

Chapter 2

An integrated local-search/set-partitioning refinement heuristic for the Capacitated Vehicle Routing Problem

In this chapter, an effective heuristic algorithm for large-scale instances of the Capacitated Vehicle Routing Problem is proposed. The technique consists in a local search method entangled with a restricted Set Partitioning problem optimization. Helsgaun's LKH-3 algorithm has been used for the local search phase, with a number of implementation improvements. The restricted Set Partitioning formulation is solved by means of an exact commercial Integer Liner Programming solver. The resulting algorithm is able to consistently improve the solutions obtained by a state-of-the-art heuristic from the literature, as well as some of the best-know solutions maintained by the CVRPLIB website.

2.1 Introduction

Firstly introduced by Dantzig and Ramser Dantzig and Ramser (1959), Vehicle Routing Problems (VRPs) are a class of problems calling for a minimum-cost set of vehicle routes to serve a given set of customers with known demands.

The Capacitated Vehicle Routing Problem (CVRP) is one of the most studied VRP versions, in which the transportation request consists of the distribution of goods from a single depot to a set of customers using homogeneous vehicles with a limited capacity. In the symmetric case, it can be defined on a complete undirected graph $G = (V, E)$ with edge costs c_e's and a special depot node d. Each customer node $i \in N = V \setminus \{d\}$ is characterized by its demand $q_i \geq 0$ which represents the amount of goods requested, while each vehicle route must start and finish at d and has to visit a set of customers whose total demand does not exceed a given capacity C. The overall number of vehicles to be used is often fixed in advance.

Historically, many mathematical formulations have been proposed for this problem Laporte, Nobert, and Desrochers (1985), Toth and Vigo (2014). Particularly relevant for our work is the so-called Set-Partitioning (SP) formulation, common to many other VRP variants. In the SP formulation, the objective is to find the best combination of feasible routes that partitions the customer nodes of the

graph, minimizing the overall cost, i.e.:

$$\min \sum_{p \in \Omega} c_p \theta_p \tag{2.1a}$$

$$\sum_{p \in \Omega} \theta_p = k \tag{2.1b}$$

$$\sum_{p \in \Omega_i} \theta_p = 1, \quad \forall i \in N \tag{2.1c}$$

$$\theta_p \in \{0, 1\}, \quad p \in \Omega \tag{2.1d}$$

where Ω is the set of feasible routes for the CVRP, c_p is the cost associated to each route $p \in \Omega$, $\Omega_i \subset \Omega$ is the subset of routes that visit the customer $i \in N$, k is the required number of routes, and θ_p is a binary variable which is 1 if the route p is in the optimal solution, 0 otherwise.

An important aspect of the SP formulation is its generality, as it easily extends to all VRP variants where the additional constraints only affect the feasibility of the routes, hence they are implicitly represented by the route set Ω. However, a main drawback is represented by the cardinality of Ω, which grows exponentially with the number of customers. To tackle this issue, only a subset of potentially-relevant routes is explicitly generated, and optimization techniques like Column Generation Ford Jr and Fulkerson (1958), Dantzig and Wolfe (1961) or Branch and Price Pecin et al. (2017), Fukasawa et al. (2006) are used. Within these schemes, a Restricted SP (RSP) formulation is iteratively solved, containing only a subset of routes.

Although several advanced mathematical programming decomposition algorithms have been proposed in the last few decades, only relatively small instances—containing only few hundred customers—have been solved to optimality Toth and Vigo (2014). Problems encountered in real-life scenarios are often substantially larger, thus efficient heuristic algorithms are the only option available to obtain good-quality solutions within acceptable computing times.

The aim of our paper is to design a powerful (yet time consuming) refinement heuristic which is able to improve top-quality solutions. Thus, our method is meant to be used on top of a state-of-the-art heuristic, more than to replace it. This is very much is the spirit of other refinement heuristics from the literature, whose quality is certified by the capability of improving state-of-the-art solutions in a final post-processing step.

The paper is organized as follows. Previous literature on CVRP heuristics is sketched in Section 2.2. In Section 2.3, the comprehensive strategy of our algorithm is described, along with the modifications and improvements applied. Extensive computational results are reported in Section 2.4, showing that our method is able to consistently improve the solutions obtained by a state-of-the-art heuristic from the literature, as well as some of the best-know solutions maintained in the CVRPLIB website Pecin et al. (2023). Some conclusions are finally drawn is Section 2.5.

2.2 Previous work

A brief outline of the CVRP heuristics that are most relevant for our work follows.

Helsgaun's Helsgaun (2009, 2017) heuristic, LKH-3 (whose code can be found in the dedicated website Helsgaun (2020)), is a penalty-based extension of the famous Lin and Kerninghan Lin and Kernighan (1973) heuristic (LK), able to tackle many VRP variants. Although less efficient with respect to other state-of-the-art CVRP heuristics, LKH-3 (from now on, just LKH) plays a prominent role in our work in that it is the building block of our local-search phase, so we next give a brief description of this method.

Originally designed for the Traveling Salesperson Problem (TSP), the LKH algorithm is based on the concept of r-Opt moves and r-optimality. In a r-Opt move, r edges from the current solution are replaced by other r edges in such a way that another solution is obtained Helsgaun (2009). A solution is said to be r-optimal if it is impossible to obtain a shorter tour by means of any r-Opt move Helsgaun (2009). It is also intuitive that, for $0 \leq r' \leq r$, an r-optimal tour is also r'-optimal, and for a tour of n city to be optimal, it must also be n-optimal. Furthermore, is also reasonable that the

probability for a r-optimal tour to be optimal grows with r Helsgaun (2009). However, the number of possible r-Opt moves grows rapidly with the number of nodes of the graph, making it impossible to fully explore the available moves for large values of r. For this reason, r is usually set to 2 or 3, as the algorithm rapidly loses efficiency for larger numbers. To overcome this limit, the LK heuristic introduces a scheme where the r value is decided at run-time, iteration after iteration. Initially, r is set to 2, its minimal value, and then it is gradually increased searching for new potential pairs with the following rationale: starting from the most "out-of-place" pair, the algorithm iterates searching for the new most "out-of-place" pairs of the remaining set, repeating the search multiple times Lin and Kernighan (1973). If an improvement is found, the search restarts from scratch, while it stops otherwise. For further information, the reader can refer to Helsgaun (2009) for a brief explanation, or to the original Lin and Kernighan's paper Lin and Kernighan (1973).

Vidal et al. Vidal et al. (2012) propose HGS, a hybrid genetic algorithm combining the effectiveness of their population based method with the Local-Search exploration of neighborhoods defined from a set of operators.

Arnold and Sörensen's Arnold and Sörensen (2019) knowledge-guided local search (KGLS) is an effective Local-Search heuristic which adopts three different neighborhood-defining operators along with a knowledge based penalization to avoid local optima.

Christiaens and Vanden Berghe Christiaens and Vanden Berghe (2020) develop a simple yet effective algorithm named *Slack Induction by String Removals* (SISR), consisting in a ruin-and-recreate local search heuristic.

In their recent work, Accorsi and Vigo Accorsi and Vigo (2021) propose FILO, a very efficient and effective iterated local search heuristic, which through the combination of acceleration and localization techniques is able to find state-of-the-art solutions for very large scale CVRP instances in a short computing time. The algorithm adopts a large number of operator-defined neighborhoods and a combination of a ruin-and-recreate scheme coupled with simulated annealing.

Sharing some similarities with the work presented in the present paper, Subramanian et al. Subramanian, Uchoa, and Ochi (2013) propose *Iterated Local Search with Set Partitioning* (ILS-SP), a hybrid algorithm merging the effectiveness of a competitive iterated local search heuristic along with the optimization a SP formulation that tries to heuristically find the best combination of the explored routes. The adoption of a SP optimization phase has been also studied for many other heuristic techniques, as in the works of Foster et al. Foster and Ryan (1976), Ryan et al. Ryan, Hjorring, and Glover (1993), Rochat et al. Rochat and Taillard (1995), Kelly et al. Kelly and Xu (1999), De Franceschi et al. De Franceschi, Fischetti, and Toth (2006), or Monaci and Toth Monaci and Toth (2006) for the Bin-Packing Problem.

Finally, Queiroga et al. Queiroga, Sadykov, and Uchoa (2021) propose a heuristic working as a refinement technique to improve the solution obtained by other heuristics. Exploring a large solution neighborhood, their algorithm is able to consistently improve near-optimal solutions. The adopted technique is POPMUSIC Taillard and Helsgaun (2019), a matheuristic Fischetti and Fischetti (2018) based on the VRPSolver Pessoa et al. (2020, 2021) exact solver for VRPs.

2.3 Algorithm Outline

The overall scheme of our approach can be subdivided into three main phases.

1. The LKH heuristics is executed, in parallel; from the solutions generated at the end of each "trial" of the core LK algorithm, routes are extracted to populate a pool (called the "route pool").
2. Considering the Linear Programming (LP) relaxation of the SP formulation, a column-generation pricing procedure is applied to "filter" the most meaningful routes from the pool.
3. The RSP formulation, considering only the selected routes, is solved with a given time limit.

The three phases above are iterated until a global time limit expires—or a maximum number of repetitions is reached.

The described algorithm has been called *Local Search - Column Generation Heuristic* (LS-CGH) since it uses the LKH heuristic to generate good candidate routes that are then fed to the RSP optimization.

To better differentiate between the different types of iterations (one nested into the other), the following terms will be used:

- in accordance with the naming adopted by the LKH algorithm, the term "trial" refers to a single pass of the core Lin-Kerninghan algorithm, ending when no more improving r-Opt moves can be found.
- A "run" is a set of successive "trials", each starting from the perturbed solution of the previous one.
- The sequence of a single execution of LKH, followed by Column Generation filtering and the RSP optimization, has been named "round".

Our LS-CGH algorithm then consists in a number of "rounds", repeating the three-phase scheme multiple times. Each round is linked to the next one as it exploits the best solution found as its initial solution, and also because the route pool is maintained between rounds.

A high-level representation of the three main phases of the algorithm is given in Algorithm 1. In the pseudocode, the following functions are used:

- LKH: Calls the LKH-based heuristic described in Section 2.3.1 and in Algorithm 2. Returns the best solution found by the algorithm (S), along with a populated route pool (P).
- CGFILTER: Applies the column-generation inspired filtering (described in Section 2.3.2) to the route pool.
- SOLVERSP: Solves the restricted Set Partitioning formulation with a black-box Integer Linear Programming (ILP) solver; see Section 2.3.3.

Algorithm 1: High-level pseudocode for the LS-CGH algorithm.

```
  Input  : Initial solution S.
  Output: The best solution found.
1 FUNCTION LS-CGH(S)
2 begin
3 |  for Round ← 1 to n_Rounds do
4 |  |  S, P ←LKH(S);
5 |  |  P′ ← CGFILTER(P);
6 |  |  S ← SOLVERSP(P′, S)
7 |  end
8 |  return S;
9 end
```

2.3.1 Phase 1: Lin, Kernighan and Helsgaun Heuristic

To integrate the LKH algorithm with our LS-CGH scheme—which has been implemented as multi-thread C++ project—and also to improve its efficiency, a number of customizations have been applied to the original Helsgaun's code available at Helsgaun (2020). A summary of the most relevant changes are reported next.

- Due to the extensive use of global variables and non-reentrant primitives in the *C* code, the algorithm was not "out-of-the-box" ready to be encapsulates into a multi-thread scheme. Therefore we have systematically modified all global variables storage making them "thread local", and we have substituted all the non-reentrant *C* primitives with their corresponding reentrant versions. After these changes, we were able to synchronize the code by means of a step-by-step execution implemented upon *pthread* barrier.
- An improved synchronization has been implemented to equalize the duration of parallel "runs".
- The Jonker and Volgenant's mTSP-to-TSP transformation has been implemented to adapt solutions generated by the RSP optimisation and make them compatible with the current LKH instance.

- A basic control interface has been added to control the execution of the LKH algorithm and to let successive LKH calls execute one after the other with a reduced overhead.
- A route extraction function has been implemented to obtain a suitable amount of diversified routes to fill the route pool.
- The caching system already adopted within the algorithm has been extended and slightly improved.
- The CVRP penalty function has been redesigned, improving its speed while maintaining the exact same behaviour as the original one.
- A Simulated Annealing (SA) scheme has been added on top of the original solution acceptance test, to improve the performance of the original algorithm and to perturb the initial solution in the attempt of escaping from local optima.

For the sake of clarity, in what follows we will call "newLKH" our modified version of the LKH. To give a clearer idea of the structure of the newLKH algorithm and of the introduced changes, a sketch of this variant is given in Algorithm 2. The overall scheme resembles the original LKH, since most of its logic is not affected by our changes. The two main additions are the route-extraction step (EXTRACTROUTES), and the Simulated Annealing acceptance test (SATEST) called on every solution returned by the LINKERNIGHAN function. To be more specific, the following functions appear in the pseudocode:

- COST: Returns the cost of the input solution.
- KICK: Perturbs the input solution; see Section 24.
- LINKERNIGHAN: Calls Helsgaun's implementation of the Lin-Kernighan heuristic on the input solution, possibly refining it; see Algorithm 3 for a simplified overview of the main steps of this phase.
- EXTRACTROUTES: Given a (possibly infeasible) tour, returns all its feasible routes.
- SATEST: Manages the current solution update according to the Simulated Annealing metaheuristic approach described in Section 24.
- TIMELIMITREACHED: Simple test that returns true if the given time limit for the phase 1 of the LS-CGH has been reached, false otherwise.

Algorithm 2: High-level pseudocode for the LKH algorithm.

```
    Input  : Initial solution S_init.
    Output: The populated route pool P and the best solution found S*.
1   FUNCTION LKH(S_init)
2   begin
3   |   for Run ← 1 to n_Runs do
4   |   |   S* ← S ← S_init;
5   |   |   for Trial ← 1 to n_Trials do
6   |   |   |   S ← KICK(S);
7   |   |   |   S ← LINKERNIGHAN(S);
8   |   |   |   P ← EXTRACTROUTES(S);
9   |   |   |   if COST(S) < COST(S*) then
10  |   |   |   |   S* ← S
11  |   |   |   end
12  |   |   |   S ← SATEST(S*, S);
13  |   |   |   if TIMELIMITREACHED() then
14  |   |   |   |   return S*, P
15  |   |   |   end
16  |   |   end
17  |   end
18  |   return S*, P
19  end
```

An overview of the LINKERNIGHAN function is provided in Algorithm 3, highlighting the positions of the "Penalty" and "Flip" functions (to be described in Section 24 and Section 2.3.4, respectively). The functions that appear in the pseudocode are as follows.

- BESTSPECIALOPTMOVE: Original LKH function which, given a solution, searches for a r-Opt move that improves it, considering a restrict set of moves specialized for routing problems. An array $M_{rOpt}[1..r]$ of 2-Opt moves and its size r are returned. The proposed move is thus represented as a sequence of r 2-Opt moves to be applied, in sequence, to produce the final r-Opt move; see Sections 24 and 2.3.4 for further details.
- FLIP: Original (for CVRP) or modified (for asymmetric problems) function that applies a single 2-Opt move to a solution; see Section 2.3.4 for details.
- PENALTY: Modified version of the original "Penalty" function that, given a solution, returns its infeasibility level; see Section 24 for details.

Algorithm 3: Simplified representation of the LINKERNIGHAN function inside the LKH algorithm

```
    Input  : Initial solution S.
    Output: The refined solution S.
 1  FUNCTION LINKERNIGHAN(S)
 2  begin
 3  |   P ← PENALTY(S);
 4  |   C ← COST(S);
 5  |   M_rOpt, r ←BESTSPECIALOPTMOVE(S);
 6  |   do
 7  |   |   Improved ← false;
 8  |   |   for t ← 1 to r do
 9  |   |   |   S ← FLIP (S, M_rOpt[t])
10  |   |   end
11  |   |   P' ←PENALTY(S);
12  |   |   C' ← COST(S);
13  |   |   if (P' < P) OR (P' = P AND C' < C) then
14  |   |   |   P ← P';
15  |   |   |   C ← C';
16  |   |   |   Improved ← true
17  |   |   else
18  |   |   |   for t ← r downto 1 do
19  |   |   |   |   S ←FLIP(S, M_rOpt[t])
20  |   |   |   end
21  |   |   end
22  |   while Improved;
23  |   return S
24  end
```

Our newLKH version containing all the speed-related optimizations (namely: the new *Penalty* function, the caching system and the new *Flip* function) is freely available, for research purposes Cavaliere (2021).

Speed improvements

Some of the most relevant changes aimed at speeding up the execution of the original LKH code are outlined next.

Cost function: To reduce the overhead related to the computation of distances between vertices, the LKH algorithm uses, since its first version, a clever caching system proposed by Bentley Bentley (1990). This caching system works with two arrays of the same size: one array is used to save the used distances, while in the other one the smaller of the two node indices is saved as a signature. The position of each distance-signature pair in their respective arrays is chosen with a fast hash function. Thanks to this simple mechanism, both Helsgaun and Bentley report that the time with TSP problems can be halved or more Bentley (1990), Helsgaun (2009).

In the LKH original cost function, several checks are performed before calling the computationally expensive distance function. Indeed, depending on the VRP version and other internal parameters, the required distance might have already been stored by previous operations. Thus, before calling the distance function, all these fields are checked. The cache is checked as a last step, only if none of the fields contains the required value. Even though the performed checks are usually less expensive

than a call to the distance function, searching all the places where the distance could have been stored (which are not located adjacently in memory) can be slower than a direct check of the cache which, very often, already contains the actual value required. For this reason, we have modified the original cost function moving the cache check ahead, in a small prologue (often inlined by the compiler even without *linking time optimization*, since it is defined in a shared header file) that first checks if the requested cost is already stored inside the cache. Only when this step fails, it proceeds by calling the remaining part of the cost function, performing all the field checks and, eventually, the final call to the distance function. Furthermore, since distance and signature are always accessed together, the subdivision into two distinct array have been modified into a single array containing the signature and its distance adjacent in memory, to improve the cache-locality of this system.

Forbidden function: The "Forbidden" function tells if a given edge is part or not of the given instance. A simple example of forbidden edges is the set of edges between depot copies—note that, in the Jonker and Volgenan's mTSP-to-TSP transformation Jonker and Volgenant (1988), multiple copies of the depot are introduced. This function is heavily used by the algorithm, as shown by our profiling. Since the caching mechanism proved to be a really effective improvement for the cost function, we have implemented an analogous mechanism for the Forbidden function, using again a small prologue to possibly skip not only all the checks made by the original one, but also the function-call overhead.

Balanced workload: As previously described, we have modified the original LKH source code to make it reentrant. The reason for this extensive modification has been the need of enabling a parallel execution of multiple instances of the LKH algorithm. However, running different threads in parallel, synchronized only at the beginning and at the end of each LKH call, often leads to an unbalanced situation where some threads take less time than others. This difference varies randomly with the status of the algorithm. To avoid the waste of potential computational resources, all the threads are synchronized such that each parallel run ends only when the slowest one has ended. In this way, fast runs (which sometimes are even twice as faster as the slowest one), can carry on with their "trials", avoiding to reach the phtread barrier early and then wait for the others to finish.

Some utility procedures have also been implemented to connect LKH with the remaining part of our LS-CGH scheme. We next describe two main components of such an interface: the route pool and the Jonker and Volgenant's solution transformation.

Route Pool: To store the routes extracted by the solutions generated by the LKH we have implemented a simple route pool. We have decided to use a data structure inheriting from C++ STL *std::unordered_set* to avoid duplicates while keeping the best version of each route within the same group of nodes. Every route is distinguished from the others by the set of visited customers (which are saved as a sorted list), while the actual customer sequence and the length of the routes are updated every time a better "duplicate" is found.

Jonker and Volgenant's solution transformation: An important transformation, proposed by Jonker and Volgenant Jonker and Volgenant (1988) and applied in LKH, is the mTSP-to-TSP conversion which transforms an instance with m salespersons into a TSP instance with $m-1$ copies of the depot. This transformation is used to reduce the search space, decreasing the symmetry of mTSP and other problems with multiple routes (e.g., CVRP). It is easy to see that when $m-1$ identical copies of the depot are introduced into the graph, for each tour there exists $m!$ equivalent tours which only differ by the order of the depot copies. This transformation deletes part of the edge of the graph, by assigning to some selected nodes two depot copies to which they are allowed to be connected with, and by forbidding the edges to the other depot copies—thus reducing the number of possible route permutations.

A problem we encountered interfacing the RSP phase with the LKH one, concerns the compatibility of the CVRP solutions produced. Indeed, the combination of routes with the Set-Partitioning ILP optimization does not consider the Jonker and Volgenant's mTSP-to-TSP transformation Jonker and Volgenant (1988) applied within the LKH algorithm. When the ILP optimization generates CVRP solutions, the transformation is applied to avoid the use of the forbidden edges. Our algorithm

follows the general directives advised in the original Jonker and Volgenant's paper Jonker and Volgenant (1988), namely:

1. Starting from a general CVRP solution, the routes are extracted and the depot is removed, obtaining a list of chains of customers.
2. The depot is copied, obtaining a number of depots equal to the number of vehicles.
3. All the chain endpoints (two for each chain) are considered. Accordingly to the transformation already in place within the current LKH instance, for each end point that results to be a *special* customer (in the sense of the Jonker and Volgenant's paper: a customer for which the transformation has assigned only two depot copies) the required depots are assigned.
4. Then the main cycle of the transformation begins. Starting from one, all the chains are concatenated one after the other, ensuring that all the *special* customers are not linked with forbidden depot copies.

New Penalty Function

Although quite effective in practice, the above improvements are of a minor theoretical relevance since they simply accelerate the algorithm without modifying its original scheme—or provide an interface for other modules to interact with it more freely. On the other hand, the *Penalty* function modification has been characterized by a more prominent re-design of one of the main bottleneck functions. LKH is characterized by a hard division between the penalty value of a solution, which correlates to a measure of the "amount of constraint violation", and the actual cost of the objective function. At run-time, LKH gives higher priority to the improvement (i.e., decrease) of the penalty, considering the edge-cost gain achieved by the proposed r-Opt move only when the penalty variation is zero.

For any given solution, the *Penalty* function computes the penalty value with a computational complexity linear in the size of the CVRP solution. Inside LKH, such a solution is represented by a TSP tour containing a number of depot copies equal to the number of vehicles (following the Jonker and Volgenant Jonker and Volgenant (1988) symmetry-breaking transformation). In what follows, the term "tour" will refer to this internal representation and it will not be a synonym for "route", which instead refers to the cycle covered by a singe vehicle.

The *Penalty* function is called inside the LKH to check a new proposed solution in the following way:

1. A new r-Opt move is found and stored (decomposed as a series of 2-Opt moves) within the LK function.
2. The move is applied to the best tour found in the current "trial", named *current* tour, obtaining a new *proposed* tour.
3. The penalty function is called to check the *proposed* tour.
4. If the *proposed* tour improves the penalty of the *current* tour, or keeps the penalty unchanged while improving its cost, it becomes the new *current* tour, otherwise the saved r-Opt move is reversed to obtain the original *current* tour.

Notice that, at any given time, the *proposed* and *current* tours are abstract concepts used to explain their role, while the tour stored in memory is actually one which is first modified and then eventually restored if it does not improve the previous one.

However, due to its strict policy requiring that the infeasibility level can never increase, the *Penalty* function frequently rejects new candidates solutions. As a matter of fact, in almost all our tests the function rejects the proposed tour more than 95% of the times, thus representing one of the main bottlenecks for the entire algorithm. This observation enabled us to optimize the original LKH scheme by speeding-up the frequent "rejecting" case, introducing a rarely executed "update" step, thus resulting in a significant performance improvement. Indeed, the main change to the original penalty function has been the restriction of the penalty checks to only the routes "touched" by the *proposed* r-Opt move. Since the penalty function is called at every new potential change of the tour, these are the only routes modified between successive calls of the penalty function.

As in the original code there is no route-related data structure, a basic one has been implemented to store the route penalty for the current tour. Then, for each node, a reference to its route-data is stored, in accordance to the current CVRP solution. Thanks to this additional information, one can efficiently retrieve the *current* penalties of the routes touched by the *proposed* r-Opt move, as they appear in the *current* tour.

As a further optimization, we observe that route penalties need to be stored only if the current tour penalty is not yet zero. Indeed, when a feasible CVRP solution has been found (and the current penalty is, therefore, zero), then the previous cumulative penalty of any subset of routes is also zero. Therefore the previously described step can be completely avoided to further speed up the function.

Finally, when a *proposed* tour is accepted, an update procedure needs to be executed to restore route-data consistency.

Simulated Annealing

To avoid to get stuck in local optima, the original LKH algorithm uses a so-called "kick" strategy, i.e., every time a "trial" of the core LK procedure cannot find any other move that improves the *current* solution, a random r-Opt move (usually a double bridge 4-Opt move Applegate et al. (1999), Applegate, Cook, and Rohe (2003), Helsgaun (2006), Martin, Otto, and Felten (1992)) is applied to the current solution and the LK procedure is called again. As previously explained, a single iteration of such scheme is named "trial" in the LKH context. This technique has however two shortcomings:

- When LK is applied over a TSP instance that maps the VRP one, the additional constraints applied through the penalties make the search space very sparse. Therefore, although effective with true TSP instances, it can result to be not powerful enough to perturb the solution and move from the current VRP local-optima.
- When a warmstart is provided to the algorithm, LKH starts from a potentially very good local optimum from which it is not able to move (especially if such a warmstart has been produced by previous iterations of the LKH algorithm itself). Therefore, a perturbing strategy able to lead the search trajectory away from this starting point and to explore new solution neighborhoods is needed.

As in the recent FILO heuristic Accorsi and Vigo (2021), we decided to integrate a Simulated Annealing (SA) Kirkpatrick, Gelatt, and Vecchi (1983) scheme into LKH, motivated also by the compatibility of the original penalty-based scheme with such a technique.

Two overlapping SA schemes have been implemented, one based on the number of "trials", and one based on the LKH time limit. During the execution, the temperature is decreased for both the SAs and the smaller one is considered for the actual SA acceptance test. In this way, when both the trial and the time limits are given, the algorithm can automatically adapt to fit the tighter of the two.

Inspired again by the SA implementation in FILO Accorsi and Vigo (2021), we have set up our SA scheme as follows:

- The ratio between the initial temperature and the final one has been fixed to 100.
- Adopting the terminology introduced in Section 2.3.1, let z be the cost of the proposed solution, z' be the cost of the current solution used as a starting point, and T^t be the temperature at the "trial" t of the algorithm. The solution z is accepted as new current solution if

  $$z - z' < T^t \cdot ln(U[0,1])$$

  where $U[0,1]$ is a uniform random variable in the $[0,1]$ range.
- Two distinct temperatures are maintained during the execution, namely: T^t_{trial} which represent the trial-based SA temperature, and T^t_{time} which is the temperature of the time-based one. The actual temperature T^t is computed as the minimum of the two. Therefore, the update formulas are:

  $$T^{t+1}_{trial} = 0.01^{1/MTRIAL} \cdot T^t_{trial}$$

$$T_{time}^{t+1} = 0.01^{\Delta t/TMAX} \cdot T_{time}^{t}$$
$$T^{t+1} = min\{\, T_{trial}^{t+1}, T_{time}^{t+1} \,\}$$

where $MTRIAL$ is the maximum number of "trials", Δt is the time lasted from "trial" t and "trial" $t+1$, and $TMAX$ is the time limit for the "run".

- Finally, the initial temperature is computed as the value of the best solution obtained after 50 "trials", multiplied by a factor c (say) defined as follows. As we aim for long runs, we have distinguished the initial part of the algorithm (where the objective is to find a good solution without getting stuck into local optima) from the second one (which tries to find improvements to the given initial solution). For the first part a factor c_z (say) has been used to scale the initial temperature when no initial solution is provided to the algorithm, while c_w (say) is the same factor when an initial solution is present—because provided externally or from previous rounds of the algorithm. After some preliminary computational tests, we have fixed $c_z = 2.5 \cdot 10^{-3}$ and $c_w = 5 \cdot 10^{-4}$.

2.3.2 Phase 2: Column Generation Filtering

The number of routes generated during the LKH execution is typically exceedingly large, hence a technique to select the best routes is essential for the efficiency of the whole algorithm.

Considering our heuristic context, we need to balance two aspects: efficiency of the column generation phase, and RSP optimization speed. To achieve the former, a set of policies built around the common objective of finding a good and relatively small subset of routes has been defined, from which the RSP optimization could start. The initial core set of candidate routes consists in the selection of the "best" $8,000$ routes from the ordered list of all routes, sorted by non-decreasing solution costs.

(Indeed, in our computational tests we have seen that values between $5,000$ and $10,000$ are adequate for fast runs where the Set-Partitioning phase needs to be fast to avoid introducing large slow-down for the whole LS-CGH algorithm.)

Starting from this core set, the following filtering techniques are applied:

1. The LP relaxation of the RSP containing only the initial set of route is iteratively solved using the dual simplex algorithm. At each iteration, the reduced costs of the routes still in the route pool are computed, saving the value of the most negative one, say $\bar{c}_{min} < 0$. At this point, the routes with a reduced cost less than $0.8 \cdot \bar{c}_{min}$ are added to the RSP, therefore inserting a number of potentially useful columns at each iteration. This pricing procedure stops when all reduced costs are nonnegative, or when a time limit is reached.
2. Since the previous policy often does not select enough routes, we also use a filtering criterion akin to the one proposed by Caprara et al. Caprara, Fischetti, and Toth (1999) for the solution of large-scale set covering problems. At every pricing iteration we also select, for each customer, the ten routes with smallest (possibly positive) reduced costs. The pricing procedure stops when the time limit is reached or when the cumulative sum of the reduced costs added during the previous iteration, becomes nonnegative.

To handle the case in which the pricing procedure selects too many routes, we have set as a hard bound value equal to $16,000$, i.e., twice the initial set size.

2.3.3 Phase 3: Restricted Set Partitioning Problem Optimization

The final step of our scheme consists in the solution of the RSP formulation. For this task we used a state-of-the-art commercial MIP solver (IBM ILOG CPLEX 12.10). Although this is an exact algorithm, it has been successfully integrated in our heuristic scheme by setting an aggressive time limit and by an early activation of its "polishing procedure" Rothberg (2007).

It is worth observing that, as an alternative to the SP formulation, a Set Covering formulation might be used, that would allow for route overlaps. (Note that multiple customer visits can be removed by a short-cut post-processing procedure, that for instances with costs satisfying the triangle property would even reduce the final solution cost.) However, as reported by Rochat and Taillard Rochat and

Taillard (1995), and confirmed by our own computational tests, the Set Covering formulation is significantly slower to solve by our MIP solver, so we preferred to stay with the SP formulation.

2.3.4 VRP Taxonomy

To position our technique within the VRP scientific literature and to give a clearer idea of its applicability to other VRP variants, we make use of the Pillac et al. Pillac et al. (2013) VRP taxonomy. Broadly speaking, VRPs can be classified by the point of view of the instance data evolution, in this sense that we have *static* problems where all the information is known beforehand, vs. *dynamic* problems where the information regarding the instance is known only during the optimization. Then, we have *deterministic* vs. *stochastic* problems: in the former, all information is known exactly, while in the latter the input data is modelled in the form of random variables. From the product of this two classifications, one obtains four different classes:

- static and deterministic;
- dynamic and deterministic;
- static and stochastic;
- dynamic and stochastic.

The technique proposed in the present work specifically aims at problems of the first category: static and deterministic, as this is the nature of our local search and set partitioning phases.

More precisely, our scheme can readily be extended to all the VRP variants characterized by solutions with independent routes (i.e., variants that can be represented through the SP formulation, needed for the SP-phase of our algorithm) and supported by LKH. Here is a brief list of possible candidates:

- Multiple Travelling Salesman Problem (m-TSP)
- Capacitated Vehicle Routing Problem (CVRP)
- Capacitated Vehicle Routing Problem with Time Windows (CVRPTW)
- Vehicle routing problem with backhauls (VRPB)
- Vehicle routing problem with backhauls and Time Windows (VRPBTW)
- Vehicle routing problem with mixed pickup and delivery (VRPMPD)
- Vehicle routing problem with simultaneous pickup and delivery (VRPSPD)
- Vehicle routing problem with mixed pickup and delivery and time windowsb (VRPMPDTW)
- Vehicle routing problem with simultaneous pickup and delivery and time windows (VRP-SPDTW)

Of course, for any such VRP variant one needs to implement a specialized feasibility check for the routes found in the LKH solutions, to ensure that only feasible routes are inserted into the route pool.

One could also extend our technique to other variants which are compatible with the TSP–tour representation and the LKH penalty system. In this case, the implementation would be more involved than in the previously cited variants (which are already supported by Helsgaun's algorithm) since, along with the definition of the Penalty function, also the the internal data structure should be modified and extended. Similarly, all prepossessing steps (including the instance file parsing, the application of potential reductions or other preprocessing operations that can simplify the search) should be revised to account for the new variant.

New Flip Function

Within LKH, most VRP variants undergo an ATSP-to-TSP transformation Jonker and Volgenant (1986), hence in what follows we will use the *symmetric* and *asymmetric* terms not to refer to the cost of the arcs in the original problem formulation, but to the cost of the arcs of the LKH internal representation of the problem. For instance, a symmetric CVRPTW instance is converted to an

asymmetric one so as to remove all the finite-cost arcs which are not feasible due to the time-window constraints. In this sense, among the above-mentioned variants, only the "m-TSP" and the "CVRP" variants are viewed as symmetric problems, while all the others are asymmetric.

Within the LKH algorithm, whenever a 2-Opt move is applied, a function named *Flip* is called to copy a portion (segment) of the TSP tour representation in its reversed order. The operation is part of every 2-Opt move, although sometimes it could be avoided by applying more complex r-Opt moves that maintain the orientation of every part of the solution. Within LKH, every r-Opt move is decomposed into a sequence of 2-Opt moves, hence every r-Opt move must go through different "flips". If naively implemented, each flip operation has a $O(n)$ complexity, and is often the main bottleneck of any r-Opt move-based algorithm.

To improve its overall performance, LKH exploits a clever data structure due to Fredman et al. Fredman et al. (1995). Three versions of the *Flip* function are implemented, with complexity $O(n)$ (naive doubled-linked list version), $O(\sqrt{n})$ (two-level tree), and $O(\sqrt[3]{n})$ (three-levels tree), respectively. In particular, the second one is usually adopted since it is able to maintain a good trade-off with the size of common instances.

We observed that most of the proposed r-Opt are rejected by the *Penalty* function. As the *Flip* function is called every time an r-Opt move is applied, in the very likely "rejection" case the solution undergoes two "flip" operations: one to produce a *proposed* tour, and another to restore the *current* tour. As a result, this function can be optimized by introducing an "update" step when a better solution is found, with a significant speedup for the most-common "rejection" case.

2.4 Computational Results

In the present section, we address the following questions:

- How effective are our improvements to the original LKH implementation, in particular in terms of speed?
- Is our overall refinement heuristic able to improve the solutions found, in long computing times, by a state-of-the-art CVRP heuristic such as FILO?
- Are we able to improve some best-known solutions from CVRPLIB library, thus providing an implicit comparison will the best methods from the literature—that arguably have been applied to the instances of this well-known library?

In the computational tests that follow, the Uchoa et al. Uchoa et al. (2017) X dataset has been used. Following Queiroga et. al Queiroga, Sadykov, and Uchoa (2021), this dataset was restricted to its largest 57 instances (called 57-X in what follows).

For speedup evaluation and for the final tests with the FILO heuristic, we also considered the XXL set Arnold, Gendreau, and Sörensen (2019) which contains 10 instances of size up to $30,000$ customers.

All the tests have been performed on Intel Xeon E3-1220 V2 CPUs, using up to 4 threads. We will refer to the *Gap* of a solution with respect to the currently Best-Known Solution (BKS), defined as:

$$Gap := \frac{Solution_value - BKS_value}{BKS_value}.$$

When not available, an initial solution can be obtained by using one of several constructive methods that LKH provides. In its default setting, a pseudo-random procedure is selected that takes into account the possible presence of some restrictions on the edges of the graph, like the presence of "fixed" edges. Another useful constructive CVRP algorithm implemented within LKH is the Clarke and Wright (CW) saving algorithm Clarke and Wright (1964). Our computational experience shows that, for the X dataset, the final solution quality does not depend too much on the selected constructive heuristic. For the bigger XXL instances,instead, CW is often superior to the pseudo-random one, as it starts from a solution that, even when infeasible, is of better quality. Thus, for the single-thread speedup tests described in Section 2.4.1 we use CW for the initialization. For the comparison with the original LKH in Section 2.4.2, instead, we use CW for the first thread, while for the remaining

threads we use the pseudo-random one to help increasing route pool variability. Notice that, for both newLKH and LS-CGH, only the very first round makes use of such an initialization, while the best solution found is used in the other rounds.

2.4.1 Original LKH vs New LKH

In this section, the original LKH is compared with our modified version. The comparison only addresses the LKH phase of LS-CGH (i.e., without RSP and route extraction), both run in single-thread for the same number of "trials". As the implemented LKH changes do not alter the search trajectory between the original version and the new one (when run in single-thread mode and when the same random seed is used), the two versions visit the same solutions sequence and perform the same algorithmic steps, hence producing the same final solution.

In Table 2.1 the speedup achieved by the new version is reported along with the size of the instance. (Since inside the LKH each solution in represented by a TSP tour of length equal to the number of customers plus the number of vehicles, we report this figure as the size of the instance.) Along with the 57-X test-bed, the 10 XXL instances of the Belgium data-set has been considered in order to evaluate the behaviour of the algorithm for a broader range of sizes.

For each test a single "run" of the LKH was executed, starting from a near-optimal warmstart. The number of "trials" has been set to 10,000 and 5 random seeds where tried for each instance. The reported speedup is the average of the 5 speedups obtained by each seed.

Figure 2.1 (left) shows how the speedup scales with the size of the instance of the 57-X set. The linear increase of the speedup with the size of the instances is further confirmed in Figure 2.1 (right) where also the very large XXL instances are considered.

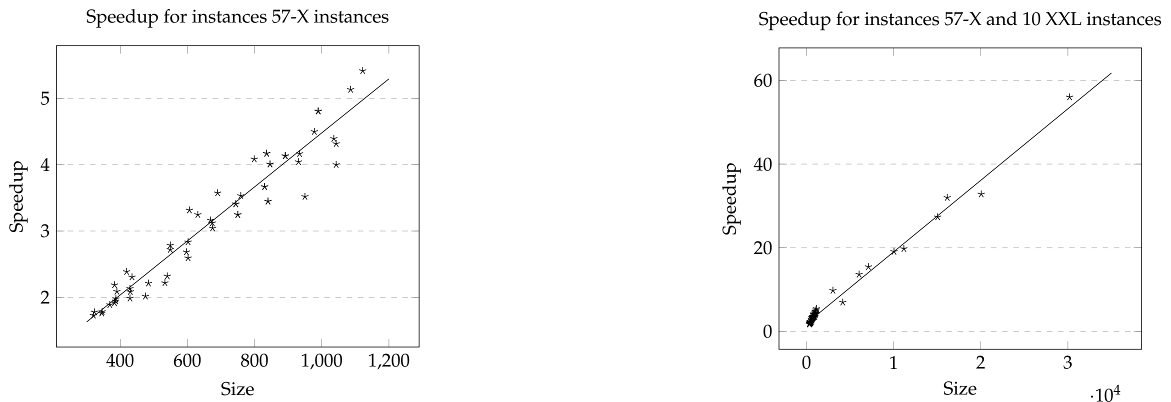


Figure 2.1: On the left, the average speedup of the "New" LKH with respect to the original version for 5 random seeds on the 57-X set. On the right the same chart including the 10 XXL instances.

2.4.2 Original LKH vs new LKH vs LS-CGH

In order to asses the effectiveness of the proposed scheme, three different variants have been compared. All the tests have been executed with the same time limit of 200 minutes, using 4 threads for both the LKH "runs" and the CPLEX solver (when used).

In Table 2.2 we compare the original LKH, the new LKH and our LS-CGH methods, and report the best gap reached (w.r.t the BKS) after 200 minutes. The "LKH" columns give the performance of the original LKH algorithm, without the proposed improvements and executed without the "round" subdivision adopted in our scheme. Four parallel threads with "runs" of 10,000 "trials" have been executed, until the time limit was reached. The "newLKH" columns give instead the solutions obtained by our new LKH scheme, without the SP phase. All the improvements applied to the original algorithm have been activated and the LKH "runs" (with 50,000 "trials" each and a time limit of 2000 seconds for each "run") have been subdivided into "rounds" of 4 parallel "runs", providing each round with the best solution found by the previous one. Finally, in the "LS-CGH" columns the

Table 2.1: Speedups of the modified LKH (newLKH) with respect to the original one. The size of the instances is computed as the number of customers plus the number of vehicles. Results are for the 57-X and XXL sets. (*) For the Flanders2 instance, the number of “trials” has been halved, since in the original algorithm $10,000$ “trials” would have been computationally too expensive.

Instance	Size	LKH Time	newLKH Time	SpeedUp
X-n303-k21	323	86	49	1.78
X-n308-k13	320	69	40	1.73
X-n313-k71	384	732	335	2.19
X-n317-k53	369	165	87	1.89
X-n322-k28	349	166	89	1.87
X-n327-k20	346	73	41	1.78
X-n331-k15	345	51	29	1.76
X-n336-k84	421	788	330	2.39
X-n344-k43	386	163	84	1.94
X-n351-k40	390	377	180	2.09
X-n359-k29	387	153	77	1.98
X-n367-k17	383	91	48	1.92
X-n376-k94	469	210	96	2.19
X-n384-k52	436	606	263	2.31
X-n393-k38	430	146	70	2.08
X-n401-k29	429	231	108	2.13
X-n411-k19	429	109	55	1.99
X-n420-k130	549	262	94	2.79
X-n429-k61	490	398	166	2.39
X-n439-k37	475	64	32	2.02
X-n449-k29	477	400	170	2.36
X-n459-k26	484	182	82	2.21
X-n469-k138	607	894	270	3.31
X-n480-k70	549	275	103	2.66
X-n491-k59	549	518	190	2.72
X-n502-k39	540	108	47	2.32
X-n513-k21	533	60	27	2.22
X-n524-k153	678	812	206	3.94
X-n536-k96	631	1145	353	3.25
X-n548-k50	597	185	69	2.68
X-n561-k42	602	130	50	2.59
X-n573-k30	602	248	87	2.83
X-n586-k159	744	571	149	3.82
X-n599-k92	691	2184	611	3.57
X-n613-k62	674	397	127	3.12
X-n627-k43	669	322	102	3.16
X-n641-k35	675	398	131	3.04
X-n655-k131	785	211	66	3.20
X-n670-k130	802	944	231	4.08
X-n685-k75	759	499	141	3.53
X-n701-k44	744	328	96	3.41
X-n716-k35	750	417	128	3.25
X-n733-k159	892	343	83	4.13
X-n749-k98	846	845	211	4.01
X-n766-k71	836	863	207	4.17
X-n783-k48	830	528	144	3.67
X-n801-k40	840	234	68	3.45
X-n819-k171	990	1791	373	4.81
X-n837-k142	978	582	129	4.50
X-n856-k95	950	186	53	3.52
X-n876-k59	934	760	182	4.16
X-n895-k37	932	950	235	4.04
X-n916-k207	1122	811	150	5.41
X-n936-k151	1092	884	172	5.13
X-n957-k87	1043	239	60	4.00
X-n979-k58	1036	919	209	4.39
X-n1001-k43	1043	434	101	4.32
Antwerp1	6342	1052	77	13.62
Antwerp2	7119	1992	129	15.43
Brussels1	15511	4287	157	27.34
Brussels2	16181	7660	240	31.98
Flanders1	20683	7037	215	32.80
Flanders2	30255	17166	306	56.01
Ghent1	10484	2028	106	19.07
Ghent2	11109	4469	226	19.75
Leuven1	3202	670	69	9.77
Leuven2	4045	710	102	6.94

results for our complete LS-CGH algorithm are reported, thus including the same setup as in the newLKH columns with the addition of the SP phase.

Both newLKH and LS-CGH show a significant decrease in the average gap, as well as a consistently lower gap for each instance in the 57-X set.

It is worth noting that the LKH algorithm involves a large number of parameters to tune: in our

tests, we used the default values provided in the scripts available in Helsgaun's website. In Table 2.2, a significant improvement is shown already by our own version of LKH (namely, newLKH). This is due to three main factors.

- The improved time performance of the algorithm allowed for the exploration of a larger number of *r*-Opt moves with respect to the original LKH.
- The SA in the first round, applied with a high initial temperature, takes better advantage of a large number of "trials". The search descent is therefore less steep (w.r.t. the number of "trials"), and also less prone to get stuck into local optima.
- The adopted "round" subdivision, in which the best solution obtained is used as warmstart for the next "round", greatly improves the efficacy of the algorithm to refine the solutions in long runs.

Finally, with the addition of CG filtering and RSP optimization, further improvements have been obtained.

2.4.3 Statistical analysis of LS-CGH

A statistical analysis of percentage gaps obtained for multiple runs on a representative subset of the studied instances has been carried out. From the 57-X dataset, we have chosen seven representative instances selected as suggested by Queiroga et al. Queiroga, Sadykov, and Uchoa (2021) so as to cover all the different characteristics considered during the generation of the whole X dataset. As to the Belgium data set, two (Antwerp1 and Flanders1) out of the ten instances have been randomly chosen. For these two instances, simulated annealing has been disabled because, for these sizes, the time limit is not enough to get stuck into local optima. Thus, the use of simulated annealing would only make local search slower without the benefit of the broader exploration that would happen with a much longer time limit. For each instance, ten runs with different random seeds have been executed, and the corresponding box-plots are reported in Figure 2.2.

According to the plot, a low variation is experienced for the Belgium instances. This can be explained by the fact that, for these very large problems, the 200-minute time limit is quite restrictive, hence the algorithm had less time to find local optima in which getting stuck. For the seven instances from the 57-X dataset, instead, the computing time allowed let the algorithm reach several local optima, hence the higher variance due to implemented diversification mechanisms—exceptional cases being the X-n469-k138 and X-n979-k58 instances with their outliers.

Figures 2.3 and 2.4 report a similar analysis for the ten instances in Table 2.2 for which LS-CGH got the best and worst relative gaps, respectively.

2.4.4 LS-CGH as a refinement tool for FILO

To asses the ability of improving the solution obtained by state-of-the-art heuristic algorithms, our proposed scheme has been tested starting from the best solution obtained by FILO Accorsi and Vigo (2021). As previously described, FILO is a recent fast and effective heuristic, especially designed for instances of very large size as those in the XXL dataset. The solutions obtained by FILO on a very large number of instances from the literature are available online Accorsi and Vigo (2020).

Our test consisted in a long run (200 minutes) of our algorithm starting from the best solutions obtained by the 10M-iteration runs of FILO. For each instance, we selected the best solution among those produced by FILO in 10 runs with different random seeds.

As shown in Tables 2.3 and 2.4, our LS-CGH algorithm is consistently able to improve many of the solution produced by FILO, lowering the average gap to 0.076% for the largest 57 instances of the X data-set, and to 0.079% for the XXL data-set.

2.4.5 CVRPLIB best-known solution improvements

During the months preceding the writing of the paper, our LS-CGH algorithm was consistently and repeatedly able to improve the best-known solutions (BKSs) for a number of instances from the literature, competing with many other algorithms developed by different groups around the world.

Table 2.2: Comparison between the solution obtained in 200 minutes runs by: the original LKH algorithm, Helsgaun's LKH with our changes and inserted in our scheme (newLKH), and our final LS-CGH algorithm (i.e., newLKH followed by RSP optimization). The best result for each instance is highlighted in boldface.

Instance	LKH		newLKH		LS-CGH	
	Sol	Gap	Sol	Gap	Sol	Gap
X-n303-k21	21877	0.65%	**21803**	**0.31%**	21805	0.32%
X-n308-k13	25995	0.53%	**25900**	**0.16%**	25919	0.23%
X-n313-k71	96097	2.18%	95330	1.37%	**94604**	**0.60%**
X-n317-k53	78409	0.07%	78361	0.01%	**78355**	**0.00%**
X-n322-k28	30061	0.76%	29968	0.45%	**29850**	**0.05%**
X-n327-k20	27800	0.97%	27640	0.39%	**27619**	**0.32%**
X-n331-k15	31289	0.60%	**31103**	**0.00%**	**31103**	**0.00%**
X-n336-k84	143175	2.92%	142122	2.16%	**141194**	**1.50%**
X-n344-k43	42417	0.87%	42201	0.36%	**42156**	**0.25%**
X-n351-k40	26343	1.73%	26133	0.92%	**26016**	**0.46%**
X-n359-k29	51807	0.59%	51652	0.29%	**51579**	**0.14%**
X-n367-k17	22955	0.62%	22824	0.04%	**22814**	**0.00%**
X-n376-k94	147807	0.06%	147720	0.00%	**147713**	**0.00%**
X-n384-k52	67082	1.73%	66403	0.71%	**66389**	**0.68%**
X-n393-k38	38519	0.68%	38335	0.20%	**38260**	**0.00%**
X-n401-k29	66485	0.50%	66481	0.49%	**66373**	**0.33%**
X-n411-k19	19890	0.90%	19780	0.34%	**19756**	**0.22%**
X-n420-k130	108247	0.42%	107946	0.14%	**107798**	**0.00%**
X-n429-k61	66135	1.05%	65742	0.45%	**65460**	**0.02%**
X-n439-k37	36559	0.46%	**36402**	**0.03%**	36422	0.09%
X-n449-k29	56118	1.60%	55569	0.61%	**55363**	**0.24%**
X-n459-k26	24508	1.53%	24226	0.36%	**24176**	**0.15%**
X-n469-k138	223542	0.77%	222320	0.22%	**222021**	**0.09%**
X-n480-k70	90031	0.65%	89698	0.28%	**89566**	**0.13%**
X-n491-k59	67355	1.31%	**66739**	**0.39%**	66894	0.62%
X-n502-k39	69317	0.13%	69254	0.04%	**69226**	**0.00%**
X-n513-k21	24428	0.94%	24268	0.28%	**24275**	**0.31%**
X-n524-k153	154662	0.04%	154616	0.01%	**154605**	**0.01%**
X-n536-k96	95924	1.14%	95224	0.40%	**95032**	**0.20%**
X-n548-k50	87031	0.38%	86836	0.16%	**86762**	**0.07%**
X-n561-k42	42998	0.66%	42854	0.32%	**42794**	**0.18%**
X-n573-k30	51053	0.75%	50835	0.32%	**50799**	**0.25%**
X-n586-k159	191487	0.62%	190593	0.15%	**190482**	**0.09%**
X-n599-k92	115113	6.14%	111324	2.65%	**110475**	**1.87%**
X-n613-k62	60467	1.57%	60136	1.01%	**59736**	**0.34%**
X-n627-k43	63000	1.34%	62395	0.37%	**62356**	**0.31%**
X-n641-k35	64551	1.36%	64205	0.82%	**64109**	**0.67%**
X-n655-k131	106943	0.15%	106857	0.07%	**106780**	**0.00%**
X-n670-k130	147052	0.49%	146812	0.33%	**146407**	**0.05%**
X-n685-k75	69310	1.62%	68554	0.51%	**68474**	**0.39%**
X-n701-k44	82933	1.23%	82521	0.73%	**82344**	**0.51%**
X-n716-k35	44186	1.87%	43637	0.61%	**43603**	**0.53%**
X-n733-k159	137622	1.05%	136477	0.21%	**136359**	**0.13%**
X-n749-k98	78682	1.83%	77863	0.77%	**77738**	**0.61%**
X-n766-k71	115728	1.15%	114910	0.43%	**114776**	**0.31%**
X-n783-k48	73497	1.53%	72822	0.60%	**72704**	**0.44%**
X-n801-k40	73976	0.92%	**73469**	**0.22%**	73484	0.24%
X-n819-k171	161871	2.37%	159287	0.74%	**159101**	**0.62%**
X-n837-k142	195666	1.00%	194453	0.37%	**194269**	**0.27%**
X-n856-k95	89473	0.57%	**89036**	**0.08%**	89102	0.15%
X-n876-k59	100297	1.01%	**99930**	**0.64%**	99986	0.69%
X-n895-k37	56497	4.90%	54827	1.80%	**54575**	**1.33%**
X-n916-k207	331620	0.74%	330093	0.28%	**329643**	**0.14%**
X-n936-k151	134163	1.09%	133169	0.34%	**133146**	**0.32%**
X-n957-k87	86197	0.86%	85606	0.16%	**85526**	**0.07%**
X-n979-k58	120354	1.16%	119977	0.84%	**119685**	**0.60%**
X-n1001-k43	74142	2.47%	**72820**	**0.64%**	72966	0.84%
Average		**1.16%**		**0.48%**		**0.32%**

The current BKSs are maintained in the CVRPLIB website Pecin et al. (2023), where the history of the obtained improvements is also reported. As stated in the website, everyone can submit new BKSs, without a description of the applied techniques. This fact has enabled a number of different "competitors" to submit many improvements, especially for the difficult instances of the X and XXL datasets. Different techniques have been applied to these instances, both refining heuristic starting from the previous BKS, and "standalone" ones starting from scratch.

In our case, for 30 large-scale well-studied instances from the CVRPLIB, we have been able to improve the BKSs from literature several times, providing a total of 105 improved BKSs. At the time of writing (March 2021), 14 BKSs produced by our LS-CGH heuristic are still unbeaten; see Table 2.5. After an initial testing phase where the ensemble of proposed techniques was still incomplete, all the new BKS have been obtained using the same parameter setting, with the only exception of the

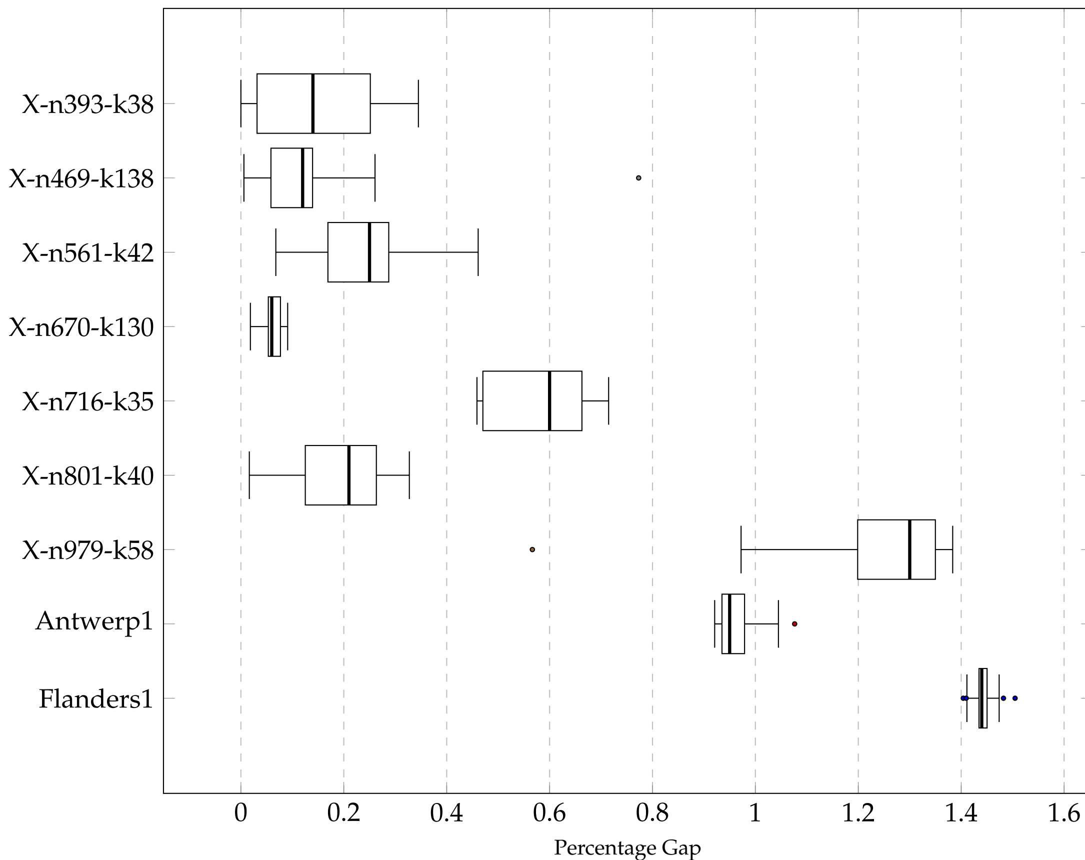


Figure 2.2: Statistical analysis of percentage gap w.r.t. the best-known solution, on a representative subset of the 57-X and Belgium datasets.

overall time limit which was set to infinity. Thus, for each instance we "manually" monitored the time lasted from the last improvement, and aborted the code when no improvement was found in the least 24 hours.

2.5 Conclusions

In this work a new CVRP refining heuristic, LS-CGH, has been proposed. We use a custom parallel and optimized version of the Lin-Kerninghan-Helsgaun heuristic to generate a large pool of feasible CVRP routes, and exploit an LP-based pricing procedure to "filter" the most meaningful ones to feed a Set Partitioning model producing the final CVRP solution. Our optimized version of the LKH heuristic is available, for research purposes, at https://github.com/c4v4/LKH3.

The LS-CGH algorithm succeeded in improving several of the best solutions obtained by a recent state-of-the-art heuristic (FILO) in 10M iterations. In addition, a log of the best-known solutions obtained in the past months by our method is publicly available on the CVRPLIB website Pecin et al. (2023), witnessing its ability to improve 105 solutions obtained by the best CVRP heuristics internationally competing on the same testbed.

In future work, our proposed method can be adapted to other routing problems, including the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW), the Capacitated Arc Routing Problem (CARP), the Vehicle Routing Problem with Backhauls (VRPB), and many others. Since LKH itself is able to address some of these VRP variants, it can be used as route generator as suggested in the present work.

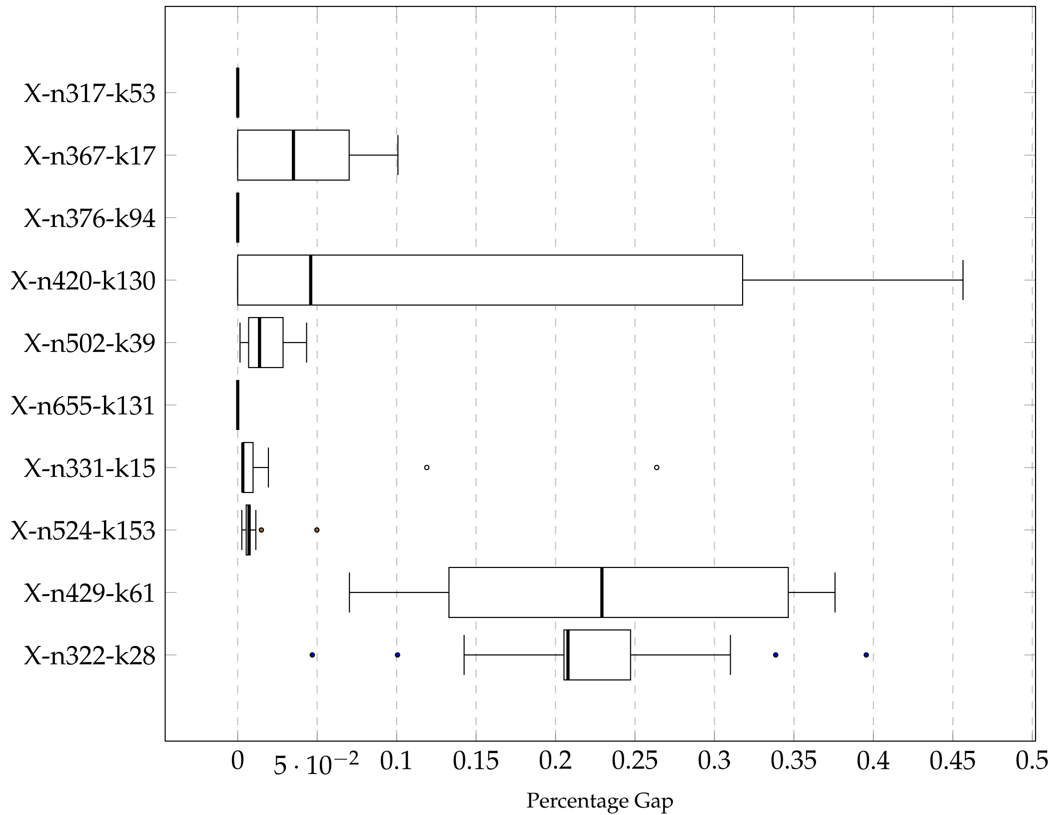


Figure 2.3: Statistical analysis of percentage gap w.r.t. the best-known solution, on the ten instances of Table 2.2 where LS-CGH performed best in terms of relative gap.

Acknowledgements

This work was partially supported by MiUR, Italy. We thank three anonymous referees for their constructive comments.

Statements and Declarations

Funding This work was partially supported by MiUR, Italy.

Conflict of interest The authors declare that have no conflict of interest.

Competing Interests The authors have no relevant financial or non-financial interests to disclose.

Author Contributions All authors contributed to the study and development of this work. All authors read and approved the final manuscript.

Data Availability The datasets used the current study are either public, or available from the corresponding author on reasonable request.

Code Availability The optimized version of the LKH heuristic is available, for research purposes, at https://github.com/c4v4/LKH3 (DOI 10.5281/zenodo.6644959)

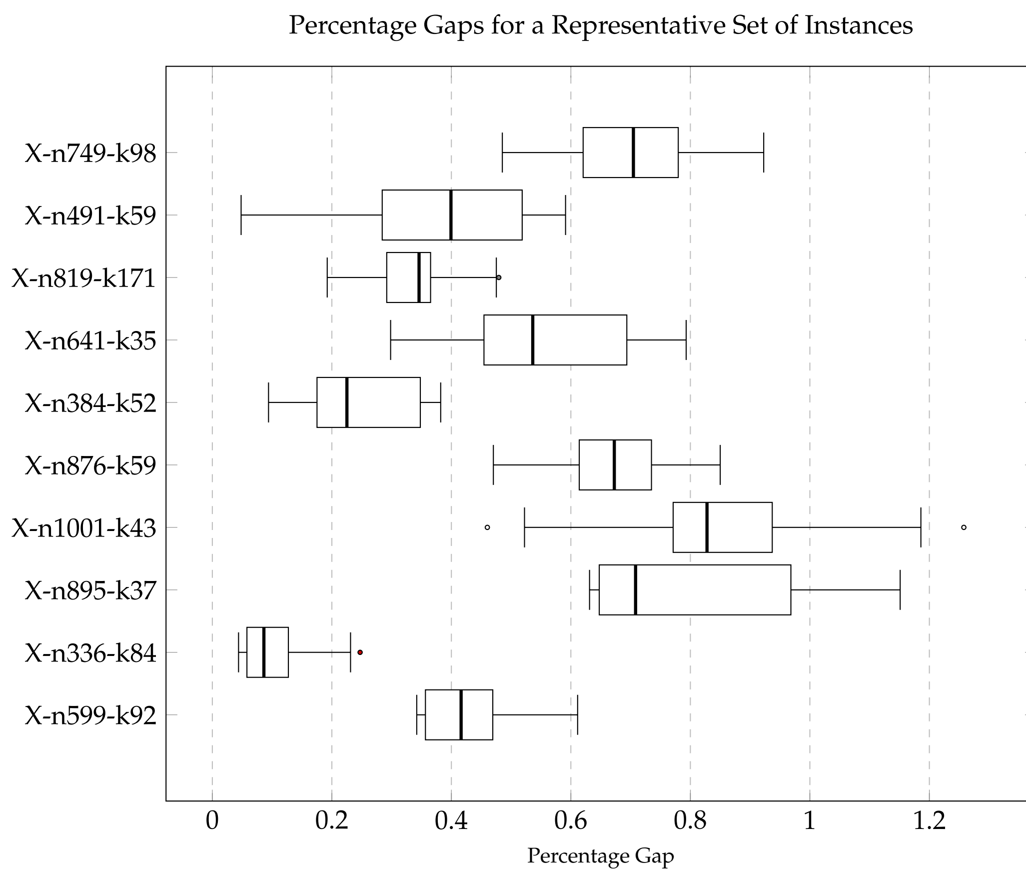


Figure 2.4: Statistical analysis of percentage gap w.r.t. the best-known solution, on the ten instances of Table 2.2 where LS-CGH performed worst in terms of relative gap.

Table 2.3: Best result for 10 runs of FILO with 10 million iterations for the largest 57 instances of the X data-set along with the improvement obtained after 200 minutes by our LS-CGH algorithm. For the XXL instances, SA was disabled due to their extremely large size. Entries in boldface highlight the cases where LS-CGH was able to improve the FILO solution.

Instance	FILO-10M		LS-CGH	
	Sol	Gap	Sol	Gap
X-n303-k21	21744	0.037%	21744	0.037%
X-n308-k13	25862	0.012%	25862	0.012%
X-n313-k71	94084	0.044%	94084	0.044%
X-n317-k53	78355	0.000%	78355	0.000%
X-n322-k28	29854	0.067%	29854	0.067%
X-n327-k20	27556	0.087%	27556	0.087%
X-n331-k15	31103	0.003%	31103	0.003%
X-n336-k84	139249	0.099%	**139195**	**0.060%**
X-n344-k43	42064	0.033%	42064	0.033%
X-n351-k40	25936	0.154%	**25922**	**0.100%**
X-n359-k29	51507	0.004%	**51505**	**0.000%**
X-n367-k17	22814	0.000%	22814	0.000%
X-n376-k94	147713	0.000%	147713	0.000%
X-n384-k52	66024	0.130%	**65996**	**0.088%**
X-n393-k38	38287	0.071%	**38269**	**0.024%**
X-n401-k29	66187	0.050%	66187	0.050%
X-n411-k19	19756	0.223%	**19755**	**0.218%**
X-n420-k130	107825	0.025%	**107798**	**0.000%**
X-n429-k61	65502	0.081%	**65455**	**0.009%**
X-n439-k37	36395	0.011%	36395	0.011%
X-n449-k29	55312	0.143%	**55280**	**0.085%**
X-n459-k26	24141	0.008%	**24140**	**0.004%**
X-n469-k138	222363	0.243%	**222038**	**0.096%**
X-n480-k70	89471	0.025%	**89457**	**0.009%**
X-n491-k59	66529	0.069%	**66491**	**0.012%**
X-n502-k39	69227	0.001%	**69226**	**0.000%**
X-n513-k21	24201	0.000%	24201	0.000%
X-n524-k153	154607	0.009%	**154605**	**0.008%**
X-n536-k96	95343	0.524%	**95278**	**0.453%**
X-n548-k50	86707	0.008%	**86704**	**0.005%**
X-n561-k42	42751	0.080%	42751	0.080%
X-n573-k30	50736	0.124%	50736	0.124%
X-n586-k159	190694	0.199%	**190686**	**0.194%**
X-n599-k92	108612	0.148%	**108609**	**0.145%**
X-n613-k62	59618	0.139%	**59572**	**0.062%**
X-n627-k43	62189	0.040%	**62184**	**0.032%**
X-n641-k35	63740	0.088%	**63735**	**0.080%**
X-n655-k131	106780	0.000%	106780	0.000%
X-n670-k130	147066	0.502%	**146481**	**0.102%**
X-n685-k75	68339	0.196%	**68318**	**0.165%**
X-n701-k44	81951	0.034%	**81950**	**0.033%**
X-n716-k35	43424	0.118%	**43417**	**0.101%**
X-n733-k159	136274	0.064%	**136265**	**0.057%**
X-n749-k98	77430	0.208%	**77399**	**0.168%**
X-n766-k71	114638	0.193%	114638	0.193%
X-n783-k48	72464	0.108%	**72457**	**0.098%**
X-n801-k40	73311	0.008%	**73307**	**0.003%**
X-n819-k171	158734	0.388%	**158703**	**0.367%**
X-n837-k142	193967	0.119%	**193948**	**0.109%**
X-n856-k95	89001	0.040%	**88966**	**0.001%**
X-n876-k59	99412	0.114%	99412	0.114%
X-n895-k37	53906	0.085%	**53898**	**0.071%**
X-n916-k207	329789	0.185%	**329660**	**0.146%**
X-n936-k151	133019	0.229%	**132999**	**0.214%**
X-n957-k87	85467	0.002%	85467	0.002%
X-n979-k58	119043	0.056%	119043	0.056%
X-n1001-k43	72414	0.082%	**72405**	**0.069%**
Average		**0.101%**		**0.076%**

Table 2.4: Best result for 10 runs of FILO with 10 million iterations for the XXL dataset along with the improvement obtained after 200 minuted by our LS-CGH algorithm.

Inst	**FILO-10M**		**LS-CGH**	
	Sol	**Gap**	**Sol**	**Gap**
Antwerp1	477619	0.072%	**477598**	**0.067%**
Antwerp2	291528	0.054%	**291493**	**0.042%**
Brussels1	502278	0.102%	**502217**	**0.090%**
Brussels2	345747	0.056%	**345706**	**0.044%**
Flanders1	7248491	0.106%	**7246624**	**0.080%**
Flanders2	4382341	0.163%	**4380571**	**0.122%**
Ghent1	469894	0.077%	**469860**	**0.070%**
Ghent2	258118	0.122%	**258090**	**0.111%**
Leuven1	192915	0.035%	**192909**	**0.032%**
Leuven2	111544	0.130%	**111541**	**0.127%**
Average		**0.092%**		**0.079%**

Table 2.5: CVRPLIB best-known solution improvements by date. For the current best at the time of writing (March 2021), the following code identifies the authors of the algorithm: (1) Francesco Cavaliere, Emilio Bendotti, and Matteo Fischetti; (2) Eduardo Queiroga, Eduardo Uchoa, and Ruslan Sadykov; (3) Vinícius R. Máximo and Mariá C.V. Nascimento; (4) Thibaut Vidal; (5) Quoc Trung Dinh, Dinh Quy Ta, Duc Dong Do.

Instance	Prev.BKS	3-May-20	18-May-20	17-June-20	30-July-20	8-Aug-20	20-Aug-20	10-Oct-20	15-Dec-20	30-Jan-21	BKS	Authors
X-n351-k40	25928	25919									25896	2
X-n384-k52	65943	65941									65938	3
X-n459-k26	24141						24140				24139	4
X-n536-k96	94950	94921									94846	2
X-n561-k42	42722	42717									42717	1
X-n573-k30	50717	50708	50673								50673	1
X-n641-k35	63737	63723							63684		63684	1
X-n670-k130	146446	146332									146332	1
X-n685-k75	68252	68245									68205	4
X-n716-k35	43414	43412							43373		43373	1
X-n766-k71	114487		114456						114417		114417	1
X-n783-k48	72393								72386		72386	1
X-n801-k40	73311						73305				73305	1
X-n819-k171	158249	158247									158121	2
X-n876-k59	99331	99330	99303								99299	4
X-n895-k37	53946	53935	53928				53870				53860	4
X-n936-k151	132907	132881	132812								132715	2
X-n957-k87	85478	85474									85465	4
X-n979-k58	119008	118996									118976	2
X-n1001-k43	72402	72397	72369								72355	2
Antwerp1	479021	478775	478674	478091	478019	477535					477277	2
Antwerp2	294319	293953	293802	292597	292511	291468	291450	291400	291387	291371	291371	1
Brussels1	504392	504175	504023	503407	503350	502144	501916	501854	501767		501767	1
Brussels2	353285	352658	352012	349602	348740	345627	345616	345565	345553		345551	5
Flanders1	7273695	7272444	7270362	7256529	7256400	7245214	7242182	7241290	7240845		7240845	1
Flanders2	4480972	4469477	4455217	4405678	4402841	4378434	4377986	4377626	4377524	4375193	4375193	1
Ghent1	471084	470902	470818	470329	470306	469838	469602	469586			469532	4
Ghent2	261676	260987	260553	259712	259486	258010	258002	257958	257954	257802	257802	1
Leuven1	193343	193244	193220	193092	193059	192894					192848	4
Leuven2	112751	112378	112280	111860	111794	111499		111489	111447	111399	111399	1

Chapter 3

An Efficient Heuristic for Very Large-Scale Vehicle Routing Problems with Simultaneous Pickup and Delivery

The demand for e-commerce delivery has increased rapidly in the last few years, especially during the COVID-19 pandemic and in the city logistics context. Such a fast growth does not come without its challenges. For example, the quick increase in deliveries occurs simultaneously with a large amount of returned goods, thus both such processes must be faced together. More specifically, online purchases are at least three times more likely to be returned than items bought in a store. In 2021, a record $761 billion of merchandise was returned, according to estimates in a recent report from the U.S. National Retail Federation. On the other hand, collecting defective goods and/or delivering new packages in the urban areas of densely populated cities can impact strongly the economic, social, and environmental policies adopted by the public authorities. Hence, designing an effective algorithm for the vehicle routing problem with a large number of simultaneous pickup and/or delivery locations is a relevant task. The aim of this work is to provide an algorithm which extends the FILO framework, originally proposed and specifically designed for the Capacitated Vehicle Routing Problem, accomplishing two objectives: first, being competitive with the state–of–the–art algorithms for the so-called vehicle routing problem with simultaneous pickup and delivery, and with its special case known as mixed pickup and delivery; second, efficiently solving new benchmark instances for these problems with a very large number of customers, while maintaining linear scalability of the computing time with respect to the problem size. The extensive computational study performed in this paper shows that both these objectives are achieved successfully by the proposed algorithm.

3.1 Introduction

The impact of the gig economy has triggered increased waste and returned goods which need to be dealt with through recycling, alternative uses, or even donation. Although Amazon does not share its overall returns numbers, in 2021 the U.S. National Retail Federation estimates that 16.6% of all merchandise sold during the holiday season was returned, up more than 56% with respect to the previous year. For online purchases, the average rate of return was even higher, at nearly 21%, up from 18% in 2020. With $469 billion of net sales revenue last year, Amazon's returns numbers are likely staggering. In this context, the relevance of the policies devoted to the management of the reverse flow of goods from the customers stimulated recent studies in the field of reverse logistics. These studies are focused on operational problems related to the daily plan of routes where pickup and delivery services can occur at the same location. On the other hand, the growing concern about global warming and the greenhouse effect due to environmental pollution, produced great inter-

est in the optimal usage of the fleet of vehicles in transportation problems. More specifically, U.S. returns generate 16 million metric tons of carbon emissions during their complicated reverse journey and up to 5.8 billion pounds of landfill waste each year, according to returns solution provider Optoro (`https://info.optoro.com/hubfs/The%20Optoro%202020%20Impact%20Report.pdf`). The best way to manage the collection of waste and recycling goods in city logistics is the key to the success in the remanufacturing processes leading to economic, social, and environmental benefits, which result in reducing costs of raw materials, distribution, collection costs, and so forth (Hornstra et al. (2020), Shafiee, Ghomi, and Sajadieh (2021), Santana et al. (2021)). Since several stakeholders work in urban areas (Taniguchi and Heijden (2000)), the goals can differ depending on the actors involved: public authorities pursue objectives focused on the collective utility, while private companies seek mainly to increase their economic benefits by both reducing their costs and ensuring a good quality of service. The combination of different objectives and different points of view stimulates the development of efficient algorithms to face both deliveries and pickups simultaneously in city logistics areas (Taniguchi et al. (2001)), where the number of customers to be visited on the same day can be very large. Suffice it to say that Amazon delivered 4.2 billion packages in the year 2020 alone, and each Amazon driver usually delivers up to 300 parcels daily in urban areas. Based on the previous concerns, the topic of reverse logistics combined with environmental policies was recently included in the management of supply chains.

In this context, the design of optimal route plans where the vehicles collect unsold, damaged, or obsolete packages and deliver goods to the customers is a very complex task. This complexity comes from dealing with several constraints related to the vehicle capacity, the nature of the products to be collected (e.g., small and large packages), time windows, and demand conditions (Arenas et al. (2017), Delgado-Antequera et al. (2020), Shafiee, Ghomi, and Sajadieh (2021)). The focus of this paper is on a specific class of vehicle routing problems that arise in many real applications where a large number of customers require a pickup and/or a delivery service in very dense urban areas. In these areas, the big players of e-commerce, such as Amazon and Alibaba, require daily route plans where, as well as several operational constraints like time windows and route duration, the delivery of packages and the pickup of returned goods can occur simultaneously in the same locations, or where a returned good is picked up or, alternatively, a package is delivered. The last-mile service for these companies is typically based on predefined delivery areas which are assigned to one or more vehicles so as to simplify the routing design and implicitly enforce positive features such as driver consistency. However, variability in demand patterns and density as well as in traffic conditions, together with increasing pressure on improving the level of service for customers may result in substantial inefficiencies of these models which may need to be evaluated against models which allow the daily planning of routes on large undivided areas. Furthermore, in some cases, pickup and delivery requests may be served in separate routes. In this context, the capability of solving large-scale routing problems with pickup and delivery may represent an important asset for evaluating and redesigning delivery areas and shift towards operations models based on larger areas. For these reasons, we concentrate in this paper on an important family of routing problems with simultaneous pickup and delivery with the aim of developing solution methods that are capable of solving, within short computing time, problem instances with several thousands of customers, thus being one or two orders of magnitude larger than those solved by the existing algorithms from the literature. These problems are referred to, respectively, as the Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD) and as the Vehicle Routing Problem with Mixed Pickup and Delivery (VRPMPD), where VRPMPD is clearly a special case of VRPSPD. Although these problems only partially model all the features arising in the problems found in practice, they allow us to model the main aspects (pickup and delivery) arising in large distribution areas. The contribution of this work to the literature focused on these two problems is twofold:

- problem-oriented: we design a tailored heuristic scheme, based on the FILO framework, proposed by Accorsi and Vigo (2021) for the Capacitated VRP. The algorithm, called FSPD uses an iterated local search (ILS) engine that handles the problem-specific constraints through the well-known *resource extension functions* (REFs), initially proposed by Desaulniers et al. (1998) and whose use within metaheuristics was introduced by Irnich (2008a). The algorithm, called FSPD, solves medium to very large-scale instances of the VRPSPD and the VRPMDP. To assess its effectiveness, we solve several VRPSPD and VRPMDP literature benchmark instances and show that the proposed algorithm performs very well compared to the existing state-of-the-art algorithms;

- algorithm-oriented: the proposed algorithm is characterized by a local search engine supporting the REFs framework and by a tailored recreate strategy allowing to explore effectively the neighborhoods of the search space. These features of the algorithm allow the solution of very large-scale instances of strongly constrained VRPs such as VRPSPD and VRPMDP by keeping a linear trend of the computational time of the algorithm as the size of the instances increases.

The paper is organized as follows. Section 3.2 describes the problems and the corresponding complexity. Section 3.3 provides a comprehensive literature review of the most recent work on VRPSPD and VRPMPD. In Section 3.4, we describe the proposed heuristic and we highlight all the new features that are introduced to solve very large-scale instances of the VRPSPD and VRPMPD. Section 3.5 presents the details of the experimental design and the results of our computational study. Finally, Section 3.7 provides some conclusions and discusses future work.

3.2 Problem description

The VRPSPD can be described as a graph theoretical problem defined on an undirected graph $G = (V, E)$, where V is the vertex set and E is the edge set. The vertex set V is partitioned into $V = \{0\} \cup V_c$ where 0 is the depot and V_c is the set of customers. A cost c_{ij} is associated with each edge $(i, j) \in E$. Moreover, we assume that the cost matrix satisfies the triangle inequality. Each customer $i \in V_c$ requires an integer quantity $d_i \geq 0$, called delivery demand, from the depot and sends an integer quantity $p_i \geq 0$, called pickup demand, to the depot. Moreover, we have $d_0 = p_0 = 0$.

A fleet of homogeneous vehicles with capacity Q is located at the depot and available to serve the customers. In different problem settings associated with the existing benchmark instances, such fleet can either be unlimited or has a fixed dimension. Recalling that a Hamiltonian circuit is a closed cycle visiting a set of customers exactly once, a VRPSPD solution is composed of a set R of Hamiltonian circuits, called *routes*, each of them, say r, starting from the depot, visiting a subset of customers and coming back to the depot. A solution is feasible if a) each customer is visited by exactly one vehicle; b) each vehicle travels at most one route; c) each vehicle route starts from and ends at the depot; d) the route pickup demand $p_r = \sum_{i \in r} p_i$ at the depot, for each $r \in R$, does not exceed Q; e) the vehicle capacity constraint should be satisfied after the visit of each customer of any route, in such a way the vehicle is not overloaded after each customer visit.

The VRPSPD is NP-hard since it generalizes the well-known capacitated vehicle routing problem (CVRP). There exist different formulations of the VRPSPD, based on two- or three-index decision variables. We refer the reader to the recent survey of Koç, Laporte, and Tükenmez (2020), in which four possible formulations of the problem under consideration are detailed.

The VRPMPD is a slight variant of the VRPSPD in which, for each customer i, either p_i or q_i is strictly positive but not both. In the scientific literature, the VRPMPD is always defined in this way and the algorithms proposed to solve this problem are simple adaptations of those used for the VRPSPD.

3.3 Literature review

The VRPSPD was originally introduced by Min (1989), who developed a model and a heuristic method based on a "Cluster First - Route Second" approach, applied to solve a real case of a public library distribution system. After this seminal paper, during the last three decades a significant number of papers devoted to VRPSPD and its variants has been published, because of the practical importance of the problem for distribution companies.

A great help in reviewing the scientific literature on the topic comes from the already mentioned survey of Koç, Laporte, and Tükenmez (2020). The survey includes a detailed discussion, among others, of the heuristics developed for solving the classical VRPSPD. The relevant cited papers are 29 and are grouped according to the following heuristic types: classical construction and improvement heuristics (3 papers), local search metaheuristics (15 papers), population search heuristics (6 papers), and ant colony heuristics (5 papers). All papers are presented by providing a performance comparison of the proposed heuristics on some benchmark datasets. The most frequently used benchmark datasets for VRPSDP are:

- Salhi and Nagy (1999) dataset, containing 28 instances with 50−199 customers randomly distributed derived from the well-known CMT instances of the CVRP. The dataset is, in turn, decomposed into two subsets, denoted as X and Y and called CMTX and CMTY in this paper, each including 14 instances. The CMTY set is obtained by exchanging the demands of every customer in set CMTX. The number of customers in each subset is always between 50 and 199 customers and the pickup and delivery demands are swapped in the two subsets. Moreover, half of the instances in each subset include an additional constraint on the maximum length of every route. However, some algorithms have been tested only on the 14 instances (i.e., seven for each subset), not including the maximum length constraint. Double precision values have been used for distances and demands;
- Dethloff (2001) dataset, containing 40 instances with 50 customers each and with different customers distribution;
- Montané and Galvão (2006) dataset, containing 18 instances with 100, 200, and 400 customers, derived from VRP with time windows test problems.

The collected results in Koç, Laporte, and Tükenmez (2020) show that the best solutions are obtained by Subramanian, Uchoa, and Ochi (2013) and by Vidal et al. (2014). This claim is also confirmed by the recent paper by Christiaens and Vanden Berghe (2020).

Since the survey covers the period from 1989 to part of 2019, we limit ourselves to updating, in a strict chronological order, the existing literature on the topic.

Simsir and Ekmekci (2019) proposed a new metaheuristic for the VRPSDP, which implements an artificial bee colony (ABC) algorithm, based on the foraging behaviors of honey bee colonies in natural life. The ABC algorithm has been tested on Dethloff (2001) instances, and it allows obtaining feasible solutions within a reasonable computational time. However, much more effort should be required to improve the performance of the method, since the cost obtained for all the test problems is always worse than the best-known solution (BKS) available.

Hof and Schneider (2019) proposed an adaptive large neighborhood search combined with a path-relinking approach, called ALNS-PR, to address the VRPSPD. The competitiveness of the ALNS-PR algorithm is shown on the CMTX and CMTY set of instances of Salhi and Nagy (1999). Due to the way by which the set CMTY is derived from CMTX, the authors claim that the algorithm provides equivalent optimal solutions for both the CMTX and the CMTY variant of each instance, where the routes of one solution correspond to the reversed routes of the other. Finally, the ALNS-PR is tested also on the sets of Dethloff (2001) and Montané and Galvão (2006). The ALNS-PR algorithm was not able to reach the BKS available only in four cases. Even though a precise time comparison among the tested algorithms is impossible, the authors attempt to translate the computational times of all methods into a common time measure that takes into account the processors used, by using the Passmark scores (see PassMark® Software (2020)) of the processors used in the different computational studies. The average computational time of the ALNS-PR algorithm, even though competitive, remains significantly larger than these of the other tested methods. Another issue of the algorithm is related to the many parameters to be defined and tuned to run it. The authors observed that three parameters have a stronger impact on solution quality compared to the remaining ones. So, they used 10 instances from the 14 test problems of Salhi and Nagy (1999) to evaluate the best values of these parameters. These values are then kept unmodified for the tuning phase of the other parameters.

Christiaens and Vanden Berghe (2020) developed a new general heuristic, named slack induction by string removals (SISR), for solving the capacitated VRP and several of its variants, including VRPSPD and VRPMDP. The heuristic is based on an iterative scheme composed of a ruin method, in which randomly selected customers are removed from the current solution to generate the so-called spatial slack, and a recreate method, represented by a greedy insertion method, which is used to reconstruct a feasible solution. Finally, a fleet minimization procedure can be used when minimizing the number of vehicles is a primary objective of the problem. In the case of VRPSPD, SISR has been tested on the whole benchmark set of Salhi and Nagy (1999) and compared against the results of the algorithms of Subramanian, Uchoa, and Ochi (2013) and of Vidal et al. (2014). In 22 cases out of 28, SISR reaches, on average, the BKS.

Hornstra et al. (2020) proposed an adaptive ALNS metaheuristic for solving the variant of VRPSDP

including handling costs. However, the metaheuristic has been also applied to the classical version of the problem. The proposed algorithm is based on the same ALNS scheme defined by Ropke and Pisinger (2006), strengthened by a local search procedure to possibly improve iteratively the current solution. Five different local search operators are used (*reinsertion*, *exchange*, *intra 2-opt*, *inter 2-opt* and *inter 3-opt*), and a new solution is accepted or not through a simulated annealing acceptance criterion. The metaheuristic has been tested on the set of 14 instances of Salhi and Nagy (1999) without distance constraints and on those of Dethloff (2001). For the first set of instances, the algorithm finds 11 BKSs out of 14 reported in Polat (2017). It is worth noting that the authors seem to have generated instances in set CMTY slightly differently from other authors, and this impacts the computation of the objective function value. The BKSs reported in Polat (2017) were reached in 26 out of 40 instances for the second benchmark dataset.

Hamzadayı, Baykasoğlu, and Akpınar (2020) proposed a Single Seekers Society (SSS) algorithm for solving the VRPSPD. The SSS algorithm (also known as "social evolutionary algorithm") is a metaheuristic that enables cooperation between different local search heuristics, each of them exploring the search space with its own parameters separately. The local search heuristics incorporated in the SSS algorithm are several, and among them, we find a greedy search, a random search, a simulated annealing, and a genetic algorithm. The SSS algorithm has been tested for the VRPSDP on the benchmark set of 14 instances of Salhi and Nagy (1999) and of Dethloff (2001). The algorithm has been able to reach almost all of the BKSs provided by several competing algorithms. While the average gaps are very small when compared to the competitors, nothing is reported about the computational time. The authors just report an average value of 10 minutes for all problem instances, which is about an order of magnitude higher than the computing time required by other state-of-the-art methods.

Olgun, Koç, and Altıparmak (2021) developed a hyper-heuristic (HH-ILS) algorithm based on iterative local search and variable neighborhood descent heuristics to solve the green version of the VRPSPD. Extensive computational experiments have been conducted to analyze the performance of the proposed algorithm also for the standard VRPSPD problem, by using the classical benchmark instances of Salhi and Nagy (1999) and of Dethloff (2001). The HH-ILS has obtained 38 BKSs out of 40 instances of Dethloff (2001). Among the problems of Salhi and Nagy (1999), the HH-ILS algorithm obtained five BKSs in comparison with those achieved by several competitive algorithms.

Park et al. (2021) proposed a genetic algorithm for solving VRPSPD. The comparison has been conducted with widely-used neighborhood search methods, but not on the standard test problems, so it is hard to say if the proposed method outperforms the best available algorithms.

Finally, in the recent paper of Öztaş and Tuş (2022), a hybrid metaheuristic for the VRPSPD is presented. It is a combination of iterated local search, variable neighborhood descent, and threshold acceptance metaheuristics. Iterated local search is used as the main framework of the proposed algorithm and is initialized with a solution generated with a nearest neighbor heuristic. Variable neighborhood descent provides intensification in the search space by randomly ordering the neighborhood structures. A perturbation mechanism allows exploring different parts of the search space, taking the opportunity to exploit non-improving solutions encountered in the search space by using an adaptive threshold acceptance. The results obtained on the first 14 benchmark instances of Salhi and Nagy (1999) show that in 10 cases the BKS has been obtained. The proposed algorithm is very competitive with other algorithms both in terms of solution quality and computational time also for Dethloff (2001) problems where it obtained the BKS for all these instances. Finally, the results obtained on the instances of the dataset of Montané and Galvão (2006) show that the algorithm is able to reach the BKS value from the literature in 8 out of 18 medium–sized problems.

The VRPMPD has been used to model real situations occurring in the food industry, where production locations must be visited together with delivery locations to account for the food regulations requiring delivery within the next 12h after the start of production (Oesterle and Bauernhansl (2016)). The most relevant contributions to the VRPMPD have been reviewed in the papers of Koç and Laporte (2018) and Santos et al. (2020). There are several benchmark datasets for the VRPMPD. The most used in the literature is the one derived from the dataset designed by Salhi and Nagy (1999) and composed of 42 test instances that range in size from 50 to 199 customers. These instances are grouped into three subsets referred to as CMTT, CMTQ and CMTH obtained from the CMT instances of the capacitated VRP. In set CMTT the proportion of pickup customers is 10%, while this value increases to 25% and 50% in sets CMTQ and CMTH, respectively. Also in this case the last

seven instances in each dataset include a maximum distance constraint in addition to the capacity constraint and, therefore, were not considered by some authors. Other datasets are used in papers focusing on problems that have the VRPMPD as a special case. These datasets are briefly described in the sequel where appropriate.

The state-of-the-art on heuristics for the VRPMDP is represented by some of the papers which focus on VRPs and their variants, such as the paper of Goksal, Karaoglan, and Altiparmak (2013), where they designed a heuristic algorithm based on particle swarm optimization that uses a local search implemented through a variable neighborhood descent algorithm. The proposed algorithm outperforms the reactive tabu search developed by Wassan, Nagy, and Ahmadi (2008) on a set of instances that are derived from the VRPSPD data set of Dethloff (2001). More precisely, it improves 101 out of 120 BKSs found by the tabu search. For the 12 instances, in which the designed algorithm performs worse than the reactive tabu search, the results are still within 0.01% of the BKS obtained by the heuristic proposed by Wassan, Nagy, and Ahmadi (2008). In addition, this algorithm obtains the best solutions within a computational time smaller than that required by its competitor. The comparison on the instances derived from the dataset designed by Salhi and Nagy (1999) is performed against the reactive tabu search of Wassan, Nagy, and Ahmadi (2008), the large neighborhood search of Ropke and Pisinger (2006) and the ant colony system of Gajpal and Abad (2009). It shows that the algorithm reaches 16 best solutions out of 42 possible solutions and three new best solutions for the test instances 2, 4, and 12 of the CMTQ dataset. The comparison with the competitive algorithms in terms of computational burden shows that the proposed algorithm needs slightly greater computational time. In the following, we analyze the performance of some previously mentioned algorithms focused on the VRP and its variants, in which significant improvements on the BKSs of the VRPMPD are also provided.

The hybrid algorithm proposed by Subramanian, Uchoa, and Ochi (2013) has been tested on the VRPMPD benchmark instances derived from the dataset designed by Salhi and Nagy (1999). A comparison is performed with the large neighborhood search of Ropke and Pisinger (2006) and the ant colony system of Gajpal and Abad (2009). The proposed algorithm obtains the BKS in 25 instances and it improves the result of another 12. It outperforms both competitive algorithms in terms of solution quality.

The Unified Hybrid Genetic Search metaheuristic of Vidal et al. (2014) is tested on the VRPMPD benchmark instances derived from the dataset designed by Salhi and Nagy (1999). A comparison with the algorithms of Anagnostopoulou, Repoussis, and Tarantilis (2013) and of Subramanian, Uchoa, and Ochi (2013) is performed. The algorithm is able to obtain the best gap of +0.00% which is the same as Subramanian, Uchoa, and Ochi (2013). These results are reached within an average computing time of 2.46 minutes.

The hybrid heuristic designed by Hof and Schneider (2019) is tested on the VRPMPD instances derived from the dataset designed by Salhi and Nagy (1999) and the obtained results are compared with the ones achieved by the heuristics designed by Subramanian, Uchoa, and Ochi (2013), and by Vidal et al. (2014). The performance is comparable in terms of: a) average percentage gap of the best solution quality based on several runs to the BKS, and b) average percentage gap of the average solution quality to the BKS. These gaps are equal to +0.00% and +0.18%, respectively. The resulting time in seconds is comparable with the one provided by Subramanian, Uchoa, and Ochi (2013).

Finally, the method designed by Christiaens and Vanden Berghe (2020) is tested on the VRPMPD instances derived from the dataset designed by Salhi and Nagy (1999), and the performance of the proposed method is compared with the heuristics presented in Subramanian, Uchoa, and Ochi (2013) and in Vidal et al. (2014). The comparison is done according to the average cost, the best cost, and the average calculation time in minutes obtained after 50 runs per instance. The proposed algorithm provides results similar to the ones achieved by Vidal et al. (2014) in terms of quality. Furthermore, it improved two BKSs.

3.4 Solution Approach

In this section, we describe the proposed heuristic, called Fast ILS localized optimization for Simultaneous Pickup and Deliver problems (FSPD for short in the following), which we designed to

solve VRPSPD and VRPMPD. The FSPD algorithm is an extension of the FILO framework, originally proposed and specifically tailored for the CVRP by Accorsi and Vigo (2021). Because of the high specialization of FILO for CVRP, extending the framework to more constrained VRP variants, such as those described in Section 3.2, requires a substantial redesign effort if one wants to preserve the effectiveness and scalability characteristics of the original approach. Therefore, FSPD includes several new features with respect to the original FILO framework, which enhance its performance on constrained variants of the VRP. Although we focus here on VRPSPD-related problems, and in particular on those that were described in Section 3.2. We shall note that FSPD approach could be more easily extended to other VRP variants, provided that the associated constraints can be expressed in terms of REFs (see Desaulniers et al. (1998) and Section 3.4.2) to efficiently handle them within local search. However, the extension of FSPD to different VRP variants may require further design changes to boost its effectiveness.

In the following sections, we first provide a brief overview of FILO framework, then we describe in detail the extensions we implemented in FSPD to consider pickup and delivery constraints. In particular, we examine the REF implementation and how some of the components of the original approach have been updated to specifically cope with the VRPSPD and VRPMPD.

3.4.1 An overview of FILO framework

In this section, we briefly outline the main characteristics of FILO approach, while the modifications introduced in FSPD are discussed in detail in the following ones. The reader is referred to Accorsi and Vigo (2021) for a more comprehensive description of FILO framework and its basic features.

Algorithm 4: High-level structure of FILO.

```
   Input  : Instance I, random seed s, number of core optimization iterations Δ_CO
   Output: Final solution S*
 1 PROCEDURE FILO(I, s)
 2 begin
 3     R ← RANDOMENGINE(s);
 4     S ← CONSTRUCTIONPHASE(I);
 5     S ← ROUTEMINIMIZATION(S, R);
 6     S* ← S;
 7     for Δ_CO iterations do
 8         S' ← S;
 9         S' ← RUIN(S', R);
10         S' ← RECREATE(S', R);
11         S' ← LOCALSEARCH(S', R);
12         if COST(S') < COST(S) then
13             S* ← S';
14         end
15         if SIMULATEDANNEALINGACCEPT(S', S) then
16             S ← S';
17         end
18     end
19     return S*;
20 end
```

Algorithm FILO is a randomized and efficient algorithm, which, as already mentioned, was specifically designed for the effective solution of large–scale instances of the CVRP. A schematic outline of FILO is depicted in Algorithm 4. FILO is designed to be run multiple times on the same instance, and each run initializes the random engine with a seed received as an input (Line 3). An initial solution is created with a restricted version of the Clarke and Wright (1964) savings algorithm, proposed by Arnold, Gendreau, and Sörensen (2019) for large-scale instances, which computes only a linear number of savings. Then, a route minimization procedure is possibly applied, whenever the initial solution uses more routes than a greedily estimated number.

The main *core optimization iteration* (Lines 7-18) is based on the well-known iterated local search (ILS) paradigm (see Lourenço, Martin, and Stützle (2003)). During this procedure, a *shaking* step (Lines 9 and 10), which is performed in a ruin-and-recreate fashion, and a *local search* step (Line 11) interleave for a prefixed number Δ_{CO} of iterations. When a local optimum is generated after the local search

application, it is possibly accepted as the current working solution with a probabilistic acceptance rule based on a simulated annealing criterion (Lines 15-17).

The feature of FILO which contributes the most to its efficiency is the *locality* of the application of its procedures. That is, both the shaking and the local search applications perform changes in a very limited, and spatially delimited, area of the solution. This makes every ILS iteration almost independent from the instance size.

The locality of the shaking procedure is obtained by the ruin-and-recreate procedure itself, which is designed to remove a set of customers that are spatially located close to each other, as better detailed in Section 20. On the other hand, the local search locality is obtained by means of a heuristic pruning technique called *selective vertex caching* (SVC), which keeps track, at any time, of a solution area that has been recently modified. This is, in practice, obtained by managing the vertices recently involved in solution changes through a limited-size set in which exceeding vertices are evicted following a least-recently-changed policy. We should note that even if locality makes the computational effort of a core optimization iteration practically independent from instance size, the algorithm still includes some linear-time routinary operations, such as solution copy at Lines 13 and 16, which, however, have a limited impact on the overall computing time of an iteration.

We next describe in more detail the two components of a core optimization iteration.

Shaking.

The shaking step, builds from the current solution S a new one, S', to be used as starting point for the local search. As previously mentioned, the shaking is performed in a ruin-and-recreate fashion. More precisely, during the ruin step, starting from a random seed customer i, a total number ω_i of customers is removed from the current solution. The customers to be removed are identified by a random walk starting from the seed customer. The length ω_i of such random walk is an adaptive parameter that is iteratively adjusted to the current instance and solution structure to produce neighbor solutions having a quality that is neither too close nor too far from the current one. The recreate step greedily reinserts the removed customers once they have been, with equal probability, either randomly shuffled or sorted according to either their distance from the depot or their demand. For more details on the shaking procedure, the reader is referred to Section 2 of Accorsi and Vigo (2021).

Local Search.

At each core optimization iteration, a local optimum is obtained by means of a sophisticated local search engine that combines accelerations and heuristic pruning techniques. In particular, the procedure uses vertex-wise {granular neighborhoods (GNs, see Toth and Vigo (2003) and Schneider, Schwahn, and Vigo (2017)) and the above-mentioned SVC to heuristically reduce the neighborhood cardinality and spatially localize the local search applications. More in detail, a GN is a restricted local search neighborhood that allows for the identification of a subset of neighbor solutions through what are known as move generators. A move generator is an instance arc that is used to uniquely identify a move of a local search procedure, and thus a neighboring solution. We define the set of move generators to be composed of arcs that connect every vertex to its K nearest neighbor. In its vertex-wise definition, GN allows to consider, for every vertex i, a variable number $0 \leq k_i \leq K$ of move generators, thus enabling the algorithm to intensify the search where it could be more effective. In addition, we use static move descriptors (SMDs, see Zachariadis and Kiranoudis (2010) and Beek et al. (2018)) as an acceleration technique to efficiently explore these neighborhoods. An SMD is a data structure that, for a given neighborhood, uniquely identifies a local search move and the associated cost variation, denoted as *delta*, obtained by applying such a move to the current solution. SMDs are stored inside specialized data structures that keep the moves sorted according to their effectiveness and enable the precise update of all, and only, the SMDs whose *delta* might have been affected by the application of any given move. For a more detailed description of how GNs, SVC, and SMDs are designed and efficiently combined for the CVRP, the reader can refer to Section 2.2 of Accorsi and Vigo (2021).

3.4.2 The FSPD framework

The extension of FILO framework to more constrained VRPs requires a substantial effort to incorporate the handling of problem-specific constraints while retaining the efficiency and scalability of the main procedures. In FSDP, this is achieved by mainly extending three components of the algorithm, namely:

- the initial solution construction (Line 4 of Algorithm 4) that should now produce a feasible solution for the problem at hand;
- the shaking step, where both ruin and recreate must incorporate problem-specific features, and
- the local search engine which must perform efficiently the feasibility check of the solutions with respect to the problem constraints.

To handle efficiently VRPSPD and VRPMPD characteristics in FSPD, a number of new features were added to the original approach. Most of these features are required to support the handling of additional constraints which are more complex with respect to the CVRP ones. To this end, we introduced two major innovations in the original FILO approach, namely:

- an extension of the local search engine to support the well-known *resource extension functions* (REFs) (see, Desaulniers et al. (1998) and Irnich (2008a));
- a generalization of the original recreate strategy aimed at handling multi-attribute VRPs.

These changes mainly affect the recreate and the local search steps (lines 10 and 11 of Algorithm 4). In particular, a major redesign of these steps is required because of the additional workload associated with the REFs framework. In the following, we describe in detail the new features of FSPD starting from the REFs handling which is preparatory to all other changes.

Resource Extension Functions.

A significant challenge when adapting CVRP algorithms to other VRP variants is effectively handling the additional constraints these variants may introduce. It is important to note that the task of handling constraints efficiently is not equally simple to manage. As discussed earlier, the FILO framework has already proven to be efficient in handling capacity constraints and constraints related to the maximum distance a route can cover in the CVRP. For these constraints, evaluating the impact of a single solution change only requires a small set of additional "cumulative" route attributes, like demand-sum or distance-sum. More in detail, the demand-sum represents the total load of the entire route, serving as a sufficient indicator of route feasibility. Additionally, when new customers are added to or removed from a route, efficiently updating the demand-sum involves simply adding or subtracting their respective demands, a process that scales with the number of customers involved. Similar consideration apply to distance-sum which is the total length of the route used to check feasibility of maximum distance constraints.

With different VRP variants, however, we may no longer have the ability to compute the feasibility by using only a constant number of route attributes. For example, when we consider VRPSPD or VRPMPD, this property is lost because the insertion of a customer into a route can cause local infeasibilities throughout the whole route, which forces us to consider the feasibility at a customer granularity (e.g., by scanning the entire route). For this reason, cumulative route attributes as in CVRP are no longer sufficient.

When considering the VRPSPD, the load while traversing a given route can either increase or decrease from customer to customer, thus making the load profile non-monotonic along the route. In this case, there are no constant-size route attributes that can both represent the route feasibility status, and at the same time can be updated at every new change of the route with a number of operations proportional to the nodes involved in the move. Thus, more general approaches are needed if one wants to handle this type of constraint efficiently.

An attempt to address this issue, while also trying to propose a generic framework to treat complex routing constraints, has been initially proposed by Kindervater and Savelsbergh (1997). Their approach consists of three main ingredients: (i) limiting the search strategy to a lexicographic search, (ii) storing some key information in global variables that are queried for feasibility checks, and (iii)

updating such global variables at every move application. With these elements, a generic heuristic can be developed to address different VRP variants efficiently. By leveraging the lexicographic search pattern, once we have carried out a first feasibility check, the following ones can be sped up by reusing the information obtained from the previous one. The main drawback of this technique regards the constraint imposed on the search, which is limited to a specific scheme.

Desaulniers et al. (1998) proposed the concept of resource extension functions (REFs), a more general and flexible framework to handle different types of constraints regarding both VRP and crew-scheduling problems. In his work, Irnich (2008b) describes the first framework which merges some of the information-handling techniques proposed by Kindervater and Savelsbergh (1997) along with the REF framework introduced by Desaulniers et al. (1998). The resulting segment REFs provide a general yet efficient way to develop heuristics for a broad variety of VRPs. The key idea of this framework consists in decoupling the search strategy from the computation of global variables. The segment REF can be computed by using specialized data structures to reduce both the size and the computational time needed in the preprocessing and during the updates, while the search strategy can be of any type with the only caveat that just concatenations can be executed for feasibility checks. Another important state-of-the-art example of such a general heuristic algorithm is represented by the UHGS algorithm of Vidal et al. (2014), which is able to successfully tackle a large class of different VRP variants leveraging the flexibility of the REFs framework. For a more in-depth description and theoretical analysis of how such a framework can be constructed, the reader is referred to Irnich (2008b).

Now, let us delve into the task of defining an appropriate collection of resources and REFs for a given problem. Considering how REFs operate, we are looking for a set of resources of size R (possibly constant), that is able to check the feasibility of an associated segment in $O(R)$ operations. Moreover, given the resources of two segments that we want to concatenate, we need a function that computes the resources associated with the segment obtained by the concatenation performing $O(R)$ operations. Let us consider the CVRP as the first practical example. As previously stated, the feasibility status of a "CVRP-route" can be represented by its demand-sum. In general, we can extend the use of the demand-sum as a resource to represent the feasibility of any route segment. Regarding the concatenation REF, it simply consists of the sum of the demand-sum of the input segments. If we want to use the REF framework to compute the feasibility of the route obtained from the insertion of a new customer i inside a route, we first have to obtain the demand-sums of the segment from the depot to the customer before i, and that of the segment from the customer after i to the depot. Concatenating the former with the demand of i, and then the result with the latter we obtain the demand-sum of the new route after the insertion. Suppose we have the two segments resources available, we can execute the two concatenations and the corresponding feasibility check with only $O(R)$ operations, which, for the demand-sum is $O(1)$. However, note that in the CVRP case restricting the feasibility check to only concatenations can actually harm the overall efficiency of the algorithm. This variant in fact supports other operations which (depending on algorithmic design choices) can be simpler and faster but are incompatible with the REF framework. In the previous example, we could have performed the addition of the demand of customer i to the demand-sum of the route and checked the result against the capacity. Unfortunately, the ability to perform feasibility checks using only resources at a route granularity and the ability to add or subtract resources is valid for only few VRP variants, such as the CVRP.

When we consider more complex variants, such as the VRPSPD, REFs can actually help to handle a more complex set of constraints in a uniform and efficient way. Regarding the feasibility, we need to check if the load exceeds the vehicle capacity at any point of the route sequence. The most obvious value that carries such information is the maximum load reached within the route. However, given two segments with only their respective maximum loads M_1 and M_2, there is no way to compute the maximum load M_3 resulting from their concatenation by using only $O(R)$ operations, with $R = 1$ in this case.

Intuitively, the resulting maximum load M_3 would be obtained as the maximum between M_1 and M_2 plus a term that accounts for the concatenation. Considering the first segment, along with M_1, we need to account for the new deliveries that are carried out in the second segment (whose sum is D_2) since all those deliveries have to be already inside the vehicle at the start of the resulting segment. In the same way, we need to add the pickup-sum P_1 of the first segment to M_2, because, in addition to the vehicle load due to the second segment, now it also has to consider all the pickups previously

carried out during the first one. Therefore, to handle pickup and delivery constraints with arbitrary concatenations between segments we need to store and update three resources, namely:

- M: maximum load;
- P: pickup-sum;
- D: delivery-sum;

Then, as described in Vidal et al. (2014), the $O(R)$ concatenation operation will be:

- $M_3 = max\{M_1 + D_2, M_2 + P_1\}$
- $P_3 = P_1 + P_2$
- $D_3 = D_1 + D_2$

In addition, the resources for a single customer i will be initialized as $M_i = max\{p_i, d_i\}$, $P_i = p_i$, and $D_i = d_i$.

Local Search Engine extension to support REFs.

The extension to the more generic REFs framework requires some major algorithmic changes also in the ruin-and-recreate procedure and in each operator of the local search step. As previously discussed, the CVRP capacity constraints with which the original FILO had to deal with can be handled efficiently in a fairly simple way. Instead, when we need to handle more complex constraints, such simplicity is lost. As described in Section 3.4.2, every feasibility check is executed on top of a given segment resource. Therefore the resource of each route must be computed first. To do so, the route is split into sub-segments, which are then concatenated together. The split is not random, as it clearly tries to use, as much as possible, pre-computed segments, so as to minimize the number of needed concatenations. As in the example of Section 3.4.2, if we consider the simple case where we want to insert a customer in a given position of a route, what is needed to compute the resource of the route obtained after the move will be:

- the resource of the segment starting at the depot and ending at the vertex before the insertion point;
- the resource associated with the single-customer segment of the customer we want to insert;
- the resource starting from the vertex after the insertion point up to the depot.

With these resources, by simply applying two concatenation operations we can obtain the complete resource of the route that would be produced by the move, and, therefore, we can evaluate its feasibility.

Looking in general at the moves applied within our algorithm we can find two main scenarios:

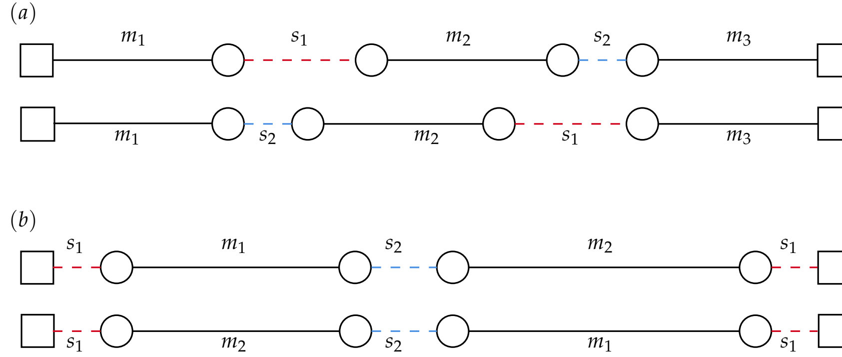
1. We want to insert a segment into a route that did not contain that segment before.
2. We want to relocate a segment to a different position within the same route.

The first case is, for example, found during the recreate phase of the ruin-and-recreate procedure, where we want to insert single customers inside the solution, one at a time. But it can also be found in the local search in any operator dealing with different routes, such as exchange, relocate, SPLIT, and TAIL moves. In all such cases, dealing always with two different routes simplifies the feasibility check for both of them, since the associated resource function computation simply resembles what we have previously described.

On the other hand, a more complex operation is needed when dealing with the second case. When we want to move a segment within the same route, the most efficient resource computation (that minimizes the number of concatenations needed) is not always the same. In particular, it is different if the segment is moved in a forward or backward position, and if the segment contains or not the depot. Furthermore, when dealing with exchange moves, which relocate two segments within the same route by swapping their position, these considerations must be combined.

For the sake of completeness, in what follows we describe the different scenarios we have to address for the operators of our LS scheme.

Figure 3.1: Illustration of segment concatenation associated with a swap move, in the case the depot is not contained in the segments to be swapped (a) or is contained in one of them (b).



Let us first consider the case where two segments are swapped within a single route because it is the case that needs special handling. Let $s_1, s_2 \subset S$ be the two route segments that we want to swap. Like in the original algorithm Accorsi and Vigo (2021), we consider segments having a length between 0 and k, with all their meaningful combinations and avoiding repetitions (e.g., the case 0-0 does not produce a change, while the case 2-3 explores the same moves as the case 3-2, so just one of then is considered). In addition, we consider the possibility of reversing one or both segments. For every intra-route swap we need to account for the following two aspects:

1. which one of the segments appears first in the route;
2. whether one of the two segments may contain the depot.

Without loss of generality, we assume that s_1 end is always before s_2 start. Whenever this is not the case we can swap the two segments and proceed as described.

If the depot is not within one of the segments, then the new route (and its resource) can be composed by concatenating five segments (four if one of the segments has length 0), as in Figure 3.1(a).

On the other hand, let us assume that the depot is included within segment s_1 (the same will apply in case it is in s_2). Then, s_1 needs to be split into two parts: the first from its start to the depot, and the second from the depot to its end. Then, the new route will be obtained by the concatenation of five segments: (i) the segment from the depot up to the end of s_1, (ii) the segment that was between s_2 and the start of s_1, (iii) segment s_2, (iv) the segment that was between the end of s_1 and s_2, and, finally, (v) the first half of s_1, from its start to the depot, as is shown in Figure 3.1(b).

Segment Resources Computation.

A number of different approaches can be adopted to obtain the resources of a given segment. They all leverage on the compromise between the number of moves tested versus the number of moves actually applied. Indeed, pre-computing some of the resources used with REFs that might be queried afterwards can be a very advantageous technique when only a few moves are actually applied and update procedures are seldom invoked. The main examples from the literature are those described in Vidal et al. (2014) where for each route, along with the depot–to–node and node–to–depot resources, all the segment resources corresponding to sub-segments up to a certain size are stored, and that of Irnich (2008b), which allows the maintenance of a constant time complexity for the queries, at the cost of heavier pre-processing and update operations.

Considering that our LS exchange operators grow up to a maximum size of three, we adopted the following implementation choices:

- For every customer we compute the resources from the depot to that customer included, and from that customer (included) to the depot;
- For every segment resource, we also compute and store the resource of the same segment obtained when we reverse it;

- Optionally, we considered a version where we compute and store the resources of all the sub-segments containing up to three customers (we tested the usefulness of these resources in Section 3.6.1).

Every update operation has been specialized to minimize the number of updated resources, trying to compute only those that have actually been changed by the last applied move, because recomputing the resources for the entire route caused a substantial overhead.

Finally, in one of the preprocessing variants that we considered, we decided to store resources for segments of size up to three, avoiding larger sizes because of the bottleneck due to the simple copy of a solution (with its meta-data) into another. This operation is executed at the start of every core iteration. Because of the lightweight iterations performed by FSPD, we noticed that it is one of the most time-consuming operations of the entire algorithm (even when properly optimized and vectorized by the compiler). To further reduce the size of the stored sub-segments, we decided to store the resources for both directions of each segment together, because there is usually a substantial overlap between the data of the two. More specifically, the pickup-sum and delivery-sum are respectively symmetric and identical for both directions, therefore, by storing segments-resources of opposite directions together, we reduced the number of attributes from 6 to 4, thus matching the number of attributes used in Irnich (2008a).

Improved Recreate procedure.

In FSPD the recreate step of the ruin-and-recreate algorithm has been completely redesigned with respect to FILO because, with the addition of REF handling, we noticed a substantial slow-down in the original procedure when dealing with very large instances. We argue this is mostly due to the feasibility checks which are now performed insertion-position-wise instead of route-wise as in the FILO case. For more computational details, a comparison with different recreate-phase implementations is reported in Section 3.6.2.

In FILO, to identify the insertion position of a removed customer, the entire solution was scanned in search of the best possible position. For the CVRP this was a relatively inexpensive procedure since only one feasibility check was necessary for each route in which the customer could be inserted. Instead, in FSPD the feasibility of every single candidate insertion position needs to be evaluated with an independent check involving REFs. To overcome the resulting considerable slow-down, the recreate procedure has been redesigned to limit the number of insertion position checks. More precisely, in FSPD the insertion positions are not explored by iterating on the solution as before. Instead, for each customer to be re-inserted in the solution, a pre-computed list of all the neighbors, sorted by their distance, is considered. Following the order of the list enables the introduction of an early-exit criterion that drastically reduces the number of checked insertion positions while keeping most of the quality of the solution produced.

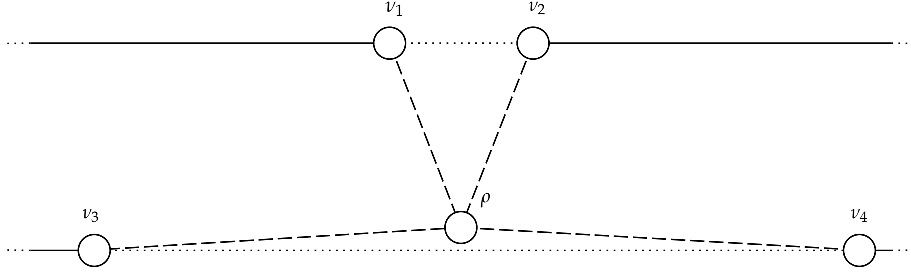
Algorithm 5 displays the main procedure that, given a customer not in the solution, tries to find the best insertion position. More formally, let T be the set containing the customers to be inserted in the current partial solution S during the recreate phase. For every customer $\rho \in T$, we explore its neighbor-list $\mathcal{N}_\rho$ containing all the customers, sorted by their distance from ρ (Line 6).

We initialize the best insertion position as that in a newly created single-customer route containing only ρ and the depot d (Line 3). This best insertion position is associated with its insertion cost c^* and with the vertices $\pi_\rho, \sigma_\rho \in S$ between which ρ is inserted. Note that initially $\pi_\rho = \sigma_\rho = d$.

Then, in the main loop, we consider insertion positions in the order given by $\mathcal{N}_\rho$. We introduce a hard upper bound N_R on the number of neighbors that can be considered to localize and speed up the recreate procedure, while also preserving most of the resulting solution quality. In this way, we apply the same rationale of the granular neighborhood technique used within the local search. A detailed analysis of the tradeoff between the speed-up and the solution quality associated with different pruning techniques is given in Sections 3.6.2 and 3.6.2.

However, when iterating over $\mathcal{N}_\rho$ we must consider some further implementation details. Let $\nu \in \mathcal{N}_\rho$ be the neighbor customer considered in the current iteration. This neighbor is associated with two possible insertion positions: one between $\textsc{Pred}(\nu)$, i.e. the predecessor of ν, and ν, and the second one between ν and its successor, $\textsc{Succ}(\nu)$. Considering both positions at every iteration would evaluate twice each insertion position: i.e., once when considering ν as a neighbor, and

Figure 3.2: Illustration of different insertion options. Although ν_1 and ν_2 are the closest customers to ρ, inserting ρ in this position would introduce a considerable detour. On the other hand, even if ν_3 and ν_4 are farther from ρ, the insertion cost of ρ between these two customers is smaller.



a second time when considering $\textsc{Pred}(\nu)$ or $\textsc{Succ}(\nu)$ as a neighbor of ρ). To avoid such double checks, we apply two precedence rules. We always consider the insertion position only if the distance $D(\textsc{Pred}(\nu), \rho)$ (Line 16) or the distance $D(\rho, \textsc{Succ}(\nu))$ (Line 22) is greater than the distance $D(\rho, \nu)$. Thus, each insertion position is evaluated only once. In the rare case in which the two distances are equal, we then consider the customer indexes and we evaluate the insertion only when $\textsc{Pred}(\nu) < \nu$ or $\nu < \textsc{Succ}(\nu)$. In this way, leveraging on the fact that $\mathcal{N}_\rho$ is sorted, we ensure that insertion positions are considered sorted by their shortest introduced edge, and we can exit at any point of the loop knowing that every other remaining insertion position would introduce longer edges. Note, however, that this does not imply that these remaining insertion positions have also a greater cost than the already considered ones. In fact, every insertion consists of the addition of two edges to the solution and the removal of one, which can always result in a net cost of zero in metric instances (see Figure 3.2 for an example where an insertion position between farther customers is preferable). When the cost of an insertion position is checked at Lines 18 and 24, we first compute the associated cost, and then, if it is better than the cost of the current best insertion position c^*, we also check the feasibility of the insertion by using the IsInsertFeasible function, which computes the maximum load of the route using the involved segments resources as described in section 3.4.2. More specifically, we concatenate the resources of the segment before the insertion point, the resources of the inserted node, and the resources of the segment after the insertion point. In this way, we obtain the new maximum load of the route that is compared with the vehicle capacity to check the feasibility.

3.5 Computational Results

Computational testing has the main objective of assessing the performance of the proposed approach. More precisely, we want to show that FSPD is a useful generalization capable of solving constrained VRPs without sacrificing scalability. To this end, we compare the FSPD performance with the best VRPSPD and VRPMPD state-of-the-art algorithms, and we present an extensive computational study on new large-scale VRPSPD and VRPMPD instances.

3.5.1 Implementation and Experimental Environment

The proposed algorithm was implemented in C++ and compiled using gcc-11.3. The experiments were performed on a 64-bit mini-computer with an AMD Ryzen 5 PRO 4650GE CPU, running at 3.3 GHz with 16GB of RAM (single-channel) on a GNU/Linux Ubuntu 22.04 operating system.

Three different versions of FSPD algorithm version have been tested. The first one is the basic fast version, called FSPD, performing 100,000 core optimization iterations, the second, called FSPD-mid, performing 500,000 iterations, and the last one, called FSPD-long, performing 1,000,000 iterations. Because of the randomized nature of FSPD algorithm, for each instance, the algorithms have been run 10 times with different random seeds. All benchmark instances we used and the solutions we obtained are available at `https://github.com/falcopt/FSPD`.

Algorithm 5: Structure of a function that, given a customer to insert back into a solution, finds a good insertion position. Note that we assume costs are symmetric.

Input : Partial solution S, customer to insert ρ
Output: Customers $\pi_\rho, \sigma_\rho \in S$ between which ρ will be inserted

1 FUNCTION FINDBESTINSERT(S, ρ)
2 **begin**
  `// Initialize candidate previous and successor nodes to the depot`
3   $\pi_\rho \leftarrow \sigma_\rho \leftarrow d$;
4   $c^* \leftarrow 2 \cdot D_{d,\rho}$;
5   $i \leftarrow 0$;
6   **for** $\nu \in \mathcal{N}_\rho$ **do**
7   **if** $i \geq N_R$ **then** `// Early exit criterion`
8   **return** π_ρ, σ_ρ;
9   **end**
10   **if** $\nu \notin S$ **then**
11   **continue**;
12   **end**
13     $i \leftarrow i + 1$;
14     $\pi_\nu \leftarrow$ PRED(ν, S);
15     $\sigma_\nu \leftarrow$ SUCC(ν, S);
    `// Recall that, since` $\mathcal{N}_\rho$ `is sorted, distances less than` $D_{\rho,\nu}$ `have already been checked`
16   **if** $D_{\pi_\nu,\rho} > D_{\rho,\nu}$ **or** $(D_{\pi_\nu,\rho} = D_{\rho,\nu}$ **and** $\pi_\nu > \nu)$ **then**
17       $c' \leftarrow D_{\pi_\nu,\rho} + D_{\rho,\nu} - D_{\pi_\nu,\nu}$;
18   **if** $c' < c^*$ **and** ISINSERTFEASIBLE(ρ, S, π_ν, ν) **then**
19         $\pi_\rho, \sigma_\rho, c^* \leftarrow \pi_\nu, \nu, c'$;
20   **end**
21   **end**
22   **if** $D_{\rho,\sigma_\nu} > D_{\nu,\rho}$ **or** $(D_{\rho,\sigma_\nu} = D_{\nu,\rho}$ **and** $\nu > \sigma_\nu)$ **then**
23       $c'' \leftarrow D_{\nu,\rho} + D_{\rho,\sigma_\nu} - D_{\nu,\sigma_\nu}$;
24   **if** $c'' < c^*$ **and** ISINSERTFEASIBLE$(\rho, S, \nu, \sigma_\nu)$ **then**
25         $\pi_\rho, \sigma_\rho, c^* \leftarrow \nu, \sigma_\nu, c''$;
26   **end**
27   **end**
28   **end**
29   **return** π_ρ, σ_ρ;
30 **end**

3.5.2 Parameter Tuning

The parameter tuning process was initialized with values directly taken from the original FILO approach. We then followed a sequential tuning strategy in which the algorithm behavior was evaluated while changing one parameter at a time. The goal was to tune FSPD in the same way as FILO was tuned. In particular, FSPD was tuned so as to achieve good results in reasonable computing time on existing literature instances, still maintaining the scalability to be effectively applied to very large-scale instances. Most parameters were kept at the values proposed in the original FILO approach. In this regard, the only change with the original parameters of FILO has been the simulated annealing final temperature. We noticed that with larger instances better results could be obtained with a lower final temperature. Therefore, as a small change from the original approach used by FILO (which sets the final temperature as a $\frac{1}{100}$ of the initial one), we decreased the final value accordingly with the instance size. More specifically, given an initial temperature S_i we set the final temperature $S_f = \frac{10}{1000+N} S_i$ where N is the number of customers of the instance. As described in the next sections, we also tuned the new components, namely:

- the preprocessing technique used to precompute segment resources (Section 3.6.1),
- the way to consider the insertion positions considered in the new recreate phase (Section 3.6.2),
- the value of the pruning factor for the new recreate phase (Section 3.6.2)

3.5.3 Testing on Instances from the Literature

Simultaneous pickup and delivery problems have been extensively studied during the past years, and several relevant benchmark instance sets have been proposed in the literature. In this section, we provide an overview of the performances of state-of-the-art algorithms on literature benchmark instances and compare them with our proposed method. Since most algorithms have been applied to more than one dataset, as well as more than one problem, in the following we provide a brief description of every competitor, then in Sections 3.5.3 and 3.5.3 we illustrate the results for every dataset. Only the best-performing competitors that adopted the most used rounding and instance definition conventions have been considered. Our overall testing shows the actual capability of FSPD approach to achieve state-of-the-art quality within the short computing times required by 100,000 iterations of the standard version. Furthermore, as expected, allowing a larger number of iterations to the approach, as in FSPD-mid and long, is beneficial in terms of solution quality improvement and still keeps the approach competitive and acceptable in terms of computing time.

- ALNS-PR: the hybrid algorithm combining adaptive local search (ALS) and path relinking of Hof and Schneider (2019).
- ILS-RVND-SP: the ILS heuristic of Subramanian, Uchoa, and Ochi (2013).
- SISR: the ruin-and-recreate algorithm of Christiaens and Vanden Berghe (2020).
- UHGS: the population-based method of Vidal et al. (2014).
- h_PSO: the hybrid discrete particle swarm optimization of Goksal, Karaoglan, and Altiparmak (2013).
- ACSEVNS: the hybrid heuristic based on ant colony and variable neighborhood search of Kalayci and Kaya (2016).
- PVNS: the perturbation-based variable neighborhood search algorithm of Polat et al. (2015). Note that, for this algorithm, the computing times reported are those of the best out of 10 runs (in terms of solution quality).
- ILS-RVND-TA: the hybrid ILS of Öztaş and Tuş (2022).
- VLBR: the adaptive memory approach of Zachariadis, Tarantilis, and Kiranoudis (2010). Note that, for this algorithm, the computing times reported are those to reach the best solution and not the total ones.

Detailed information about which dataset the above algorithms have been tested is provided in the section corresponding to each dataset.

To better compare our results with other algorithms executed on different hardware configurations, we roughly scaled computing times by using the single-thread rating defined by PassMark® Software (2020). At the time of writing, our CPU has a score of 2632. The CPU time of competing methods is thus scaled to match our CPU score. In particular, their scaled time is defined by $\hat{t} = t \cdot (P_A/P_B)$, where P_A is the competing method's CPU single-thread rating, P_B is our CPU rating, and t is the raw computing time proposed in the competing method paper. We are fully aware that CPU-benchmark scores are not intended as a precise measure to scale and compare algorithms times, but instead only as a qualitative measure to partially account for different hardware. Indeed, the algorithms adopted to produce such scores can differ substantially from the algorithms considered in this work. Furthermore, along with the CPU also main memory speed and the whole system performance should be measured to produce a real representative score, especially in memory-bound routing heuristics where sparsity is often exploited to prune the search at the cost of cache–locality. Nevertheless, we think such a raw scaling represents a more fair comparison of algorithms speed with respect to simply reporting the originally published running times.

Table 3.1 summarizes, for every algorithm, the CPU on which it was run and the associated single thread score at the time of writing. Note that no score is available for the AMD Opteron 250 CPU used by Vidal et al. (2014). Therefore, for this algorithm, we scaled times by considering the ratio of CPU frequency instead of that of scores.

If not differently mentioned, all computing times reported in the tables in the column marked (Time) are the average total times over multiple runs, whereas in columns marked (Time*), we report, where available, the average times to reach the best solution of the run. Finally, columns marked (Avg) report the average percentage gaps of the solution S found by the algorithm with respect to the best-known solution value (BKS), computed as $100 \cdot (\text{COST}S - BKS)/BKS$. Note that for some algorithms only the gap of the best solution found along different runs is reported. In these cases, we reported in the tables the best gap and marked it with an asterisk.

Algorithm	CPU	Score
ALNS-PR	Intel Core i5-6600 @ 3.30GHz	2283
ILS-RVND-SP	Intel Core i7-870 @ 2.93GHz	1392
SISR	Intel Xeon E5-2650 v2 @ 2.60GHz	1675
UHGS	AMD Opteron 250 @ 2.4GHz	649
h_PSO	Intel Xeon X5460 @ 3.16GHz	1377
ACSEVNS	Intel Xeon E5-2650 @ 2.00GHz	1257
PVNS	Intel Core2 Duo T5750 @ 2.00GHz	738
ILS-RVND-TA	Intel Core i7-7500U @ 2.70GHz	1940
VLBR	Intel Core2 Duo T5500 @ 1.66GHz	590
FSPD	AMD Ryzen 5 PRO 4650GE @ 3.3GHz	2632

Table 3.1: The CPU models used by algorithms from the literature and their single thread PassMark scores.

VRPSPD Benchmark Instances.

The current standard benchmark instances sets for the VRPSPD have been proposed in Salhi and Nagy (1999), Dethloff (2001) and Montané and Galvão (2006). Since not all the algorithms we selected have been tested on all datasets, in the following, we separately illustrate the results on each of them.

Testing on Salhi and Nagy (1999) Instances. In Salhi and Nagy (1999), two sets of instances for VRPSPD, called CMTX and CMTY, have been generated from the classical CVRP instances proposed by Christofides, Mingozzi, and Toth (1979). More precisely, CMTX instances redefine for every customer i, the delivery demand d_i as $d_i = r_i \cdot q_i$ and the pickup demand p_i as $p_i = (1 - r_i) \cdot q_i$ where the ratio $r_i = min\{x_i/y_i, y_i/x_i\}$ and, x_i and y_i are the x and y coordinates, respectively. The set CMTY is defined by swapping d_i and p_i demands of set CMTX. In both sets, seven out of 14 instances define an additional constraint on the maximum length of every route. Double-precision values have been used for distances and demands. Several algorithms only solved the first seven instances without this constraint. Therefore, we separately report the results on CMTX and CMTY in two tables, whereas complete results are reported in Appendix 3.9.

Table 3.2 reports the results on the first seven instances only, whereas Table 3.3 includes all 14 instances of each subset. Both tables clearly show that FSPD in its fast setting is among the best-performing algorithms and requires quite short computing times, whereas its mid and long versions strongly outperform the competing algorithms while requiring computing times of a handful of minutes and comparable with those of the competitors. To better investigate the dominance relation among FSPD and the existing algorithms, we performed an extensive statistical analysis following the procedure described in Christiaens and Vanden Berghe (2020). In particular, we conducted a one-tailed Wilcoxon signed-rank test (Wilcoxon (1945)) to test whether FSPD, in its three versions, is equivalent to or dominates each competing method. Our analysis shows, with a significance level $\alpha = 0.025$, that:

- On CMTX instances FSPD-mid and long dominate ILS-RNVD-SP and PVNS.
- On CMTY instances FSPD-mid and long dominate ILS-RNVD-SP.
- In all other cases FSPD neither dominates nor is dominated by competing algorithms.

Table 3.2: Results on the first seven instances of Salhi and Nagy (1999) CMTX and CMTY datasets. Average gaps marked by an asterisk are actually the best gap obtained along several runs.

Algorithm	**X**			**Y**		
	Avg	**Time***	**Time**	**Avg**	**Time***	**Time**
ALNS-PR	0.000*	–	105.611	0.009*	–	121.481
ILS-RVND-SP	0.307	–	149.569	0.290	–	143.291
SISR	0.240	–	391.930	0.212	–	391.767
UHGS	0.013	–	43.053	0.015	–	39.629
h_PSO	0.081*	–	100.532	0.121*	–	110.809
ACSEVNS	0.018	–	49.747	0.019	–	48.724
PVNS	0.024	–	34.240	0.022	–	24.333
ILS-RVND-TA	0.970	–	68.396	1.006	–	68.068
VLBR	0.111*	4.557	–	0.111*	3.702	–
FSPD	0.013	5.261	28.524	0.017	5.139	28.626
FSPD-mid	0.002	16.286	148.623	0.001	15.762	148.293
FSPD-long	0.000	23.465	299.800	0.000	17.766	305.624

Table 3.3: Results on the complete Salhi and Nagy (1999) CMTX and CMTY datasets. Average gaps marked by an asterisk are actually the best gap obtained along several runs.

Algorithm	**X**			**Y**		
	Avg	**Time***	**Time**	**Avg**	**Time***	**Time**
ALNS-PR	0.240	–	91.488	0.215	–	99.419
ILS-RVND-SP	0.234	–	101.411	0.220	–	98.899
SISR	0.145	–	487.390	0.128	–	481.853
ACSEVNS	0.031	–	85.996	0.031	–	84.440
PVNS	0.116	–	68.298	0.114	–	61.214
FSPD	0.056	7.574	35.661	0.065	8.330	35.673
FSPD-mid	0.018	29.300	182.381	0.026	27.598	183.794
FSPD-long	0.013	47.846	372.412	0.022	38.941	376.797

The excellent compromise between quality and speed of FSPD with respect to the existing methods from the literature is further supported by the performance charts of Figures 3.3 and 3.4, which illustrate the trade-off between the average percentage gap and the average computing time. The various versions of FSPD clearly dominate the competing methods. The only exception is represented by ACSEVNS which obtains on the full dataset slightly better solutions than FSPD in its short setting but is inferior to FSPD-mid and long.

Testing on Dethloff (2001) and on Montané and Galvão (2006) Instances. We complete the testing on VRPSPD by considering the datasets proposed by Dethloff (2001) and Montané and Galvão (2006) which were considered by some algorithms from the literature. Dethloff (2001) proposed a dataset

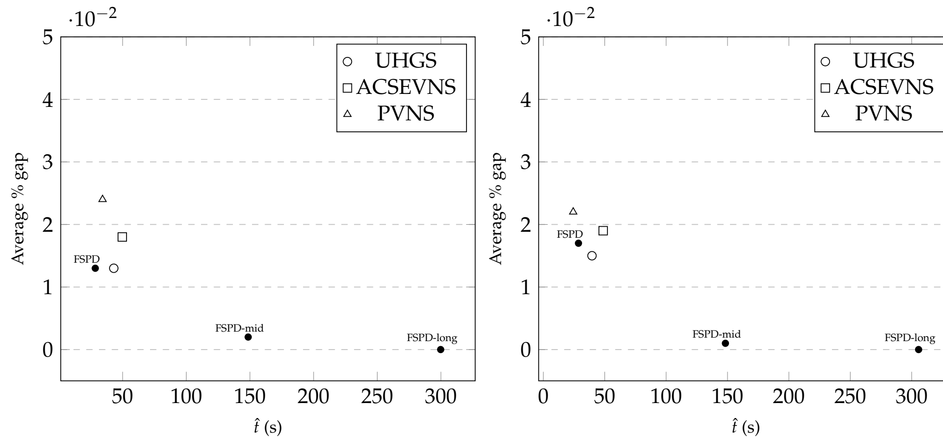


Figure 3.3: Illustration of the results on the first seven instances of Salhi and Nagy (1999) X (left) and Y (right) datasets.

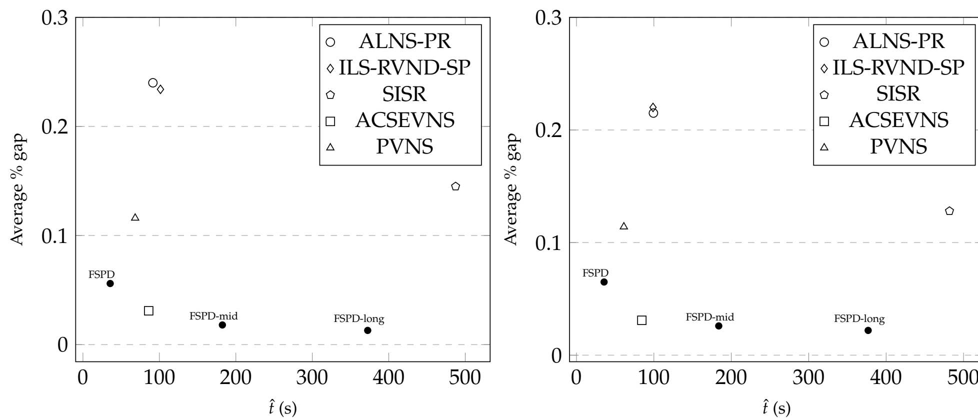


Figure 3.4: Illustration of the results on the complete Salhi and Nagy (1999) X (left) and Y (right) datasets.

with 40 Euclidean instances with 50 customers and two types of customer distribution. The first type (with prefix SCA) have the customers randomly distributed in the square $(0, 0) - (100, 100)$, while the other type (CON) scatters half of the customers as in the SCA type, and the other half is concentrated in the square $(\frac{100}{3}, \frac{100}{3}) - (\frac{200}{3}, \frac{200}{3})$. The delivery amounts are chosen randomly while the pickup amount is obtained from the delivery amount multiplied by a random number in the range $[0.5 - 1.5]$. Montané and Galvão (2006) proposed 18 instances, ranging from 100 to 400 customers, created by adapting literature instances of the VRP with time windows. The aggregate results of the comparison between FSPD and its competitors are reported in Table 3.4. The table clearly shows that FSPD in all its versions is very competitive both in terms of performance and speed with respect to the existing algorithms. Furthermore, our statistical analysis shows that on Montane instances all versions of FSPD dominate the competing methods except ILS-RVND-SP which however takes more than five times of computing time. On Dethloff instances, all versions of FSPD dominate both ACSEVNS and ILS-RVND-TA which are the only methods that report average results over several runs. We also note that on Dethloff instances FSPD obtains average gaps which are equivalent to the best ones obtained by ALNS-PR, h_PSO, and VLBR.

VRPMPD Benchmark Instances.

The most used benchmark from the literature for VRPMPD are the three sets proposed by Salhi and Nagy (1999). Also in this case the instances are derived from the CVRP instances of Christofides, Mingozzi, and Toth (1979), by defining every second, fourth, or tenth customer of the instance as a pickup-only customer with a demand equal to the original CVRP demand. The other customers are instead defined as delivery-only customers and keep the original CVRP demand as delivery de-

Table 3.4: Results on the Dethloff (2001) and Montané and Galvão (2006) datasets. Average gaps marked by an asterisk are actually the best gap obtained along several runs.

Algorithm	Montane			Dethloff		
	Avg	Time*	Time	Avg	Time*	Time
ALNS-PR	0.374	–	856.708	0.000*	–	9.167
ILS-RVND-SP	0.077	–	1933.605	–	–	–
h_PSO	–	–	–	0.000*	–	1.598
ACSEVNS	–	–	–	0.011	–	2.926
ILS-RVND-TA	0.890	–	452.206	0.084	–	7.425
VLBR	0.469*	21.573	–	0.000*	0.732	–
FSPD	0.173	14.691	33.593	0.000	0.489	30.971
FSPD-mid	0.115	67.696	170.895	0.000	0.602	155.923
FSPD-long	0.080	137.224	340.496	0.000	0.674	311.992

mand. The three resulting datasets are denoted as CMTH, CMTQ, and CMTT, respectively. Results for these datasets are reported in Table 3.5 for the first seven instances without maximum distance constraint, and in Table 3.6 for the complete datasets. Both tables show that also for VRPMPD our algorithm generally obtains better or comparable average gaps with respect to the best existing algorithms at the expense of an acceptable increase in the computing time. We note that ALNS-VR, h_PSO, and VLBR report only the best result obtained in several runs, and also in this case FSPD average gaps are better or comparable. This is further confirmed by the results of the statistical analysis that shows that all FSPD variants dominate ALNS-PR on both CMTH and CMTQ datasets. FSPD-mid and long dominate ILS-RVND-SP on the CMTQ dataset, while in all other cases, FSPD neither dominates nor is dominated by the competing algorithms.

Table 3.5: Results on the first seven instances of Salhi and Nagy (1999) CMTH, CMTQ, and CMTT datasets. Average gaps marked by an asterisk are actually the best gap obtained along several runs.

Algorithm	H			Q			T		
	Avg	Time*	Time	Avg	Time*	Time	Avg	Time*	Time
ALNS-PR	0.171	–	146.069	0.215	–	117.152	0.124	–	95.241
ILS-RVND-SP	0.060	–	129.589	0.036	–	131.835	0.017	–	145.005
SISR	0.074	–	419.095	0.076	–	393.021	0.091	–	338.200
UHGS	0.053	–	42.884	0.055	–	37.685	0.021	–	31.111
h_PSO	0.295*	–	90.211	0.215*	–	101.235	0.174*	–	91.758
FSPD	0.066	2.386	32.064	0.089	4.979	31.083	0.058	6.456	31.924
FSPD-mid	0.045	4.246	166.354	0.044	16.329	162.497	0.044	19.365	168.544
FSPD-long	0.022	10.600	332.113	0.031	22.777	325.808	0.042	36.249	333.019

Table 3.6: Results on the complete Salhi and Nagy (1999) CMTH, CMTQ, and CMTT datasets. Average gaps marked by an asterisk are actually the best gap obtained along several runs.

Algorithm	H			Q			T		
	Avg	Time*	Time	Avg	Time*	Time	Avg	Time*	Time
ALNS-PR	0.169	–	116.208	0.211	–	102.163	0.177	–	86.929
ILS-RVND-SP	0.104	–	92.048	0.091	–	92.650	0.086	–	98.131
SISR	0.041	–	508.691	0.065	–	486.899	0.090	–	425.641
UHGS	0.079	–	39.175	0.053	–	38.023	0.069	–	32.126
FSPD	0.062	6.590	37.520	0.077	8.550	36.527	0.098	8.157	34.317
FSPD-mid	0.032	24.862	191.916	0.028	27.289	188.760	0.086	25.449	178.201
FSPD-long	0.012	42.777	383.658	0.022	41.418	381.668	0.075	50.021	352.333

3.5.4 Testing on New Large-Scale Instances

Real-world applications of pickup and delivery problems, such as those arising in city logistics, may involve hundreds if not thousands of points to be served. As shown in the previous section

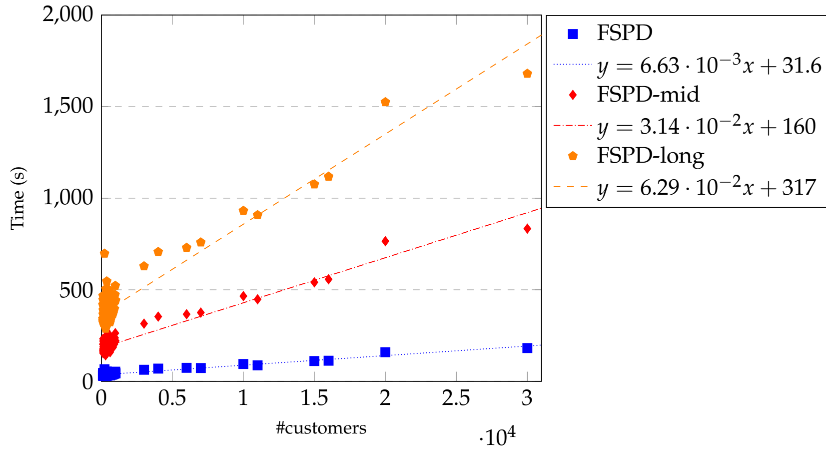


Figure 3.5: Computing time required by different FSPD versions as a function of problem size for X and XXL VRPSPD problem instances.

the existing algorithms, including FSPD, prove able to find near-optimal solutions for VRPSPD and VRPMPD involving up to 100-200 points. However, the computing times required to find such high-quality solutions typically grow quadratically with the instance size, thus requiring prohibitive efforts for the solution of realistic-size instances involving thousands of customers. Our FSPD approach, as the original FILO algorithm for CVRP, is instead designed to preserve a linear growth of the computational effort with respect to the problem size thanks to the careful implementation of the local search and the various methods which enhance the locality of the search. As a contribution to future research in the routing with pickup and the delivery area, we propose new benchmarks involving much larger instances of the problems so that future algorithms may be assessed in more practically relevant scenarios. To this end, and in line with previous literature, we extended to VRPSPD and VRPMPD the existing large and very-large-scale benchmarks available for the CVRP. More precisely, we considered two widely used benchmark datasets for CVRP, namely the well-known X dataset proposed by Uchoa et al. (2017) involving 100 to 1,000 customers and the XXL instances proposed by Arnold, Gendreau, and Sörensen (2019), including ten instances ranging from 3,000 to 30,000 customers. The new benchmark instances together with those from the literature are available at `https://github.com/falcopt/FSPD`.

In the next sections, we describe the results of the application of FSPD to the new X and XXL instances so as to provide a new reference point for the evaluation of algorithms. In addition, the testing confirms what was already proved in the instances from the literature. On the one side, FSPD obtains good quality solutions and the quality is significantly improved when a larger number of iterations is allowed. On the other side, the growth of the computing time required by FSPD is almost linear with respect to instance size as clearly depicted in Figure 3.5, where all data points (#customers, Time (s)) fit well the linear regressions.

In the following sections, we give in detail the results obtained for the X and XXL instances of VRPSPD and VRPMPD. In all tables, the average percentage gaps are computed with respect to the best solution found by our algorithm during all the experiments and the parameter tuning.

VRPSPD X and XXL Benchmark Instances.

In Table 3.7 we give the results obtained by our algorithm run in its three different settings for the XX and XY datasets obtained from the X benchmark of Uchoa et al. (2017) as proposed by Salhi and Nagy (1999) and described in Section 3.5.3. The same procedure has been applied to the XXL instances of Arnold, Gendreau, and Sörensen (2019) obtaining the VRPSPD instances sets XXLX and XXLY for which the results are given in Table 3.8.

By observing the tables it can be seen that the quality on large-scale instances is within 1% to 3% from the best solutions found across all our experiments already with the standard version of FSPD, although running it for a longer time provides a substantial benefit in terms of gap reduction. The average time for X instances is below 0.5, 2.5, and 5 minutes for FSPD, FSPD-mid, and FSPD-long,

Table 3.7: Results on the new large-scale VRPSPD XX, XY instances.

Algorithm	XX			XY		
	Avg	Time*	Time	Avg	Time*	Time
FSPD	0.769	28.905	36.019	0.776	28.359	35.922
FSPD-mid	0.365	135.261	180.511	0.382	134.260	179.671
FSPD-long	0.279	267.001	361.192	0.253	261.489	358.289

Table 3.8: Results on the new very large-scale VRPSPD XXLX, XXLY instances.

Algorithm	XXLX			XXLY		
	Avg	Time*	Time	Avg	Time*	Time
FSPD	2.814	104.867	105.059	2.741	104.007	104.221
FSPD-mid	0.936	516.885	518.687	0.897	511.611	513.295
FSPD-long	0.305	1040.386	1045.203	0.226	1025.770	1028.897

respectively. For XXL instances the quality improvement is more drastic when more time is allowed and the solution improvement is constant until the last iterations, as can be observed by the comparison of the Time* and Time columns. This is further illustrated in Figure 3.6 where the evolution along time of the best solution found by the three different FSPD versions is depicted on an XXL instance. Note that, the different convergence speed of the algorithms shown in the figure is due to the simulated annealing criteria whose annealing schedule is based on the total number of iterations while the initial and final temperatures are kept fixed. By observing the figure it is arguable that with even longer runs FSPD may further improve the solution quality, although such improvement could be relatively marginal. The running time for XXL instances is below 2, 10, and 20 minutes for FSPD, FSPD-mid, and FSPD-long, respectively.

VRPMPD Benchmark Instances.

As for the VRPSPD large-scale instances, six new datasets have been generated using the same approach described in Section 3.5.3. In particular, starting from the X dataset for CVRP proposed by Uchoa et al. (2017) we generated dataset XH, XQ, and XT VRPMPD datasets for which the results are reported in Table 3.9.

Table 3.9: Results on the new large-scale VRPMPD XH, XQ, and XT instances.

Algorithm	XH			XT			XQ		
	Avg	Time*	Time	Avg	Time*	Time	Avg	Time*	Time
FSPD	0.665	28.013	36.923	0.323	34.172	41.283	0.350	32.689	40.693
FSPD-mid	0.317	131.377	185.643	0.160	163.705	206.805	0.179	158.888	205.110
FSPD-long	0.225	255.780	372.402	0.105	322.414	412.839	0.133	310.485	412.412

Similarly, starting from the XXL instances for CVRP proposed by Arnold, Gendreau, and Sörensen (2019) we generated datasets XXLH, XXLQ, and XXLT, for which the results are presented in Table 3.10.

The testing on VRPMPD substantially confirms our observations made for large and very large-scale VRPSPD instances.

3.6 Algorithmic Components Analysis

This section provides some insights on the main new algorithmic components introduced in FSPD to better understand their role and impact on the overall algorithm.

3.6.1 Segment Attributes Preprocessing

The first aspect we considered regards the resource data that we compute initially and update with each change in the solution as introduced in Section 3.4.2. This particular aspect solely affects the

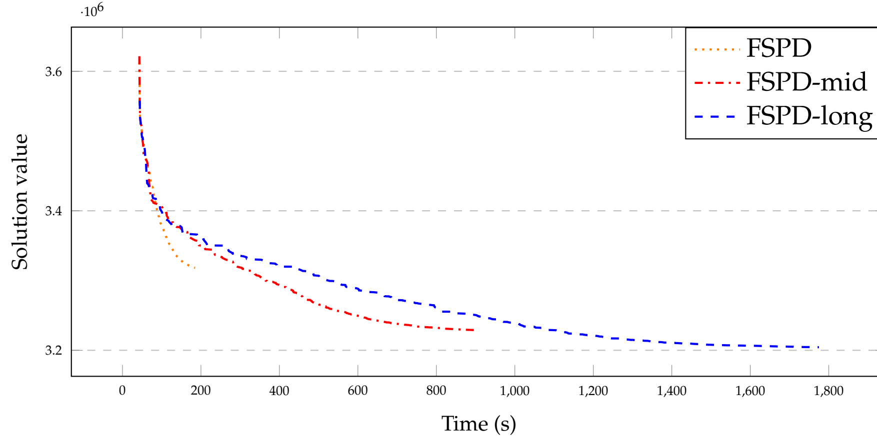


Figure 3.6: Typical evolution over time of the best solution found for XXL instances. The data refers to instance Flanders2X with seed 0. The different convergence speed of the algorithms is due to the simulated annealing criteria whose annealing schedule is based on the total number of iterations while the initial and final temperatures are kept fixed.

Table 3.10: Results on the new very large-scale VRPMPD XXLH, XXLT, XXLQ instances.

Algorithm		**XXLH**			**XXLT**			**XXLQ**	
	Avg	**Time***	**Time**	**Avg**	**Time***	**Time**	**Avg**	**Time***	**Time**
FSPD	2.172	98.195	98.478	1.165	103.306	103.503	1.283	102.984	103.181
FSPD-mid	0.607	474.207	475.908	0.286	502.077	503.345	0.322	503.715	505.082
FSPD-long	0.161	952.016	955.609	0.077	1003.172	1005.620	0.094	1004.766	1008.189

time efficiency of the algorithm and has no impact on the search trajectory itself.

There are various approaches that can enhance algorithm speed through the utilization of resources and REFs. The effectiveness of these techniques can vary depending on the algorithm and the frequency and type of resource usage.

We explored three different types of preprocessing techniques:

- No preprocessing (JIT): In this approach, every resource is computed from scratch whenever it is required.
- Forward and backward resources storage (BA): We store the resources from the depot to each node (forward) and from the node back to the depot (backward) along the current route. All other resources are computed from scratch when needed.
- Forward and backward resources (FULL), along with sub-segment resources storage: in addition to the forward and backward resources, we also store the resources of route segments up to size three (the longest segments treated in our local search operators).

Given that each main core iteration of the FSPD algorithm focuses on a localized set of nodes, determining the most effective technique is not straightforward. On one hand, employing advanced preprocessing techniques can prevent repetitive computations of the same resources. At the same time, incorporating meta-data into the solution data structure increases the cost of the copy operation. This copy operation is applied at every iteration and can significantly contribute to the overall computing time, particularly when paired with lightweight iterations as in the case of FSPD. This aspect generated a tradeoff between the amount of resources computed on the spot and those that are precomputed. As we noticed during the FSPD development, this tradeoff is highly sensitive to the specific algorithmic choices and it may vary with different algorithmic designs (primarily depending on the number of resources queried at each iteration). To reduce the tuning space, during our tests, we fixed all the other algorithmic components to their best-performing version. To evaluate the scaling behavior of the techniques under investigation, we employed a dataset comprising seven instances from the minimal subset of the dataset proposed by Uchoa et al. (2017), as defined

in Queiroga et al. (2020). These seven instances encompass the complete range of characteristics considered in the entire dataset. Furthermore, we added ten instances from the dataset introduced by Arnold, Gendreau, and Sörensen (2019), resulting in a total of 17 instances. For each instance, we considered the five variants, namely H, T, Q, X, and Y, obtained as previously described. Consequently, our tuning set comprises 85 instances, ranging from 393 to 30,000 customers, considering both VRPSPD and VRPMPD variants.

To enhance the reliability of the results, each test was repeated ten times, by using different random seeds.

Table 3.11: Average computing time (in seconds) for the three REF preprocessing techniques

Instance	Size	RR50-JIT	RR50-BA	RR50-FULL	RR100-JIT	RR100-BA	RR100-FULL
X-n393-k38	393	39.071	31.752	31.911	42.127	32.983	32.949
X-n469-k138	469	37.836	36.405	35.884	38.338	36.771	36.177
X-n561-k42	561	47.572	35.305	35.920	59.005	39.433	40.094
X-n670-k130	670	55.390	47.492	47.328	63.143	51.593	51.764
X-n716-k35	716	57.283	40.835	41.574	72.802	45.945	47.099
X-n801-k40	801	57.021	41.469	42.001	68.398	46.283	47.004
X-n979-k58	979	70.397	53.188	53.937	84.073	58.135	59.267
Leuven1	3001	82.923	63.738	66.503	111.187	78.692	82.398
Leuven2	4001	252.233	69.351	74.471	463.928	93.141	100.279
Antwerp1	6001	98.124	73.807	86.438	139.122	94.786	109.886
Antwerp2	7001	186.427	72.295	88.176	354.677	101.770	121.114
Ghent1	10001	114.597	94.322	126.418	164.403	123.085	153.062
Ghent2	11001	257.955	86.567	120.883	455.457	110.374	142.928
Brussels1	15001	126.798	110.382	124.407	192.600	139.499	152.215
Brussels2	16001	244.423	111.574	123.324	424.538	136.827	148.378
Flanders1	20001	174.658	158.814	172.003	262.721	190.944	205.542
Flanders2	30001	411.580	181.283	217.633	702.112	205.574	241.551
Avg		**136.135**	**76.975**	**87.577**	**217.566**	**93.284**	**104.218**

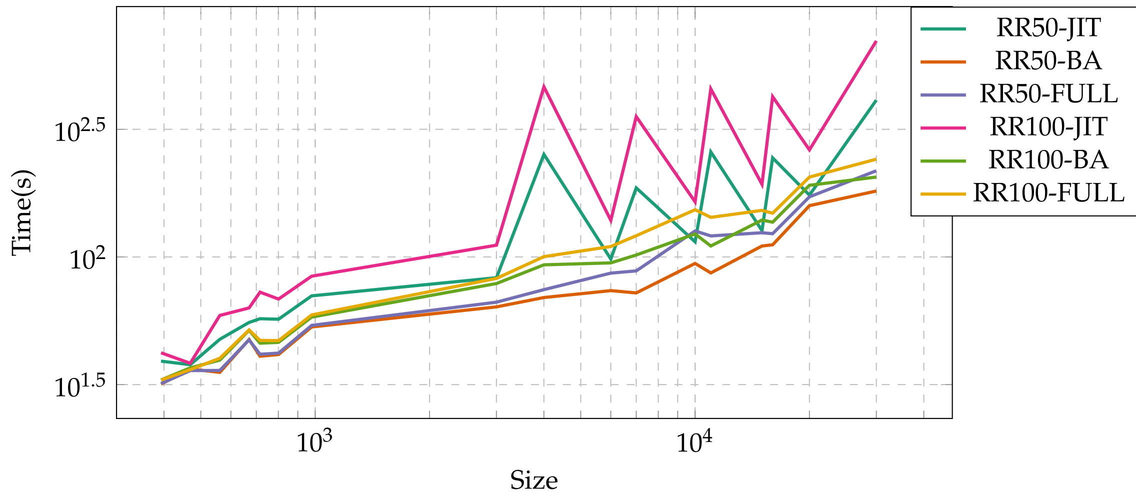


Figure 3.7: Plots of the average computing time with respect to instance size for the three REF preprocessing techniques

We tested the three variants with recreate-effort values of 50 and 100, which display the best trade-offs between quality and computing time as shown in Section 3.6.2. The results of our testing are reported in Table 3.11 and in Figure 3.7. Since we are mostly interested in studying the scaling profile of the different configurations, for each instance we aggregate the five variants (H, Q, T, X, Y) and the ten different random seeds, showing the average value obtained.

When examining the computing time relative to the instance size, we observe that the first variant, without preprocessing, performs competitively in instances characterized by relatively short routes, while requiring significantly more time for instances with longer routes. Among the other two preprocessing techniques, the second exhibits better overall running times. Although one might intuitively expect that computing route segments up to size three (matching the length of the segments

exchanged by our local search operators) would be an obvious choice, in reality, these sequences are small enough to be quickly computed. Consequently, storing them as additional data has not been proved advantageous for the proposed algorithm.

3.6.2 Recreate Tuning

In this section, we will examine in depth the algorithmic alternatives we considered in the recreate phase of FSPD which required several new features with respect to the original FILO approach, as we have previously described in Section 3.4.2.

Recreate Phase Variants.

In the proposed approach, we have redesigned this step for the purpose of improving its efficiency. Previously, we provided a simplified overview of the underlying rationale behind the new implementation. In this section, we will examine four distinct variants that introduce changes to implementation details and assess their impact on the overall algorithm performance.

First of all, a simple yet effective technique already used in the literature (e.g. Christiaens and Vanden Berghe (2020)), regards the use of a fast, route-wise, heuristic check for the feasibility, that skips all the routes whose pickup-sum or delivery-sum would exceed the capacity in case a given customer would be inserted into it. This is clearly a necessary but not sufficient condition for feasibility, and, being performed at the route level it is often able to quickly discard entire routes, greatly reducing the number of possible insertion locations considered especially when iterating over the whole solution as in the original FILO recreate phase (which we considered in the tests that follow).

Recalling what has been described in Section 3.4.2, the proposed recreate phase incorporates an intuitive parameter that, given a node, indicates the number of its neighbors considered for reinsertion into the solution. However, there exist multiple ways for counting these neighbors, each associated with a different combination of computational effort, resulting quality, and generalization capabilities.

Given a customer c that we wish to reintroduce back into the solution, we considered four alternatives for counting the neighbors during the recreate phase:

- Variant 1 (V1): Considers only the N_R nearest neighbors of c.
- Variant 2 (V2): Considers the N_R nearest neighbors for which at least one of the two adjacent insertion positions is feasible.
- Variant 3 (V3): Considers the N_R nearest neighbors which are inside routes that satisfy the route-wise feasibility check.

Furthermore, we introduced a fourth variant (V4), which incorporates the route-wise feasibility check in the second version as well. This inclusion aims to reduce the computing time while having no impact on the search trajectory, which remains identical to the one of V2.

We used the same dataset of Section 3.6.1, since, also in this case, our main focus was on achieving a good scaling profile.

We conducted tests on the four recreate variants using two different recreate pruning levels: 50 and 100. Table 3.14 and Table 3.15 present respectively the average relative gaps with the BKS and the computing time. Also in this case, the results are sorted by instance size and the average over the five instance variants (X, Y, H, Q, T), and the ten random seeds is displayed for each generating CVRP instance. Figure 3.8 shows how the quality is affected by different neighbor counting methodologies while Figure 3.9 compare computing time with the different version, using logarithmic scales to more clearly depict the profile of each version.

The first version, while being the fastest, produces solutions of noticeably lower quality in instances where routes are already nearly filled. This results in an average gap of 2.48 and 2.14 for the pruning levels of 50 and 100, respectively.

The second version exhibits characteristics opposite to the first one. It effectively handles solutions with full routes, as full routes do not result in wasting candidate positions. Consequently, it achieves average gaps of 1.52 and 1.51 for the pruning levels of 50 and 100 respectively. However, the repeated

Table 3.12: Average gaps for the 4 recreate variants with recreate effort of 50 and 100. For each instance, the values obtained for the 5 H, Q, T, X, and Y are averaged together

Instance	Size	RR50-V1	RR50-V2	RR50-V3	RR50-V4	RR100-V1	RR100-V2	RR100-V3	RR100-V4
X-n393-k38	393	1.028	0.495	0.515	0.495	0.694	0.483	0.497	0.483
X-n469-k138	469	6.011	0.922	0.937	0.922	4.415	0.938	0.954	0.938
X-n561-k42	561	1.524	0.909	0.886	0.909	1.234	0.844	0.862	0.844
X-n670-k130	670	3.553	1.176	1.206	1.176	3.086	1.416	1.403	1.416
X-n716-k35	716	1.294	0.806	0.794	0.806	1.082	0.890	0.876	0.890
X-n801-k40	801	1.123	0.503	0.487	0.503	0.889	0.430	0.406	0.430
X-n979-k58	979	1.057	0.657	0.651	0.657	0.875	0.693	0.673	0.693
Leuven1	3001	1.149	0.895	0.898	0.895	0.995	0.872	0.860	0.872
Leuven2	4001	3.125	2.712	2.746	2.712	2.791	2.593	2.620	2.593
Antwerp1	6001	2.810	1.828	1.850	1.828	2.615	1.880	1.883	1.880
Antwerp2	7001	3.028	2.076	2.126	2.076	2.658	2.064	2.062	2.064
Ghent1	10001	1.696	1.440	1.441	1.440	1.595	1.424	1.428	1.424
Ghent2	11001	2.924	2.274	2.292	2.274	2.647	2.198	2.201	2.198
Brussels1	15001	2.439	2.044	2.030	2.044	2.279	2.047	2.050	2.047
Brussels2	16001	3.469	2.752	2.753	2.752	3.111	2.572	2.591	2.572
Flanders1	20001	1.860	1.149	1.154	1.149	1.619	1.212	1.213	1.212
Flanders2	30001	4.097	3.181	3.182	3.181	3.714	3.037	3.028	3.037
Avg		**2.482**	**1.519**	**1.526**	**1.519**	**2.135**	**1.505**	**1.506**	**1.505**

Table 3.13: Average computing time (in seconds) for the 4 recreate variants with recreate effort of 50 and 100. For each instance, the values obtained for the 5 H, Q, T, X, and Y are averaged together

Instance	Size	RR50-V1	RR50-V2	RR50-V3	RR50-V4	RR100-V1	RR100-V2	RR100-V3	RR100-V4
X-n393-k38	393	29.721	40.228	31.752	31.617	31.003	42.226	32.983	32.635
X-n469-k138	469	32.709	43.423	36.405	36.182	34.067	43.676	36.771	36.280
X-n561-k42	561	31.402	45.051	35.305	35.457	32.805	52.988	39.433	39.421
X-n670-k130	670	44.079	57.169	47.492	47.313	47.864	64.350	51.593	51.486
X-n716-k35	716	39.034	58.798	40.835	40.735	41.189	71.988	45.945	45.912
X-n801-k40	801	38.163	78.957	41.469	41.246	41.393	101.709	46.283	45.934
X-n979-k58	979	48.422	99.309	53.188	52.982	52.160	117.119	58.135	58.088
Leuven1	3001	52.772	234.843	63.738	63.490	58.201	351.164	78.692	78.677
Leuven2	4001	58.590	80.476	69.351	70.403	68.647	128.535	93.141	94.956
Antwerp1	6001	61.899	311.514	73.807	73.325	67.619	513.920	94.786	94.675
Antwerp2	7001	65.545	113.792	72.295	73.093	73.023	257.753	101.770	104.163
Ghent1	10001	78.463	526.760	94.322	93.810	86.429	1009.430	123.085	121.309
Ghent2	11001	85.059	116.448	86.567	87.132	93.868	253.690	110.374	111.445
Brussels1	15001	99.824	356.427	110.382	109.748	108.559	784.853	139.499	138.058
Brussels2	16001	109.082	142.812	111.574	111.889	117.931	379.514	136.827	137.285
Flanders1	20001	144.216	518.407	158.814	157.883	153.589	1027.148	190.944	189.886
Flanders2	30001	182.405	217.978	181.283	181.798	192.000	303.370	205.574	205.998
Avg		**70.670**	**178.964**	**76.975**	**76.947**	**76.491**	**323.731**	**93.284**	**93.306**

feasibility computations significantly slow down the overall algorithm, causing the second version to take more than twice the computing time, on average, when compared to the first version.

The last two versions address the limitations of the first ones by introducing the heuristic route-wise feasibility check, which provides a rapid rejection mechanism for infeasible insertion positions. As can be observed from the results, this simple check proves to be particularly effective in improving the quality of the first version with V3 (Table 3.12) and in accelerating the second version with V4 (Table 3.13).

However, employing this check to speed up the first version results in a technique that is not easily generalizable to other VRP variants. Not all variants may have a similar route-wise feasibility check, or if such a check exists, it may possess different infeasibility detection capabilities, thereby affecting the quality of the obtained solutions. Moreover, counting only the positions that pass this check creates a technique that is effective for the tested instances but lacks any quality guarantees. There may exist instances where the solutions have routes capable of passing the heuristic route-wise feasibility check but fail the position-wise feasibility check. Thus, although both these variants perform well in practice, we selected variant V4 over V3, as we consider it a more robust technique. The choice of V4 provides us with the expectation of better generalization of our technique when extended to other variants that may have different route-wise feasibility checks.

Lastly, we can observe that counting only the neighbors that have at least one feasible insertion position nearby produces very similar results for both versions with recreate-effort levels of 50 and 100, while significantly reducing the computing time of the former case.

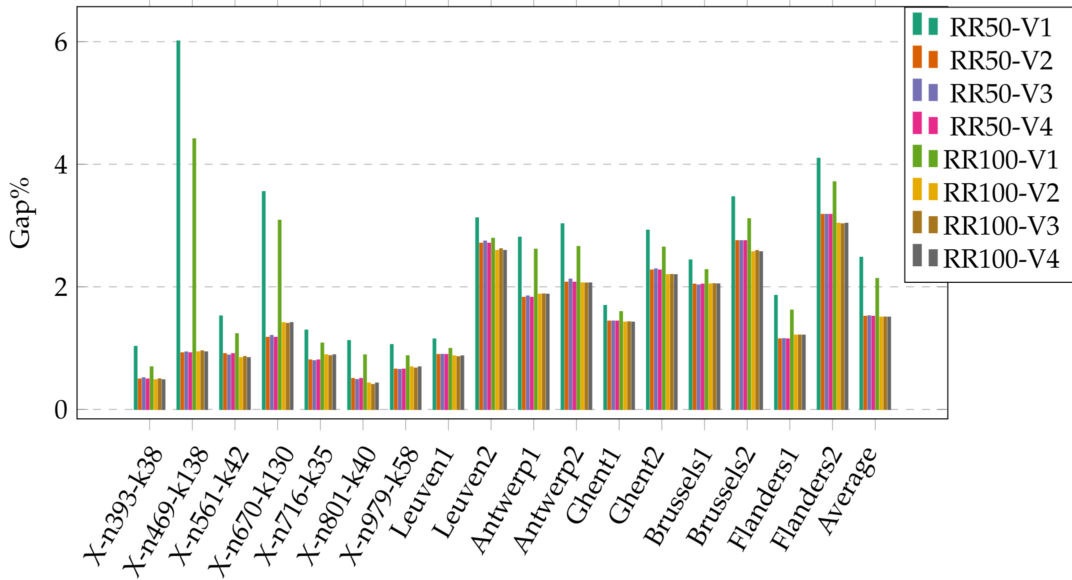


Figure 3.8: Plots of the average gaps sorted by size for the 4 recreate variants with recreate effort of 50 and 100. For each instance, the values obtained for the 5 H, Q, T, X, and Y are averaged together

Tuning of Ruin and Recreate Pruning Factor.

After selecting a suitable recreate variant, we can focus on the tuning of the *recreate-effort*, i.e., the parameter controlling the level of pruning during the recreate phase. By tuning this parameter, we can strike a favorable balance, preserving most of the original solution quality while significantly reducing computing time.

We examined a total of eight different configurations. The first configuration corresponds to the original recreate phase of FILO (marked as RR-OLD in the following tables), which scans the entire solution in search of insertion points. Notably, when the pruning factor is sufficiently high, simple solution iterations prove to be faster than scanning the neighbor list of the node intended for insertion. We explored seven additional versions with varying values of recreate-effort, aiming to identify a range that could work reasonably well even for instances with a customer count in the tens of thousands. Specifically, we considered versions with recreate-effort values of 5, 10, 25, 50, 100, 250, and 500 (marked as RR5, . . . , RR500 OLD in the following tables).

Table 3.14: Average gaps for the different levels of recreate effort and FILO's original recreate technique.

Instance	Size	RR5	RR10	RR25	RR50	RR100	RR250	RR500	RR-OLD
X-n393-k38	393	0.534	0.488	0.432	0.495	0.483	0.435	0.431	0.477
X-n469-k138	469	0.914	0.936	0.969	0.922	0.938	0.922	0.893	0.923
X-n561-k42	561	0.879	0.800	0.761	0.909	0.844	0.803	0.843	0.885
X-n670-k130	670	1.022	0.999	1.154	1.176	1.416	1.376	1.364	1.428
X-n716-k35	716	0.991	0.869	0.811	0.806	0.890	0.873	0.898	0.906
X-n801-k40	801	0.654	0.590	0.489	0.503	0.430	0.438	0.489	0.447
X-n979-k58	979	0.442	0.464	0.598	0.657	0.693	0.731	0.772	0.780
Leuven1	3001	1.035	0.956	0.855	0.895	0.872	0.888	0.885	0.910
Leuven2	4001	3.209	2.931	2.935	2.712	2.593	2.619	2.692	2.676
Antwerp1	6001	2.066	1.954	1.852	1.828	1.880	1.862	1.910	1.993
Antwerp2	7001	2.592	2.389	2.156	2.076	2.064	2.041	2.181	2.257
Ghent1	10001	1.551	1.466	1.433	1.440	1.424	1.500	1.497	1.569
Ghent2	11001	2.930	2.749	2.459	2.274	2.198	2.165	2.187	2.165
Brussels1	15001	2.251	2.157	2.041	2.044	2.047	2.107	2.131	2.235
Brussels2	16001	3.481	3.238	2.946	2.752	2.572	2.525	2.575	2.590
Flanders1	20001	1.282	1.179	1.122	1.149	1.212	1.287	1.328	1.434
Flanders2	30001	3.774	3.631	3.374	3.181	3.037	3.009	3.035	3.143
Avg		**1.742**	**1.635**	**1.552**	**1.519**	**1.505**	**1.505**	**1.536**	**1.578**

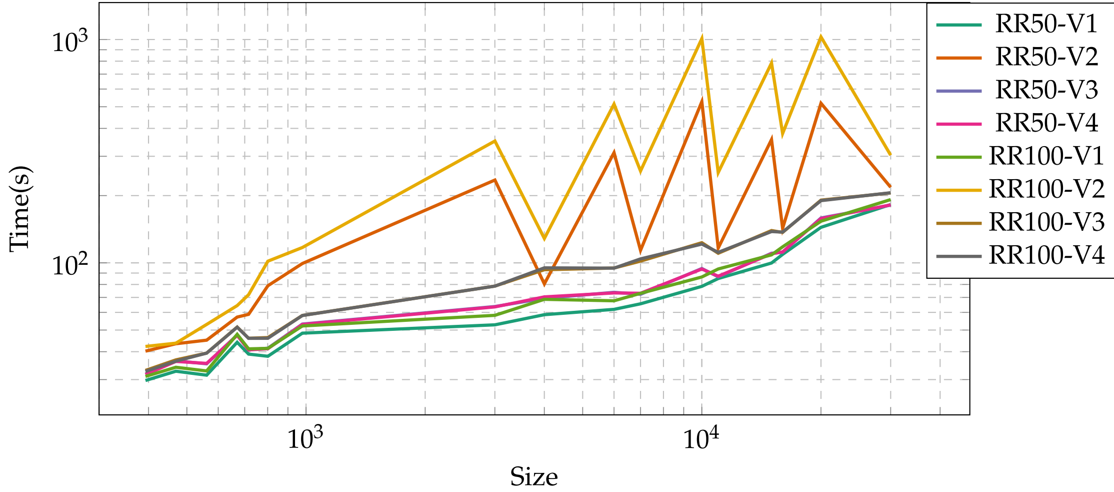


Figure 3.9: Plots of the average computing time with respect to instance size for the 4 recreate variants with recreate effort of 50 and 100. For each instance, the values obtained for the 5 H, Q, T, X, and Y are averaged together

Table 3.15: Average computing time (in seconds) for the different levels of recreate effort and FILO's original recreate technique.

Instance	Size	RR5	RR10	RR25	RR50	RR100	RR250	RR500	RR-OLD
X-n393-k38	393	27.553	28.110	29.709	31.582	32.863	33.511	33.956	28.427
X-n469-k138	469	34.584	35.146	36.476	36.055	36.575	37.099	37.070	35.461
X-n561-k42	561	29.457	30.267	32.617	35.629	39.742	44.418	47.067	32.979
X-n670-k130	670	43.197	43.920	45.507	47.653	51.927	58.988	62.903	48.803
X-n716-k35	716	33.639	34.946	37.582	41.204	46.498	53.319	57.413	41.976
X-n801-k40	801	31.918	33.342	37.049	41.527	46.367	50.254	53.409	38.767
X-n979-k58	979	44.394	46.005	49.547	53.366	58.584	66.981	74.152	56.579
Leuven1	3001	46.198	47.837	53.723	63.824	79.039	103.857	122.230	85.116
Leuven2	4001	47.049	50.508	58.086	70.036	94.854	174.096	309.472	375.175
Antwerp1	6001	51.897	53.873	60.979	73.638	94.871	138.071	180.523	144.853
Antwerp2	7001	50.404	53.368	60.583	73.210	104.037	202.471	336.417	331.962
Ghent1	10001	66.414	68.478	77.513	93.558	121.560	171.436	213.405	235.539
Ghent2	11001	66.160	69.183	76.095	87.107	111.546	194.489	324.823	504.103
Brussels1	15001	85.665	88.211	96.026	110.744	138.808	206.171	282.118	387.810
Brussels2	16001	91.481	94.439	101.356	112.080	137.055	218.398	356.726	759.884
Flanders1	20001	131.096	133.115	141.788	158.230	190.755	278.504	405.867	679.633
Flanders2	30001	160.188	162.793	169.937	181.775	206.219	332.927	566.347	1974.249
Avg		**61.253**	**63.149**	**68.504**	**77.131**	**93.606**	**139.117**	**203.759**	**338.901**

Tables 3.14 and 3.15 report respectively the average gap and computing times for the tested recreate-effort values. As in the previous sections, in this case, we have been mainly focused on achieving a good tradeoff with very large instances, and thus we have used the same dataset, comprising 7 X instances and 10 XXL ones. As in the previous tables, we report the results by aggregating both the five variants (X, Y, H, Q, T) and the ten random seeds run for each one of them.

From the obtained results, we observed that recreate-effort values of 50 and 100 represent the best tradeoff between quality and speed. As clearly displayed in Figures 3.10 and 3.11, versions with recreate-effort below 50 exhibit faster execution but suffer from a decrease in quality, particularly with large instances. On the other hand, instances with recreate-effort values above 100 demonstrate longer computing times while also yielding equal or lower average solution quality in extreme cases. Therefore, recreate-effort values of 50 and 100 can be considered suited to get a favorable balance between quality and speed. We selected the value of 50 for our complete computational tests displayed in Appendix 3.9.

Tuning of Ruin and Recreate Pruning Factor – Small instances.

With some surprise, during our testing we observed that recreate-effort values of 50 or 100 can have a detrimental effect in a few of the smaller classical instances. This observation aligns with

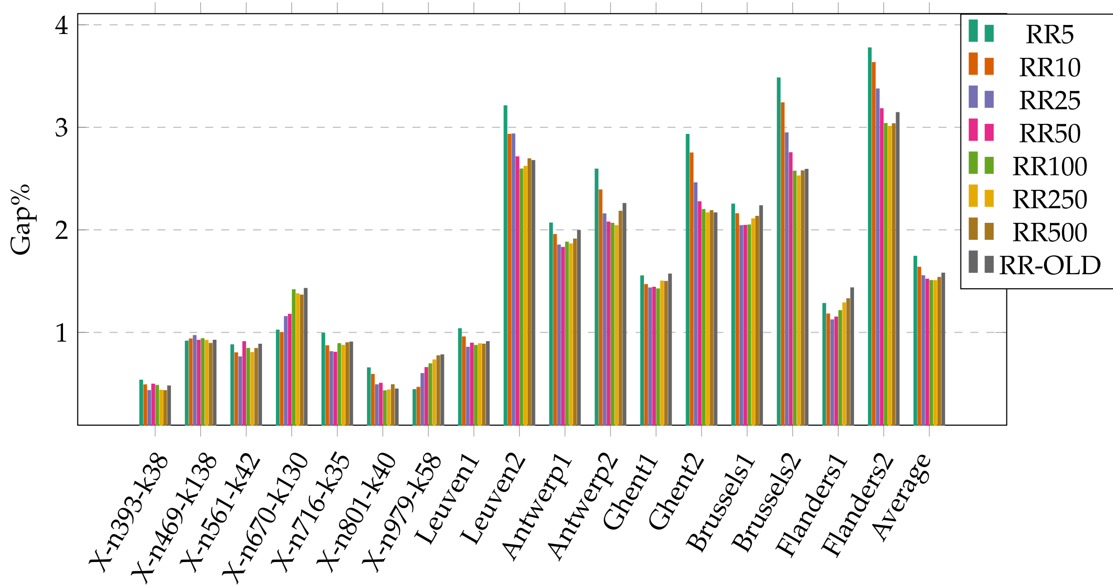


Figure 3.10: Plots of the average gaps sorted by size for the different levels of recreate effort and FILO's original recreate technique.

the findings from the tuning process discussed in the previous section, which revealed that a large recreate-effort can negatively impact solution quality even for very large instances.

To address this issue, we devised an additional criterion to control the value of recreate-effort specifically for smaller instances, thereby targeting the opposite end of the size spectrum compared to the previous tests. We assigned different fractions of the instance size to determine the recreate-effort value. To merge the two criteria, we selected the minimum recreate-effort computed using both approaches.

We conducted tests using the following instance-size fractions: 0%, 1%, 2%, 5%, 10%, 20%, 50%, and 100%.

The tuning was performed using a second dataset composed of the five variants X, Y, H, Q, and T from the dataset by Salhi and Nagy (1999), along with the 18 instances from Montané and Galvão (2006). Consequently, this second dataset encompasses a total of 88 instances, ranging from 50 to 400 customers.

Table 3.16: Average gaps and computing time obtained using different fractions of the instance size as recreate-effort.

	RR50-0%	**RR50-1%**	**RR50-2%**	**RR50-5%**	**RR50-10%**	**RR50-20%**	**RR50-50%**	**RR50-100%**
Gap	0.127	0.094	0.103	0.098	0.094	0.092	0.130	0.130
Time(s)	32.520	32.177	32.135	33.304	34.475	35.439	36.153	36.192

We observed remarkable robustness in our approach, as shown in Table 3.16 which reports the aggregated average relative gap and computing time for all the instances considered in this small dataset. When we set a recreate-effort value lower than 50 neighbors, the problematic instances ceased to yield noticeably poor results. All the fractions proved to work comparably well in terms of computing times. Consequently, we arbitrarily decided to employ 20% of the instance size as the second criterion. This choice was based on the fact that, when combined with a maximum recreate-effort of 50 neighbors, it yielded a slightly better average gap for the smaller instances.

3.6.3 Super-linear algorithmic phases management

One of the main innovations introduced in FSPD is a more appropriate handling of some steps which absorb an increasing portion of the overall computational effort when dealing with large-scale instances.

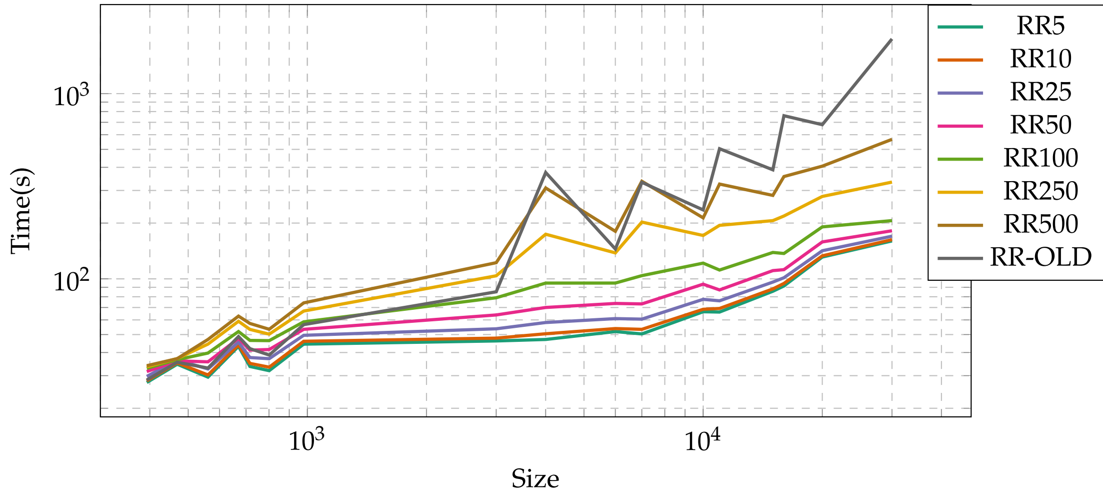


Figure 3.11: Plots of the average computing time with respect to instance size for the different levels of recreate effort and FILO's original recreate technique.

Currently, the largest of the popular instances that can be found in the literature for routing problems is a CVRP instance which contains 30,000 customers proposed by Arnold, Gendreau, and Sörensen (2019). With respect to instance sizes in the most used benchmarks, this is already one or two orders of magnitude larger and presents a new and different set of challenges to tackle in the development of new algorithms. FSPD already inherits from FILO an overall design explicitly tailored for instances of this size (e.g., the GNs, SVC, and SMD have been mixed together to make XXL instances tractable). However, with the current hardware capabilities, we are reaching a limit in both the performance of some steps of the algorithm and in its memory occupation. Currently, our algorithm with 30,000 customers uses around $10 GByte$ of main memory at its peak. The two main sources of such memory occupation are the quadratic-size cost matrix, and the nearest-neighbors matrix, which for each vertex contains a sorted list of the other vertices sorted by distance thus also requiring a total number of $O(N^2)$ elements. Therefore, even if with the current maximum-sized instances we can still maintain the same approach that we have with smaller ones, it is also clear that we are close to the limit and that to further increase the instance size we need to adapt our algorithm. Luckily, the problem of the cost matrix can be solved simply by either computing the distance using the relative distance function when edge costs are implicit (which can be coupled with a small cache to greatly speed up the process like in Bentley (1990) and Helsgaun (2000), or by retrieving it from a file when the matrix cost is explicit, with the associated overhead. At the same time, the neighbors-matrix can be reduced in two different ways:

- by heuristically limiting the number of neighbors for each customer, which makes the used space linear, but also introduces a trade-off with the solution quality which rapidly degrades when not enough neighbors are selected;
- by adopting more complex data structures (e.g., k-d trees), which can provide the full list of sorted neighbors while requiring less space.

Another problem that might occur regards the computation effort needed to obtain the neighbors-matrix, which, using standard techniques, requires $O(N^2 log N)$ because the full list of vertices must be sorted for each vertex. Currently, this step requires about 20% of the total time of the algorithm for instances of 30,000 customers and 100,000 core iterations of the algorithm. We drastically reduced such time by replacing the standard sorting algorithm that can be found in the C++ STL with the efficient implementation of radix-sort by Skarupke (2016), thus halving the time taken by this step. Figure 3.12 reports the speedup obtained due to this algorithmic change for different instance sizes. Furthermore, to have a qualitative idea of the asymptotic behavior we might expect when we address instances with more than 30,000 customers, we performed a brief analysis of the steps of the algorithm which displays a super-linear time complexity. To this end, we run the X-n1001-k43 instance of the X dataset proposed by Uchoa et al. (2017) involving 100 to 1,000 customers, and each XXL instance proposed by Arnold, Gendreau, and Sörensen (2019), including up to 30,000

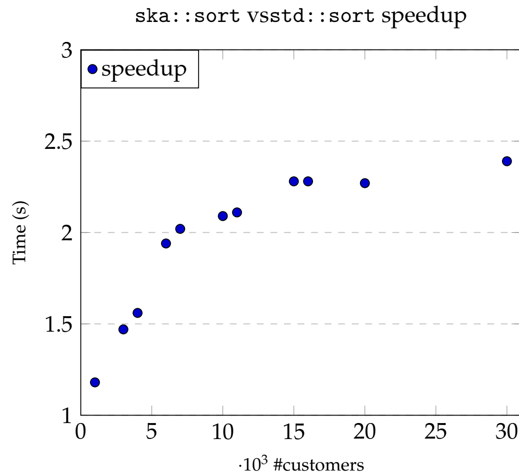


Figure 3.12: Speedup obtained substituting the C++ standard library sort algorithm, with Malte Skarupke's radix-sort implementation.

customers, with 10 different seeds by using both standard sort and radix-sort. In these runs, we measured the time needed by the pre-processing phase and that of the core optimization (coreopt) phase when performing 100, 000 iterations, to evaluate the ratio between the two.

To produce a very rough forecast of what we might expect from larger instances, we first computed a trend-line for both the preprocessing based on the standard sorting and radix-sort (see Figures 3.13)

As we can see, the trend lines fit the points for both the preprocessing data very well, while the coreopt iterations are quite scattered (the average linear complexity is still maintained). If we extend the trend-lines until the preprocessing time reaches the coreopt iterations time, we can notice that at an instance size of around 80, 000 customers, the standard sort preprocessing last as much as the coreopt iterations, while the radix-sort preprocessing holds up until around 210, 000 customers.

From this very rough analysis, we can see that current preprocessing techniques already occupy a good fraction of the total computational time (around 20% for the bigger 30, 000-customers instance). With these sizes, switching to a faster sorting algorithm can be a simple and fast fix to the problem, since the change can be applied affecting only a few lines of code.

However, we might expect that when (and if) sizes of hundreds of thousands are reached, quadratic steps will need to be completely avoided and other more sophisticated techniques (based for example on k-d trees as in Bentley (1990)) might become the best-suited choice, even though definitely more complex to implement.

3.7 Conclusions

In this paper, we present an effective heuristic, called FSPD, for two challenging VRPs arising in the city logistics context, namely the VRPSPD and the VRPMPD. In addition to the comparison of FSPD with the best state-of-the-art VRPSPD and VRPMPD algorithms, which shows its competitiveness, extensive computational experiments are performed on very large-scale instances of the problems. This is the first computational study ranging from classical medium-sized benchmarks to very large-scale ones for these problems. The statistical analysis shows that FSPD, in its short version, is almost always comparable or better than the other algorithms despite being a very fast algorithm, while FSPD-long is always comparable or better. The added value of FSPD relies on its capability to solve nowadays realistic-sized instances that are one or two orders of magnitude larger than those included in the most used benchmarks while keeping its time performance within a linear scalability trend. It is worth pointing out that such scalability is reached while dealing with new features with respect to the original FILO framework proposed by Accorsi and Vigo (2021) for the CVRP. Computational results demonstrate the effectiveness of FSPD in solving large-scale instances. Remarkably, results show that FSPD reaches good quality solutions with 100, 000 core optimization

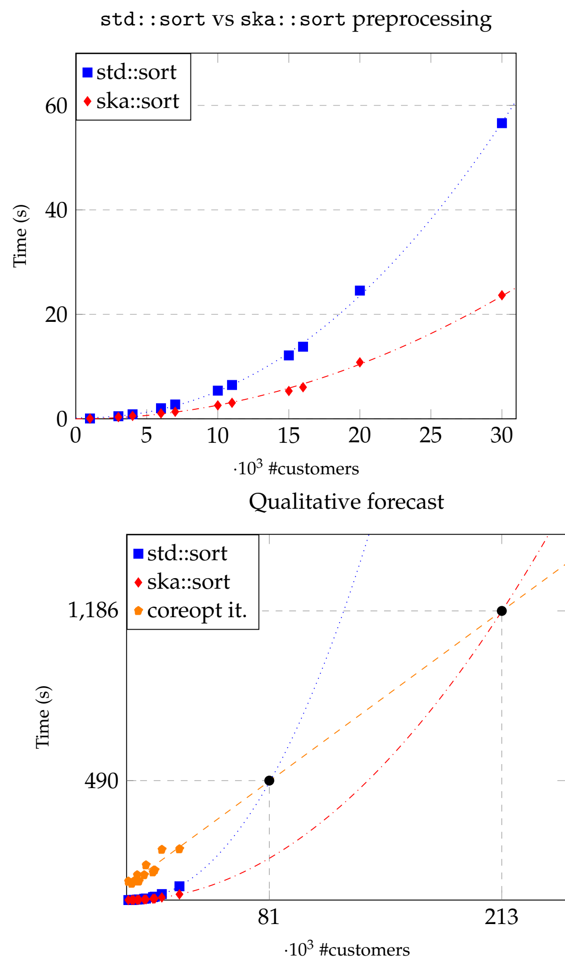


Figure 3.13: On the left, time comparison between preprocessing based on C++ standard library sort algorithm (blue) and Malte Skarupke's radix-sort implementation (red). For both series, the trend-line is also reported. On the left, the same plot at a different scale with the addition of the time needed by 100, 000 coreopt iterations. The two intersection points represent a forecast of the size at which the preprocessing step will need the same amount of time as the actual refinement step.

iterations while remaining below a computing time of 0.5 minutes.
There are several directions for future work. First, the new technologies introduced new challenges in classical VRP frameworks, such as the possibility of providing fast services to a large number of customers in a very short time (Same Day Delivery) within the urban areas of mega-cities. Thus, there is value in studying how to serve such a large number of customers, while considering several features within a more general project that studies the very large-scale multi-attribute VRPs. Second, in addition to customer service, it is worth investigating VRPs on large-scale instances where the workload for drivers/couriers is balanced.

3.8 Acknowledgments

This research was partially funded by the U.S. Air Force Office of Scientific Research [Award FA8655-21-1-7046], and by the University of Calabria through project grant CIG8929018ABE.

3.9 Appendix A: Complete results

3.9.1 VRPSPD Instances

Table 3.17: Full results for dataset CMTX

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	**Avg**	**Time***	**Time**	**Best**	**Avg**	**Time***	**Time**	**Best**	**Avg**	**Time***	**Time**
CMT1X	50	466.77	0.001	0.001	24.844	0.067	0.001	0.001	124.455	0.072	0.001	0.001	249.026	0.073
CMT6X	50	555.43	0.000	0.000	37.211	0.282	0.000	0.000	186.859	0.322	0.000	0.000	372.287	0.320
CMT2X	75	684.21	0.000	0.036	22.101	1.734	0.000	0.012	118.980	5.747	0.000	0.000	229.207	9.507
CMT7X	75	900.12	0.000	0.011	39.300	15.499	0.000	0.000	194.769	42.788	0.000	0.000	447.297	31.736
CMT3X	100	721.27	0.000	0.002	31.228	8.436	0.000	0.000	158.772	22.878	0.000	0.000	320.911	36.976
CMT12X	100	662.22	0.000	0.019	26.918	1.782	0.000	0.000	147.968	1.383	0.000	0.000	295.159	1.551
CMT8X	100	865.5	0.000	0.000	45.847	0.264	0.000	0.000	232.025	0.386	0.000	0.000	459.830	0.345
CMT14X	100	821.75	0.000	0.000	55.554	0.061	0.000	0.000	277.739	0.068	0.000	0.000	555.542	0.068
CMT11X	120	833.92	0.000	0.000	29.447	8.885	0.000	0.000	148.534	43.832	0.000	0.000	302.089	63.411
CMT13X	120	1541.14	0.112	0.125	42.300	28.276	0.112	0.112	226.241	105.158	0.112	0.112	476.908	183.574
CMT4X	150	852.46	0.000	0.026	31.608	5.269	0.000	0.000	156.520	17.889	0.000	0.000	327.979	16.213
CMT9X	150	1160.68	0.000	0.000	42.537	8.289	0.000	0.000	228.073	24.365	0.000	0.000	465.816	27.213
CMT5X	199	1029.25	0.000	0.006	33.524	10.653	0.000	0.000	185.133	22.198	0.000	0.002	374.231	36.523
CMT10X	199	1373.4	0.000	0.561	36.841	16.542	0.000	0.131	167.271	123.111	0.000	0.069	337.484	262.328
Avg			**0.008**	**0.056**	**35.661**	**7.574**	**0.008**	**0.018**	**182.381**	**29.300**	**0.008**	**0.013**	**372.412**	**47.846**

Table 3.18: Full results for dataset CMTY

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	**Avg**	**Time***	**Time**	**Best**	**Avg**	**Time***	**Time**	**Best**	**Avg**	**Time***	**Time**
CMT1Y	50	466.77	0.001	0.001	24.477	0.072	0.001	0.001	122.938	0.051	0.001	0.001	246.201	0.051
CMT6Y	50	555.43	0.000	0.000	36.897	0.409	0.000	0.000	186.365	0.250	0.000	0.000	371.916	0.251
CMT2Y	75	684.21	0.000	0.062	23.087	2.698	0.000	0.000	112.867	8.157	0.000	0.000	239.402	3.970
CMT7Y	75	900.12	0.000	0.022	39.822	15.862	0.000	0.000	205.490	59.457	0.000	0.000	436.220	37.259
CMT3Y	100	721.27	0.000	0.000	30.545	10.145	0.000	0.000	163.016	16.880	0.000	0.000	334.039	22.731
CMT12Y	100	662.22	0.000	0.000	26.361	1.738	0.000	0.000	141.755	2.487	0.000	0.000	289.171	1.831
CMT8Y	100	865.5	0.000	0.000	45.782	0.379	0.000	0.000	229.994	0.356	0.000	0.000	460.101	0.393
CMT14Y	100	821.75	0.000	0.000	55.490	0.059	0.000	0.000	278.928	0.059	0.000	0.000	560.044	0.057
CMT11Y	120	833.92	0.000	0.019	29.770	8.454	0.000	0.000	153.974	29.438	0.000	0.000	322.282	34.718
CMT13Y	120	1541.14	0.112	0.125	43.237	28.884	0.112	0.112	224.685	103.909	0.112	0.112	477.372	183.735
CMT4Y	150	852.46	0.000	0.022	33.176	3.734	0.000	0.005	164.604	27.015	0.000	0.000	342.566	13.044
CMT9Y	150	1160.68	0.000	0.000	41.617	11.240	0.000	0.000	224.586	27.521	0.000	0.000	469.208	33.002
CMT5Y	199	1029.25	0.000	0.017	32.966	9.134	0.000	0.000	178.898	26.303	0.000	0.000	365.711	48.018
CMT10Y	199	1373.4	0.020	0.642	36.190	23.807	0.000	0.252	185.018	84.485	0.000	0.197	360.926	166.119
Avg			**0.009**	**0.065**	**35.673**	**8.330**	**0.008**	**0.026**	**183.794**	**27.598**	**0.008**	**0.022**	**376.797**	**38.941**

Table 3.19: Full results for dataset Dethloff

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	**Avg**	**Time***	**Time**	**Best**	**Avg**	**Time***	**Time**	**Best**	**Avg**	**Time***	**Time**
CON3-0	50	616.52	0.000	0.000	31.196	0.025	0.000	0.000	155.940	0.025	0.000	0.000	311.215	0.025
CON3-1	50	554.47	0.000	0.000	32.289	0.030	0.000	0.000	161.520	0.030	0.000	0.000	323.138	0.030
CON3-2	50	518.0	0.000	0.000	27.323	1.106	0.000	0.000	141.699	1.088	0.000	0.000	283.938	0.965
CON3-3	50	591.19	0.000	0.000	39.432	0.036	0.000	0.000	197.229	0.036	0.000	0.000	394.520	0.036
CON3-4	50	588.79	0.000	0.000	30.274	0.018	0.000	0.000	151.248	0.018	0.000	0.000	302.311	0.018
CON3-5	50	563.7	0.000	0.000	33.022	0.008	0.000	0.000	164.857	0.008	0.000	0.000	329.584	0.008
CON3-6	50	499.05	0.000	0.000	30.946	0.227	0.000	0.000	155.519	0.170	0.000	0.000	311.991	0.182
CON3-7	50	576.48	0.000	0.000	25.273	0.029	0.000	0.000	126.506	0.029	0.000	0.000	253.042	0.029
CON3-8	50	523.05	0.000	0.000	38.753	0.019	0.000	0.000	193.890	0.019	0.000	0.000	387.429	0.019
CON3-9	50	578.25	0.000	0.000	29.456	0.053	0.000	0.000	147.319	0.051	0.000	0.000	294.417	0.051
CON8-0	50	857.17	0.000	0.000	32.992	0.258	0.000	0.000	165.926	0.351	0.000	0.000	331.254	0.331
CON8-1	50	740.85	0.000	0.000	42.967	0.031	0.000	0.000	214.963	0.032	0.000	0.000	429.293	0.032
CON8-2	50	712.89	0.000	0.000	45.895	1.067	0.000	0.000	234.921	0.769	0.000	0.000	470.116	0.737
CON8-3	50	811.07	0.000	0.000	40.588	0.044	0.000	0.000	202.551	0.043	0.000	0.000	404.838	0.044
CON8-4	50	772.25	0.000	0.000	31.389	0.056	0.000	0.000	157.198	0.056	0.000	0.000	313.771	0.056
CON8-5	50	754.88	0.000	0.000	34.712	0.164	0.000	0.000	174.614	0.104	0.000	0.000	349.575	0.103
CON8-6	50	678.92	0.000	0.000	27.894	0.177	0.000	0.000	139.807	0.145	0.000	0.000	280.360	0.244
CON8-7	50	811.96	0.000	0.000	37.280	1.084	0.000	0.000	189.655	2.239	0.000	0.000	381.345	1.606
CON8-8	50	767.53	0.000	0.000	30.549	0.081	0.000	0.000	153.232	0.081	0.000	0.000	306.221	0.081
CON8-9	50	809.0	0.000	0.000	37.828	0.041	0.000	0.000	190.007	0.041	0.000	0.000	379.266	0.041
SCA3-0	50	635.62	0.000	0.000	25.546	9.762	0.000	0.000	139.594	11.110	0.000	0.000	282.481	13.891
SCA3-1	50	697.84	0.000	0.000	28.652	0.008	0.000	0.000	143.200	0.008	0.000	0.000	285.806	0.008
SCA3-2	50	659.34	0.000	0.000	28.706	0.004	0.000	0.000	143.466	0.004	0.000	0.000	287.867	0.004
SCA3-3	50	680.04	0.000	0.000	31.088	0.130	0.000	0.000	155.878	0.114	0.000	0.000	311.785	0.090
SCA3-4	50	690.5	0.000	0.000	31.435	0.004	0.000	0.000	157.408	0.004	0.000	0.000	314.486	0.004
SCA3-5	50	659.9	0.000	0.000	24.976	0.005	0.000	0.000	124.813	0.005	0.000	0.000	249.747	0.005
SCA3-6	50	651.09	0.000	0.000	33.908	0.025	0.000	0.000	169.327	0.025	0.000	0.000	338.857	0.026
SCA3-7	50	659.17	0.000	0.000	24.921	0.203	0.000	0.000	124.901	0.281	0.000	0.000	249.535	0.281
SCA3-8	50	719.47	0.000	0.000	29.351	0.007	0.000	0.000	146.442	0.007	0.000	0.000	292.750	0.007
SCA3-9	50	681.0	0.000	0.000	33.762	0.005	0.000	0.000	169.186	0.005	0.000	0.000	337.769	0.005
SCA8-0	50	961.5	0.000	0.000	26.492	0.058	0.000	0.000	132.645	0.058	0.000	0.000	265.138	0.058
SCA8-1	50	1049.65	0.000	0.000	23.267	0.195	0.000	0.000	117.412	0.209	0.000	0.000	234.047	0.203
SCA8-2	50	1039.64	0.000	0.000	28.403	0.735	0.000	0.000	144.010	1.556	0.000	0.000	288.402	0.992
SCA8-3	50	983.34	0.000	0.000	22.471	0.094	0.000	0.000	112.655	0.091	0.000	0.000	225.801	0.092
SCA8-4	50	1065.49	0.000	0.000	30.986	0.074	0.000	0.000	155.611	0.074	0.000	0.000	311.137	0.074
SCA8-5	50	1027.08	0.000	0.000	24.618	0.228	0.000	0.000	124.031	0.267	0.000	0.000	248.229	0.266
SCA8-6	50	971.82	0.000	0.000	30.722	0.451	0.000	0.000	154.777	0.859	0.000	0.000	310.478	0.714
SCA8-7	50	1051.28	0.000	0.000	21.481	2.619	0.000	0.000	110.498	3.705	0.000	0.000	223.581	5.193
SCA8-8	50	1071.18	0.000	0.000	31.479	0.103	0.000	0.000	158.312	0.103	0.000	0.000	315.587	0.102
SCA8-9	50	1060.5	0.000	0.000	26.540	0.299	0.000	0.000	134.131	0.267	0.000	0.000	268.556	0.311
Avg			**0.000**	**0.000**	**30.971**	**0.489**	**0.000**	**0.000**	**155.923**	**0.602**	**0.000**	**0.000**	**311.992**	**0.674**

Table 3.20: Full results for dataset Montane

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	**Avg**	**Time***	**Time**	**Best**	**Avg**	**Time***	**Time**	**Best**	**Avg**	**Time***	**Time**
r101	100	1009.95	0.000	0.013	23.357	4.729	0.000	0.000	120.181	18.959	0.000	0.000	250.054	22.297
r201	100	666.2	0.001	0.001	37.703	0.365	0.001	0.001	189.603	0.682	0.001	0.001	380.578	0.721
c101	100	1220.18	0.066	0.217	32.057	8.183	0.000	0.060	188.556	10.109	0.000	0.042	352.814	96.090
c201	100	662.07	0.000	0.000	42.832	0.056	0.000	0.000	214.868	0.070	0.000	0.000	429.813	0.070
rc101	100	1059.32	0.000	0.000	25.323	3.507	0.000	0.000	137.292	2.473	0.000	0.000	275.495	3.755
rc201	100	672.92	0.000	0.000	31.163	0.075	0.000	0.000	156.573	0.077	0.000	0.000	313.372	0.077
R1_2_1	200	3353.8	0.057	0.325	25.588	15.934	0.050	0.157	135.676	73.274	0.057	0.108	259.883	137.040
R2_2_1	200	1665.58	0.000	0.000	40.989	2.209	0.000	0.000	208.628	1.265	0.000	0.000	425.191	1.938
C1_2_1	200	3628.51	0.008	0.088	29.241	26.946	0.000	0.043	143.397	121.977	0.000	0.005	278.911	236.639
C2_2_1	200	1726.59	0.000	0.000	34.469	26.393	0.000	0.009	182.240	103.582	0.000	0.000	367.048	193.600
RC1_2_1	200	3303.7	0.117	0.429	29.017	15.429	0.099	0.335	136.881	83.119	0.000	0.209	280.759	171.067
RC2_2_1	200	1560	0.000	0.000	43.277	1.036	0.000	0.000	219.438	0.593	0.000	0.000	439.902	0.642
R1_4_1	400	9519.45	0.143	0.447	30.032	26.930	0.041	0.162	146.209	125.251	0.001	0.157	297.610	239.586
R2_4_1	400	3546.49	0.000	0.176	36.358	20.128	0.000	0.059	183.213	95.827	0.000	0.027	382.383	258.013
C1_4_1	400	11047.19	0.387	0.596	33.742	32.482	0.380	0.566	163.859	156.751	0.099	0.459	321.797	310.134
C2_4_1	400	3539.5	0.067	0.285	37.273	26.112	0.000	0.171	177.445	163.477	0.000	0.136	352.313	311.347
RC1_4_1	400	9447.53	0.325	0.529	31.111	30.242	0.369	0.514	161.269	150.300	0.010	0.300	298.659	283.678
RC2_4_1	400	3403.7	0.000	0.003	41.148	23.680	0.000	0.000	210.787	110.746	0.000	0.000	422.350	203.338
Avg			**0.065**	**0.173**	**33.593**	**14.691**	**0.052**	**0.115**	**170.895**	**67.696**	**0.009**	**0.080**	**340.496**	**137.224**

Table 3.21: Full results for dataset XX

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
X-n101-k25	101	19781.0	0.000	0.134	28.216	5.144	0.000	0.009	159.032	12.428	0.000	0.000	324.199	12.633
X-n106-k14	106	17156.0	0.035	0.052	38.808	17.232	0.000	0.027	202.470	100.145	0.000	0.035	433.734	126.785
X-n110-k13	110	11813.0	0.000	0.000	30.688	2.913	0.000	0.000	165.194	2.764	0.000	0.000	331.890	2.858
X-n115-k10	115	10238.0	0.000	0.000	52.178	0.041	0.000	0.000	262.068	0.041	0.000	0.000	522.815	0.041
X-n120-k6	120	10120.0	0.000	0.000	47.174	0.423	0.000	0.000	238.012	0.367	0.000	0.000	476.316	0.408
X-n125-k30	125	40033.0	0.032	0.076	33.827	20.812	0.000	0.038	173.536	108.724	0.012	0.044	356.124	211.030
X-n129-k18	129	18933.0	0.000	0.081	27.437	15.647	0.000	0.028	142.167	66.141	0.000	0.019	283.025	138.852
X-n134-k13	134	8046.0	0.771	1.269	34.023	16.373	0.000	0.900	165.459	75.223	0.460	1.061	362.866	107.666
X-n139-k10	139	11182.0	0.000	0.845	35.385	9.143	0.000	0.605	186.204	35.023	0.000	0.630	363.487	89.417
X-n143-k7	143	11924.0	0.000	0.579	40.434	2.387	0.017	0.568	216.098	8.913	0.017	0.530	440.382	12.177
X-n148-k46	148	31022.0	0.100	0.479	27.167	21.842	0.000	0.460	156.802	86.795	0.000	0.358	317.981	160.194
X-n153-k22	153	17226.0	0.000	0.024	36.040	29.959	0.000	0.031	180.968	126.362	0.000	0.037	373.986	233.553
X-n157-k13	157	12041.0	0.199	0.448	36.020	21.342	0.125	0.220	188.959	88.088	0.000	0.167	376.635	173.469
X-n162-k11	162	11699.0	0.000	0.207	37.615	6.134	0.000	0.020	161.265	67.108	0.000	0.008	330.996	141.764
X-n167-k10	167	14173.0	0.000	0.027	34.700	8.748	0.000	0.004	188.902	18.324	0.000	0.000	391.160	20.254
X-n172-k51	172	31714.0	0.000	0.944	29.599	21.560	0.255	0.586	161.978	92.227	0.057	0.448	323.299	172.255
X-n176-k26	176	29553.0	0.328	1.764	29.684	27.520	0.000	0.617	136.630	105.569	0.146	0.818	311.982	160.014
X-n181-k23	181	18461.0	0.022	0.043	37.638	22.704	0.022	0.035	195.635	114.062	0.000	0.016	398.350	199.548
X-n186-k15	186	17194.0	0.401	0.777	28.928	19.078	0.244	0.471	163.065	66.678	0.000	0.452	323.788	188.695
X-n190-k8	190	13287.0	0.008	0.080	39.336	31.487	0.000	0.035	201.932	139.005	0.000	0.033	398.245	290.447
X-n195-k51	195	32656.0	0.260	0.663	29.061	22.876	0.000	0.214	148.789	76.953	0.000	0.193	298.409	156.401
X-n200-k36	200	36513.0	0.260	1.660	30.562	26.641	0.227	1.424	147.115	115.882	0.000	1.277	289.430	236.953
X-n204-k19	204	14985.0	0.000	0.208	30.116	21.426	0.000	0.103	154.462	84.706	0.000	0.057	306.780	150.625
X-n209-k16	209	20609.0	0.112	0.309	34.136	11.610	0.000	0.135	172.966	56.592	0.092	0.227	359.116	113.650
X-n214-k11	214	8501.0	0.024	0.481	33.409	16.804	0.000	0.276	179.992	59.551	0.000	0.273	359.042	116.459
X-n219-k73	219	83037.0	0.067	0.231	38.230	30.808	0.000	0.085	190.589	143.377	0.011	0.075	391.347	314.527
X-n223-k34	223	29633.0	0.074	0.789	29.335	21.451	0.000	0.490	157.003	69.580	0.051	0.232	292.896	214.642
X-n228-k23	228	19563.0	0.573	0.858	38.361	34.578	0.000	0.632	190.434	156.872	0.026	0.611	384.702	303.694
X-n233-k16	233	15284.0	0.092	0.207	33.733	20.968	0.000	0.155	178.685	109.772	0.131	0.175	363.126	155.523
X-n237-k14	237	18359.0	0.000	0.263	31.555	23.951	0.000	0.160	157.345	101.630	0.000	0.194	319.440	217.916
X-n242-k48	242	51806.0	0.382	0.797	25.193	19.803	0.087	0.367	127.949	104.665	0.000	0.192	256.902	195.971
X-n247-k50	247	23839.0	0.428	1.205	37.472	32.650	0.000	0.662	185.190	137.471	0.008	0.564	365.878	290.088
X-n251-k28	251	26275.0	0.225	0.987	31.142	25.086	0.015	0.674	152.123	125.658	0.000	0.636	319.660	239.171
X-n256-k16	256	14339.0	0.021	0.145	35.625	21.417	0.000	0.013	176.077	117.158	0.000	0.029	359.677	192.600
X-n261-k13	261	18246.0	0.225	0.789	31.436	22.236	0.148	0.451	156.086	112.956	0.000	0.366	314.730	204.756
X-n266-k58	266	49863.0	0.331	0.783	31.798	25.618	0.146	0.423	166.929	112.218	0.000	0.342	352.214	208.645
X-n270-k35	270	26257.0	0.366	0.713	30.311	26.658	0.000	0.517	158.582	114.461	0.263	0.487	309.083	192.907
X-n275-k28	275	16937.0	0.000	0.172	36.758	33.457	0.071	0.120	190.689	146.386	0.053	0.095	370.151	317.898
X-n280-k17	280	22688.0	0.304	1.223	30.372	25.834	0.013	0.383	146.943	130.811	0.000	0.421	301.596	230.629
X-n284-k15	284	14443.0	0.055	0.251	33.786	25.950	0.014	0.118	175.800	133.898	0.000	0.040	344.778	273.209
X-n289-k60	289	66904.0	0.646	0.879	33.876	30.255	0.112	0.464	168.647	117.730	0.000	0.249	321.061	286.412
X-n294-k50	294	34845.0	0.494	0.858	26.731	23.148	0.000	0.424	129.824	108.057	0.175	0.405	254.911	212.970
X-n298-k31	298	24601.0	0.089	0.429	29.152	26.592	0.000	0.120	152.912	93.358	0.012	0.109	304.213	206.789
X-n303-k21	303	16113.0	1.800	1.952	35.630	32.998	0.025	1.185	176.264	146.336	0.000	1.356	364.285	320.024
X-n308-k13	308	18142.0	0.039	1.665	38.235	31.760	0.000	1.416	201.479	146.296	0.006	1.052	377.005	308.887
X-n313-k71	313	67716.0	0.325	0.581	36.148	33.292	0.028	0.226	177.967	142.761	0.000	0.183	356.894	279.324
X-n317-k53	317	50180.0	0.949	1.059	41.697	37.867	0.052	0.835	206.190	193.622	0.000	0.645	412.185	394.903
X-n322-k28	322	22567.0	0.133	0.372	31.159	28.394	0.000	0.299	157.387	110.455	0.049	0.230	320.151	234.448
X-n327-k20	327	20292.0	0.172	0.667	37.612	28.375	0.123	0.371	182.880	151.220	0.000	0.199	357.400	286.228
X-n331-k15	331	21492.0	0.437	0.737	34.603	29.551	0.000	0.486	171.714	128.524	0.368	0.476	347.510	202.825
X-n336-k84	336	95486.0	0.651	0.815	30.362	26.869	0.175	0.336	149.814	133.930	0.000	0.223	294.770	249.285
X-n344-k43	344	31239.0	0.288	0.743	32.098	28.944	0.048	0.251	159.062	123.724	0.000	0.300	314.082	259.012
X-n351-k40	351	18536.0	0.059	0.295	31.367	29.527	0.000	0.111	160.948	141.187	0.000	0.072	319.200	258.192
X-n359-k29	359	31898.0	1.009	1.753	30.423	26.678	0.326	0.959	153.818	128.632	0.000	0.575	303.800	254.670
X-n367-k17	367	18246.0	0.088	0.148	35.595	33.462	0.000	0.080	181.603	162.519	0.044	0.092	360.151	314.166
X-n376-k94	376	101172.0	0.060	0.265	37.003	35.507	0.155	0.239	193.787	162.571	0.000	0.125	360.205	336.822
X-n384-k52	384	42851.0	0.576	0.985	29.787	26.728	0.000	0.313	153.674	136.534	0.037	0.380	315.398	248.664
X-n393-k38	393	29201.0	0.151	0.416	32.151	29.961	0.096	0.241	161.878	140.327	0.000	0.162	328.697	284.925
X-n401-k29	401	54341.0	0.099	0.198	42.021	38.589	0.020	0.137	214.758	186.680	0.000	0.117	426.584	336.168
X-n411-k19	411	15043.0	0.166	0.672	37.370	33.904	0.027	0.166	182.459	159.078	0.000	0.140	367.066	340.271
X-n420-k130	420	72085.0	0.770	1.034	29.882	28.336	0.000	0.349	143.351	136.242	0.141	0.292	290.800	265.351
X-n429-k61	429	46224.0	0.469	0.841	29.790	27.306	0.225	0.444	151.951	116.524	0.000	0.252	293.492	263.596
X-n439-k37	439	27125.0	0.358	0.632	36.596	31.222	0.000	0.167	179.587	158.161	0.081	0.214	373.583	331.491
X-n449-k29	449	35679.0	0.135	0.816	32.022	29.560	0.064	0.399	161.028	141.748	0.000	0.391	332.829	282.497
X-n459-k26	459	20957.0	0.072	0.318	38.784	33.093	0.038	0.130	191.950	166.526	0.000	0.110	392.052	323.010
X-n469-k138	469	135415.0	0.414	0.974	30.426	28.930	0.051	0.455	148.704	125.690	0.000	0.145	267.973	231.851
X-n480-k70	480	60589.0	0.160	0.511	35.817	33.094	0.160	0.291	176.572	161.348	0.000	0.107	357.880	331.384
X-n491-k59	491	48961.0	0.159	0.850	33.874	32.311	0.016	0.286	168.267	162.578	0.000	0.065	325.171	311.865
X-n502-k39	502	51691.0	0.025	0.081	45.373	43.829	0.008	0.034	228.304	216.819	0.000	0.019	456.739	431.494
X-n513-k21	513	19879.0	0.282	0.824	39.482	33.714	0.000	0.407	198.599	160.333	0.000	0.384	392.103	342.272
X-n524-k153	524	95946.0	0.893	1.287	36.154	34.614	0.197	0.532	176.289	168.823	0.000	0.327	350.801	342.825
X-n536-k96	536	64843.0	0.689	0.924	37.563	36.193	0.133	0.524	177.229	174.260	0.000	0.477	355.587	339.947
X-n548-k50	548	55523.0	0.362	0.509	38.858	37.379	0.086	0.340	197.830	182.030	0.000	0.230	388.788	361.511
X-n561-k42	561	32607.0	0.448	1.155	36.558	33.959	0.000	0.544	180.785	163.408	0.058	0.337	356.570	308.359
X-n573-k30	573	33826.0	0.381	0.707	46.517	45.462	0.000	0.128	221.029	205.152	0.068	0.145	441.611	388.989
X-n586-k159	586	126539.0	0.772	0.936	33.982	32.753	0.000	0.315	167.376	163.790	0.013	0.197	335.781	322.323
X-n599-k92	599	72328.0	0.611	1.029	31.384	30.070	0.281	0.449	155.261	147.833	0.000	0.262	309.197	298.498
X-n613-k62	613	43443.0	0.960	1.589	32.672	31.084	0.470	0.583	161.595	150.049	0.000	0.449	324.451	293.108
X-n627-k43	627	43999.0	0.089	1.017	41.553	39.784	0.102	0.377	205.092	197.020	0.000	0.182	407.271	388.573
X-n641-k35	641	41913.0	0.489	0.756	36.820	35.451	0.103	0.218	179.751	167.381	0.000	0.087	360.252	330.364
X-n655-k131	655	78413.0	0.473	0.626	40.892	39.383	0.075	0.242	190.391	180.709	0.000	0.163	376.331	357.125
X-n670-k130	670	97671.0	0.550	1.387	41.429	39.724	0.349	0.655	203.668	197.498	0.000	0.315	399.300	385.661
X-n685-k75	685	50534.0	0.704	1.207	35.746	33.844	0.000	0.380	177.737	167.626	0.093	0.242	356.016	347.680
X-n701-k44	701	54279.0	0.582	0.913	37.938	36.662	0.103	0.375	188.677	175.298	0.000	0.163	378.696	353.271
X-n716-k35	716	35640.0	0.398	0.723	40.980	39.471	0.028	0.230	199.525	195.262	0.000	0.174	399.714	387.941
X-n733-k159	733	98507.0	0.870	1.160	33.230	31.622	0.367	0.520	158.157	154.542	0.000	0.222	313.688	298.824
X-n749-k98	749	50990.0	0.661	0.977	37.642	37.192	0.222	0.473	186.800	174.861	0.000	0.383	369.837	361.715
X-n766-k71	766	71818.0	1.372	2.344	41.389	40.061	0.272	0.705	205.542	194.563	0.000	0.422	406.026	392.698
X-n783-k48	783	46580.0	0.573	0.863	42.166	39.743	0.000	0.373	210.649	194.819	0.131	0.304	417.341	397.327
X-n801-k40	801	47741.0	0.375	0.639	42.345	40.726	0.052	0.238	209.506	201.053	0.000	0.137	419.418	410.529
X-n819-k171	819	111809.0	0.744	1.032	36.543	35.658	0.000	0.323	171.632	168.303	0.004	0.191	343.169	336.967
X-n837-k142	837	122876.0	1.095	1.582	35.133	34.468	0.225	0.561	169.514	167.399	0.000	0.235	330.193	322.632
X-n856-k95	856	60399.0	0.377	0.743	47.089	45.243	0.091	0.331	228.973	221.535	0.000	0.288	454.395	443.504
X-n876-k59	876	66901.0	0.317	0.521	46.394	45.080	0.000	0.170	226.288	217.535	0.027	0.129	450.136	436.291
X-n895-k37	895	37758.0	1.759	2.572	46.553	44.486	0.519	0.908	226.645	212.513	0.000	0.563	444.732	427.116
X-n916-k207	916	217951.0	0.908	1.235	37.091	36.574	0.240	0.429	170.611	168.139	0.000	0.194	335.866	328.784
X-n936-k151	936	93219.0	1.012	1.254	46.284	45.092	0.031	0.549	225.982	219.523	0.000	0.290	440.922	428.150
X-n957-k87	957	59054.0	0.520	0.804	49.751	49.035	0.103	0.276	246.242	238.920	0.000	0.196	489.032	479.280
X-n979-k58	979	97862.0	0.426	0.780	53.391	52.449	0.068	0.409	259.646	250.985	0.000	0.229	520.609	502.497
X-n1001-k43	1001	49219.0	2.105	2.726	45.450	43.095	0.100	0.694	227.155	221.222	0.000	0.140	455.039	429.477
Avg			**0.383**	**0.769**	**36.019**	**28.905**	**0.074**	**0.365**	**180.511**	**135.261**	**0.026**	**0.279**	**361.192**	**267.001**

Table 3.22: Full results for dataset XY

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
X-n101-k25	101	19781.0	0.000	0.189	28.198	6.665	0.000	0.011	145.051	21.335	0.000	0.026	303.865	27.671
X-n106-k14	106	17162.0	0.012	0.012	40.147	21.102	0.000	0.010	208.392	87.192	0.000	0.010	420.887	160.514
X-n110-k13	110	11813.0	0.000	0.059	29.168	4.656	0.000	0.000	154.141	8.457	0.000	0.000	311.433	11.218
X-n115-k10	115	10238.0	0.000	0.000	52.176	0.045	0.000	0.000	260.412	0.045	0.000	0.000	522.836	0.045
X-n120-k6	120	10120.0	0.000	0.000	44.874	1.268	0.000	0.000	230.341	1.379	0.000	0.000	461.027	1.989
X-n125-k30	125	40033.0	0.035	0.078	32.482	22.277	0.027	0.064	178.436	89.863	0.000	0.029	357.366	167.513
X-n129-k18	129	18933.0	0.011	0.170	27.852	15.134	0.000	0.008	133.670	75.170	0.000	0.013	274.294	147.551
X-n134-k13	134	8083.0	0.297	0.777	32.877	19.454	0.000	0.417	173.704	44.783	0.000	0.468	356.644	85.530
X-n139-k10	139	11182.0	0.617	0.945	35.491	9.655	0.000	0.662	170.674	45.260	0.000	0.569	348.598	77.092
X-n143-k7	143	11924.0	0.000	0.517	39.994	2.315	0.000	0.372	215.454	3.836	0.000	0.396	428.244	14.539
X-n148-k46	148	30970.0	0.365	0.669	26.786	20.718	0.000	0.402	149.258	92.148	0.136	0.320	273.337	202.332
X-n153-k22	153	17226.0	0.000	0.059	36.831	31.208	0.000	0.062	187.127	133.109	0.000	0.030	366.783	265.323
X-n157-k13	157	12041.0	0.108	0.419	35.267	21.049	0.000	0.161	168.057	118.906	0.000	0.133	343.767	184.730
X-n162-k11	162	11699.0	0.000	0.414	37.372	9.796	0.000	0.069	172.972	58.667	0.000	0.021	351.998	135.439
X-n167-k10	167	14173.0	0.000	0.022	35.910	6.310	0.000	0.000	197.780	9.109	0.000	0.000	391.102	19.068
X-n172-k51	172	31722.0	0.404	0.870	30.684	22.581	0.236	0.590	156.427	82.614	0.000	0.385	322.422	160.397
X-n176-k26	176	29545.0	0.633	1.733	28.903	23.371	0.190	0.796	136.876	89.754	0.000	0.158	276.413	197.777
X-n181-k23	181	18450.0	0.081	0.107	38.746	20.420	0.000	0.072	193.723	110.287	0.070	0.081	401.405	195.839
X-n186-k15	186	17194.0	0.308	0.881	29.427	15.765	0.000	0.484	166.592	64.833	0.000	0.265	320.589	140.307
X-n190-k8	190	13287.0	0.000	0.096	41.559	27.471	0.000	0.022	196.637	140.212	0.000	0.018	392.242	280.939
X-n195-k51	195	32592.0	0.236	0.749	30.667	21.353	0.218	0.439	159.082	68.879	0.000	0.318	307.209	136.386
X-n200-k36	200	36511.0	0.687	1.985	30.086	25.211	0.208	1.440	145.453	114.768	0.000	0.826	270.709	213.217
X-n204-k19	204	14985.0	0.060	0.239	29.197	19.026	0.060	0.161	163.902	52.629	0.000	0.094	305.771	145.050
X-n209-k16	209	20574.0	0.224	0.570	35.232	12.761	0.170	0.330	184.949	44.074	0.000	0.206	358.966	83.158
X-n214-k11	214	8501.0	0.000	0.994	33.689	17.387	0.047	0.578	176.498	71.017	0.000	0.363	340.923	139.907
X-n219-k73	219	83029.0	0.107	0.215	40.048	32.339	0.004	0.127	182.984	154.203	0.000	0.059	430.275	323.824
X-n223-k34	223	29691.0	0.162	0.744	30.450	23.903	0.034	0.351	150.720	110.134	0.000	0.408	319.869	231.319
X-n228-k23	228	19586.0	0.434	0.691	37.149	30.296	0.092	0.594	189.466	134.672	0.000	0.598	378.272	273.507
X-n233-k16	233	15284.0	0.190	0.259	34.572	21.554	0.098	0.207	175.531	104.564	0.000	0.186	350.524	168.873
X-n237-k14	237	18359.0	0.000	0.165	29.549	20.808	0.000	0.141	153.540	95.888	0.000	0.136	320.108	184.047
X-n242-k48	242	51832.0	0.174	0.613	26.581	22.951	0.000	0.287	140.750	99.614	0.042	0.133	266.162	189.961
X-n247-k50	247	23841.0	0.331	1.201	37.454	31.047	0.189	0.762	195.079	146.482	0.000	0.391	374.115	312.413
X-n251-k28	251	26267.0	0.129	0.709	29.453	25.130	0.000	0.777	155.400	132.368	0.046	0.421	314.772	229.152
X-n256-k16	256	14339.0	0.021	0.121	35.992	23.264	0.000	0.028	179.937	92.515	0.000	0.007	349.365	234.475
X-n261-k13	261	18248.0	0.438	1.139	30.836	20.691	0.077	0.823	170.340	102.718	0.000	0.404	316.455	207.450
X-n266-k58	266	50000.0	0.322	0.652	31.803	25.847	0.190	0.295	158.608	128.500	0.000	0.150	343.951	192.799
X-n270-k35	270	26287.0	0.365	0.720	30.705	24.890	0.285	0.718	155.927	99.218	0.000	0.418	293.254	205.475
X-n275-k28	275	16949.0	0.024	0.167	38.257	30.384	0.000	0.091	192.642	146.746	0.012	0.081	393.140	238.259
X-n280-k17	280	22688.0	0.145	1.409	30.762	25.575	0.071	0.549	147.805	120.607	0.000	0.258	293.537	233.247
X-n284-k15	284	14444.0	0.028	0.102	35.150	30.071	0.000	0.053	177.921	139.939	0.000	0.046	362.597	261.371
X-n289-k60	289	67020.0	0.521	0.735	32.925	29.660	0.201	0.311	170.640	136.618	0.000	0.181	326.516	267.301
X-n294-k50	294	34745.0	0.460	1.094	26.983	22.581	0.469	0.726	132.096	98.923	0.000	0.563	253.473	181.204
X-n298-k31	298	24562.0	0.326	0.880	28.264	22.000	0.008	0.313	144.709	100.456	0.000	0.207	292.898	201.143
X-n303-k21	303	16117.0	1.930	2.131	37.035	30.777	0.000	1.543	184.669	128.343	0.099	1.395	383.551	194.663
X-n308-k13	308	18143.0	0.424	1.665	39.665	32.441	0.000	0.622	192.316	146.452	0.033	0.689	407.631	191.736
X-n313-k71	313	67781.0	0.179	0.446	33.708	30.926	0.041	0.200	164.457	147.786	0.000	0.139	335.053	290.255
X-n317-k53	317	50243.0	0.683	0.897	43.756	40.845	0.669	0.832	222.616	203.707	0.000	0.711	425.603	394.474
X-n322-k28	322	22579.0	0.009	0.584	32.198	26.537	0.000	0.230	161.121	114.422	0.022	0.202	321.931	209.837
X-n327-k20	327	20319.0	0.128	0.451	35.677	27.563	0.034	0.274	185.610	141.911	0.000	0.260	365.416	270.879
X-n331-k15	331	21525.0	0.153	0.372	33.978	27.780	0.000	0.214	174.017	110.894	0.181	0.294	349.373	218.026
X-n336-k84	336	95361.0	0.618	0.894	29.954	28.061	0.077	0.422	145.111	131.587	0.000	0.264	292.372	262.290
X-n344-k43	344	31209.0	0.157	0.692	30.391	28.486	0.000	0.443	152.581	126.556	0.093	0.368	307.291	245.756
X-n351-k40	351	18518.0	0.130	0.300	31.129	26.394	0.189	0.225	153.957	139.711	0.000	0.161	309.572	272.273
X-n359-k29	359	31924.0	0.861	1.174	30.888	28.510	0.000	0.586	154.390	138.513	0.025	0.377	318.944	254.444
X-n367-k17	367	18244.0	0.093	0.176	36.039	31.850	0.000	0.123	182.587	152.224	0.000	0.095	363.268	312.511
X-n376-k94	376	101104.0	0.195	0.289	34.211	30.948	0.000	0.163	176.833	162.941	0.026	0.142	343.171	310.078
X-n384-k52	384	42821.0	0.404	0.968	29.686	26.042	0.000	0.504	152.642	131.243	0.208	0.314	306.144	240.064
X-n393-k38	393	29209.0	0.113	0.458	32.230	28.741	0.041	0.228	163.012	146.010	0.000	0.139	330.794	276.859
X-n401-k29	401	54339.0	0.094	0.259	41.471	34.725	0.000	0.122	208.145	188.392	0.028	0.108	412.326	364.742
X-n411-k19	411	15043.0	0.226	0.658	37.796	34.987	0.000	0.163	181.976	165.909	0.000	0.117	365.668	339.283
X-n420-k130	420	72128.0	0.284	0.687	29.119	27.030	0.236	0.413	143.954	126.131	0.000	0.279	289.823	270.673
X-n429-k61	429	46248.0	0.659	0.908	29.902	27.677	0.000	0.347	145.106	122.926	0.076	0.332	298.660	239.705
X-n439-k37	439	27049.0	0.580	0.911	37.373	30.333	0.222	0.542	186.579	173.728	0.000	0.434	369.382	316.002
X-n449-k29	449	35737.0	0.143	0.559	31.553	27.960	0.000	0.335	156.741	148.222	0.031	0.191	318.375	278.731
X-n459-k26	459	20964.0	0.091	0.279	39.121	33.407	0.005	0.117	189.499	164.562	0.000	0.077	385.905	335.240
X-n469-k138	469	135368.0	0.927	1.197	30.207	28.938	0.166	0.345	138.335	128.768	0.000	0.286	273.984	247.428
X-n480-k70	480	60581.0	0.413	0.565	34.557	32.785	0.050	0.244	174.289	158.657	0.000	0.109	343.940	325.825
X-n491-k59	491	48961.0	0.198	0.964	34.246	30.799	0.000	0.203	168.769	160.894	0.049	0.103	334.536	301.896
X-n502-k39	502	51689.0	0.019	0.082	45.869	43.872	0.000	0.038	230.113	221.331	0.014	0.035	458.879	435.131
X-n513-k21	513	19889.0	0.176	0.672	39.903	35.287	0.272	0.405	203.916	158.547	0.000	0.426	409.178	350.187
X-n524-k153	524	95980.0	0.996	1.352	36.128	34.443	0.123	0.609	177.430	168.915	0.000	0.292	344.524	327.074
X-n536-k96	536	64904.0	0.549	0.864	35.101	33.375	0.388	0.604	176.338	162.538	0.000	0.420	344.914	322.071
X-n548-k50	548	55522.0	0.200	0.471	38.880	36.037	0.067	0.247	198.768	190.322	0.000	0.283	385.391	364.904
X-n561-k42	561	32583.0	0.565	0.941	35.934	33.451	0.000	0.439	180.501	169.444	0.068	0.401	356.358	323.682
X-n573-k30	573	33846.0	0.086	0.603	45.591	40.960	0.000	0.185	232.462	215.015	0.009	0.098	463.384	439.086
X-n586-k159	586	126581.0	0.485	0.938	34.492	33.712	0.094	0.386	163.863	160.928	0.000	0.200	326.529	313.914
X-n599-k92	599	72296.0	0.830	1.065	30.874	28.980	0.252	0.482	150.547	141.047	0.000	0.280	297.217	261.618
X-n613-k62	613	43463.0	0.810	1.463	32.882	30.107	0.439	0.767	163.167	154.771	0.000	0.425	323.452	307.402
X-n627-k43	627	43946.0	0.412	1.370	40.730	38.331	0.100	0.671	201.478	196.904	0.000	0.191	399.520	378.690
X-n641-k35	641	41906.0	0.530	0.934	37.339	34.957	0.019	0.296	186.419	172.844	0.000	0.184	371.671	352.296
X-n655-k131	655	78386.0	0.500	0.693	40.038	37.093	0.073	0.278	183.004	171.865	0.000	0.136	360.145	339.691
X-n670-k130	670	97628.0	0.713	1.423	40.977	38.540	0.186	0.449	199.369	191.539	0.000	0.212	388.362	373.890
X-n685-k75	685	50523.0	0.798	1.023	35.419	34.536	0.194	0.473	177.526	165.165	0.000	0.285	348.850	330.194
X-n701-k44	701	54293.0	0.534	1.001	36.428	34.248	0.004	0.241	183.231	171.819	0.000	0.157	370.379	338.288
X-n716-k35	716	35634.0	0.323	0.688	40.155	38.127	0.073	0.306	196.146	189.404	0.000	0.193	390.962	383.000
X-n733-k159	733	98482.0	1.005	1.273	32.162	31.648	0.000	0.466	156.005	148.805	0.062	0.300	310.377	296.843
X-n749-k98	749	50983.0	0.634	1.050	35.726	34.688	0.161	0.496	173.440	166.757	0.000	0.290	345.858	336.774
X-n766-k71	766	71805.0	1.228	1.913	42.461	41.590	0.386	0.699	203.084	196.851	0.000	0.365	397.887	383.951
X-n783-k48	783	46551.0	0.535	0.958	41.326	37.617	0.200	0.547	203.288	193.395	0.000	0.342	401.207	370.613
X-n801-k40	801	47729.0	0.394	0.742	42.418	40.212	0.000	0.238	209.650	197.428	0.019	0.179	416.448	407.792
X-n819-k171	819	111646.0	0.862	1.206	35.976	34.926	0.316	0.575	170.101	164.248	0.000	0.267	332.434	316.679
X-n837-k142	837	123100.0	0.696	1.372	36.425	35.929	0.000	0.413	170.241	164.075	0.056	0.209	339.689	331.444
X-n856-k95	856	60370.0	0.030	0.673	46.510	45.096	0.197	0.283	227.383	222.920	0.000	0.162	447.109	434.449
X-n876-k59	876	66926.0	0.267	0.465	48.123	46.978	0.057	0.177	234.228	230.553	0.000	0.090	469.391	451.899
X-n895-k37	895	37836.0	1.364	2.284	46.131	41.862	0.309	0.721	224.526	210.373	0.000	0.358	447.984	418.147
X-n916-k207	916	217877.0	1.071	1.268	37.927	37.596	0.237	0.472	181.822	177.447	0.000	0.227	360.079	352.325
X-n936-k151	936	93117.0	0.758	1.057	45.444	43.901	0.111	0.435	221.406	215.791	0.000	0.282	438.820	430.606
X-n957-k87	957	58998.0	0.386	1.013	49.050	46.979	0.327	0.565	242.858	232.498	0.000	0.362	479.218	468.843
X-n979-k58	979	97882.0	0.345	0.832	53.314	52.568	0.109	0.336	257.612	250.335	0.000	0.237	505.877	473.284
X-n1001-k43	1001	49013.0	2.626	3.230	46.023	43.703	0.382	1.116	228.125	219.127	0.000	0.478	455.963	441.079
Avg			**0.387**	**0.776**	**35.922**	**28.359**	**0.097**	**0.382**	**179.671**	**134.260**	**0.014**	**0.253**	**358.289**	**261.489**

3.9.2 VRPMPD Instances

Table 3.23: Full results for dataset XXLX

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
Leuven1	3001	137432.0	0.932	1.218	65.488	65.073	0.419	0.481	329.570	327.087	0.000	0.250	659.847	654.346
Leuven2	4001	82194.0	4.394	4.971	70.306	69.953	1.644	2.396	359.552	352.060	0.000	0.758	727.248	710.583
Antwerp1	6001	346563.0	2.698	3.224	77.058	76.901	0.775	0.952	388.364	387.647	0.000	0.224	783.741	779.454
Antwerp2	7001	214177.0	2.301	2.748	81.572	81.198	0.455	0.798	424.125	421.947	0.000	0.308	860.962	855.353
Ghent1	10001	365418.0	1.871	1.966	96.911	96.751	0.487	0.633	473.888	472.513	0.000	0.169	962.729	958.717
Ghent2	11001	214337.0	2.680	2.873	87.083	87.027	0.923	1.353	445.665	444.920	0.000	0.672	896.875	891.461
Brussels1	15001	376294.0	2.640	2.845	111.955	111.813	0.517	0.717	544.159	543.906	0.000	0.161	1097.920	1096.056
Brussels2	16001	279692.0	2.853	3.043	112.825	112.665	0.503	0.709	563.807	562.629	0.000	0.180	1124.116	1122.973
Flanders1	20001	5217403.0	1.298	1.447	160.615	160.561	0.331	0.435	769.671	769.114	0.000	0.129	1539.105	1536.640
Flanders2	30001	3201225.0	3.644	3.803	186.779	186.725	0.706	0.889	888.068	887.024	0.000	0.197	1799.487	1798.281
Avg			**2.531**	**2.814**	**105.059**	**104.867**	**0.676**	**0.936**	**518.687**	**516.885**	**0.000**	**0.305**	**1045.203**	**1040.386**

Table 3.24: Full results for dataset XXLY

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
Leuven1	3001	137644.0	0.849	1.066	66.024	65.626	0.210	0.334	324.768	320.977	0.000	0.173	646.863	642.210
Leuven2	4001	82346.0	4.028	4.788	69.928	69.140	1.298	2.359	363.295	357.124	0.000	0.669	719.824	712.136
Antwerp1	6001	347231.0	2.948	3.281	74.749	74.657	0.714	0.898	369.409	368.823	0.000	0.101	737.793	734.318
Antwerp2	7001	214035.0	2.469	2.794	80.162	79.950	0.391	0.764	418.696	416.235	0.000	0.282	854.779	851.140
Ghent1	10001	365724.0	1.720	1.889	93.789	93.645	0.416	0.594	455.572	454.904	0.000	0.113	912.854	910.597
Ghent2	11001	215520.0	2.147	2.313	86.591	86.430	0.438	0.777	446.559	445.425	0.000	0.118	892.264	887.701
Brussels1	15001	375939.0	2.883	3.053	110.735	110.676	0.779	0.874	537.589	537.053	0.000	0.219	1065.935	1064.555
Brussels2	16001	279479.0	2.889	3.145	113.083	112.958	0.486	0.706	563.014	562.087	0.000	0.174	1129.187	1127.220
Flanders1	20001	5221438.0	1.149	1.406	160.430	160.315	0.331	0.416	771.296	770.930	0.000	0.073	1535.166	1534.707
Flanders2	30001	3197711.0	3.473	3.677	186.718	186.677	1.061	1.251	882.756	882.555	0.000	0.338	1794.305	1793.114
Avg			**2.456**	**2.741**	**104.221**	**104.007**	**0.612**	**0.897**	**513.295**	**511.611**	**0.000**	**0.226**	**1028.897**	**1025.770**

Table 3.25: Full results for dataset CMTH

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
CMT1H	50	465.02	0.000	0.000	25.594	0.371	0.000	0.000	128.983	0.215	0.000	0.000	258.028	0.202
CMT6H	50	555.43	0.000	0.000	37.175	0.273	0.000	0.000	186.798	0.335	0.000	0.000	375.356	0.276
CMT2H	75	662.63	0.000	0.000	32.389	0.620	0.000	0.000	165.432	0.377	0.000	0.000	330.803	0.374
CMT7H	75	900.12	0.000	0.049	44.743	9.866	0.000	0.000	210.221	81.726	0.000	0.000	433.205	97.906
CMT3H	100	700.94	0.000	0.000	30.247	0.211	0.000	0.000	151.747	0.257	0.000	0.000	304.416	0.337
CMT12H	100	629.37	0.000	0.000	31.441	1.167	0.000	0.000	162.175	2.671	0.000	0.000	325.936	3.275
CMT8H	100	865.5	0.000	0.000	45.430	0.399	0.000	0.000	229.476	0.332	0.000	0.000	460.592	0.330
CMT14H	100	821.75	0.000	0.000	55.013	0.069	0.000	0.000	275.624	0.070	0.000	0.000	555.154	0.070
CMT11H	120	818.05	0.000	0.000	32.215	4.920	0.000	0.000	173.814	5.330	0.000	0.000	351.889	5.136
CMT13H	120	1542.86	0.000	0.000	43.943	30.506	-0.013	0.000	227.205	101.478	0.000	0.000	454.047	159.533
CMT4H	150	828.12	0.000	0.288	35.538	4.566	0.000	0.314	187.501	14.875	0.000	0.153	359.589	58.586
CMT9H	150	1160.68	0.000	0.000	40.430	12.508	0.000	0.000	219.484	37.002	0.000	0.000	455.038	45.266
CMT5H	199	978.74	0.000	0.173	37.026	4.849	0.000	0.000	194.823	5.999	0.000	0.000	394.131	6.290
CMT10H	199	1372.2	0.000	0.354	34.100	21.941	0.000	0.140	173.534	97.409	0.000	0.012	313.031	221.300
Avg			**0.000**	**0.062**	**37.520**	**6.590**	**0.000**	**0.032**	**191.916**	**24.862**	**0.000**	**0.012**	**383.658**	**42.777**

Table 3.26: Full results for dataset CMTQ

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
CMT1Q	50	489.74	0.001	0.001	27.953	0.075	0.001	0.001	140.368	0.057	0.001	0.001	280.722	0.057
CMT6Q	50	555.43	0.000	0.000	37.572	0.368	0.000	0.000	189.772	0.239	0.000	0.000	379.003	0.282
CMT2Q	75	731.26	0.000	0.071	26.973	2.964	0.000	0.000	141.154	3.310	0.000	0.000	274.018	11.413
CMT7Q	75	900.69	0.000	0.000	32.303	17.960	0.000	0.000	183.527	14.585	0.000	0.000	370.130	20.657
CMT3Q	100	747.15	0.000	0.000	35.816	0.147	0.000	0.000	180.011	0.132	0.000	0.000	358.834	0.132
CMT12Q	100	729.25	0.000	0.004	26.926	3.065	0.000	0.001	146.778	4.790	0.000	0.000	295.095	7.949
CMT8Q	100	865.5	0.000	0.000	45.126	0.372	0.000	0.000	226.560	0.339	0.000	0.000	453.440	0.316
CMT14Q	100	821.75	0.000	0.000	55.104	0.062	0.000	0.000	278.106	0.062	0.000	0.000	552.757	0.062
CMT11Q	120	939.36	0.000	0.000	36.716	9.724	0.000	0.000	205.656	16.166	0.000	0.000	412.531	30.601
CMT13Q	120	1542.86	0.000	0.000	44.424	30.886	-0.013	0.000	227.828	101.720	0.000	0.000	469.176	156.924
CMT4Q	150	913.93	0.147	0.165	36.103	5.385	0.147	0.147	187.235	8.766	0.147	0.147	379.848	9.679
CMT9Q	150	1161.24	0.000	0.000	44.665	5.889	0.000	0.000	236.144	11.717	0.000	0.000	480.258	11.126
CMT5Q	199	1104.87	0.204	0.384	27.096	13.493	0.000	0.162	136.280	81.083	0.000	0.073	279.608	99.609
CMT10Q	199	1374.18	0.000	0.451	34.608	29.308	0.000	0.089	163.225	139.083	0.000	0.084	357.936	231.044
Avg			**0.025**	**0.077**	**36.527**	**8.550**	**0.010**	**0.028**	**188.760**	**27.289**	**0.011**	**0.022**	**381.668**	**41.418**

Table 3.27: Full results for dataset CMTT

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	**Avg**	**Time***	**Time**	**Best**	**Avg**	**Time***	**Time**	**Best**	**Avg**	**Time***	**Time**
CMT1T	50	520.06	0.000	0.000	35.610	0.048	0.000	0.000	180.936	0.048	0.000	0.000	361.065	0.048
CMT6T	50	555.43	0.000	0.000	30.041	0.086	0.000	0.000	150.097	0.055	0.000	0.000	299.428	0.055
CMT2T	75	782.77	0.000	0.028	23.019	1.409	0.000	0.000	123.303	2.550	0.000	0.000	238.690	0.875
CMT7T	75	903.05	0.000	0.000	30.700	0.091	0.000	0.000	153.304	0.088	0.000	0.000	306.864	0.089
CMT3T	100	798.07	0.000	0.000	26.821	3.261	0.000	0.000	142.190	5.839	0.000	0.000	286.631	3.643
CMT12T	100	787.52	0.000	0.000	37.572	0.801	0.000	0.000	192.759	0.767	0.000	0.000	386.977	0.541
CMT8T	100	865.54	0.000	0.000	41.721	0.950	0.000	0.000	209.960	0.778	0.000	0.000	419.796	0.867
CMT14T	100	826.77	0.000	0.000	33.469	0.271	0.000	0.000	169.168	0.256	0.000	0.000	338.447	0.258
CMT11T	120	998.8	0.000	0.000	34.769	15.637	0.000	0.000	199.117	39.988	0.000	0.000	386.647	77.781
CMT13T	120	1541.14	0.112	0.112	44.028	30.568	0.099	0.111	227.734	101.674	0.112	0.112	450.265	164.587
CMT4T	150	990.39	0.000	0.000	34.257	7.293	0.000	0.000	184.385	12.711	0.000	0.000	362.584	30.581
CMT9T	150	1162.55	0.000	0.001	37.527	22.779	0.000	0.000	198.782	70.439	0.000	0.000	412.127	127.745
CMT5T	199	1218.77	0.325	0.377	31.423	16.742	0.156	0.308	157.121	73.652	0.000	0.293	308.542	140.276
CMT10T	199	1381.04	0.549	0.852	39.483	14.265	0.429	0.792	205.962	47.439	0.179	0.650	374.594	152.947
Avg			**0.070**	**0.098**	**34.317**	**8.157**	**0.049**	**0.086**	**178.201**	**25.449**	**0.021**	**0.075**	**352.333**	**50.021**

Table 3.28: Full results for dataset XH

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
X-n101-k25	101	19265.0	0.052	0.450	26.877	13.289	0.000	0.269	144.467	44.386	0.000	0.212	302.006	88.880
X-n106-k14	106	14873.0	0.000	0.001	38.007	21.045	0.000	0.000	203.684	57.930	0.000	0.000	424.702	96.782
X-n110-k13	110	11153.0	0.000	0.000	30.488	4.600	0.000	0.000	160.247	6.332	0.000	0.000	323.156	8.925
X-n115-k10	115	10156.0	0.000	0.000	44.705	0.018	0.000	0.000	224.178	0.018	0.000	0.000	447.366	0.018
X-n120-k6	120	9716.0	0.000	0.000	36.686	2.107	0.000	0.000	192.072	2.584	0.000	0.000	384.645	3.996
X-n125-k30	125	33107.0	0.187	1.001	38.410	22.468	0.000	0.412	185.626	81.839	0.000	0.037	371.510	179.618
X-n129-k18	129	18054.0	0.000	0.080	28.833	14.000	0.000	0.012	142.018	63.366	0.000	0.005	295.358	114.299
X-n134-k13	134	7418.0	0.000	0.303	35.231	16.603	0.000	0.093	163.643	63.385	0.000	0.049	359.618	105.125
X-n139-k10	139	10834.0	0.572	0.572	49.819	0.254	0.342	0.540	247.166	10.771	0.000	0.444	473.967	49.619
X-n143-k7	143	11750.0	0.979	1.216	43.551	6.766	0.085	0.980	200.553	33.840	0.000	0.789	402.507	60.465
X-n148-k46	148	27230.0	0.408	0.605	33.251	20.516	0.176	0.383	159.927	100.048	0.000	0.326	333.241	157.347
X-n153-k22	153	14699.0	0.000	0.189	35.842	25.433	0.000	0.135	207.346	78.028	0.000	0.122	388.903	205.774
X-n157-k13	157	11077.0	0.063	0.097	46.760	27.832	0.000	0.045	224.156	105.784	0.000	0.018	451.567	192.152
X-n162-k11	162	11175.0	0.045	0.396	34.556	7.073	0.000	0.082	169.984	35.279	0.000	0.083	350.748	62.872
X-n167-k10	167	13965.0	0.000	0.191	35.022	13.929	0.000	0.138	186.582	34.733	0.000	0.132	392.576	72.683
X-n172-k51	172	27443.0	0.011	0.442	32.466	25.727	0.000	0.285	182.400	88.319	0.051	0.257	370.331	143.925
X-n176-k26	176	28961.0	0.000	0.038	34.683	26.185	0.000	0.000	181.859	121.409	0.000	0.000	368.444	197.694
X-n181-k23	181	15584.0	0.000	0.174	39.916	22.323	0.051	0.119	197.059	92.408	0.000	0.138	422.026	191.696
X-n186-k15	186	16029.0	0.000	0.769	31.836	12.460	0.000	0.293	155.351	49.441	0.000	0.142	306.625	93.193
X-n190-k8	190	10400.0	0.000	0.086	37.992	17.279	0.000	0.030	205.879	72.536	0.000	0.039	416.484	131.163
X-n195-k51	195	28575.0	0.049	0.603	33.508	20.659	0.000	0.294	162.764	102.239	0.000	0.106	330.896	177.586
X-n200-k36	200	33129.0	0.085	0.214	32.014	28.563	0.039	0.187	161.818	127.632	0.000	0.139	334.033	236.205
X-n204-k19	204	14271.0	0.000	0.015	34.299	21.280	0.000	0.000	182.418	65.814	0.000	0.000	366.948	125.657
X-n209-k16	209	19416.0	0.170	0.685	31.698	18.107	0.000	0.126	177.323	62.906	0.000	0.086	358.848	98.663
X-n214-k11	214	8058.0	0.621	0.836	40.533	12.493	0.372	0.608	198.362	56.136	0.000	0.539	371.816	142.134
X-n219-k73	219	63552.0	0.065	0.167	52.248	28.644	0.000	0.059	263.656	132.978	0.000	0.045	569.213	198.096
X-n223-k34	223	25194.0	0.000	0.701	28.409	23.041	0.000	0.261	144.134	105.625	0.000	0.147	290.228	197.668
X-n228-k23	228	17212.0	0.035	0.077	32.392	27.573	0.006	0.031	162.694	134.906	0.000	0.012	322.619	279.242
X-n233-k16	233	14374.0	0.000	0.202	36.309	18.746	0.000	0.053	177.192	89.983	0.000	0.015	359.701	167.401
X-n237-k14	237	17854.0	0.000	0.376	31.799	19.913	0.000	0.042	172.666	66.166	0.000	0.007	342.185	130.778
X-n242-k48	242	47216.0	0.555	1.180	29.382	21.477	0.000	0.431	150.604	96.929	0.085	0.295	293.307	173.045
X-n247-k50	247	22842.0	0.018	0.903	42.068	31.073	0.088	0.524	215.750	161.398	0.000	0.291	461.784	281.606
X-n251-k28	251	23880.0	0.000	0.435	28.835	24.737	0.004	0.192	150.473	113.323	0.034	0.150	327.799	201.427
X-n256-k16	256	13587.0	0.000	0.035	42.423	17.913	0.000	0.000	219.574	76.843	0.000	0.000	435.485	131.482
X-n261-k13	261	17989.0	0.334	1.162	35.892	20.129	0.334	0.665	174.889	95.365	0.000	0.387	353.477	165.180
X-n266-k58	266	43351.0	0.302	0.613	31.903	26.305	0.016	0.315	163.074	103.514	0.000	0.211	324.665	231.973
X-n270-k35	270	22557.0	0.058	0.182	30.732	22.943	0.004	0.047	158.787	123.240	0.000	0.134	324.440	228.079
X-n275-k28	275	13957.0	0.000	0.132	32.495	27.490	0.000	0.049	175.952	136.674	0.000	0.037	335.608	251.569
X-n280-k17	280	21920.0	0.456	1.106	37.764	27.505	0.611	0.775	187.053	135.807	0.000	0.594	376.792	298.321
X-n284-k15	284	13914.0	0.122	0.474	34.991	21.179	0.007	0.216	178.567	84.929	0.000	0.155	361.437	180.127
X-n289-k60	289	53645.0	0.414	0.692	33.468	29.283	0.000	0.255	175.315	140.174	0.071	0.214	339.661	244.888
X-n294-k50	294	29439.0	0.187	0.837	28.722	21.623	0.112	0.359	150.764	96.130	0.000	0.217	296.789	222.627
X-n298-k31	298	22601.0	0.022	0.522	31.675	22.711	0.018	0.079	170.107	137.090	0.000	0.080	352.717	205.343
X-n303-k21	303	15311.0	0.144	0.624	35.608	21.446	0.000	0.570	182.489	86.608	0.039	0.466	359.382	189.857
X-n308-k13	308	17909.0	0.670	2.354	39.258	27.695	0.000	0.807	187.050	128.519	0.084	0.584	388.328	246.495
X-n313-k71	313	52922.0	0.425	0.877	30.932	25.955	0.440	0.507	155.791	123.154	0.000	0.514	310.548	284.125
X-n317-k53	317	42704.0	0.028	0.164	46.278	42.032	0.000	0.074	231.325	220.442	0.002	0.056	454.320	434.685
X-n322-k28	322	20292.0	0.000	0.411	31.322	25.545	0.000	0.182	163.424	118.562	0.000	0.074	324.453	196.700
X-n327-k20	327	19004.0	0.495	0.926	33.702	23.965	0.184	0.573	169.805	136.865	0.000	0.476	354.160	201.596
X-n331-k15	331	20404.0	0.206	0.395	32.950	27.159	0.034	0.124	161.853	133.836	0.000	0.113	325.945	257.273
X-n336-k84	336	78262.0	0.222	0.685	32.627	29.375	0.138	0.343	160.689	151.112	0.000	0.139	319.239	288.741
X-n344-k43	344	26616.0	0.346	0.620	28.620	23.400	0.150	0.368	144.121	122.251	0.000	0.280	285.596	235.785
X-n351-k40	351	17065.0	0.334	0.766	31.069	25.155	0.164	0.382	154.967	113.030	0.000	0.261	304.664	251.956
X-n359-k29	359	30994.0	0.519	0.828	31.261	28.212	0.065	0.427	153.407	130.043	0.000	0.272	314.338	240.770
X-n367-k17	367	15534.0	1.017	1.577	37.698	26.272	0.315	0.957	184.180	133.172	0.000	0.838	371.367	256.203
X-n376-k94	376	79505.0	0.033	0.174	41.935	37.992	0.000	0.138	216.401	187.199	0.000	0.098	421.511	357.465
X-n384-k52	384	39670.0	0.600	0.791	29.397	25.470	0.015	0.319	148.061	125.687	0.000	0.175	290.576	232.101
X-n393-k38	393	24675.0	0.401	0.718	32.033	27.276	0.215	0.436	161.351	135.782	0.000	0.251	318.975	248.559
X-n401-k29	401	37559.0	0.266	0.453	37.943	33.916	0.136	0.289	189.875	152.098	0.000	0.205	382.386	304.193
X-n411-k19	411	13762.0	0.094	0.818	36.677	30.927	0.051	0.350	181.539	157.876	0.000	0.294	362.810	270.545
X-n420-k130	420	62929.0	0.421	0.764	30.617	29.312	0.167	0.455	150.807	135.809	0.000	0.359	304.022	259.598
X-n429-k61	429	40564.0	0.311	0.690	29.223	25.846	0.175	0.410	143.676	123.965	0.000	0.243	296.465	252.110
X-n439-k37	439	23932.0	0.134	0.830	35.253	31.626	0.004	0.152	168.745	155.788	0.000	0.167	340.636	312.158
X-n449-k29	449	34134.0	0.498	0.821	33.507	30.800	0.278	0.489	168.964	144.580	0.000	0.286	348.298	312.807
X-n459-k26	459	16389.0	0.085	0.564	36.229	32.516	0.092	0.339	184.294	148.325	0.000	0.096	355.593	313.124
X-n469-k138	469	121262.0	0.428	0.832	34.033	32.255	0.075	0.265	167.072	145.923	0.000	0.153	330.014	305.983
X-n480-k70	480	50929.0	0.295	0.545	33.275	30.176	0.175	0.370	164.771	152.972	0.000	0.245	335.428	311.918
X-n491-k59	491	39991.0	0.883	1.128	32.175	28.597	0.153	0.475	161.166	145.244	0.000	0.389	324.781	288.520
X-n502-k39	502	38142.0	0.052	0.227	46.465	43.856	0.000	0.157	238.532	222.511	0.037	0.155	465.634	445.257
X-n513-k21	513	18863.0	0.382	1.119	41.177	31.313	0.000	0.840	215.034	129.073	0.053	0.638	405.579	238.902
X-n524-k153	524	87859.0	0.730	1.315	45.667	43.211	0.382	0.556	225.766	217.428	0.000	0.466	451.687	427.960
X-n536-k96	536	53430.0	0.700	1.084	38.426	36.748	0.095	0.423	185.558	177.558	0.000	0.230	363.669	345.321
X-n548-k50	548	49629.0	0.095	0.469	34.959	32.918	0.095	0.210	174.853	156.365	0.000	0.210	345.754	325.083
X-n561-k42	561	28190.0	0.518	1.312	35.541	31.015	0.035	0.387	175.908	152.258	0.000	0.182	348.681	303.853
X-n573-k30	573	28756.0	0.372	0.620	43.309	40.454	0.104	0.345	215.113	192.946	0.000	0.189	423.622	381.834
X-n586-k159	586	104695.0	0.448	0.667	36.264	34.420	0.030	0.174	178.114	165.926	0.000	0.149	367.038	324.666
X-n599-k92	599	62516.0	0.539	0.762	32.177	29.592	0.043	0.309	160.345	142.624	0.000	0.239	318.628	283.465
X-n613-k62	613	37394.0	0.685	0.902	31.478	29.344	0.029	0.236	158.068	143.637	0.000	0.312	320.645	281.217
X-n627-k43	627	35778.0	0.187	0.555	38.999	36.758	0.061	0.191	193.291	181.824	0.000	0.070	384.244	371.383
X-n641-k35	641	38426.0	0.856	1.131	37.321	35.484	0.115	0.324	189.987	173.972	0.000	0.269	376.832	347.684
X-n655-k131	655	58522.0	0.154	0.246	47.202	46.250	0.031	0.127	232.031	221.863	0.000	0.083	462.042	446.161
X-n670-k130	670	86862.0	0.813	1.188	49.523	48.425	0.000	0.629	244.310	226.710	0.158	0.387	483.134	468.533
X-n685-k75	685	42293.0	0.674	0.993	35.857	34.464	0.440	0.598	176.361	157.693	0.000	0.342	350.437	329.949
X-n701-k44	701	47773.0	0.687	1.203	39.284	37.619	0.174	0.423	196.723	183.331	0.000	0.233	398.670	339.511
X-n716-k35	716	26624.0	1.232	1.537	39.807	38.290	0.740	1.051	197.578	180.701	0.000	0.829	400.144	376.828
X-n733-k159	733	78945.0	0.252	0.680	35.845	34.464	0.009	0.209	175.929	169.333	0.000	0.145	347.710	336.795
X-n749-k98	749	45591.0	0.656	0.941	36.382	34.323	0.178	0.372	178.379	169.262	0.000	0.155	354.531	340.702
X-n766-k71	766	67064.0	0.705	1.173	43.799	42.554	0.000	0.559	218.649	205.074	0.084	0.502	432.094	420.499
X-n783-k48	783	44079.0	0.873	1.242	40.566	39.243	0.261	0.471	204.979	186.456	0.000	0.306	401.918	370.197
X-n801-k40	801	43900.0	0.326	0.591	38.535	38.027	0.027	0.243	191.284	182.453	0.000	0.194	383.057	367.121
X-n819-k171	819	88466.0	0.665	0.875	38.651	36.946	0.123	0.408	186.158	180.993	0.000	0.305	368.376	342.659
X-n837-k142	837	106845.0	0.680	0.947	38.380	37.887	0.082	0.373	186.506	183.259	0.000	0.157	369.648	353.826
X-n856-k95	856	52058.0	0.290	0.520	44.342	42.665	0.079	0.260	218.471	209.153	0.000	0.209	433.320	421.140
X-n876-k59	876	55683.0	0.542	0.902	44.853	43.758	0.092	0.330	221.500	218.296	0.000	0.148	437.196	420.846
X-n895-k37	895	34905.0	0.771	1.260	42.079	39.837	0.498	0.694	207.947	190.448	0.000	0.639	418.303	379.532
X-n916-k207	916	176175.0	0.607	0.936	40.853	39.889	0.244	0.330	191.314	188.088	0.000	0.167	376.200	364.294
X-n936-k151	936	80655.0	0.484	0.801	51.721	48.788	0.052	0.307	252.505	242.870	0.000	0.180	500.763	484.539
X-n957-k87	957	50863.0	0.250	0.667	45.954	43.353	0.116	0.236	226.447	220.094	0.000	0.229	455.615	438.534
X-n979-k58	979	67077.0	0.304	0.518	51.280	50.481	0.148	0.284	258.124	252.542	0.000	0.106	508.833	493.716
X-n1001-k43	1001	44761.0	0.395	0.986	43.537	42.754	0.194	0.434	224.553	215.883	0.000	0.373	446.121	427.417
Avg			**0.316**	**0.665**	**36.923**	**28.013**	**0.098**	**0.317**	**185.643**	**131.377**	**0.007**	**0.225**	**372.402**	**255.780**

Table 3.29: Full results for dataset XQ

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
X-n101-k25	101	21901.0	0.078	0.132	34.924	7.966	0.078	0.084	174.497	41.869	0.000	0.070	367.857	60.218
X-n106-k14	106	20203.0	0.000	0.012	31.496	23.965	0.000	0.005	171.991	102.771	0.000	0.001	338.208	235.771
X-n110-k13	110	12855.0	0.210	0.266	31.728	4.996	0.000	0.169	157.923	21.453	0.000	0.090	303.471	52.313
X-n115-k10	115	11208.0	0.000	0.289	36.447	1.390	0.000	0.000	188.487	2.073	0.000	0.000	378.539	2.428
X-n120-k6	120	11540.0	0.000	0.000	43.047	0.961	0.000	0.000	220.194	1.431	0.000	0.000	441.201	1.964
X-n125-k30	125	43331.0	0.713	0.770	35.730	26.224	0.000	0.355	187.027	121.614	0.000	0.252	370.680	297.258
X-n129-k18	129	21898.0	0.023	0.390	29.425	13.455	0.009	0.169	146.704	67.861	0.000	0.178	328.808	60.423
X-n134-k13	134	8977.0	0.000	0.059	39.648	18.532	0.000	0.016	223.921	47.006	0.000	0.025	447.801	78.338
X-n139-k10	139	12106.0	0.000	0.467	41.277	6.635	0.000	0.000	221.595	15.820	0.000	0.000	457.014	16.633
X-n143-k7	143	13424.0	0.000	0.264	33.655	16.173	0.000	0.070	164.389	67.477	0.000	0.039	327.346	106.037
X-n148-k46	148	34428.0	0.000	0.264	45.934	17.176	0.000	0.238	210.947	106.565	0.000	0.139	470.222	124.627
X-n153-k22	153	17004.0	0.000	0.019	43.348	29.084	0.000	0.000	217.659	161.526	0.000	0.000	439.835	297.206
X-n157-k13	157	13543.0	0.000	0.001	48.248	43.853	0.000	0.000	240.856	172.128	0.000	0.000	518.634	260.982
X-n162-k11	162	12186.0	0.000	0.000	39.233	5.202	0.000	0.000	211.968	9.025	0.000	0.000	436.814	4.785
X-n167-k10	167	17258.0	0.000	0.070	38.673	14.399	0.000	0.009	191.647	56.374	0.000	0.012	408.520	124.105
X-n172-k51	172	36146.0	0.014	0.058	39.244	24.687	0.014	0.065	210.838	134.380	0.000	0.034	412.348	313.624
X-n176-k26	176	35442.0	0.003	0.157	34.684	27.924	0.000	0.005	178.933	120.607	0.000	0.002	366.850	242.183
X-n181-k23	181	20123.0	0.000	0.039	43.768	19.491	0.000	0.000	219.102	110.558	0.000	0.000	446.944	222.080
X-n186-k15	186	19828.0	0.000	0.106	32.566	14.670	0.000	0.028	154.445	79.817	0.000	0.017	316.547	140.319
X-n190-k8	190	13620.0	0.029	0.096	32.845	23.055	0.000	0.081	170.826	120.886	0.029	0.046	339.351	222.806
X-n195-k51	195	34301.0	0.000	0.151	37.783	22.269	0.000	0.085	202.701	92.478	0.000	0.099	423.629	235.087
X-n200-k36	200	44628.0	0.179	0.242	38.613	30.692	0.000	0.188	198.612	160.079	0.150	0.187	379.705	287.784
X-n204-k19	204	16328.0	0.000	0.107	34.560	20.833	0.000	0.058	169.374	64.189	0.000	0.069	344.975	104.005
X-n209-k16	209	24736.0	0.263	0.335	30.995	24.637	0.000	0.242	154.331	122.805	0.008	0.242	322.334	222.576
X-n214-k11	214	9409.0	0.266	0.480	33.883	20.610	0.000	0.118	166.860	71.150	0.000	0.080	326.258	183.930
X-n219-k73	219	89853.0	0.004	0.013	88.899	79.595	0.002	0.010	461.613	381.198	0.000	0.007	855.413	729.933
X-n223-k34	223	32722.0	0.229	0.437	36.637	32.325	0.006	0.271	192.758	140.493	0.000	0.145	390.444	227.392
X-n228-k23	228	21070.0	0.000	0.474	38.103	32.575	0.000	0.006	179.447	160.524	0.000	0.000	344.783	296.825
X-n233-k16	233	16458.0	0.462	0.493	33.750	23.456	0.000	0.262	163.624	111.734	0.298	0.393	383.672	105.968
X-n237-k14	237	21940.0	0.055	0.080	34.508	27.588	0.059	0.063	172.313	132.329	0.000	0.056	351.957	260.777
X-n242-k48	242	63944.0	0.034	0.212	37.357	34.196	0.002	0.123	213.144	144.935	0.000	0.091	440.475	287.564
X-n247-k50	247	29446.0	0.380	1.072	57.361	48.316	0.000	0.459	247.909	201.556	0.000	0.448	505.751	419.195
X-n251-k28	251	30356.0	0.049	0.282	32.146	24.881	0.020	0.084	166.245	139.138	0.000	0.084	321.647	279.282
X-n256-k16	256	16000.0	0.087	0.087	43.440	22.670	0.006	0.079	233.433	80.141	0.000	0.079	493.583	122.240
X-n261-k13	261	21715.0	0.101	0.241	32.414	23.301	0.000	0.252	170.544	107.609	0.101	0.174	354.609	191.350
X-n266-k58	266	56216.0	0.411	0.632	36.844	28.373	0.169	0.443	185.465	161.070	0.000	0.294	399.054	294.148
X-n270-k35	270	28147.0	0.394	0.556	33.583	20.397	0.025	0.373	158.284	142.251	0.000	0.352	319.503	230.796
X-n275-k28	275	17468.0	0.097	0.395	38.742	28.542	0.046	0.201	197.736	161.844	0.000	0.129	387.643	281.095
X-n280-k17	280	27553.0	0.152	0.343	35.360	30.479	0.000	0.221	185.680	158.821	0.051	0.204	367.108	332.549
X-n284-k15	284	16677.0	0.000	0.167	37.454	29.899	0.060	0.149	187.357	126.080	0.000	0.086	367.889	248.520
X-n289-k60	289	71464.0	0.249	0.375	39.657	31.271	0.069	0.303	219.550	139.592	0.000	0.192	431.134	323.863
X-n294-k50	294	36682.0	0.322	0.618	31.944	24.082	0.038	0.300	153.161	140.176	0.000	0.200	322.926	231.508
X-n298-k31	298	27344.0	0.037	0.217	31.346	25.865	0.022	0.046	144.303	124.779	0.000	0.052	294.272	244.960
X-n303-k21	303	17957.0	0.000	0.089	31.349	28.388	0.000	0.022	161.649	128.181	0.000	0.000	319.257	253.424
X-n308-k13	308	21418.0	0.009	0.331	45.261	33.047	0.005	0.264	239.812	166.430	0.000	0.202	460.880	294.074
X-n313-k71	313	73302.0	0.282	0.501	38.914	36.883	0.190	0.259	198.007	185.485	0.000	0.222	411.615	361.187
X-n317-k53	317	61527.0	0.031	0.090	57.615	54.119	0.000	0.026	285.794	268.560	0.008	0.032	563.729	517.289
X-n322-k28	322	24881.0	0.004	0.124	32.364	25.921	0.000	0.038	165.688	129.077	0.000	0.029	336.207	256.177
X-n327-k20	327	23159.0	0.194	0.650	33.316	27.605	0.000	0.383	171.420	123.400	0.047	0.272	347.676	248.082
X-n331-k15	331	25273.0	0.032	0.126	34.457	25.880	0.004	0.073	177.814	114.307	0.000	0.057	347.821	246.321
X-n336-k84	336	106481.0	0.186	0.387	38.568	33.853	0.038	0.218	203.181	152.682	0.000	0.098	401.184	328.429
X-n344-k43	344	33844.0	0.112	0.328	31.709	28.425	0.041	0.117	158.582	126.414	0.000	0.068	305.535	276.982
X-n351-k40	351	20558.0	0.141	0.496	32.109	26.615	0.112	0.427	164.699	139.871	0.000	0.289	327.803	285.073
X-n359-k29	359	40159.0	0.127	0.349	34.258	30.235	0.045	0.188	172.986	142.927	0.000	0.122	342.630	283.229
X-n367-k17	367	18738.0	0.027	0.058	41.741	33.337	0.011	0.064	219.627	172.176	0.000	0.037	424.937	366.851
X-n376-k94	376	111347.0	0.026	0.070	72.888	69.504	0.004	0.021	364.198	340.870	0.000	0.026	741.758	712.779
X-n384-k52	384	50170.0	0.476	0.684	35.274	32.593	0.165	0.325	173.431	146.242	0.000	0.246	343.836	304.706
X-n393-k38	393	30847.0	0.412	0.553	32.271	29.881	0.178	0.323	163.234	146.413	0.000	0.203	316.296	280.582
X-n401-k29	401	51626.0	0.054	0.188	44.077	40.967	0.012	0.065	221.234	200.426	0.000	0.053	439.337	410.978
X-n411-k19	411	16569.0	0.205	0.261	38.048	33.918	0.000	0.187	192.247	174.892	0.109	0.202	389.709	336.746
X-n420-k130	420	84151.0	0.106	0.196	37.952	35.805	0.021	0.093	191.204	180.898	0.000	0.053	383.847	353.787
X-n429-k61	429	52307.0	0.260	0.428	33.791	30.080	0.000	0.159	173.747	156.308	0.042	0.173	352.406	324.350
X-n439-k37	439	29742.0	0.128	0.205	38.776	35.778	0.017	0.079	189.417	168.984	0.000	0.079	380.929	351.129
X-n449-k29	449	44048.0	0.402	0.841	33.181	28.775	0.070	0.630	172.399	139.505	0.000	0.353	343.694	296.947
X-n459-k26	459	19890.0	0.045	0.295	37.743	34.515	0.055	0.144	192.685	172.872	0.000	0.113	387.184	330.957
X-n469-k138	469	167977.0	0.607	0.833	44.076	40.801	0.105	0.364	210.114	198.209	0.000	0.209	445.348	371.439
X-n480-k70	480	69091.0	0.061	0.229	37.626	34.888	0.036	0.148	187.540	173.226	0.000	0.096	375.426	345.082
X-n491-k59	491	51820.0	0.152	0.274	36.828	33.608	0.008	0.162	184.705	164.927	0.000	0.122	366.527	342.691
X-n502-k39	502	52852.0	0.070	0.113	55.070	53.281	0.008	0.049	276.718	268.225	0.000	0.046	549.899	527.697
X-n513-k21	513	21482.0	0.261	0.691	36.419	30.689	0.293	0.525	186.204	172.768	0.000	0.273	364.874	295.106
X-n524-k153	524	118435.0	0.244	0.434	56.974	55.039	0.011	0.421	296.126	280.555	0.000	0.334	606.046	510.726
X-n536-k96	536	72872.0	0.180	0.475	40.233	39.315	0.000	0.120	192.321	186.293	0.010	0.100	387.350	359.664
X-n548-k50	548	67134.0	0.164	0.321	40.894	38.739	0.013	0.115	207.131	196.065	0.000	0.066	400.512	369.458
X-n561-k42	561	34480.0	0.110	0.356	35.538	32.449	0.000	0.226	181.948	153.849	0.003	0.191	356.166	338.222
X-n573-k30	573	39936.0	0.120	0.329	45.109	41.826	0.090	0.240	229.109	208.793	0.000	0.179	456.976	432.951
X-n586-k159	586	146421.0	0.087	0.310	44.327	41.964	0.071	0.134	222.755	211.550	0.000	0.049	460.075	439.380
X-n599-k92	599	83343.0	0.323	0.485	40.535	38.460	0.174	0.285	213.272	206.746	0.000	0.196	413.631	389.275
X-n613-k62	613	47640.0	0.386	0.551	33.081	30.884	0.000	0.212	165.757	159.251	0.052	0.157	328.221	305.672
X-n627-k43	627	48432.0	0.095	0.247	44.390	43.029	0.002	0.082	219.370	212.153	0.000	0.067	438.981	416.966
X-n641-k35	641	50006.0	0.246	0.424	40.686	38.976	0.010	0.202	201.682	188.845	0.000	0.202	412.592	370.459
X-n655-k131	655	81739.0	0.031	0.061	58.345	54.811	0.000	0.021	280.948	264.901	0.006	0.033	578.428	554.002
X-n670-k130	670	108912.0	0.814	1.215	52.050	50.723	0.000	0.564	250.923	241.878	0.132	0.458	507.354	490.801
X-n685-k75	685	54616.0	0.271	0.489	37.801	35.555	0.020	0.251	186.070	175.600	0.000	0.180	374.045	351.157
X-n701-k44	701	63946.0	0.291	0.551	40.473	39.011	0.000	0.206	203.440	191.394	0.120	0.246	403.449	374.119
X-n716-k35	716	33751.0	0.320	0.540	44.617	42.247	0.083	0.203	218.843	207.538	0.000	0.097	434.447	423.983
X-n733-k159	733	105784.0	0.131	0.361	41.735	39.993	0.101	0.199	205.695	197.724	0.000	0.129	415.201	391.061
X-n749-k98	749	59460.0	0.237	0.490	41.438	39.346	0.020	0.222	207.110	196.341	0.000	0.173	415.216	405.869
X-n766-k71	766	88150.0	0.353	0.432	48.808	46.778	0.000	0.295	243.168	237.195	0.001	0.300	487.736	479.374
X-n783-k48	783	58120.0	0.148	0.433	42.658	39.871	0.151	0.230	213.930	202.591	0.000	0.093	426.091	401.762
X-n801-k40	801	57194.0	0.175	0.309	42.697	41.300	0.000	0.175	214.374	202.717	0.026	0.144	424.011	409.592
X-n819-k171	819	120766.0	0.315	0.485	44.363	42.682	0.179	0.256	215.367	208.397	0.000	0.184	425.893	416.411
X-n837-k142	837	148464.0	0.252	0.389	45.413	44.167	0.000	0.140	225.113	217.419	0.036	0.081	449.068	437.149
X-n856-k95	856	69722.0	0.149	0.219	47.846	46.497	0.067	0.133	234.573	229.223	0.000	0.067	458.093	445.529
X-n876-k59	876	75152.0	0.301	0.553	49.396	47.935	0.072	0.323	239.609	231.343	0.000	0.171	476.011	461.848
X-n895-k37	895	44105.0	0.243	0.812	43.639	40.977	0.000	0.251	216.522	204.700	0.000	0.149	432.944	397.283
X-n916-k207	916	247445.0	0.355	0.444	48.217	46.362	0.112	0.202	247.619	244.334	0.000	0.127	481.028	464.557
X-n936-k151	936	104488.0	0.379	0.644	52.572	51.681	0.099	0.246	257.894	244.016	0.000	0.175	513.884	495.366
X-n957-k87	957	66802.0	0.105	0.229	47.414	45.249	0.012	0.103	232.452	227.098	0.000	0.075	466.819	454.606
X-n979-k58	979	91727.0	0.809	0.886	56.905	54.594	0.089	0.569	279.053	268.513	0.000	0.434	553.941	533.660
X-n1001-k43	1001	57512.0	0.383	0.621	44.156	42.320	0.028	0.279	222.046	211.329	0.000	0.238	440.875	420.038
Avg			**0.173**	**0.350**	**40.693**	**32.689**	**0.034**	**0.179**	**205.110**	**158.888**	**0.012**	**0.133**	**412.412**	**310.485**

Table 3.30: Full results for dataset XT

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
X-n101-k25	101	25090.0	0.000	0.017	38.867	9.451	0.000	0.001	203.930	38.320	0.000	0.002	432.294	77.970
X-n106-k14	106	23069.0	0.000	0.139	37.162	13.492	0.000	0.039	211.465	42.591	0.000	0.007	405.517	189.155
X-n110-k13	110	14214.0	0.000	0.039	32.287	3.436	0.000	0.022	162.168	13.196	0.000	0.000	318.401	31.587
X-n115-k10	115	12211.0	0.000	0.000	36.649	0.203	0.000	0.000	184.638	0.214	0.000	0.000	368.896	0.210
X-n120-k6	120	12854.0	0.016	0.100	32.641	16.383	0.000	0.075	172.279	51.600	0.016	0.057	347.370	96.162
X-n125-k30	125	49986.0	0.012	0.519	39.337	29.407	0.000	0.085	169.491	92.635	0.000	0.008	366.117	107.462
X-n129-k18	129	26103.0	0.000	0.077	30.483	17.321	0.000	0.083	174.445	82.650	0.000	0.009	332.815	168.359
X-n134-k13	134	10147.0	0.000	0.035	38.378	22.114	0.000	0.000	184.950	112.105	0.000	0.000	384.214	162.968
X-n139-k10	139	13052.0	0.000	0.001	38.988	1.080	0.000	0.000	199.823	2.035	0.000	0.000	401.495	2.836
X-n143-k7	143	14802.0	0.000	0.407	29.692	10.800	0.000	0.068	141.199	53.310	0.000	0.029	280.763	107.206
X-n148-k46	148	38826.0	0.000	0.084	41.372	27.548	0.000	0.009	225.414	98.571	0.000	0.000	458.441	171.089
X-n153-k22	153	19166.0	0.063	0.106	42.681	33.224	0.026	0.067	215.785	163.644	0.000	0.037	425.199	314.242
X-n157-k13	157	15561.0	0.000	0.013	48.459	43.888	0.000	0.000	239.194	216.499	0.000	0.001	489.156	420.174
X-n162-k11	162	13478.0	0.015	0.142	45.385	13.219	0.030	0.088	243.804	36.221	0.000	0.049	454.155	134.213
X-n167-k10	167	19275.0	0.021	0.181	30.155	17.956	0.021	0.090	160.565	72.937	0.000	0.022	324.988	118.526
X-n172-k51	172	41802.0	0.005	0.013	43.060	35.644	0.000	0.003	227.690	183.847	0.000	0.002	447.897	355.191
X-n176-k26	176	44695.0	0.007	1.306	46.389	33.932	0.000	0.550	217.047	183.796	0.000	0.014	388.323	275.168
X-n181-k23	181	23481.0	0.000	0.067	48.881	32.746	0.000	0.007	212.311	120.385	0.000	0.003	436.969	241.181
X-n186-k15	186	22118.0	0.005	0.005	32.905	12.444	0.000	0.004	184.541	69.469	0.000	0.004	368.404	120.059
X-n190-k8	190	15345.0	0.013	0.081	34.230	25.534	0.007	0.055	171.014	123.057	0.000	0.022	342.214	259.533
X-n195-k51	195	40119.0	0.192	0.321	38.194	26.138	0.000	0.250	223.782	78.285	0.000	0.119	405.621	193.703
X-n200-k36	200	53699.0	0.011	0.141	42.396	29.917	0.000	0.012	194.074	164.825	0.000	0.022	408.570	240.148
X-n204-k19	204	18405.0	0.033	0.060	35.081	22.605	0.000	0.016	181.267	140.930	0.000	0.021	356.481	296.312
X-n209-k16	209	28480.0	0.063	0.149	30.522	23.521	0.004	0.097	158.085	120.299	0.000	0.042	308.040	213.834
X-n214-k11	214	10147.0	0.148	0.425	31.950	15.843	0.000	0.082	163.243	98.447	0.000	0.011	350.375	115.759
X-n219-k73	219	106528.0	0.002	0.003	111.900	97.312	0.001	0.002	586.011	484.533	0.000	0.001	1174.502	971.021
X-n223-k34	223	37176.0	0.221	0.339	36.160	32.660	0.000	0.193	177.959	132.850	0.000	0.099	335.730	299.860
X-n228-k23	228	23853.0	0.017	0.127	33.388	30.269	0.025	0.129	173.691	146.727	0.000	0.089	342.973	299.749
X-n233-k16	233	18037.0	0.000	0.074	36.203	25.129	0.000	0.011	172.584	137.146	0.000	0.008	348.078	281.787
X-n237-k14	237	24974.0	0.088	0.307	32.727	25.427	0.024	0.118	173.223	100.333	0.000	0.094	349.037	252.090
X-n242-k48	242	75119.0	0.000	0.156	38.414	32.812	0.000	0.189	192.516	142.436	0.000	0.071	447.797	317.951
X-n247-k50	247	34957.0	0.154	0.524	53.246	35.276	0.009	0.253	239.388	217.045	0.000	0.129	482.853	412.052
X-n251-k28	251	35525.0	0.025	0.457	32.273	25.545	0.028	0.290	176.462	148.272	0.000	0.143	337.773	252.839
X-n256-k16	256	17585.0	0.000	0.063	34.445	27.132	0.000	0.026	170.544	129.728	0.000	0.019	337.376	260.282
X-n261-k13	261	24572.0	0.000	0.153	33.799	22.500	0.016	0.270	172.753	116.480	0.000	0.085	337.110	201.275
X-n266-k58	266	67782.0	0.056	0.220	40.368	32.774	0.000	0.097	222.580	169.391	0.007	0.065	439.577	344.371
X-n270-k35	270	32171.0	0.134	0.386	31.391	26.979	0.003	0.159	152.689	110.235	0.000	0.087	307.139	180.033
X-n275-k28	275	19644.0	0.000	0.158	41.536	38.451	0.000	0.125	207.843	168.166	0.000	0.091	415.187	332.474
X-n280-k17	280	31515.0	0.143	0.591	33.946	31.153	0.013	0.055	166.074	147.305	0.000	0.066	332.482	299.460
X-n284-k15	284	18905.0	0.127	0.317	34.915	29.133	0.000	0.170	174.758	121.154	0.042	0.188	361.024	240.708
X-n289-k60	289	87470.0	0.000	0.356	47.981	41.211	0.018	0.142	248.843	166.692	0.043	0.146	479.178	393.891
X-n294-k50	294	43375.0	0.203	0.336	32.156	28.597	0.000	0.143	158.770	126.009	0.095	0.260	329.423	242.447
X-n298-k31	298	31758.0	0.050	0.315	27.162	25.317	0.000	0.186	139.938	98.574	0.006	0.151	276.056	225.443
X-n303-k21	303	20240.0	0.104	0.256	32.912	26.682	0.005	0.098	163.443	137.370	0.000	0.105	322.269	265.927
X-n308-k13	308	24367.0	0.008	0.549	38.620	29.766	0.213	0.528	204.713	115.388	0.000	0.294	381.632	278.709
X-n313-k71	313	85629.0	0.207	0.340	37.389	34.162	0.000	0.170	181.590	155.446	0.012	0.154	377.514	348.232
X-n317-k53	317	71661.0	0.000	0.029	61.625	56.284	0.000	0.014	314.147	298.068	0.006	0.021	613.840	564.252
X-n322-k28	322	28085.0	0.256	0.474	30.886	26.079	0.057	0.294	152.915	126.917	0.000	0.165	297.830	254.138
X-n327-k20	327	25753.0	0.054	0.820	35.134	28.633	0.000	0.311	166.812	147.219	0.175	0.330	346.679	268.424
X-n331-k15	331	28460.0	0.004	0.247	34.043	29.302	0.000	0.074	167.932	152.126	0.000	0.076	334.025	286.491
X-n336-k84	336	125720.0	0.138	0.331	35.398	32.744	0.000	0.109	172.711	155.979	0.004	0.104	367.312	293.394
X-n344-k43	344	38520.0	0.109	0.336	31.162	28.071	0.213	0.281	157.882	138.843	0.000	0.245	317.327	248.459
X-n351-k40	351	23684.0	0.139	0.415	31.425	28.027	0.089	0.173	153.842	145.609	0.000	0.115	302.679	277.987
X-n359-k29	359	46945.0	0.288	0.504	31.302	28.911	0.000	0.226	154.211	135.618	0.051	0.133	314.778	259.081
X-n367-k17	367	21654.0	0.171	0.730	42.369	39.013	0.102	0.769	220.850	173.418	0.000	0.806	510.614	270.415
X-n376-k94	376	134666.0	0.005	0.049	85.506	77.423	0.000	0.031	446.810	309.820	0.001	0.023	878.682	765.964
X-n384-k52	384	59344.0	0.118	0.306	37.659	32.681	0.032	0.192	199.047	154.417	0.000	0.107	383.033	297.678
X-n393-k38	393	35186.0	0.185	0.330	31.913	29.780	0.102	0.190	158.072	144.735	0.000	0.158	314.484	286.914
X-n401-k29	401	61902.0	0.128	0.206	45.443	43.382	0.031	0.144	233.750	219.281	0.000	0.078	468.703	421.186
X-n411-k19	411	18625.0	0.000	0.235	36.198	34.482	0.129	0.208	182.757	177.577	0.102	0.164	364.798	346.006
X-n420-k130	420	99301.0	0.107	0.165	40.540	38.431	0.053	0.089	200.904	183.703	0.000	0.060	408.052	341.688
X-n429-k61	429	60716.0	0.156	0.377	35.290	32.617	0.000	0.142	174.755	159.340	0.016	0.141	350.826	317.999
X-n439-k37	439	33453.0	0.093	0.157	37.323	33.847	0.006	0.078	180.754	161.315	0.000	0.090	370.443	337.720
X-n449-k29	449	51028.0	0.123	0.417	34.366	30.777	0.125	0.227	169.291	155.520	0.000	0.173	340.053	290.623
X-n459-k26	459	22510.0	0.204	0.415	35.670	32.498	0.009	0.359	184.673	167.718	0.000	0.183	358.818	327.731
X-n469-k138	469	202649.0	0.525	0.777	44.631	42.611	0.240	0.372	199.862	186.655	0.000	0.269	424.450	387.550
X-n480-k70	480	81819.0	0.125	0.241	37.944	36.406	0.070	0.123	181.170	164.596	0.000	0.059	374.860	336.311
X-n491-k59	491	60730.0	0.199	0.451	35.731	33.655	0.058	0.214	182.398	170.955	0.000	0.102	360.480	325.409
X-n502-k39	502	62729.0	0.041	0.068	57.611	55.817	0.013	0.047	292.330	280.497	0.000	0.046	575.841	555.975
X-n513-k21	513	23056.0	0.000	0.347	37.632	34.964	0.052	0.117	192.879	179.647	0.048	0.141	381.361	318.636
X-n524-k153	524	138782.0	0.164	0.481	53.950	50.283	0.017	0.182	270.822	259.613	0.000	0.214	558.787	539.286
X-n536-k96	536	85175.0	0.195	0.639	41.464	39.542	0.097	0.432	199.795	189.857	0.000	0.197	382.394	370.594
X-n548-k50	548	79026.0	0.028	0.141	43.028	41.095	0.000	0.078	214.766	204.826	0.023	0.070	428.871	398.971
X-n561-k42	561	39267.0	0.446	0.782	36.036	34.401	0.000	0.420	176.701	169.159	0.150	0.332	347.393	325.904
X-n573-k30	573	46237.0	0.065	0.309	45.489	42.981	0.000	0.122	225.883	210.050	0.000	0.046	445.316	417.314
X-n586-k159	586	172982.0	0.309	0.407	47.358	44.567	0.029	0.154	230.976	222.650	0.000	0.146	460.988	443.152
X-n599-k92	599	99152.0	0.182	0.379	40.723	38.178	0.100	0.190	210.104	201.985	0.000	0.083	406.288	370.350
X-n613-k62	613	54298.0	0.271	0.468	33.677	32.216	0.000	0.222	161.744	149.331	0.026	0.183	324.490	302.184
X-n627-k43	627	57242.0	0.093	0.267	44.828	43.352	0.000	0.113	223.044	216.600	0.016	0.119	442.751	422.629
X-n641-k35	641	58278.0	0.182	0.408	40.681	38.793	0.014	0.166	204.895	193.096	0.000	0.104	406.613	370.436
X-n655-k131	655	96769.0	0.018	0.054	67.568	61.840	0.000	0.028	339.587	299.246	0.008	0.020	659.981	625.474
X-n670-k130	670	133617.0	0.254	0.667	56.244	53.326	0.238	0.425	278.752	264.774	0.000	0.202	549.036	525.027
X-n685-k75	685	62677.0	0.268	0.461	38.691	36.522	0.043	0.230	186.016	180.891	0.000	0.155	375.284	353.567
X-n701-k44	701	74837.0	0.417	0.592	41.181	39.263	0.000	0.194	205.358	193.940	0.045	0.147	406.942	388.706
X-n716-k35	716	39434.0	0.320	0.543	43.358	42.821	0.010	0.219	217.888	209.844	0.000	0.094	431.691	411.798
X-n733-k159	733	123851.0	0.270	0.353	42.534	40.725	0.052	0.143	206.267	196.256	0.000	0.095	414.929	392.652
X-n749-k98	749	71545.0	0.245	0.546	40.072	38.405	0.145	0.211	201.313	193.609	0.000	0.112	400.338	384.891
X-n766-k71	766	103657.0	0.179	0.357	47.525	44.975	0.035	0.130	237.472	229.031	0.000	0.138	475.340	450.568
X-n783-k48	783	66085.0	0.133	0.365	42.253	40.477	0.020	0.204	209.536	189.739	0.000	0.126	419.047	407.896
X-n801-k40	801	67122.0	0.054	0.236	43.357	40.522	0.055	0.120	214.408	206.407	0.000	0.071	430.650	412.779
X-n819-k171	819	144690.0	0.274	0.492	41.399	39.827	0.037	0.178	195.672	190.212	0.000	0.064	390.693	359.881
X-n837-k142	837	175138.0	0.285	0.439	50.763	49.457	0.073	0.167	227.473	214.507	0.000	0.079	456.169	436.037
X-n856-k95	856	81204.0	0.084	0.191	51.755	50.797	0.010	0.117	251.409	245.692	0.000	0.064	503.410	479.781
X-n876-k59	876	90507.0	0.336	0.762	47.952	47.577	0.033	0.427	228.328	223.577	0.000	0.170	444.913	435.728
X-n895-k37	895	49904.0	0.383	0.644	43.394	41.472	0.170	0.347	213.465	207.092	0.000	0.153	419.581	395.048
X-n916-k207	916	298667.0	0.398	0.507	48.751	47.117	0.067	0.179	236.869	228.939	0.000	0.093	449.745	421.801
X-n936-k151	936	120742.0	0.476	0.634	50.860	48.843	0.074	0.280	247.007	238.961	0.000	0.294	493.780	479.244
X-n957-k87	957	78057.0	0.186	0.245	50.217	48.598	0.000	0.105	249.290	242.193	0.035	0.075	496.077	473.039
X-n979-k58	979	108570.0	0.157	0.269	54.610	52.779	0.000	0.074	274.911	263.375	0.002	0.053	546.628	524.792
X-n1001-k43	1001	66711.0	0.429	0.763	44.729	43.883	0.100	0.330	221.390	214.297	0.000	0.229	438.371	428.170
Avg			**0.124**	**0.323**	**41.283**	**34.172**	**0.032**	**0.160**	**206.805**	**163.705**	**0.009**	**0.105**	**412.839**	**322.414**

Table 3.31: Full results for dataset XXLH

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
Leuven1	3001	114189.0	0.902	1.192	57.257	56.821	0.123	0.335	287.228	284.111	0.000	0.133	572.301	566.699
Leuven2	4001	76187.0	1.244	1.657	76.717	75.283	0.224	0.498	390.784	384.092	0.000	0.143	782.492	774.700
Antwerp1	6001	277391.0	1.270	1.539	64.310	64.228	0.269	0.357	316.557	315.305	0.000	0.101	633.918	628.605
Antwerp2	7001	185142.0	2.138	2.396	67.286	67.134	0.214	0.620	343.886	341.744	0.000	0.255	699.515	694.561
Ghent1	10001	271224.0	1.670	1.812	83.367	83.336	0.382	0.550	403.475	402.782	0.000	0.054	803.289	801.747
Ghent2	11001	163972.0	2.641	2.958	88.894	88.586	0.564	0.836	459.459	458.249	0.000	0.381	940.444	935.345
Brussels1	15001	302630.0	1.966	2.034	101.942	101.877	0.436	0.609	492.033	491.384	0.000	0.065	982.451	981.226
Brussels2	16001	222184.0	2.984	3.393	114.324	114.149	0.659	0.776	568.799	567.907	0.000	0.164	1159.963	1157.090
Flanders1	20001	4070908.0	1.271	1.410	147.368	147.303	0.358	0.486	690.899	690.719	0.000	0.118	1374.369	1373.876
Flanders2	30001	2692883.0	3.205	3.331	183.312	183.233	0.876	1.007	805.964	805.780	0.000	0.196	1607.350	1606.307
Avg			**1.929**	**2.172**	**98.478**	**98.195**	**0.411**	**0.607**	**475.908**	**474.207**	**0.000**	**0.161**	**955.609**	**952.016**

Table 3.32: Full results for dataset XXLQ

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
Leuven1	3001	151052.0	0.441	0.539	64.564	63.882	0.000	0.108	319.472	316.041	0.014	0.070	638.852	631.708
Leuven2	4001	92568.0	0.646	0.850	70.139	69.701	0.178	0.349	353.287	349.661	0.000	0.184	705.398	693.365
Antwerp1	6001	372610.0	0.542	0.588	75.213	75.146	0.098	0.164	376.007	374.818	0.000	0.062	749.548	745.529
Antwerp2	7001	234127.0	1.255	1.376	69.584	69.367	0.229	0.393	354.702	352.865	0.000	0.112	713.130	710.750
Ghent1	10001	365274.0	0.798	0.884	97.661	97.512	0.132	0.207	484.013	483.687	0.000	0.045	971.599	969.672
Ghent2	11001	209515.0	1.328	1.512	88.300	88.148	0.203	0.316	458.872	458.001	0.000	0.069	925.009	922.058
Brussels1	15001	395568.0	1.219	1.289	112.614	112.531	0.277	0.370	549.912	549.177	0.000	0.054	1097.312	1096.318
Brussels2	16001	282942.0	2.056	2.242	112.724	112.632	0.345	0.503	558.367	557.181	0.000	0.096	1113.451	1112.731
Flanders1	20001	5580410.0	0.719	0.816	161.615	161.567	0.241	0.277	786.900	786.724	0.000	0.096	1555.288	1553.964
Flanders2	30001	3513440.0	2.640	2.740	179.397	179.353	0.410	0.530	809.293	808.998	0.000	0.155	1612.299	1611.561
Avg			**1.164**	**1.283**	**103.181**	**102.984**	**0.211**	**0.322**	**505.082**	**503.715**	**0.001**	**0.094**	**1008.189**	**1004.766**

Table 3.33: Full results for dataset XXLT

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
Leuven1	3001	177105.0	0.352	0.460	66.059	65.580	0.119	0.209	329.082	325.810	0.000	0.109	654.726	648.416
Leuven2	4001	103576.0	0.703	1.292	66.227	65.534	0.058	0.307	324.347	319.930	0.000	0.116	645.503	640.436
Antwerp1	6001	434312.0	0.459	0.511	77.686	77.531	0.100	0.145	386.221	385.180	0.000	0.056	774.017	770.870
Antwerp2	7001	270063.0	0.928	1.065	69.609	69.460	0.188	0.314	354.793	353.611	0.000	0.095	707.649	704.445
Ghent1	10001	429480.0	0.597	0.650	100.929	100.780	0.144	0.177	491.837	491.042	0.000	0.020	974.587	973.866
Ghent2	11001	238958.0	1.485	1.713	85.479	85.430	0.262	0.339	440.448	440.102	0.000	0.086	883.547	880.552
Brussels1	15001	461696.0	0.874	0.998	115.473	115.421	0.237	0.279	561.922	561.132	0.000	0.047	1118.340	1117.921
Brussels2	16001	321980.0	1.817	1.938	110.023	109.885	0.408	0.475	539.192	538.846	0.000	0.110	1085.982	1084.337
Flanders1	20001	6599087.0	0.621	0.669	166.781	166.713	0.163	0.200	813.679	813.454	0.000	0.062	1628.515	1627.951
Flanders2	30001	4053751.0	2.285	2.355	176.767	176.728	0.269	0.414	791.934	791.663	0.000	0.071	1583.334	1582.927
Avg			**1.012**	**1.165**	**103.503**	**103.306**	**0.195**	**0.286**	**503.345**	**502.077**	**0.000**	**0.077**	**1005.620**	**1003.172**

Chapter 4

Daily Planning of Acquisitions and Scheduling of Dynamic Downlinks for the PLATiNO Satellite

PLATiNO is a small Synthetic Aperture Radar Earth observation satellite launched in 2023. The operational life of the satellite is controlled by an activity plan, generated on a routine basis, which includes acquisitions, maneuvers, and downlinks. The aim of the plan is to satisfy as many requests as possible, thus maximizing the satellite exploitation, without violating platform constraints. We develop a genetic algorithm, which is hybridized with repair procedures to fix infeasible solutions and local search operators to quickly identify high-quality local optima. In addition, we model the problem using Mixed Integer Linear Programming formulations to provide tight estimations of the optimal solution value. We test our heuristic algorithm on several realistic benchmark instances derived from real-world data provided by Thales Alenia Space Italia and compare the quality of the solutions with the bound values produced by the formulations. The computational results show that our approach is able to compute near-optimal solutions within computing times that are fully compatible with the real-world application.

4.1 Introduction

PLATiNO is a program managed by Agenzia Spaziale Italiana (ASI), the Italian Space Agency, focused on the development of a small satellite (50-300 kg) platform, characterized by a high level of adaptability to specific customer needs and easy-tailored for a wide set of Earth observations and telecommunication missions and constellations. PLATiNO program includes the development of two satellites: PLATiNO-1, equipped with a Synthetic Aperture Radar (SAR) sensor, and PLATiNO-2, equipped with a Thermal Infra-Red sensor, launched in 2022 and 2023, respectively.

PLATiNO-1, from now on referred to as PLATiNO, is a civil EO SAR mission aimed at monitoring the Mediterranean area; imaging can be performed both in passive and active modes, with a ground resolution of up to 1 meter.

The mission is split into two phases, lasting 1 and 1.5 years respectively:

- In Phase 1 the satellite will be in formation with either a COSMO-SkyMed or "Cosmo Seconda Generazione" satellite, at 619 km altitude. In this phase, the PLATiNO SAR sensor will mainly work as a bistatic SAR, a configuration in which the COSMO-SkyMed SAR is the transmitter while the PLATiNO SAR is the receiver.
- During Phase 2, the satellite orbit altitude will be reduced to 410 km, and the SAR sensor will mainly operate in monostatic mode, working both as a transmitter and a receiver.

Thales Alenia Space Italia (TASI), in collaboration with the CIRI-ICT inter-department research center of the Alma Mater Studiorum University of Bologna, developed an efficient and effective genetic

algorithm for the PLATiNO mission planner.

This mission planner algorithm takes into account several specific mission constraints, which include:

1. Absence of SAR electronic steering: PLATiNO antenna is fixed to the platform and a limited set of directive beams is available. This implies that the satellite has to reorient the SAR antenna at each new acquisition. In this scenario, the algorithm has to plan sets of acquisitions requiring a similar off-nadir angle (and so similar platform attitude), provided that some operational constraints are satisfied.
2. Management of high-priority acquisitions occurring during data downlink: PLATiNO requires flexible planning including data downlink interruption, fast re-orientation and acquisition, and downlink resumption. For this reason, the PLATiNO planning algorithm must take into account flexible and dynamic management of the downlink activities, in which time windows have to be automatically adjusted to maximize the total number of acquisitions while satisfying the satellite operational constraints.

A noteworthy distinction in this problem compared to several previous studies in the literature (see Section 4.2) is that downlink activities must be scheduled in their respective downlink opportunities and they may also conflict with acquisition opportunities.

The primary goal of this collaboration is the development of a "solver-free" heuristic algorithm that can run within a short computing time (i.e., at most 5 minutes, single-thread on a standard PC), thus providing a base-prototype for further development by the TASI team. To this end, this paper introduces a novel mission planning algorithm that allows flexible acquisition scheduling while considering various mission constraints, such as satellite memory management and dynamic scheduling of downlink activities. The proposed heuristic is computationally evaluated on a benchmark of realistic instances provided by TASI. Furthermore, we define both exact and relaxed mathematical formulations of the problem, which enabled us to compute upper bounds on the optimal solution values. Given the effectiveness of modern commercial optimization solvers, especially when applied to a subset of the test instances, we have designed and tested a number of math-heuristics as possible competitors to the algorithm developed for TASI.

The rest of the paper is organized as follows. Section 4.2 surveys the current state-of-the-art on relevant planning algorithms, whereas Section 4.3 provides a comprehensive description of the key features and constraints of the PLATiNO mission planning. In Section 4.4 we introduce a mathematical formulation of the problem and a relaxation that allows to compute tight dual bounds. Section 4.5 presents the details of our proposed heuristic algorithm. Section 4.6 discusses the formulation used to compute an upper bound, while Section 4.7 introduces formulation-based competitors. Experimental results are presented in Section 4.8. Finally, in Section 4.9, we review the achieved results, discuss potential algorithm extensions, and outline future activities.

4.2 Literature Review

There is a large literature facing the scheduling of satellite activities in both single and multi-satellite environments. In this section, we mainly review single-satellite problems and algorithms used to solve them, as this is the target of the current research. We refer to Barkaoui and Berger (2020), Wang, Demeulemeester, and Qiu (2016), Wang et al. (2015) for a comprehensive overview of the algorithms proposed for this class of problems. Each contribution in the literature is typically tailored towards the resolution of a specific industrial scenario, using a precise satellite model and handling specific features and constraints. Most real-world problems are, as expected, solved by means of heuristic approaches due to the need to obtain high-quality solutions for large instances in short computing time. Many heuristics are based on greedy constructive algorithms Bensana et al. (1999), possibly with look-ahead and back-tracking capabilities Bianchessi and Righini (2008), genetic algorithms Li, Xu, and Wang (2007), Mansour and Dessouky (2010), tabu search Vasquez and Hao (2001), Bensana et al. (1999), Cordeau and Laporte (2005), Lin and Liao (2004), Lin et al. (2005) and iterated local search Peng et al. (2018). The literature also includes a few exact algorithms, mainly based on dynamic programming Gabrel and Vanderpooten (2002), Hall and Magazine (1994), Lemaitre et al. (2002) and Russian doll search Bensana et al. (1999), Benoist and Rottembourg (2004).

The problems addressed also depend on the type of satellite that is used. For example, SPOT5 satellites Mansour and Dessouky (2010), Vasquez and Hao (2001), Zhang et al. (2013), Bensana et al. (1999) only perform acquisitions starting at fixed times (i.e., determined apriori), whereas Agile satellites Li, Xu, and Wang (2007), Liu et al. (2017), Tangpattanakul, Jozefowiez, and Lopez (2015) are more flexible and can postpone or anticipate acquisitions within specific time-windows possibly at the expense of the quality of the resulting image. In most problems, the management of the satellite memory, as well as the scheduling of downlink activities, is typically neglected or strongly approximated. For example, in Li, Xu, and Wang (2007), Liu et al. (2017), Mansour and Dessouky (2010), Vasquez and Hao (2001), Luo et al. (2017), Bensana et al. (1999), Lin and Liao (2004), Lin et al. (2005) the satellite memory is only modeled using a capacity constraint, whereas downlink activities are not taken into account. A notable exception is given in Bianchessi and Righini (2008), where a satellite is considered that can perform acquisition and downlink operations at the same time, with no need to account for the latter during the planning of the former activities.

Conversely, with the PLATiNO satellite, as discussed in the introduction, both acquisition and downlink activities must be scheduled; in addition, the former operations may even overlap with downlink visibility windows, generating a race over the use of the satellite. The acquisition windows (namely, the sensing start and stop times) are computed by a dedicated algorithm, invoked before the start of the planning phase. This is due to the need to maximize the quality of the image: all the programming parameters of the SAR sensor are computed taking into account a series of inputs, such as the acquisition mode, the area of interest, and the morphology of the terrain. Therefore, similarly to the SPOT5 satellites, from a "planning" point of view, the acquisitions do not require to be scheduled inside a given time window, but only to be selected (i.e., planned).

4.3 Problem Description

The PLATiNO planning and scheduling problem requires to determine an optimal set of *acquisition requests* (ARs) that the satellite has to serve during a given time horizon. Each AR is associated with a geographical area, and is characterized by a positive *priority* and by a positive *memory size*. As the satellite may pass several times over the same geographical area, each AR can be satisfied within different *data-take opportunities* (DTOs), each associated with a time window during which the satellite may acquire a picture of the required area while meeting strict quality requirements. The overall objective of the problem is to maximize the sum of the priorities of the selected ARs. An acquired DTO is stored in the satellite memory. Given the finite capacity of this memory, stored acquisitions have to be sent back to Earth during specific *downlink opportunities* (DLOs), each defined by a time window associated with the visibility cone of a ground station antenna. In the PLATiNO satellite, while DTO and DLO time windows can overlap, the satellite can perform only one of these operations at a time.

During a DLO, multiple acquisitions are downlinked, following a predefined set of criteria dependent on the particular scenario in which the PLATiNO satellite is employed. As discussed in Section 4.3, a simplified yet well-defined downlink policy has been designed to encapsulate the complexities and underlying principles of real-world scenarios.

Note that, regardless of the downlink policy, it may not be possible to downlink all stored acquisitions during the current satellite plan; in this case, the remaining stored data are considered part of the initial satellite storage during the planning activity of the following day. Finally, in some predetermined time windows, called *platform activity windows* (PAWs), the satellite is unavailable (e.g. because it has to perform orbit correction activities) and neither acquisition nor downlink can be executed. Thus, neither DTOs nor DLOs may overlap with PAWs. In general, however, PAWs can be managed during a preprocessing step, by removing (or resizing) all activities overlapping with a PAW.

Typically, the large number of potential ARs and the constraints of the satellite make it impossible to serve all requests. It is thus necessary to set up an optimization tool to select a subset of ARs that is eventually served during the planning horizon.

In the following, we list the operational and physical constraints ruling the PLATiNO satellite activities.

Time conflicts As the satellite can perform one operation at a time, all planned operations must be executed sequentially. This implies that the time windows for each pair of planned activities (i.e., DTO, DLO, or PAW) must not overlap. In addition, a minimum time between the end of an activity and the start of the following one must be guaranteed, because the satellite must complete the possibly required maneuvers. The maneuver time depends on the current satellite attitude and on the forthcoming activity that has to be performed. Indeed, each activity is in fact associated with a specific attitude describing the orientation in space that the satellite must set up in order to accomplish it. In addition, to perform an acquisition and obtain a clear picture, the satellite has to be stabilized, an operation that requires a constant stabilization time.

Operational profile The satellite must not exceed a maximum number of acquisitions for any orbit around the Earth. Given that the satellite performs many orbits of the same duration during its daily schedule, this constraint is modelled by imposing a maximum number of acquisitions in any rolling window of that duration. We observe that, this constraint implicitly limits the energy consumption of the satellite during its activities.

Memory The satellite has an internal storage with limited capacity, that cannot be exceeded. During DLOs, data that have been stored during acquisitions can be transmitted, thus releasing some memory.
Though being an approximation of the real one, the downlink policy that is considered has a practical relevance from the application side. We assume that acquired data can be downloaded at a constant rate, which depends on the specific satellite used and which imposes an upper bound on the amount of data that can be transmitted to the Earth in a given downlink window. In addition, only complete acquisitions can be transmitted, i.e., partial downlink of the data for later completion is not allowed. Finally, the total amount of used memory at the end of the planning cannot exceed a given threshold, to avoid overconstrained situations in the following schedule.

The problem can be formally stated as follows. Let A be the set of ARs to be fulfilled. Let $D = \{1, ..., N\}$ be the set of N DTOs and $\Lambda = \{1, ..., L\}$ be the set of L DLOs. Without loss of generality, we assume that both sets D and Λ are sorted by non-decreasing starting time of the activities.

Each AR may be satisfied by different DTOs; in particular, for each AR $a \in A$, we denote by $D_a^{AR} \subseteq D$ the set of DTOs that allow acquisition of AR a.

Each DTO $i \in D$ associated with an AR $a \in A$ (i.e., $i \in D_a^{AR}$) is associated with

- the priority p_i of AR a;
- the amount of memory m_i occupied by AR a;
- a start time s_i and a finish time f_i during which the satellite may capture the image.

For each pair (i, j) of distinct DTOs, we denote by $\gamma(i, j)$ the maneuver-time needed for setting the satellite up to acquire DTO j right after DTO i; we assume that this value includes the time needed for stabilization of the satellite as well.

Each DLO $l \in \Lambda$ is characterized by a time window $[\sigma_l, \phi_l]$ during which the satellite can downlink acquisitions. To perform a downlink, the satellite's orientation has to be set to maintain alignment with the ground-based antenna. The technical time for this operation depends on the last acquired DTO. For this reason, for each DTO i and DLO l, we denote by $\gamma_l^+(i)$ the maneuver time needed to transition from the DTO acquisition to hte DLO operation. Conversely, we let $\gamma_l^-(i)$ denote the time needed for the transition from DLO l to DTO i.

Finally, the problem is characterized by the following parameters:

- T_{orbit} is the time needed by the satellite to complete an orbit around the Earth;
- C_{max} is the maximum number of acquisitions that the satellite can perform during a single orbit;
- T_{acq} is the minimum time between two planned acquisitions, even if no maneuvers are needed;
- R_{dl} is the downlink rate, i.e., the amount of memory freed per unit of time during a downlink activity;

- M_{tot} is the maximum satellite memory that can be used at any given time;
- M_{init} is the initial satellite memory occupation;
- M_{fin} is the maximum final memory occupation.

We assume $M_{init} \leq M_{fin} \leq M_{tot}$, i.e., that the initial memory is smaller or equal than the final one, thus ensuring that an empty plan is always a feasible solution.

The problem calls for the determination of a scheduling of the satellite activities, and for determining a sequence of DTOs and DLOs so that feasibility constraints are satisfied and the sum of the priorities of selected ARs is a maximum.

4.4 Mathematical Formulation

We now present a mathematical formulation of the problem described above.

For each DTO $i \in D$, we introduce a binary d_i with the following meaning:

$$d_i = \begin{cases} 1 & \text{if DTO } i \text{ is inserted into the plan} \\ 0 & \text{otherwise} \end{cases} \quad i \in D.$$

The objective function is then

$$\max \quad \sum_{i \in D} d_i p_i \tag{4.1}$$

subject to the constraint hereafter defined.

4.4.1 General scheduling constraints

The first set of constraints impose that, for each AR $a \in A$, at most one associated DTO is selected, namely

$$\sum_{i \in D_a^{AR}} d_i \leq 1 \quad a \in A \tag{4.2}$$

There must be enough time between two planned DTOs for the satellite to accomplish the necessary maneuvers. Note also that a minimum time T_{acq} must always be considered between two successive acquisitions.

$$d_i + d_j \leq 1, \quad \forall\, i, j \in D : (s_i \leq s_j) \wedge (f_i + \max\{T_{acq}, \gamma(i,j)\} > s_j) \tag{4.3}$$

For any given orbit time window T_{orbit}, the number of acquisitions must not be greater than a constant value C_{max}, which depends on the antennas' uplink capabilities and on satellite energy consumption.

$$\sum_{\substack{i \in D: \\ s_j \leq s_i < s_j + T_{orbit}}} d_i \leq C_{max} \quad j \in D \tag{4.4}$$

4.4.2 Memory management

To model memory management through the plan, we define a set of time intervals $[s, f]$ with s, f being two arbitrary time instants of the plan. Let us define the parameterized subsets $D_{sf} = \{i \in D : s_i < f \wedge f_i > s\}$ containing all the DTOs whose time windows overlap the time interval starting at s and ending at f. For each interval defined in the next paragraphs, we will introduce the corresponding binary variables t_{sf} that express if a given time window is *active* in the model.

A necessary requirement for any interval to be considered active is that no DTO that overlaps with that interval has been selected. Therefore, for any variable t_{sf} that will be introduced, the following constraint is enforced:

$$t_{sf} \leq 1 - \frac{\sum_{i \in D_{sf}} d_i}{|D_{sf}|} \tag{4.5}$$

Then, we consider 4 scenarios.

DLO intervals Given a DLO $l \in \Lambda$, we define the interval $[\sigma_l, \phi_l]$. The corresponding variable $t_{\sigma_l \phi_l}$ does not require additional constraints, because (4.5) already models the activation criterion.

DLO to DTO intervals Let us consider a DLO $l \in \Lambda$ and a DTO $i \in D$, we define the values $s_i^l = s_i - \gamma_l^-(i)$ as the time instant at which any downlink activity has to be stopped to leave enough time for the maneuver towards DTO $i \in D$. DLO to DTO intervals are defined between the start of any DLO $l \in \Lambda$ and each s_i^l with $i \in D : \sigma_l < s_i^l < \phi_l$. The corresponding variables $t_{\sigma_l s_i^l}$ require the additional activation constraints:

$$t_{\sigma_l s_i^l} \leq d_i \quad l \in \Lambda, i \in D : \sigma_l < s_i^l < \phi_l \tag{4.6}$$

since, to be considered active, the corresponding DTO i has to be selected in the plan.

DTO to DLO Intervals Symmetrically with the previous case, we model the ending part of the DLO windows considering the intervals $[f_i^l, \phi_l]$ for each DLO $l \in \Lambda$ and DTO $i \in D : f_i^l < \phi_l$. Similarly to time instant s_i^l, the values $f_i^l = f_i + \gamma_l^+(i)$ are defined to account for the first time instant from which downlink operations can start when DTO $i \in D$ is executed. The corresponding variables $t_{f_i^l \phi_l}$ require the additional activation constraints:

$$t_{f_i^l \phi_l} \leq d_i \quad l \in \Lambda, i \in D : \sigma_l < f_i^l < \phi_l \tag{4.7}$$

DTO to DTO Intervals Finally, the fourth case represents all the intervals between pairs of DTOs $i, j \in D_{\sigma_l \phi_l} : f_i < s_j$ overlapping a given DLO $l \in \Lambda$. These intervals are active if constraints (4.5) are satisfied and if DTOs i and j are both selected in the plan.

$$t_{f_i^l s_j^l} \leq \frac{d_i + d_j}{2} \quad l \in \Lambda, i, j \in D_{\sigma_l \phi_l} : f_i < s_j \tag{4.8}$$

Occasionally, given two intervals $[s_1, f_1]$ and $[s_2, f_2]$, it might happen that both $s_1 = s_2$ and $f_1 = f_2$ hold true, resulting in a naming conflict for the symbols defined for such intervals. While this presents a symbol-naming issue, it can be trivially addressed by introducing an index to distinguish between duplicated sets of names. However, for clarity's sake, we do not apply such a notation in the present description, assuming that such a case does not happen.

Intervals Aggregation

In light of the definition of the set of intervals corresponding to variables t, a straightforward approach to model the acquisition downlink would involve defining binary variables that indicate whether a given DTO is downlinked during a particular time interval. However, it becomes evident that the number of intervals increases quadratically with the number of DTOs overlapping each DLO. Multiplying this quantity by the number of DTOs can result in a model that quickly grows in size, making it challenging to solve, even for relatively small instances. As a consequence, due to the fact that only one of two overlapping time intervals can be chosen at any given moment, we group such intervals together to reduce the number of required variables. For the sake of brevity, we focus on describing solely the formulation derived after this aggregation.

As just mentioned, overlapping intervals are inherently mutually exclusive, meaning that only one can be active at any given time. Furthermore, within each interval, only DTOs that have been acquired beforehand can be downlinked, while DTOs coming after the starting time of the interval remain non-downlinkable due to either their temporal position or overlapping status.

Grouping overlapping intervals that share the same initial time instant significantly reduces the number of downlink variables. To illustrate this concept, consider Figure 4.1: if a DLO encompasses four overlapping DTOs, it results in the generation of 15 potential interval variables, which is proportional to the square of the number of overlapped DTOs. However, by aggregating intervals with an identical starting time instant, the number of clusters remains linear and equals the number of DTOs plus one.

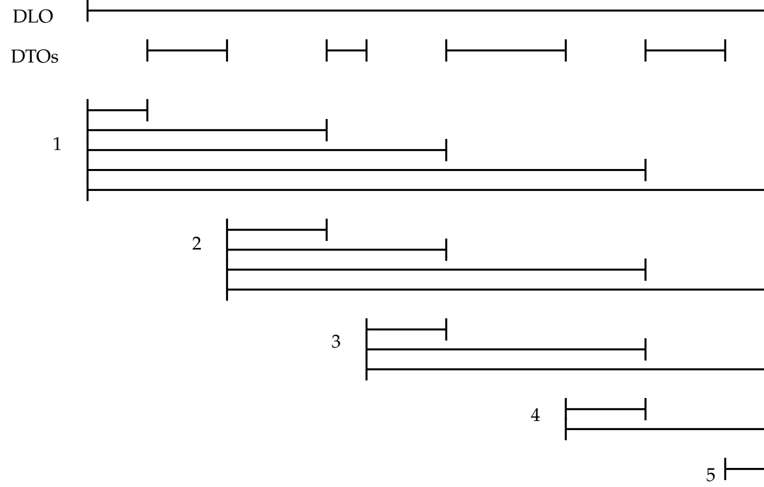


Figure 4.1: Illustration of grouping intervals resulting from DTO and DLO overlapping.

For each cluster of intervals, identified by their starting time, and for each potentially acquired DTO (those whose time window ends before the start of the cluster), we can introduce a binary variable denoted as y:

$$y_{si} = \begin{cases} 1 & \text{if DTO } i \in D \text{ is downlinked during an interval starting at } s \\ 0 & \text{otherwise} \end{cases} \quad l \in \Lambda$$

Here, for a given DLO $l \in \Lambda$, the time instants take values in the set

$$s \in S_l = \{\sigma_l\} \cup \{f_i^l \; \forall i \in D : \sigma_l < f_i^l < \phi_l\}.$$

Each y variable can only be active if one of the underlying intervals is active. Additionally, the total memory downlinked within the cluster cannot exceed the maximum memory that can be downlinked during the currently active interval. Both conditions can be enforced through a single constraint:

$$\sum_{\substack{i \in D: \\ f_i^l < s}} m_k y_{si} \leq \left[(\phi_l - s) t_{s\phi_l} + \sum_{i \in D_{s\phi_l}} (s_i^l - s) t_{ss_i^l} \right] R_D \quad l \in \Lambda,\, s \in S_l \tag{4.9}$$

The memory constraints are defined as follows:

$$\sum_{\substack{i \in D: \\ f_i^l \leq \hat{s}}} m_i d_i - \sum_{h \in \Lambda} \sum_{\substack{s \in S_h: \\ s < \hat{s}}} \sum_{\substack{i \in D: \\ f_i^h < s}} m_i y_{si} \leq M_{tot} \quad l \in \Lambda,\, \hat{s} \in S_l \tag{4.10}$$

A constraint is defined at the starting time of every potential downlink operation. The first term in the constraint represents the memory occupied by completed acquisitions, while the second term accounts for the memory freed during the previous downlinks.

In addition to the previous constraints, a final memory test must also be performed at the very end of the plan, to check that not only the maximum memory load is not exceeded, but also that the final

memory occupation requirement is met.

$$\sum_{i \in D} m_i d_i - \sum_{l \in \Lambda} \sum_{s \in S_l} \sum_{\substack{i \in D: \\ f_i^l < s}} m_i y_{si} \leq M_{fin} \tag{4.11}$$

Given that memory load decreases monotonically during downlink operations, contrasted with its monotonically non-decreasing profile in all other time intervals (as visually depicted in Figure 4.3), monitoring memory load during this finite set of time instances suffices to evaluate the memory feasibility of the plan.

Additionally, each y variable can only be equal to 1 when its corresponding DTO activation variable is also equal to 1.

$$\sum_{\substack{l \in \Lambda: \\ \sigma_l \geq f_i^l}} \sum_{s \in S_l} y_{si} \leq d_i \quad i \in D \tag{4.12}$$

4.4.3 Relaxed Memory Management

The formulation discussed in the previous section proves effective in finding optimal solutions for many instances provided by TASI. This is particularly evident in cases where optimal plans are not tightly constrained by memory limitations. However, when dealing with instances characterized by heavy memory usage, the proposed formulation often fails at closing the optimization gap within a reasonable time limit. Additionally, the resulting bound closely aligns with the one obtained when considering the model without memory constraints, which tends to be relatively weak, as demonstrated in Section 4.8.4. Given that memory usage is a prominent aspect of the problems under consideration, we explored a relaxation of the problem. This relaxation allows us to retain the memory aspect in the model while relaxing some constraints to simplify the optimization process.

Specifically, we make the following relaxations:

- We disregard the time required for maneuvers between DTOs and DLOs, considering only the constant stabilization time required before each acquisition.
- We permit partial downlinking of acquisitions, actually removing the discrete nature of downlink scheduling.

This approach treats memory occupation as a homogeneous quantity that can be increased with new acquisitions and reduced to match the exact amount transmittable to Earth within the downlink time windows.

Optimal solutions derived from this relaxed model allow us to compute a dual bound. This bound becomes valuable when the complete formulation struggles to find the optimal solution, or when optimization tasks are challenging due to problem size. Furthermore, as we describe in Section 4.6 the following relaxed model of the memory, can be merged with the exact model, even if redundant, to improve the linear relaxation of the exact model and help the optimization process in both convergence speed and feasible solution quality when the optimization is stopped because the time limit is reached.

Similar to the complete model, the relaxation uses variables:

$$d_i = \begin{cases} 1 & \text{if DTO } i \in D \text{ is inserted into the plan} \\ 0 & \text{otherwise} \end{cases} \quad i \in D$$

The objective function and constraint from (4.1) to (4.4) remain unchanged. All variables and constraints related to exact memory management described in Section 4.4.2 are disregarded. Instead, we introduce additional constraints to model the satellite memory and downlink management requirements in a relaxed way. To achieve this, we introduce two sets of continuous variables denoted as δ and μ. For each DLO $l \in \Lambda$, δ_l and μ_l represent the amount of memory downlinked during DLO $l \in \Lambda$ and the amount of memory occupied at the beginning of DLO $l \in \Lambda$, respectively.

The following constraint establishes the amount of memory used at the beginning of the first DLO, scheduled at time σ_0:

$$\mu_0 = M_{init} + \sum_{\substack{i \in D: \\ s_i \leq \sigma_0}} d_i m_i \tag{4.13}$$

Subsequent values of μ_{l+1} for $l \in \Lambda \setminus \{L\}$ are computed based on μ_l, incorporating the memory of DTOs acquired in between and subtracting the estimated memory freed during DLO l:

$$\mu_{l+1} = \mu_l - \delta_l + \sum_{\substack{i \in D: \\ \sigma_l < s_i \leq \sigma_{l+1}}} d_i m_i \quad l \in \Lambda \setminus \{L\} \tag{4.14}$$

To impose constraints on memory capacity, we bound variables μ as follows:

$$\mu_l \leq M_{tot} \quad l \in \Lambda \tag{4.15}$$

$$\tag{4.16}$$

Finally, the memory usage at the end of the plan is constrained by:

$$\mu_L - \delta_L + \sum_{\substack{i \in D: \\ s_i \geq \sigma_L}} d_i m_i \leq M_{fin} \tag{4.17}$$

To limit the amount of memory downlinked during each DLO, we impose bounds on the δ variables. First of all, constraints (4.18) ensure that the memory freed does not exceed the satellite's capability to free memory during the given time window. The time window is defined by subtracting the time spent acquiring the DTOs that overlap with the DLO from its length:

$$\delta_l \leq \left[(\phi_l - \sigma_l) - \sum_{i \in D_{\sigma_l \phi_l}} d_i (\min\{f_i, \phi_l\} - \max\{s_i, \sigma_l\}) \right] R_D \quad l \in \Lambda. \tag{4.18}$$

Note that we could not use s_i^l and f_i^l values to obtain a tighter bound accounting for maneuver times in the previous constraints. This is due to DTO–DTO transitions that make the use of s_i^l and f_i^l inappropriate since they would account for maneuvers from and towards the DLO $l \in \Lambda$ that does not take place.

Constraints (4.19) ensure that the memory downlinked does not exceed the memory used:

$$\delta_l \leq \mu_l + \sum_{\substack{i \in D: \\ \sigma_l < s_i \wedge f_i < \phi_l}} d_i m_i \quad l \in \Lambda \tag{4.19}$$

Model (4.1) – (4.4), (4.13) – (4.19) can be solved directly using a MILP solver, allowing for the quick computation of an upper bound on the optimal solution value.

4.5 Genetic Algorithm Solution Approach

As previously mentioned, a key requirement for our collaboration with TASI was to avoid using solver-based techniques. This requirement stemmed from TASI's intention to adopt the proposed approach for various satellite classes and different downlink policies, which may not always be compatible with a concise and efficient mathematical formulation. To address this challenge, we chose to employ a custom heuristic technique.

To solve the PLATiNO planning and scheduling problem, we propose a metaheuristic based on the genetic algorithm (GA) paradigm. Initially, an initial population is generated using a randomized constructive heuristic. Subsequently, the standard GA phases of selection, crossover, and mutation are executed until a termination criterion is met, such as a predefined number of iterations or a time limit. Since crossover and mutation may yield solutions that violate one or more constraints, we

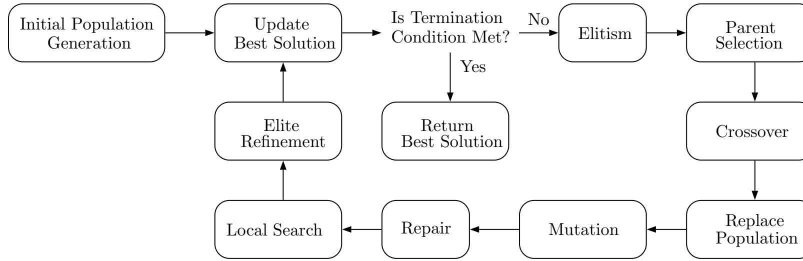


Figure 4.2: High-level structure of the proposed genetic algorithm.

employ a set of repair procedures to ensure the feasibility of the solutions. Additionally, we enhance the GA by incorporating local search procedures designed to increase the likelihood of discovering local optima corresponding to high-quality solutions.

The high-level structure of our proposed approach is illustrated in Figure 4.2, and the subsequent sections provide detailed descriptions of each step.

4.5.1 Individual Representation

Each individual of the population, encoding a problem solution, consists of the sequence of planned DTOs sorted according to their starting time. A plan does not contain DLOs or PAWs but implicitly encodes them in the time intervals between planned DTOs. Downlink activities are scheduled whenever enough time is available during a DLO window, keeping into account the satellite maneuvers. More details about the DLO scheduling are given in Section 4.5.9. Finally, the fitness of a plan is defined by the sum of the priorities of served ARs.

4.5.2 Initial Population Generation

The initial population is created by using a simple greedy randomized constructive heuristic able to generate an arbitrary number of (probably) unique initial feasible plans. More precisely, each plan is generated by the following steps:

1. DTOs are randomly shuffled.
2. DTOs are considered, one at a time, in the order resulting from the shuffling. A DTO is added to the plan if the following joint conditions are satisfied:
   - it serves an AR not already served by the current plan;
   - adding the DTO does not introduce a time conflict with any other DTO in the plan.
3. Once all DTOs have been considered, a possibly overfilled plan, yet satisfying the time conflict constraint, has been generated.
4. In order to obtain a plan that is feasible with respect to the remaining constraints, repair procedures (later described in Section 4.5.7) are applied. More precisely, these procedures are executed sequentially by checking and, in case of violations, fixing first the operational profile, and then the memory constraint.

The above procedure is executed $\bar{n}$ times, then the $|P|$ plans with the highest fitness value are selected to be the initial solutions for population P.

4.5.3 Elitism

Elitism, initially introduced in De Jong (1975), is a simple technique that consists of retaining a fraction of the best-fit plans within subsequent generations so that they are not lost (and thus not selected during the parent selection phase) due to the application of crossover or mutation. In our implementation, just before the parent selection phase, we replace the worst plan in the population with the best (or elite) plan found up to the current iteration.

4.5.4 Parent Selection

The selection of pairs of plans, called *parents*, that is used in the subsequent crossover phase, follows a variation of the classical fitness-proportional (or roulette wheel) selection method. Both parents are randomly selected, however, the probability p_i of selecting plan i as the first parent is defined as $p_i = f_i / \sum_{j \in P} f_j$ where f_i denotes the fitness, equal to the objective function, of plan i and P is the set of plans defining the population. The second parent is, instead, uniformly randomly selected. Because the crossover (described in Section 4.5.5) generates a single offspring from a pair of parents, a number of pairs equal to the number of individuals in the population are selected.

4.5.5 Crossover and Elite Refinement

The crossover design is crucial in tightly constrained problems. A completely random crossover would in fact produce strongly infeasible offspring (i.e., new plans derived from already existing ones), exacerbating the effects of repair procedures, which would recover solutions feasibility at the expense of a heavy perturbation, thus possibly wasting most of the potential advantages given by the recombination of parents. For this reason, we designed two crossover variants, called *multi-point* and *local-search* crossovers, that both preserve at least the time conflict feasibility of the generated offspring and ensure each AR is served by a single DTO. This way, the new offspring, being much closer to feasibility, are less perturbed by the application of the repair procedures. The two crossovers are applied in distinct phases of the algorithm, to better exploit their characteristics. During each generation, the multi-point crossover is applied with probability $p_c = 0.90$ to the pair of parents chosen by the parent selection procedure. Once all pairs have been processed, the population is entirely replaced by the newly generated offspring. When for a pair of parents the crossover is not applied, they are simply copied unchanged into the new population. The local search crossover, instead, is applied at the end of each generation during the elite refinement procedure to further enhance the elite plan. The following paragraphs provide a detailed description of both procedures.

Multi-point crossover The multi-point crossover combines two parents to generate a new plan. The resulting plan is defined by crossing multiple times parents in such a way as to ensure that each AR is not served more than once and there is no time conflict constraint violation.

The procedure starts by setting two indices, i_1 and i_2, to the beginning of the first and second parent, respectively, and by creating a new empty plan $\mathcal{P}$. A value n is randomly chosen such that, on average, the new plan $\mathcal{P}$ is obtained by crossing the parents a desired number of times. Next, the sequences of DTOs of the first parent indexed between i_1 and $i_1 + n - 1$ included and of the second parent indexed between i_2 and $i_2 + m - 1$ are compared. The value $i_2 + m - 1$ is the index of the last DTO of the second parent having a finish time smaller than the finish time of DTO indexed by $i_1 + n$ in the first parent. A fitness value is computed for each sequence by simulating their insertion into plan $\mathcal{P}$. During this procedure, DTOs serving an already served AR or violating the time conflict constraint are removed from their respective sequence. Finally, the resulting sequence with the largest fitness value is inserted into $\mathcal{P}$. Indices are updated to $i_1 = i_1 + n$ and $i_2 = i_2 + m$ and they are further increased until they both point to the first DTO, in the respective parent, that does not conflict with the current last DTO of plan $\mathcal{P}$, taking into account also the maneuvers time. Once both parents have been scanned, a newly created plan $\mathcal{P}$, possibly violating the operational profile and memory constraints, has been created.

Local-search crossover As the multi-point crossover, the local-search one combines two parents to generate a single offspring. The procedure can be seen as a sort of local search operator generating a new plan resulting from the possible improvement of the first parent by using the second as a source of sequences of DTOs.

First, a list of common DTOs found in both parents is created. This list induces a partitioning of both parents into consecutive intervals between two common DTOs, called segments. For each interval, the fitness of the associated segment is computed. Whenever the second parent has a segment having a better fitness than the original one, the former is used to replace the latter, thus improving the overall fitness of the first parent. Replacing a feasible sequence of the first parent with a feasible sequence of the second one preserves the time conflict feasibility, however, as for the multi-point

crossover, no check is done regarding the memory and operational profile constraints, which are then repaired at the end of the procedure in order to make the newly generated plan feasible.

Because this particular crossover is likely to create a copy of the best between the two plans, thus drastically reducing the population diversity, it is not well suited as a crossover step during the standard genetic algorithm phases. In the proposed algorithm, we use the local-search crossover in the elite refinement phase (see Figure 4.2) with the aim of further improving the current elite plan by accumulating in it the best DTO-sequences available in the current population. Let $\mathcal{E}$ be the elite plan, the overall elite refinement phase, inspired by the genetic algorithm proposed by Nagata and Kobayashi (2013), is defined as follows.

1. First, plans in the population are randomly shuffled.
2. For each plan $\mathcal{P} \in P$ (considered in the order defined by the first step):
   (a) The local-search crossover is applied between the elite plan $\mathcal{E}$ and $\mathcal{P}$ resulting in a candidate elite plan $\hat{\mathcal{E}}$.
   (b) Repair procedures are called on $\hat{\mathcal{E}}$.
   (c) The candidate elite plan $\hat{\mathcal{E}}$ is accepted if it is better than the elite plan $\mathcal{E}$.

The procedure is performed during every generation of the genetic algorithm, and, for performance reasons, only one iteration over the entire population is executed.

4.5.6 Mutation

Given a plan $\mathcal{P}$, to avoid the premature convergence of the algorithm, a mutation is applied to each individual in the population with probability $p_m = 0.45$. The mutation randomly adds a number m of DTOs serving ARs not already acquired with m equal to the nearest integer to $f_m \cdot |\mathcal{P}|$. The value of $f_m = 0.04$ is used to scale the mutation intensity to the current plan length $|\mathcal{P}|$. To increase the level of diversification, the procedure does not check whether inserting a given DTO violates the time conflict, operational profile, or memory constraints.

4.5.7 Infeasibility Repair Procedures

A repair procedure is associated with each problem constraint. When a plan violating a given constraint is detected, the corresponding repair procedure performs some changes to restore the feasibility, possibly worsening the plan's fitness. We next describe for each constraint how the associated repair procedure restores the plan feasibility.

Time conflicts These constraints are local to every pair of consecutive DTOs, thus, enforcing the plan feasibility after every incremental change can be done in constant time, as during the crossover application. However, being the mutation-free to add DTOs possibly violating the current constraint, a repair procedure is needed. The repair procedure iterates through the plan and, whenever a pair of consecutive conflicting DTOs is found, it removes the second one.

Operational profile Whenever an infeasible orbit time window is found (i.e., a time window corresponding to the duration of an orbit in which the number of planned DTOs is greater than the predetermined limit C_{max}), the procedure randomly selects a DTO from such time-window and removes it. This operation is repeated until the orbit becomes feasible.

Memory The satellite memory occupation is computed using the algorithm described in Section 4.5.9, taking into account constraints introduced in Section 4.3. The satellite storage occupation is then checked before every planned downlink activity (thus potentially multiple times during the same DLO window). Whenever the memory occupation exceeds the memory capacity, the plan feasibility is restored by randomly removing DTOs that are currently stored, until the memory occupation becomes feasible. Note that, since the downlink is computed heuristically, even the removal of a DTO may cause a previously feasible plan to become infeasible, since the downlink order may change. As a consequence, the memory repair procedure must be performed as the final repair procedure, and it must be repeated until a complete scan of the plan does not report any memory

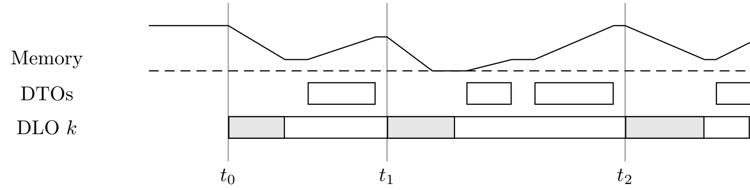


Figure 4.3: Memory saturation representation. Given the DLO l, the satellite memory saturation is computed before the start time of planned downlink activities (grayed areas) happening at times t_0, t_1 and t_2.

violation (i.e., no DTOs have been removed by the current application of the memory repair procedure), which, however, happens rarely.

4.5.8 Local Search

We designed two local search procedures, denoted as SWAP01 and SWAP11, aimed at improving the fitness of a given feasible plan. Both procedures perform simple and localized changes that preserve the feasibility of the current plan. In particular, SWAP01 tries to find a spot in the plan where a DTO can be inserted, while SWAP11 tries to replace a planned DTO with another one that is not planned, provided that this change improves the plan fitness. Both procedures follow a greedy first-improvement strategy in which the first locally feasible choice found (see the fast memory check below) is immediately implemented, and whenever the resulting plan is globally feasible, the change is accepted.

Fast (local) memory check Due to the downlink policy, evaluating whether a local change in a plan violates the memory constraint cannot be accomplished in constant time. We thus introduced the concept of *memory saturation* to heuristically filter out changes that may likely violate the memory constraint. More precisely, we define with *satellite memory saturation*, sat_t, the satellite memory occupation induced by the sequence of acquisitions and downlink activities that happened up to time t included. In particular, for a given plan, we are interested in the subset of times t corresponding to the beginning of planned downlink activities. As an example, in Figure 4.3, the values of sat_{t_0}, sat_{t_1} and sat_{t_2}—identifying the satellite memory usage before performing the associated downlink activities—can be used to quickly reject changes occurring in the interval between two planned downlink start times. Before executing any local search procedure, these values can be computed during a preprocessing phase in which we linearly scan the plan and store the memory occupation before the beginning of any planned DLO (see Section 4.5.9).

Before performing any change identified by a local search move, values sat_t, with t appropriately selected according to the specific change the move is currently evaluating, are checked for a quick rejection test.

As an example, consider a SWAP01 application that is trying to insert a candidate DTO i having s_i and f_i as start and finish time, respectively. A time t is identified as the time associated with the first planned downlink activity after f_i. This can be efficiently done by means of a binary search, provided that downlink activities are sorted according to their start times. Whenever $m_i + sat_t > M_{tot}$, the change is immediately rejected.

Note that, even if the application is locally feasible (and thus not rejected by the fast memory check), because of the complex downlink requirements, it may result in a globally infeasible plan (e.g., the memory storage in a subsequent part of the plan may be exceeded due to this change). Thus, in case of acceptance, the whole plan must be checked against the memory constraint.

Similar reasoning can be done for the SWAP11 operator in which, however, the memory m_j of the replaced DTO j can be subtracted from sat_t whenever the candidate DTO i and the removed one j share the same downlink activity starting at time t. The local memory rejection test then becomes $m_i - m_k + sat_t > M_{tot}$.

Finally, once a change to the plan is accepted, memory saturation values must be recomputed.

Details of SWAP01

The execution of a SWAP01 move is decomposed into two sequential steps called insertion and backtracking. During the insertion step, a number of candidate DTOs satisfying the above-described fast memory check as well as the time conflicts and operational profile constraints are inserted into the plan with the aim of making it as full as possible. Candidate DTOs are identified by considering ARs not already served in the plan under examination. The ARs order is randomized before each operator application. Then, in the second backtracking step, the resulting plan is evaluated with respect to the standard memory constraint. As long as a violation of this constraint is identified, a randomly selected DTO, chosen among those previously inserted in the first step, is removed. The two-step decomposition allows for a more efficient procedure execution in which the number of times the time-consuming memory constraint is evaluated is kept as low as possible.

Details of SWAP11

The SWAP11 operator considers replacing any planned DTO with a not planned one that has a better fitness (i.e., priority) and fits—with respect to the satellite maneuvers—in the position of the replaced one. To speed up the search, for each planned DTO, we can limit the set of candidate DTOs to those time-conflicting with the planned one. This list can be composed once during a preprocessing step.

Given a planned DTO we are trying to replace, the list of its time-conflicting DTOs is considered to identify a candidate DTO that satisfies the fast memory check, and the operational profile constraint and does not conflict with the previous and next planned DTO. If a candidate DTO satisfies the above constraints and better fitness is found, the planned DTO is replaced with the candidate one, and the memory constraint is evaluated over the complete plan. In case of violations, the change is undone and the search continues.

Because checking whether replacing all possible planned DTOs can be very time-consuming, we implemented a simple yet effective technique to improve the efficiency of the SWAP11 procedure by performing a heuristic pruning of its neighborhood. The technique, which is inspired by the Selective Vertex Caching described in Accorsi and Vigo (2021), makes each plan aware of the plan points that have been edited during the current GA generation by storing nearby planned DTOs into a cache. In particular, each DTO that is added to the plan is also added to the cache, while when a DTO is removed the previous and next planned DTOs are added to the cache. The SWAP11 procedure, instead of considering all planned DTOs, considers only the DTOs found in the cache as well as their successors and predecessors. The cache of plans in the population is cleared at the beginning of each generation so as to focus the SWAP11 application on the limited plan points that were recently changed during the current generation.

4.5.9 Indirect Dynamic Downlink Scheduling Algorithm

As discussed in Section 4.5.1, the representation of downlink activities does not involve a direct encoding within an individual plan. Instead, it is derived as an outcome of the specific set of planned DTOs. To achieve this, a deterministic algorithm called ITERATEOVERMEMORY (IOM) is employed. This algorithm computes the memory usage pattern associated with a given plan and, whenever a DLO provides sufficient time, it schedules the downlink of stored acquisitions through the DOWNLINKDTOS routine. Since this operation is performed for each available downlink interval, it is essential that the algorithm remains lightweight to prevent introducing significant computational overhead whenever downlink scheduling is determined. Moreover, this approach serves as a customization point that facilitates the adaptation of downlink policies. It effectively separates the planning logic for acquisition scheduling from the subsequent transmission to Earth.

The objective here is to select a subset of currently stored acquisitions, optimizing the release of memory within the constraints of the downlink time capacity and the discretization requirements outlined in Section 4.3. This problem results to be an instance of 0-1 Knapsack Problem (KP). Given the heuristic context, we solve these KP instances using the greedy heuristic proposed by Dantzig Dantzig (1957). In this heuristic, items are sorted based on their profit-to-weight ratio, and the knapsack is filled by including the sorted elements until the capacity is reached.

It's worth noting that employing a heuristic approach for downlink scheduling may potentially prevent the overall algorithm from finding the optimal solution, particularly if achieving optimality in downlink scheduling is critical. However, our computational tests, as detailed in Section 4.8, indicate that the genetic algorithm consistently identifies near-optimal solutions despite the suboptimal nature of the downlink scheduling process.

The algorithm receives in input the plan $\mathcal{P}$, a list of DLOs $\mathcal{D}$, sorted by increasing start time, and the initial satellite memory state $\mathbb{M}$, which may contain acquisitions performed during a previous planning activity that still need to be downlinked. Algorithm 6 shows the high-level pseudocode of procedure IOM. The algorithm iterates over both the input plan $\mathcal{P}$ and the sorted list of DLOs $\mathcal{D}$, emulating the acquisitions made and the memory occupation through the plan. When a downlink operation can be executed, the DOWNLINKDTOS routine is invoked to identify which stored acquisitions are downlinked and, thus, removed from the satellite's internal storage. For the sake of clarity, technical details, not relevant to the understanding of the general idea behind the algorithm, have been omitted (e.g., the maneuvers between DLOs and DTOs).

Algorithm IOM is among the core components of the proposed approach. It is used by all functionalities interacting with the satellite memory and downlink activities. In particular, it is used to compute the memory saturation used for the fast memory check (Section 4.5.8) as well as to check and repair the memory constraint (Section 4.5.7).

Algorithm 6: High-level pseudocode for the IOM algorithm.

```
1  ITERATEOVERMEMORY (P, D, M);
   Input: A plan P, a list of DLOs D sorted by start time, the initial satellite memory state M
2  begin
3      i ← j ← 0;
4      while (i < |P|) ∨ (j < |D|) do
5          if j < |D| then
6              if i < |P| then
7                  dl_stop ← min{φ_j, f_{P_i}};
8              else
9                  dl_stop ← φ_j;
10             end
11             if i > 0 then
12                 dl_start ← max{σ_j, s_{π_{i−1}}};
13             else
14                 dl_start ← σ_j;
15             end
16             if (dl_start < φ_j) ∧ (dl_stop > σ_j) then
17                 PREDLOFUNC(P, M);
18                 DOWNLINKDTOS(M, dl_start, dl_stop);
19             end
20         end
21         if (j = |D|) ∨ ((i < |P|) ∧ (f_{π_{i−1}} < φ_j)) then
22             STOREDTO(M, P_i);
23             POSTDTOFUNC(P, M);
24             i ← i + 1;
25         else
26             j ← j + 1;
27         end
28     end
29     PREENDINGFUNC(P, M)
30 end
```

We denote by $\mathcal{P}_i$ the DTO in the position i of plan $\mathcal{P}$. Moreover, dl_{start} and dl_{stop} represent, for each potential downlink activity, the start and stop time of communications with ground station antennas. These values are defined by considering the intervals between each pair of planned DTOs occurring during a DLO, and by assessing whether the interval can be used to downlink some stored acquisition. Note that algorithm IOM assumes that DLOs visibility cones (and thus their associated start and finish times) do not overlap with each other. Such an assumption is fully compatible with the problem requirements defined by TASI.

The function DOWNLINKDTOS performs the downlink scheduling given the current satellite memory state $\mathbb{M}$ and the downlink activity time interval $[dl_{start}, dl_{stop}]$. After its execution $\mathbb{M}$ is freed

from the downlinked acquisitions. On the other hand, the function StoreDTO manages newly encountered acquisitions by storing them into $\mathbb{M}$.

IOM is used to accomplish three different operations: (i) the computation of the memory saturation, (ii) the feasibility test, and (iii) the repairing of the memory constraint. To this extent, we have inserted three customization points called PreDLOFunc, PostDLOFunc, and PreEndingFunc, which are employed in the following way:

- PreDLOFunc is called before starting a potential downlink activity at time $t = dl_{start}$, which may eventually result in the decrease of the satellite memory occupation. It is used by (iii) the memory repair procedure, after having checked whether the memory occupation is greater than M_{tot}. Or by (i) the computation of the satellite memory saturation, where value sat_t is defined to be equal to the current memory occupation.
- PostDLOFunc is used during (ii) the memory constraint check (e.g., during local search procedures) to quickly return as soon as an infeasibility is detected.
- PreEndingFunc performs the same task as PreDLOFunc (i.e., operations (i) and (iii)), but at the end of the plan, using M_{fin} instead of M_{tot} as satellite memory capacity.

4.6 Exact and Relaxed Formulation Approaches

In addition to the development of additional heuristic algorithms, as discussed in Section 4.7, we have considered calculating an upper bound on the optimal solution using both relaxed and exact formulations to evaluate the quality of the solutions generated by the proposed Genetic Algorithm planner.

We have considered four formulations in total:

- *No-Memory Relaxation:* Defined by constraints (4.1) – (4.4), this is the quickest formulation to compute. When it returns a feasible solution, we can conclude that, for that specific instance, the memory aspect can be disregarded.
- *Continuous Memory Relaxation:* Defined by constraints (4.1) – (4.4) and (4.13) – (4.19), this lightweight formulation is often able to provide a tight bound due to its partial consideration of memory constraints.
- *Discrete Memory Exact:* Defined by constraints (4.1) – (4.12), this exact formulation exhibits a weak linear relaxation, making it less effective with memory-intensive instances.
- *Strengthened Discrete Memory Exact:* Combining constraints (4.1) – (4.19), this formulation merges the strengths of the second and third formulations to enhance the Discrete Memory model with the tighter linear relaxation of the Continuous Memory formulation, making it able to often find the optimal solution value or the best bound with most instances.

The last two formulations were also tested as heuristic algorithms in Section 4.8 to evaluate the performance of modern commercial MILP solvers when used as heuristics.

4.7 Math-Heuristic Competitors

As stated in Section 4.1, the main output of the collaboration with TASI has been the design and development of an efficient solver-free prototype that could obtain near-optimal solutions within the short time limit of five minutes. To this end, most of the effort of this work has gone into the development of the Genetic Algorithm approach described in Section 4.5 and here tested against other competitors' algorithms obtained by instead enabling the use of MILP solvers. What has been clear from the preliminary phases of the development has been that, once we neglect the memory aspects, commercial MILP solvers like CPLEX are very good at solving the satellite daily planning at optimality. In search of a way to exploit such proficiency even when memory constraints are considered, we implemented 3 alternative heuristic techniques hereafter described.

4.7.1 Relax and Repair Heuristic

To ensure efficient scalability, the hybridized technique focuses solely on utilizing the relaxed formulation discussed in Section 4.4.3. Specifically, we have developed a standalone math-heuristic, which we named Relax and Repair (RR), based on the relaxed formulation (4.1) – (4.4) and (4.13) – (4.19).

As the name implies, RR takes advantage of the effectiveness of commercial MILP solvers to rapidly generate a diverse set of solutions while exploring the Branch and Bound tree. When one of these solutions is found, a callback function is invoked to assess its feasibility. If the solution is found feasible, it is retained as it is; otherwise, it undergoes the repair procedure outlined in Section 4.5.7. The repaired solution is then returned as a primal-heuristic solution, while the original infeasible solution is rejected.

Since both the feasibility test and the repair procedures rely on the heuristic downlinking policy described in Section 4.5.9, rejecting solutions found infeasible by these heuristic criteria invalidates the bound naturally produced by the MILP solver. Consequently, this straightforward technique can produce feasible solutions, while the resulting bound remains equivalent to the one produced before the introduction of no-good constraints that prevent maybe-infeasible solutions.

4.7.2 Strenghtened Discrete-Memory Relaxation

One of the most straightforward ways to approach the problem once we have set up a MILP formulation is to utilize commercial solvers with a time limit to obtain heuristic solutions. These solvers are equipped with heuristic algorithms designed to rapidly generate near-optimal solutions, leveraging the information provided by the MILP framework.

In this context, we selected the better-performing of our two exact formulations and directed the optimization to run with a single thread, stopping after a time limit of five minutes. This setup allows for comparison with other heuristic techniques.

This approach offers several advantages, including the potential to find the optimal solution and halt the optimization before the time limit is reached. Moreover, if the optimal solution is not found within five minutes, this approach still provides a valid upper bound, which offers valuable insights into the solution quality. However, there are drawbacks to this method. First, when the time limit is reached, there is no guarantee that the internal heuristics may yield competitive results. It is indeed possible that the returned solution is of much lower quality compared to other problem-specific techniques. Then, the use of a MILP model comes with scalability challenges, making it unsuitable for handling large-scale instances due to memory requirements and extended optimization times.

4.7.3 Hybrid Genetic Algorithm - Relax and Repair Heuristic

To leverage the strengths of a rapid math-heuristic, capable of efficiently generating numerous solutions within a short computational time, and a Genetic Algorithm approach that can be initialized by high-quality individuals, we propose a straightforward hybridization strategy. This approach involves splitting the available time limit into two equal halves: during the first half, the RR algorithm is executed, providing the resulting solutions as the initial individuals for the subsequent Genetic Algorithm run.

This method brings two key advantages: (i) The upper limit we get from the mathematical model (before adding no-good constraints) lets us stop the heuristic search when the optimization gap becomes small enough. (ii) The RR solutions usually contain sequences of DTOs of excellent quality. This makes the heuristic search more effective, helping it quickly find high-quality solutions.

4.8 Computational Experiments

The computational testing had the objective of assessing the performance of the proposed solution approach. To this extent, the algorithm was tested on a dataset of instances derived from real-world data realized by the TASI team and tailored to specifically stress the algorithm with tight constraints

and a wide range of instance sizes. Moreover, the results are validated against the four formulations listed in Section 4.6.

4.8.1 Implementation and Experimental Environment

The proposed algorithm was implemented in C++ and compiled using `GCC 11.4`. The experiments were performed on a 64-bit desktop computer with an AMD Ryzen 9 5950X CPU, with 64 GB of RAM. We used CPLEX 12.10 CPLEX (2019) to solve the four MILP formulations. Regarding the bound computation, for all the four formulations considered, CPLEX has been run without restrictions on the number of threads (which for our machine are a maximum of 32 logical threads and 16 physical cores) and with a time limit of one hour. For the heuristic algorithms testing, all the runs were executed using a single thread with a time limit of five minutes, in accordance with TASI requirements listed in Section 4.1. Finally, since the proposed algorithm contains randomized components and the runs are time-based, for each instance, we performed 10 runs using seeds from 1 to 10.

4.8.2 Parameters Definition and Tuning

The algorithm makes use of a number of parameters ruling its behavior. They are described in the following list, along with the adopted values:

- $|P| = 70$, population size;
- $\bar{n} = 2000$, number of plans generated during the initial population generation (see Section 4.5.2);
- $p_\ell = 0.25$, probability of applying the local search;
- $p_c = 0.90$, probability of applying the multi-point crossover;
- $f_c = 0.77$, parameter determining the number of crossing points used in the multi-point crossover as a fraction of the length of the plans involved. In particular, the number of crossing points used when performing a multi-point crossover application between plans $\mathcal{P}_1$ and $\mathcal{P}_2$ is the nearest integer to $f_c \cdot \min\{|\mathcal{P}_1|, |\mathcal{P}_2|\}$;
- $p_m = 0.45$, probability of applying the mutation;
- $f_m = 0.04$, parameter determining the number of mutations as a fraction of the length of the plan. In particular, the number of mutations applied to a plan $\mathcal{P}$ is the nearest integer to $f_m \cdot |\mathcal{P}|$.

The parameters have been tuned by using a Bayesian optimization (BO) approach (see Bergstra et al. (2015)) applied on preliminary set of instances for a total number of 100 attempts. Moreover, during the BO, parameters could have values taken from the following uniform discrete distributions:

- $|P|$ in $[10, 100]$ with a step of 10;
- $\bar{n}$ in $[1000, 10000]$ with a step of 1000;
- p_ℓ in $[0.00, 1.00]$ with a step of 0.05;
- p_c in $[0.00, 1.00]$ with a step of 0.05;
- f_c in $[0.01, 1.00]$ with a step of 0.01;
- p_m in $[0.00, 1.00]$ with a step of 0.05;
- f_m in $[0.01, 1.00]$ with a step of 0.01.

The resulting tuning was then used during the solution of all instances.

4.8.3 Instances Description

Several benchmark instances have been provided by TASI to test the proposed GA approach. In particular, the instances aim at modeling potential scenarios occurring during a realistic daily planning

activity performed by TASI, ranging from "lighter" days where the satellite might be left idle for a good portion of the plan, to more intensive days where the strategic choice regarding the acquisitions to take is of key importance. Moreover, some of them also have a particular focus on stressing the several constraints of the problem in order to test the overall robustness of the algorithm (so that it may be applied to similar satellites with minor modifications). More specifically, in the following list we detail the distributions for the main parameters.

- The planning horizon duration H takes a value between 3, 6, 9, 12, 18, and 24 hours.
- The number of ARs ranges from 250 to 3500 to model days with few and several requests. The number of DTOs is highly correlated to the number of ARs. In particular, given the satellite trajectory and the planning horizon, the instances have an average of about 2 DTOs per AR. Note that DTOs overlapping with PAWs are filtered out.
- The memory occupation of stored acquisitions is defined according to one of the following uniform distributions: $U(60, 120)$ (large requests, small variability), $U(3, 120)$ (large variability), $U(3, 30)$ (small requests, large variability), $U(15, 30)$ (small requests, small variability), and $0.8 \cdot U(3, 30) + 0.2 \cdot U(60, 120)$ (several small requests, and few large requests).
- A total number of three ground stations for the downlink of acquisitions were considered. They are located at Matera (Italy), Troll (Antarctica), and Svalbard (Norway). Note, however, that not all ground stations are always made available to all instances.
- We consider scenarios in which the satellite starts with an initial memory occupation ranging from 0% to 100%. By consequence, in order for the empty plan to be feasible, the final memory occupation is selected in the range 0% to 100% to be at least equal to the initial memory occupation.
- The are three classes of memory constraint tightness: normal, tight, and very tight. According to the selected class, the satellite memory is defined in order to contain a maximum fraction of the total number of ARs.
- Similarly, for the operational capacity there are three classes (normal, tight, and very tight) ruling the maximum number of acquisitions that can be performed during a satellite orbit.
- Finally, the priority of ARs has been selected by TASI to emulate real-world scenarios.

The dataset contains 80 instances, 20 of which are large-scale, with a number of ARs greater than 1000. These large-scale instances, while not adhering to real-world scenarios for the PLATiNO-1 satellite, are considerably more common for other classes of satellites, and for this reason, they have been provided in the test bed.

Tables 4.1 and 4.2 provide an overview of the primary characteristics of the medium-sized and larger datasets respectively.

4.8.4 Upper Bound computation

We have computed a valid upper bound for all instances in the test bed, and the results are summarized in Table 4.3 and 4.4. To measure the relative gaps, we use the formula:

$$\text{Relative Gap} = 100 \cdot \frac{(UB - LB)}{UB}$$

Here, UB represents the upper bound derived from the considered formulation, and LB is the best-known feasible solution obtained through any heuristic or formulation.

Table 4.3 presents several formulations for comparison. The first formulation, without memory considerations, is included to identify cases where memory constraints are not critical, and this simpler relaxation can find feasible solutions. The second model, the continuous memory relaxation, partially models memory downlink. It serves two key purposes: (i) it computes a valid upper bound for all instances, even the large-scale ones, and (ii) it provides insights into potential improvements if partial downlinking of acquisitions were feasible.

Additionally, we include the discrete-memory exact formulation, which, while valid, features a weaker linear relaxation that hinders optimization speed. When it fails to reach the optimum, it generates a less robust upper bound, often weaker than the faster continuous memory relaxation.

Finally, the strengthened discrete-memory approach combines the best aspects of the previous two methods, making it the overall top performer on average. It's worth noting that, in some cases, the continuous-memory approach might produce a slightly better upper limit than the strengthened discrete-memory approach when

Table 4.1: Summary of medium instances and their key characteristics. Please note that M_{init} and M_{fin} are expressed as percentages of the total memory capacity, M_{tot}.

Instance	Best Bound	Best Sol	Gap	$\|A\|$	$\|D\|$	$\|\Lambda\|$	H	M_{tot}	M_{init}%	M_{fin}%	C_{max}
day0.12_250_0	833	833	0.00	250	421	0	3	277	17	64	2000
day0.12_250_1	4502	4502	0.00	250	423	5	3	147	14	67	133
day0.12_250_2	6065	6065	0.00	250	412	2	3	1597	12	58	13
day0.12_250_3	3175	3175	0.00	250	419	1	3	147	43	76	13
day0.12_250_4	7964	7964	0.00	250	427	5	3	417	0	100	13
day0.25_250_0	2514	2514	0.00	250	430	0	6	2990	14	74	67
day0.25_250_1	7764	7764	0.00	250	427	5	6	513	0	100	7
day0.25_250_2	927	949	2.37	250	432	5	6	30	0	100	67
day0.25_250_3	2560	2560	0.00	250	426	1	6	215	34	36	7
day0.25_250_4	2818	2818	0.00	250	427	0	6	152	10	79	67
day1_250_0	4917	4917	0.00	250	510	15	24	133	13	25	250
day1_250_1	4008	4008	0.00	250	488	4	24	282	62	63	17
day1_250_2	2383	2383	0.00	250	504	14	24	14	16	44	2
day1_250_3	5507	5507	0.00	250	494	26	24	19	19	65	2
day1_250_4	1900	1900	0.00	250	521	15	24	39	0	73	2
day0.25_500_5	3099	3099	0.00	500	833	9	6	32	36	42	13
day0.25_500_6	5989	5989	0.00	500	837	4	6	108	19	19	13
day0.25_500_7	15	15	0.00	500	869	0	6	2176	66	66	2000
day0.25_500_8	10037	10037	0.00	500	831	5	6	44	29	46	2000
day0.25_500_9	3181	3272	2.86	500	849	6	6	684	29	42	133
day0.5_500_5	1976	2023	2.38	500	861	10	12	119	9	15	7
day0.5_500_6	6952	6953	0.01	500	866	5	12	149	23	23	67
day0.5_500_7	7078	7147	0.97	500	876	10	12	151	10	40	1000
day0.5_500_8	3090	3090	0.00	500	850	2	12	2998	33	57	7
day0.5_500_9	1800	1800	0.00	500	831	8	12	103	34	38	1000
day1_500_5	2843	2843	0.00	500	975	15	24	398	30	30	33
day1_500_6	5587	5587	0.00	500	989	3	24	632	43	44	3
day1_500_7	5690	5703	0.23	500	998	3	24	357	28	29	500
day1_500_8	3145	3145	0.00	500	956	13	24	85	41	53	3
day1_500_9	2818	2869	1.81	500	992	16	24	74	34	43	3
day0.38_750_10	8897	8897	0.00	750	1277	0	9	512	60	77	2000
day0.38_750_11	10289	10364	0.73	750	1270	11	9	202	22	24	133
day0.38_750_12	11150	11150	0.00	750	1285	4	9	156	63	63	13
day0.38_750_13	3665	3665	0.00	750	1276	2	9	4428	19	53	133
day0.38_750_14	3787	3931	3.80	750	1267	2	9	306	17	17	13
day0.75_750_10	5756	5783	0.47	750	1296	18	18	346	0	100	7
day0.75_750_11	7069	7076	0.10	750	1418	4	18	83	9	27	67
day0.75_750_12	9120	9634	5.64	750	1419	17	18	131	33	34	1000
day0.75_750_13	12767	12912	1.14	750	1365	3	18	411	41	42	7
day0.75_750_14	12453	12453	0.00	750	1361	20	18	143	31	31	7
day1_750_10	755	756	0.13	750	1495	4	24	111	29	30	50
day1_750_11	10836	10993	1.45	750	1433	15	24	42	14	69	750
day1_750_12	4348	4364	0.37	750	1476	4	24	297	74	75	50
day1_750_13	1054	1054	0.00	750	1467	4	24	406	39	39	50
day1_750_14	4593	4786	4.20	750	1494	26	24	47	17	19	50
day0.5_1000_15	1779	1912	7.48	1000	1798	8	12	1070	99	100	2000
day0.5_1000_16	12119	12120	0.01	1000	1676	6	12	98	12	15	133
day0.5_1000_17	5851	5851	0.00	1000	1670	6	12	1971	0	100	13
day0.5_1000_18	3849	3849	0.00	1000	1757	1	12	2374	17	17	2000
day0.5_1000_19	18529	18588	0.32	1000	1769	9	12	176	55	69	13
day1_1000_15	13302	13345	0.32	1000	1943	14	24	43	55	55	67
day1_1000_15	6405	6405	0.00	1000	1974	16	24	37	42	69	7
day1_1000_16	857	857	0.00	1000	1954	4	24	589	29	30	7
day1_1000_16	2368	2368	0.00	1000	1973	27	24	82	34	63	7
day1_1000_17	5943	5943	0.00	1000	1952	2	24	420	14	16	67
day1_1000_17	2576	2665	3.45	1000	1961	17	24	35	28	33	67
day1_1000_18	6706	6706	0.00	1000	1928	27	24	81	18	57	7
day1_1000_18	17700	17704	0.02	1000	1978	4	24	739	23	50	000

Instance	Best Bound	Best Sol	Gap	$\lvert A \rvert$	$\lvert D \rvert$	$\lvert \Lambda \rvert$	H	M_{tot}	M_{init}%	M_{fin}%	C_{max}
day1_1000_19	4584	4741	3.42	1000	1949	26	24	143	20	55	67
day1_1000_19	20392	20406	0.07	1000	1979	17	24	25	21	30	7

Table 4.2: Summary of large instances and their key characteristics. Please note that M_{init} and M_{fin} are expressed as percentages of the total memory capacity, M_{tot}.

Instance	**Best Bound**	**Best Sol**	**Gap**	$\lvert \mathbf{A} \rvert$	$\lvert \mathbf{D} \rvert$	$\lvert \mathbf{\Lambda} \rvert$	**H**	$\mathbf{M_{tot}}$	$\mathbf{M_{init}}$%	$\mathbf{M_{fin}}$%	$\mathbf{C_{max}}$
day1_1500_0	11098	11614	4.65	1500	2970	16	24	296	12	12	1500
day1_1500_1	6671	6902	3.46	1500	2972	26	24	686	39	71	1500
day1_1500_2	14459	15397	6.49	1500	2946	27	24	165	36	37	1500
day1_1500_3	21211	21399	0.89	1500	2931	15	24	296	42	42	100
day1_2000_4	18137	19627	8.22	2000	3932	16	24	274	71	72	2000
day1_2000_5	12568	12568	0.00	2000	3903	26	24	561	9	30	13
day1_2000_6	3710	3735	0.67	2000	3935	27	24	284	0	0	13
day1_2000_7	17641	18314	3.81	2000	3890	26	24	63	17	37	133
day1_2500_8	6614	6887	4.13	2500	4875	26	24	227	0	100	2500
day1_2500_9	10066	10334	2.66	2500	4913	13	24	114	99	100	167
day1_2500_10	3049	3198	4.89	2500	4942	15	24	362	35	35	2500
day1_2500_11	38028	38463	1.14	2500	4872	15	24	99	36	65	167
day1_3000_12	59269	59428	0.27	3000	5836	14	24	1637	16	74	200
day1_3000_13	19363	21080	8.87	3000	5918	16	24	408	14	14	200
day1_3000_14	50273	54608	8.62	3000	5819	27	24	796	24	24	3000
day1_3000_15	3526	3670	4.08	3000	5971	14	24	432	61	77	3000
day1_3500_16	5788	5933	2.51	3500	6880	26	24	403	28	61	3500
day1_3500_17	3794	3899	2.77	3500	6854	15	24	374	39	40	23
day1_3500_18	35177	35614	1.24	3500	6819	26	24	1598	71	79	233
day1_3500_19	49758	51364	3.23	3500	6920	14	24	398	23	25	23

it's stopped due to time constraints. This happens because the continuous-memory approach finishes its optimization process, giving a valid, mixed integral upper bound, while the strengthened discrete-memory approach just returns the limit obtained from linear relaxations in the Branch and Bound tree. However, this difference never exceeds 1 unit for the medium-sized test group.

For large-scale instances, as shown in Table 4.4, both exact formulations in most cases are unable to compute valid upper bounds. Therefore, we rely on the relaxed models to establish valid upper bounds for validating the heuristic results.

Table 4.3: Relative percentage gaps computed by CPLEX on the medium dataset with two relaxed and two exact formulations, each within a one-hour time limit and executed with 32 threads. Bold indicates bounds matching the best-known solution (found by heuristics or during CPLEX optimization), and $*$ denotes cases where the time limit was reached.

Instance	**Best Known**		**No-Mem**			**Cont-Mem**			**Discr-Mem**			**Strght-Mem**		
	Sol	**Bound**	**UB**	**Gap%**	**Time**	**UB**	**Gap%**	**Time**	**UB**	**Gap%**	**Time**	**UB**	**Gap%**	**Time**
day0.12_250_0	833	833	**833**	**0.00**	0.20	**833**	**0.00**	0.21	**833**	**0.00**	0.21	**833**	**0.00**	0.22
day0.12_250_1	4502	4502	**4502**	**0.00**	0.23	**4502**	**0.00**	0.27	**4502**	**0.00**	9.99	**4502**	**0.00**	10.21
day0.12_250_2	6065	6065	**6065**	**0.00**	0.31	**6065**	**0.00**	0.35	**6065**	**0.00**	0.41	**6065**	**0.00**	0.42
day0.12_250_3	3175	3175	3217	1.31	0.27	3188	0.41	0.33	**3175**	**0.00**	1352.00	**3175**	**0.00**	26.92
day0.12_250_4	7964	7964	**7964**	**0.00**	0.57	**7964**	**0.00**	0.34	**7964**	**0.00**	10.88	**7964**	**0.00**	10.64
day0.25_250_0	2514	2514	**2514**	**0.00**	0.34	**2514**	**0.00**	0.37	**2514**	**0.00**	0.39	**2514**	**0.00**	0.38
day0.25_250_1	7764	7764	**7764**	**0.00**	0.21	**7764**	**0.00**	0.23	**7764**	**0.00**	1.71	**7764**	**0.00**	1.67
day0.25_250_2	927	949	1348	31.23	0.36	950	2.42	0.75	1073	13.61	3600*	949	2.32	3600*
day0.25_250_3	2560	2560	3802	32.67	0.26	**2560**	**0.00**	0.64	**2560**	**0.00**	0.38	**2560**	**0.00**	0.56
day0.25_250_4	2818	2818	2994	5.88	0.40	**2818**	**0.00**	0.54	**2818**	**0.00**	0.33	**2818**	**0.00**	0.43
day1_250_0	4917	4917	6253	21.37	0.15	5056	2.75	0.94	**4917**	**0.00**	73.56	**4917**	**0.00**	47.09
day1_250_1	4008	4008	6663	39.85	0.20	4064	1.38	0.33	4010	0.05	3600*	**4008**	**0.00**	3.07
day1_250_2	2383	2383	4344	45.14	0.15	2394	0.46	0.58	**2383**	**0.00**	8.44	**2383**	**0.00**	3.52
day1_250_3	5507	5507	**5507**	**0.00**	0.09	**5507**	**0.00**	0.11	**5507**	**0.00**	9.67	**5507**	**0.00**	9.75
day1_250_4	1900	1900	2127	10.67	0.12	**1900**	**0.00**	0.69	**1900**	**0.00**	8.32	**1900**	**0.00**	8.00
day0.25_500_5	3099	3099	3237	4.26	0.95	3105	0.19	1.08	**3099**	**0.00**	165.85	**3099**	**0.00**	163.51
day0.25_500_6	5989	5989	7724	22.46	0.69	5991	0.03	0.84	**5989**	**0.00**	2.36	**5989**	**0.00**	2.57
day0.25_500_7	15	15	2588	99.42	2.17	**15**	**0.00**	0.67	**15**	**0.00**	0.69	**15**	**0.00**	0.69
day0.25_500_8	10037	10037	10472	4.15	0.79	**10037**	**0.00**	1.39	**10037**	**0.00**	1.19	**10037**	**0.00**	1.35
day0.25_500_9	3181	3272	3420	6.99	1.26	3286	3.20	1.78	3420	6.99	3600*	3272	2.78	3600*
day0.5_500_5	1976	2023	4610	57.14	1.04	2027	2.52	2.11	3138	37.03	3600*	2023	2.32	3600*
day0.5_500_6	6952	6953	8350	16.74	0.95	6957	0.07	0.87	7751	10.31	3600*	6953	0.01	3600*
day0.5_500_7	7078	7147	7843	9.75	0.51	7151	1.02	1.44	7763	8.82	3600*	7147	0.97	3600*
day0.5_500_8	3090	3090	**3090**	**0.00**	0.68	**3090**	**0.00**	0.78	**3090**	**0.00**	23.03	**3090**	**0.00**	24.86
day0.5_500_9	1800	1800	4847	62.86	0.84	1818	0.99	0.91	1801	0.06	3600*	**1800**	**0.00**	90.60
day1_500_5	2843	2843	10489	72.90	1.89	2863	0.70	2.52	3354	15.24	3600*	**2843**	**0.00**	391.25
day1_500_6	5587	5587	7665	27.11	0.34	5595	0.14	0.41	5588	0.02	3600*	**5587**	**0.00**	31.77
day1_500_7	5690	5703	11075	48.62	0.72	5808	2.03	0.81	10855	47.58	3600*	5703	0.23	3600*
day1_500_8	3145	3145	**3145**	**0.00**	0.35	**3145**	**0.00**	0.38	**3145**	**0.00**	22.20	**3145**	**0.00**	22.81
day1_500_9	2818	2869	4054	30.49	0.72	2884	2.29	1.09	3098	9.04	3600*	2869	1.78	3600*
day0.38_750_10	8897	8897	10730	17.08	1.17	**8897**	**0.00**	1.20	**8897**	**0.00**	1.22	**8897**	**0.00**	1.23
day0.38_750_11	10289	10364	11308	9.01	1.55	10364	0.72	3.58	11242	8.48	3600*	10364	0.72	3600*
day0.38_750_12	11150	11150	13606	18.05	1.65	11181	0.28	2.33	**11150**	**0.00**	1939.70	**11150**	**0.00**	254.94
day0.38_750_13	3665	3665	**3665**	**0.00**	1.36	**3665**	**0.00**	1.76	**3665**	**0.00**	47.27	**3665**	**0.00**	50.80
day0.38_750_14	3787	3931	4967	23.76	1.97	3940	3.88	2.65	4958	23.62	3600*	3931	3.66	3600*
day0.75_750_10	5756	5783	6512	11.61	0.90	5787	0.54	1.72	6467	10.99	3600*	5783	0.47	3600*
day0.75_750_11	7069	7076	13572	47.91	1.69	7149	1.12	2.97	11785	40.02	3600*	7076	0.10	3600*

Instance	Best Known		No-Mem			Cont-Mem			Discr-Mem			Strght-Mem		
	Sol	Bound	UB	Gap%	Time	UB	Gap%	Time	UB	Gap%	Time	UB	Gap%	Time
day0.75_750_12	9120	9634	10114	9.83	0.95	9634	5.34	1.76	10088	9.60	3600*	9635	5.35	3600*
day0.75_750_13	12767	12912	20149	36.64	5.04	13998	8.79	3.60	18844	32.25	3600*	12912	1.12	3600*
day0.75_750_14	12453	12453	**12453**	**0.00**	0.91	**12453**	**0.00**	1.00	**12453**	**0.00**	85.81	**12453**	**0.00**	87.84
day1_750_10	755	756	5521	86.32	2.78	759	0.53	2.65	2378	68.25	3600*	756	0.13	3600*
day1_750_11	10836	10993	12825	15.51	0.65	11142	2.75	1.70	11031	1.77	3600*	10993	1.43	3600*
day1_750_12	4348	4364	10060	56.78	2.33	4476	2.86	3.07	9541	54.43	3600*	4364	0.37	3600*
day1_750_13	1054	1054	10695	90.14	3.17	1060	0.57	2.00	8578	87.71	3600*	**1054**	**0.00**	96.57
day1_750_14	4593	4786	10504	56.27	1.62	4802	4.35	3.73	5532	16.97	3600*	4786	4.03	3600*
day0.5_1000_15	1779	1912	7126	75.04	2.70	1916	7.15	3.02	4526	60.69	3600*	1912	6.96	3600*
day0.5_1000_16	12119	12120	14454	16.15	2.65	12126	0.06	2.63	12917	6.18	3600*	12120	0.01	3600*
day0.5_1000_17	5851	5851	**5851**	**0.00**	3.08	**5851**	**0.00**	3.12	**5851**	**0.00**	111.68	**5851**	**0.00**	111.17
day0.5_1000_18	3849	3849	16593	76.80	5.44	**3849**	**0.00**	2.13	165169	97.67	3600*	**3849**	**0.00**	72.72
day0.5_1000_19	18529	18588	21846	15.18	2.48	18588	0.32	3.69	20317	8.80	3600*	18589	0.32	3600*
day1_1000_15	13302	13345	16580	19.77	3.03	13345	0.32	4.71	13883	4.18	3600*	13346	0.33	3600*
day1_1000_15	6405	6405	6990	8.37	1.52	6426	0.33	1.80	**6405**	**0.00**	1093.87	**6405**	**0.00**	681.66
day1_1000_16	857	857	8494	89.91	9.33	859	0.23	4.55	177530	99.52	3600*	**857**	**0.00**	193.81
day1_1000_16	2368	2368	**2368**	**0.00**	1.21	**2368**	**0.00**	1.95	**2368**	**0.00**	534.68	**2368**	**0.00**	536.19
day1_1000_17	5943	5943	14587	59.26	2.80	5960	0.29	2.72	14222	58.21	3600*	**5943**	**0.00**	145.80
day1_1000_17	2576	2665	5820	55.74	2.82	2665	3.34	8.60	3650	29.42	3600*	2665	3.34	3600*
day1_1000_18	6706	6706	**6706**	**0.00**	1.49	**6706**	**0.00**	1.69	**6706**	**0.00**	1049.83	**6706**	**0.00**	1064.72
day1_1000_18	17700	17704	21091	16.08	1.53	17706	0.03	3.64	21091	16.08	3600*	17704	0.02	3600*
day1_1000_19	4584	4741	5120	10.47	1.76	4741	3.31	3.27	5120	10.47	3600*	4741	3.31	3600*
day1_1000_19	20392	20406	20795	1.94	1.58	20418	0.13	2.76	20434	0.21	3600*	20406	0.07	3600*
Averages				**26.24**	**1.48**		**1.13**	**1.83**		**14.90**	**2089.26**		**0.70**	**1569.16**

Table 4.4: Relative percentage gaps achieved by CPLEX for the large dataset using the two relaxed formulations and two exact formulations. The optimization process was constrained to a one-hour time limit and executed with 32 threads. Bold values indicate bounds that match the best-known solutions, which were found either through heuristics or during CPLEX optimization. The symbol ∗ indicates that the optimization reached the time limit, while ⋆ signifies that there are instances with unbounded results excluded from the average.

Instance	Best Known		No-Mem			Cont-Mem			Discr-Mem			Strght-Mem		
	Sol	Bound	UB	Gap%	Time	UB	Gap%	Time	UB	Gap%	Time	UB	Gap%	Time
day1_1500_0	11098	11614	17771	37.55	2.71	11614	4.44	4.24	147762	92.49	3600*	*inf*	–	3600*
day1_1500_1	6671	6902	13146	49.25	5.67	6902	3.35	6.85	*inf*	–	3600*	*inf*	–	3600*
day1_1500_2	14459	15397	17417	16.98	2.22	15397	6.09	3.44	*inf*	–	3600*	*inf*	–	3600*
day1_1500_3	21211	21399	25343	16.30	6.55	21399	0.88	8.80	105514	79.90	3600*	*inf*	–	3600*
day1_2000_4	18137	19627	24018	24.49	6.59	19627	7.59	9.07	*inf*	–	3600*	*inf*	–	3600*
day1_2000_5	12568	12568	**12568**	**0.00**	14.23	**12568**	**0.00**	17.57	*inf*	–	3600*	*inf*	–	3600*
day1_2000_6	3710	3735	3953	6.15	5.19	3735	0.67	5.52	*inf*	–	3600*	*inf*	–	3600*
day1_2000_7	17641	18314	19043	7.36	5.38	18314	3.67	7.16	*inf*	–	3600*	*inf*	–	3600*
day1_2500_8	6614	6887	14015	52.81	8.50	6887	3.96	10.07	442968	98.51	3600*	442968	98.51	3600*
day1_2500_9	10066	10334	21312	52.77	13.95	10334	2.59	9.37	233144	95.68	3600*	*inf*	–	3600*
day1_2500_10	3049	3198	7229	57.82	10.38	3198	4.66	11.76	*inf*	–	3600*	*inf*	–	3600*
day1_2500_11	38028	38463	42935	11.43	30.41	38463	1.13	32.88	*inf*	–	3600*	*inf*	–	3600*
day1_3000_12	59269	59428	59853	0.98	14.24	59428	0.27	16.46	*inf*	–	3600*	*inf*	–	3600*
day1_3000_13	19363	21080	25768	24.86	12.11	21080	8.15	14.35	280713	93.10	3600*	*inf*	–	3600*
day1_3000_14	50273	50745	54608	7.94	17.05	50745	0.93	22.50	*inf*	–	3600*	*inf*	–	3600*
day1_3000_15	3526	3670	7527	53.16	10.15	3670	3.92	11.24	104127	96.61	3600*	*inf*	–	3600*
day1_3500_16	5788	5933	8162	29.09	15.62	5933	2.44	16.68	*inf*	–	3600*	*inf*	–	3600*
day1_3500_17	3794	3899	13954	72.81	65.13	3899	2.69	59.16	*inf*	–	3600*	*inf*	–	3600*
day1_3500_18	35177	35614	42735	17.69	38.49	35614	1.23	45.97	*inf*	–	3600*	*inf*	–	3600*
day1_3500_19	49758	51364	62145	19.93	39.00	51364	3.13	57.41	*inf*	–	3600*	*inf*	–	3600*
Averages				**27.97**	**16.18**		**3.09**	**18.53**		92.72⋆	3600.00⋆		98.51⋆	3600.00⋆

4.8.5 Heuristic Results

In this section, we outline the tests conducted to evaluate the performance of the proposed GA and compare it with the heuristics described in Section 4.7. The results for medium and large-sized instances are reported in Tables 4.5 and 4.6, respectively. Additionally, we provide boxplots illustrating the results for the medium-sized test bed in Figure 4.4. Please note that in this section, the relative percentage gaps are computed between the best upper bound obtained by any of the methods considered in Section 4.8.4 and the lower bound obtained by the current method for the specific instance. In other words, while the previous section fixed the lower bound for each instance and allowed the upper bound to vary depending on the method used, this section fixes the upper bound and then varies the lower bound accordingly.

Table 4.5: Average and best relative percentage gaps achieved by the two exact formulations and the three heuristics. These results were obtained within a time limit of 5 minutes and executed with a single thread. Values matching the best-known solutions are highlighted in bold.

Instance	Best Known		Relax&Repair		Strght-Mem		Genetic		Hybrid-GRR	
	Sol	Bound	Best	Avg	Best	Avg	Best	Avg	Best	Avg
day0.12_250_0	833	833	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
day0.12_250_1	4502	4502	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
day0.12_250_2	6065	6065	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
day0.12_250_3	3175	3175	**0.00**	0.01	**0.00**	**0.00**	**0.00**	0.03	**0.00**	0.00
day0.12_250_4	7964	7964	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
day0.25_250_0	2514	2514	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
day0.25_250_1	7764	7764	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
day0.25_250_2	927	949	3.79	5.22	**2.32**	3.08	**2.32**	2.69	**2.32**	2.57
day0.25_250_3	2560	2560	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.02	**0.00**	**0.00**
day0.25_250_4	2818	2818	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
day1_250_0	4917	4917	1.50	3.27	**0.00**	0.02	0.22	0.39	0.28	0.48
day1_250_1	4008	4008	0.22	0.25	**0.00**	**0.00**	0.22	0.40	0.22	0.24
day1_250_2	2383	2383	**0.00**	0.31	**0.00**	**0.00**	**0.00**	0.01	**0.00**	**0.00**
day1_250_3	5507	5507	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
day1_250_4	1900	1900	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**

Instance	Best Known		Relax&Repair		Strght-Mem		Genetic		Hybrid-GRR	
	Sol	**Bound**	**Best**	**Avg**	**Best**	**Avg**	**Best**	**Avg**	**Best**	**Avg**
day0.25_500_5	3099	3099	0.23	0.93	**0.00**	**0.00**	**0.00**	0.20	**0.00**	0.09
day0.25_500_6	5989	5989	0.02	0.02	**0.00**	**0.00**	0.03	0.05	0.02	0.02
day0.25_500_7	15	15	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
day0.25_500_8	10037	10037	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.01	**0.00**	**0.00**
day0.25_500_9	3181	3272	3.97	4.42	**2.78**	3.23	**2.78**	3.30	**2.78**	**2.78**
day0.5_500_5	1976	2023	5.68	11.65	5.19	6.72	3.51	4.03	3.26	3.62
day0.5_500_6	6952	6953	0.04	0.15	**0.01**	0.02	0.24	0.34	0.04	0.09
day0.5_500_7	7078	7147	1.73	2.48	1.11	1.43	1.09	1.13	0.99	1.07
day0.5_500_8	3090	3090	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.07	**0.00**	**0.00**
day0.5_500_9	1800	1800	0.83	1.35	**0.00**	0.31	1.50	2.56	0.33	0.65
day1_500_5	2843	2843	9.67	11.32	**0.00**	0.10	7.70	7.98	7.18	7.72
day1_500_6	5587	5587	0.02	0.05	**0.00**	**0.00**	0.07	0.13	0.02	0.04
day1_500_7	5690	5703	2.40	9.20	**0.23**	0.28	1.32	1.79	0.72	1.04
day1_500_8	3145	3145	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.08	**0.00**	**0.00**
day1_500_9	2818	2869	8.12	10.10	2.89	4.28	4.08	5.05	4.50	5.13
day0.38_750_10	8897	8897	**0.00**	**0.00**	**0.00**	**0.00**	0.01	0.12	**0.00**	**0.00**
day0.38_750_11	10289	10364	1.35	1.66	1.07	1.32	1.03	1.15	0.97	1.05
day0.38_750_12	11150	11150	0.04	0.08	**0.00**	**0.00**	0.02	0.04	0.02	0.03
day0.38_750_13	3665	3665	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
day0.38_750_14	3787	3931	5.83	7.94	**3.66**	5.12	3.74	4.09	3.79	3.96
day0.75_750_10	5756	5783	1.07	1.37	0.66	1.44	0.80	1.08	0.71	0.84
day0.75_750_11	7069	7076	2.11	3.60	0.73	2.19	2.05	2.85	1.13	1.45
day0.75_750_12	9120	9634	16.73	19.06	8.05	9.32	**5.34**	6.25	5.36	6.10
day0.75_750_13	12767	12912	1.25	9.75	**1.12**	5.58	1.25	1.26	1.25	1.26
day0.75_750_14	12453	12453	**0.00**	**0.00**	**0.00**	**0.00**	0.04	0.06	**0.00**	**0.00**
day1_750_10	755	756	0.79	3.10	**0.13**	0.22	2.51	3.31	0.26	0.50
day1_750_11	10836	10993	12.54	14.34	2.70	3.67	1.70	2.29	1.64	1.70
day1_750_12	4348	4364	5.93	6.48	**0.37**	0.37	0.82	1.14	0.78	0.86
day1_750_13	1054	1054	**0.00**	**0.00**	**0.00**	0.38	0.66	1.40	**0.00**	**0.00**
day1_750_14	4593	4786	9.97	11.46	9.67	11.90	4.05	4.73	**4.03**	4.35
day0.5_1000_15	1779	1912	9.10	11.49	8.94	12.78	7.79	9.08	7.74	8.06
day0.5_1000_16	12119	12120	0.12	0.15	0.02	0.09	0.36	0.41	0.05	0.06
day0.5_1000_17	5851	5851	**0.00**	**0.00**	**0.00**	**0.00**	0.05	0.15	**0.00**	**0.00**
day0.5_1000_18	3849	3849	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
day0.5_1000_19	18529	18588	0.50	0.65	0.41	0.97	0.36	0.43	0.38	0.40
day1_1000_15	13302	13345	1.02	1.37	0.85	1.24	0.49	0.66	0.43	0.49
day1_1000_15	6405	6405	0.14	0.32	0.06	0.40	0.19	0.45	0.14	0.14
day1_1000_16	857	857	5.95	6.11	**0.00**	**0.00**	5.13	5.87	4.78	5.06
day1_1000_16	2368	2368	**0.00**	**0.00**	**0.00**	**0.00**	0.13	0.60	**0.00**	**0.00**
day1_1000_17	5943	5943	0.13	0.21	**0.00**	**0.00**	0.94	2.77	0.12	0.18
day1_1000_17	2576	2665	9.98	13.05	10.92	15.85	3.83	4.73	3.79	4.07
day1_1000_18	6706	6706	**0.00**	**0.00**	**0.00**	3.42	**0.00**	0.20	**0.00**	**0.00**
day1_1000_18	17700	17704	0.06	0.15	**0.02**	0.23	0.17	0.25	0.05	0.07
day1_1000_19	4584	4741	9.01	9.88	13.18	17.71	3.42	4.45	**3.31**	3.60
day1_1000_19	20392	20406	0.53	0.70	0.76	1.50	0.07	0.08	**0.07**	0.08
Averages			**2.21**	**3.06**	**1.30**	**1.92**	**1.20**	**1.50**	**1.06**	**1.16**

Table 4.6: Average and best relative percentage gaps achieved by the two exact formulations and the three heuristics. These results were obtained within a time limit of 5 minutes and executed with a single thread. Values matching the best-known solutions are highlighted in bold.

Instance	Best Known		Relax&Repair		Strght-Mem		Genetic		Hybrid-GRR	
	Sol	**Bound**	**Best**	**Avg**	**Best**	**Avg**	**Best**	**Avg**	**Best**	**Avg**
day1_1500_0	11098	11614	10.78	13.95	100.00	100.00	4.49	6.01	**4.44**	5.23
day1_1500_1	6671	6902	5.56	6.82	80.06	80.06	**3.35**	3.65	3.45	3.65
day1_1500_2	14459	15397	17.52	20.62	100.00	100.00	6.53	7.10	**6.09**	6.94
day1_1500_3	21211	21399	1.35	1.47	0.93	1.45	1.21	1.26	**0.88**	0.94
day1_2000_4	18137	19627	14.11	17.32	99.70	99.70	8.11	9.66	**7.59**	8.29
day1_2000_5	12568	12568	**0.00**	**0.00**	95.32	95.32	0.26	0.49	**0.00**	**0.00**
day1_2000_6	3710	3735	2.68	3.64	100.00	100.00	0.94	1.49	**0.67**	0.84
day1_2000_7	17641	18314	7.30	8.85	100.00	100.00	4.23	5.13	**3.67**	3.90
day1_2500_8	6614	6887	7.11	7.58	100.00	100.00	6.19	6.77	**3.96**	4.21
day1_2500_9	10066	10334	4.43	5.03	100.00	100.00	5.99	7.09	**2.59**	3.12
day1_2500_10	3049	3198	12.45	13.25	99.47	99.47	6.16	6.43	**4.66**	5.34
day1_2500_11	38028	38463	1.50	1.64	99.40	99.40	1.22	1.28	**1.13**	1.16
day1_3000_12	59269	59428	0.34	0.41	100.00	100.00	0.33	0.40	**0.27**	0.29
day1_3000_13	19363	21080	28.25	31.15	100.00	100.00	9.81	11.10	**8.15**	8.68
day1_3000_14	50273	50745	1.61	2.21	99.97	99.97	1.24	1.84	**0.93**	0.97
day1_3000_15	3526	3670	9.78	11.20	100.00	100.00	5.59	6.02	**3.92**	4.47
day1_3500_16	5788	5933	5.38	6.54	100.00	100.00	3.49	4.07	**2.44**	3.17
day1_3500_17	3794	3899	4.13	4.35	98.67	98.67	8.18	9.62	**2.69**	3.00
day1_3500_18	35177	35614	1.69	2.05	100.00	100.00	1.32	1.36	**1.23**	1.26
day1_3500_19	49758	51364	5.02	5.79	100.00	100.00	3.37	4.92	**3.13**	4.25
Averages			**7.05**	**8.19**	**93.68**	**93.70**	**4.10**	**4.79**	**3.10**	**3.48**

The initial heuristic we examine is the Relax and Repair heuristic, as detailed in Section 4.7.1. Table 4.5 shows that this approach performs remarkably well with instances that are not strongly limited by memory constraints, resulting in a small median relative gap, as portrayed in Figure 4.4. In such scenarios, the relaxation effectively identifies either the optimal solution or a close approximation,

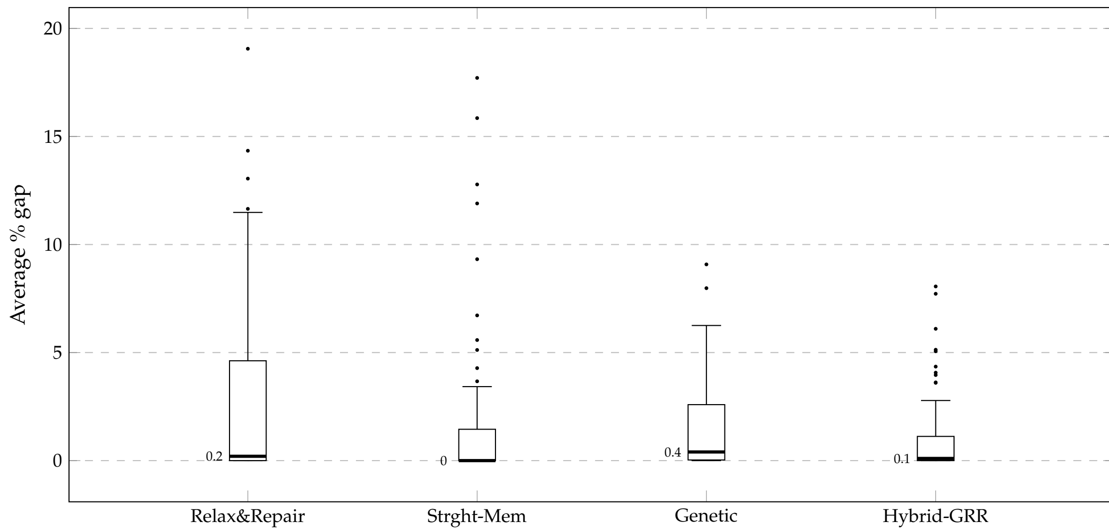


Figure 4.4: Comparison of average gaps obtained by the 5 ″heuristic‟ algorithms. The median is shown at the left of each boxplot.

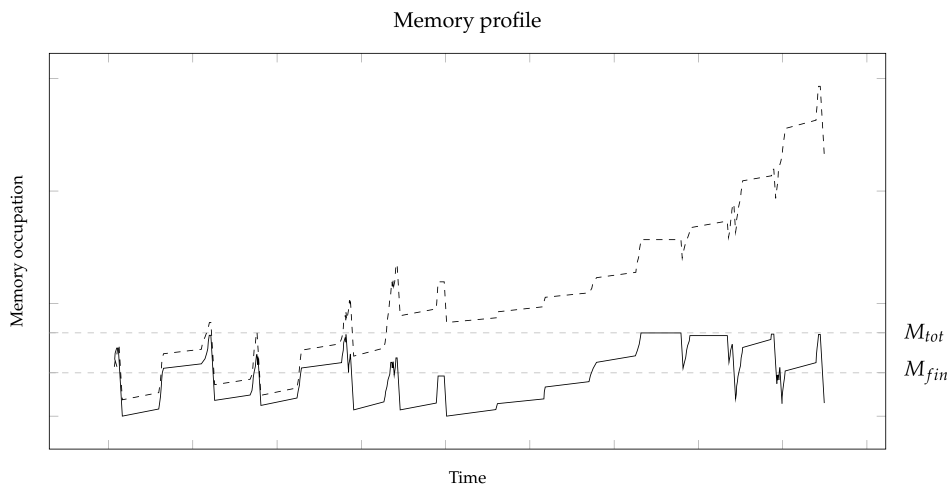


Figure 4.5: Memory profile computed with Algorithm IOM for a feasible plan (solid line) and a relaxed plan resulting from the continuous-memory model (dashed line).

which is then rapidly improved to reach the optimum. However, for memory-intensive instances, this method performs poorly, producing solutions with relative gaps of nearly 20% when compared to the best bounds obtained. This discrepancy is clearly evident in the box plot, where the extended upper whisker surpasses that of all other considered heuristics.

This disparity can largely be attributed to a substantial variance between the continuous-memory profile and the actual profile. For instance, consider Figure 4.5, which displays the memory profile computed with Algorithm IOM for a feasible plan and the one obtained by solving the relaxed model. To meet the requirements outlined in Section 4.3, the plan derived from the relaxed model would necessitate a satellite with approximately four times the available memory. This discrepancy significantly influences the effects of the repair procedures, leading to a notable degradation in solution quality.

Nonetheless, when dealing with large-scale instances, this technique still manages to obtain relatively competitive results.

The second heuristic we consider is the strengthened discrete-memory formulation. For medium-sized instances, this approach outperforms others in terms of the median relative gap, as shown in Figure 4.4. However, the same plot reveals a significant number of outliers, indicating instability in

this method. While it often finds exceptionally good (often optimal) solutions, it can also produce solutions of lower quality when it fails. On average, for medium-sized instances, it achieves a relative gap of 1.92%. The results are completely different for large instances, where this method, based on the full problem formulation, struggles to find non-trivial solutions, making it the worst-performing technique in this size range.

Next, we have the solver-free version of the Genetic algorithm. Although it has the worst overall median relative gap (as seen in Figure 4.4), it still ranks as the second-best technique on average. The results for medium-sized instances maintain stability within a 10% relative gap. However, it tends to find fewer instances at the optimum compared to other methods, causing the higher median relative gap.

Finally, we examine the Genetic Algorithms - Relax and Repair hybridization, which proves to be the best-performing heuristic. In this hybrid approach the Relax and Repair serves two purposes: (i) it obtains optimal solutions for instances with low memory demands, reducing the median and average relative gap, and (ii) it generates a diversified collection of solutions containing high-quality DTO sequences. When the Genetic Algorithm is invoked, it generally addresses the shortcomings of the Relax and Repair methods, producing good-quality solutions even for instances where the continuous-memory relaxation yields weak bounds.

Concerning large-scale instances, the last two methods, based on the Genetic Algorithm, excel, resulting in an average relative gap of 4.79% for the Genetic Algorithm alone and 3.48% for the Hybridized version.

4.9 Conclusions

In this paper, we introduced a hybrid genetic algorithm (GA) designed to address the daily planning of acquisitions and downlink activity scheduling for the PLATiNO satellite. Our GA incorporates local search and repair methods to improve its convergence towards high-quality solutions. Additionally, we proposed mathematical formulations to either fully or partially model the problem, leveraging them to compute valid upper bounds, which in turn validate our computational outcomes.

Our proposed algorithm, requiring just a few minutes of computational time, consistently produces solutions with an average gap of approximately 1.5% compared to the upper bounds for medium-sized instances and less than 4% for larger ones. Furthermore, we implemented alternative heuristic approaches to assess the performance of solver-based techniques.

Our results demonstrate that, in the context of this particular problem, commercial solvers can significantly enhance heuristic quality, yet they may not be the optimal standalone solution, especially when faced with memory-requiring instances.

Bibliography

Accorsi L, Vigo D, 2020 *FILO repository*. `https://github.com/acco93/filo`.

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

Anagnostopoulou A, Repoussis P, Tarantilis C, 2013 *Grasp with path relinking for vehicle routing problems with clustered and mixed backhauls*. Technical report, Athens University of Economics and Business.

Applegate D, Bixby R, Chvátal V, Cook W, 1999 *Finding tours in the TSP*. Technical report, University of Bonn, Germany.

Applegate D, Cook W, Rohe A, 2003 *Chained Lin-Kernighan for large traveling salesman problems. INFORMS Journal on Computing* 15(1):82–92.

Arenas I, Sánchez A, Armando C, Solano L, Medina L, 2017 *Cvrptw model applied to the collection of food donations. Proceedings of the International Conference on Industrial Engineering and Operations Management*, 1307—-1314 (Bogotá D.C.).

Arnold F, Gendreau M, Sörensen K, 2019 *Efficiently solving very large-scale routing problems. Computers & Operations Research* 107:32–42.

Arnold F, Sörensen K, 2019 *Knowledge-guided local search for the vehicle routing problem. Computers & Operations Research* 105:32 – 46, URL `http://dx.doi.org/10.1016/j.cor.2019.01.002`.

Barkaoui M, Berger J, 2020 *A new hybrid genetic algorithm for the collection scheduling problem for a satellite constellation. Journal of the Operational Research Society* 71(9):1390–1410.

Beek O, Raa B, Dullaert W, Vigo D, 2018 *An efficient implementation of a static move descriptor-based local search heuristic. Computers & Operations Research* 94:1 – 10, URL `http://dx.doi.org/https://doi.org/10.1016/j.cor.2018.01.006`.

Benoist T, Rottembourg B, 2004 *Upper bounds for revenue maximization in a satellite scheduling problem. 4OR* 2(3):235–249.

Bensana E, Verfaillie G, Agnese J, Bataille N, Blumstein D, 1999 *Exact and inexact methods for the daily management of an earth observation satellite. Proceeding of the international symposium on space mission operations and ground data systems*, 507–514.

Bentley JL, 1990 *K-d trees for semidynamic point sets. Proceedings of the sixth annual symposium on Computational geometry*, 187–197.

Bergstra J, Komer B, Eliasmith C, Yamins D, Cox DD, 2015 *Hyperopt: a python library for model selection and hyperparameter optimization. Computational Science & Discovery* 8(1):014008.

Bianchessi N, Righini G, 2008 *Planning and scheduling algorithms for the cosmo-skymed constellation. Aerospace Science and Technology* 12(7):535–544.

Caprara A, Fischetti M, Toth P, 1999 *A heuristic method for the set covering problem. Operations Research* 47(5):730–743.

Cavaliere F, 2021 *newLKH repository*. `https://github.com/c4v4/LKH3`, URL `http://dx.doi.org/10.5281/zenodo.6644959`.

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

Christofides N, Mingozzi A, Toth P, 1979 *The vehicle routing problem*, volume 1 (Wiley Interscience).

Clarke G, Wright JW, 1964 *Scheduling of vehicles from a central depot to a number of delivery points. Operations Research* 12(4):568–581.

Cordeau JF, Laporte G, 2005 *Maximizing the value of an earth observation satellite orbit. Journal of the Operational Research Society* 56(8):962–968.

CPLEX, 2019 *IBM Ilog CPLEX optimizer 12.10 callable library*.

Dantzig GB, 1957 *Discrete-variable extremum problems. Operations Research* 5(2):266–277.

Dantzig GB, Ramser JH, 1959 *The truck dispatching problem*. *Management Science* 6(1):80–91.

Dantzig GB, Wolfe P, 1961 *The decomposition algorithm for linear programs*. *Econometrica: Journal of the Econometric Society* 767–778.

De Franceschi R, Fischetti M, Toth P, 2006 *A new ILP-based refinement heuristic for vehicle routing problems*. *Mathematical Programming* 105(2):471–499.

De Jong KA, 1975 *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. thesis, University of Michigan, USA.

Delgado-Antequera L, Laguna M, Pacheco J, Caballero R, 2020 *A bi-objective solution approach to a real-world waste collection problem*. *Journal of the Operational Research Society* 71(2):183—-194.

Desaulniers G, Desrosiers J, Solomon MM, Soumis F, Villeneuve D, et al., 1998 *A unified framework for deterministic time constrained vehicle routing and crew scheduling problems*. *Fleet management and logistics*, 57–93 (Springer).

Dethloff J, 2001 *Vehicle routing and reverse logistics: The vehicle routing problem with simultaneous delivery and pick-up*. *OR-Spektrum* 23(1):79–96, URL `http://dx.doi.org/https://doi.org/10.1007/PL00013346`.

Fischetti M, Fischetti M, 2018 *Matheuristics*. Martí R, Pardalos PM, Resende MGC, eds., *Handbook of Heuristics*, 121–153 (Cham: Springer International Publishing).

Ford Jr LR, Fulkerson DR, 1958 *A suggested computation for maximal multi-commodity network flows*. *Management Science* 5(1):97–101.

Foster BA, Ryan DM, 1976 *An integer programming approach to the vehicle scheduling problem*. *Journal of the Operational Research Society* 27(2):367–384.

Fredman ML, Johnson DS, McGeoch LA, Ostheimer G, 1995 *Data structures for traveling salesmen*. *Journal of Algorithms* 18(3):432–479.

Fukasawa R, Longo H, Lysgaard J, De Aragão MP, Reis M, Uchoa E, Werneck RF, 2006 *Robust branch-and-cut-and-price for the capacitated vehicle routing problem*. *Mathematical Programming* 106(3):491–511.

Gabrel V, Vanderpooten D, 2002 *Enumeration and interactive selection of efficient paths in a multiple criteria graph for scheduling an earth observing satellite*. *European Journal of Operational Research* 139(3):533–542.

Gajpal Y, Abad P, 2009 *Multi-ant colony system (macs) for a vehicle routing problem with backhauls*. *European Journal of Operational Research* 196(1):102–117, URL `http://dx.doi.org/https://doi.org/10.1016/j.ejor.2008.02.025`.

Goksal F, Karaoglan I, Altiparmak F, 2013 *A hybrid discrete particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery*. *Computers & Industrial Engineering* 65(1):39–53, URL `http://dx.doi.org/https://doi.org/10.1016/j.cie.2012.01.005`.

Hall NG, Magazine MJ, 1994 *Maximizing the value of a space mission*. *European Journal of Operational Research* 78(2):224–241.

Hamzadayı A, Baykasoğlu A, Akpınar Ş, 2020 *Solving combinatorial optimization problems with single seekers society algorithm*. *Knowledge-Based Systems* 201-202:106036, URL `http://dx.doi.org/https://doi.org/10.1016/j.knosys.2020.106036`.

Helsgaun K, 2000 *An effective implementation of the lin–kernighan traveling salesman heuristic*. *European Journal of Operational Research* 126(1):106–130, URL `http://dx.doi.org/https://doi.org/10.1016/S0377-2217(99)00284-2`.

Helsgaun K, 2006 *An effective implementation of K-opt moves for the Lin-Kernighan TSP heuristic*. Ph.D. thesis, Roskilde University. Department of Computer Science.

Helsgaun K, 2009 *General k-opt submoves for the Lin–Kernighan TSP heuristic*. *Mathematical Programming Computation* 1(2):119–163.

Helsgaun K, 2017 *An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems*. Technical report, Roskilde Universitet, URL `http://dx.doi.org/10.13140/RG.2.2.25569.40807`.

Helsgaun K, 2020 *LKH-3*. `http://akira.ruc.dk/~keld/research/LKH-3`.

Hof J, Schneider M, 2019 *An adaptive large neighborhood search with path relinking for a class of vehicle-routing problems with simultaneous pickup and delivery*. *Networks* 74(3):207–250, URL `http://dx.doi.org/https://doi.org/10.1002/net.21879`.

Hornstra R, Silva A, Roodbergen K, Coelho L, 2020 *The vehicle routing problem with simultaneous pickup and delivery and handling costs*. *Computers & Operations Research* 115:104858, URL `http://dx.doi.org/https://doi.org/10.1016/j.cor.2019.104858`.

Irnich S, 2008a *Resource extension functions: properties, inversion, and generalization to segments*. *OR Spectrum* 30:113 – 148, URL `http://dx.doi.org/10.1007/s00291-007-0083-6`.

Irnich S, 2008b *A unified modeling and solution framework for vehicle routing and local search-based metaheuristics*. *INFORMS Journal on Computing* 20(2):270–287.

Jonker R, Volgenant T, 1986 *Transforming asymmetric into symmetric traveling salesman problems: erratum*. *Operations Research Letters* 5(4):215–216.

Jonker R, Volgenant T, 1988 *An improved transformation of the symmetric multiple traveling salesman problem*. *Operations Research* 36(1):163–167.

Kalayci CB, Kaya C, 2016 *An ant colony system empowered variable neighborhood search algorithm for the vehicle routing problem with simultaneous pickup and delivery*. *Expert Systems with Applications* 66:163–175, URL `http://dx.doi.org/https://doi.org/10.1016/j.eswa.2016.09.017`.

Kelly JP, Xu J, 1999 *A set-partitioning-based heuristic for the vehicle routing problem*. *INFORMS Journal on Computing* 11(2):161–172.

Kindervater GA, Savelsbergh MW, 1997 *10. vehicle routing: handling edge exchanges*. *Local search in combinatorial optimization* (John Wiley & Son).

Kirkpatrick S, Gelatt CD, Vecchi MP, 1983 *Optimization by simulated annealing*. *science* 220(4598):671–680.

Koç Ç, Laporte G, 2018 *Vehicle routing with backhauls: Review and research perspectives*. *Computers & Operations Research* 91:79–91, URL `http://dx.doi.org/https://doi.org/10.1016/j.cor.2017.11.003`.

Koç Ç, Laporte G, Tükenmez I, 2020 *A review of vehicle routing with simultaneous pickup and delivery*. *Computers & Operations Research* 122:104987, URL `http://dx.doi.org/https://doi.org/10.1016/j.cor.2020.104987`.

Laporte G, Nobert Y, Desrochers M, 1985 *Optimal routing under capacity and distance restrictions*. *Operations Research* 33(5):1050–1073.

Lemaitre M, Verfaillie G, Jouhaud F, Lachiver JM, Bataille N, 2002 *Selecting and scheduling observations of agile satellites*. *Aerospace Science and Technology* 6(5):367–381.

Li Y, Xu M, Wang R, 2007 *Scheduling observations of agile satellites with combined genetic algorithm*. *Third International Conference on Natural Computation (ICNC 2007)*, volume 3, 29–33.

Lin S, Kernighan BW, 1973 *An effective heuristic algorithm for the traveling-salesman problem*. *Operations Research* 21(2):498–516.

Lin WC, Liao DY, 2004 *A tabu search algorithm for satellite imaging scheduling*. *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, volume 2, 1601–1606.

Lin WC, Liao DY, Liu CY, Lee YY, 2005 *Daily imaging scheduling of an earth observation satellite*. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 35(2):213–223.

Liu X, Laporte G, Chen Y, He R, 2017 *An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time*. *Computers & Operations Research* 86:41–53.

Lourenço HR, Martin OC, Stützle T, 2003 *Iterated Local Search*, 320–353 (Boston, MA: Springer US), ISBN 978-0-306-48056-0, URL `http://dx.doi.org/10.1007/0-306-48056-5_11`.

Luo K, Wang H, Li Y, Li Q, 2017 *High-performance technique for satellite range scheduling*. *Computers & Operations Research* 85:12–21.

Mansour MA, Dessouky MM, 2010 *A genetic algorithm approach for solving the daily photograph selection problem of the spot5 satellite*. *Computers & Industrial Engineering* 58(3):509–520.

Martin O, Otto SW, Felten EW, 1992 *Large-step markov chains for the TSP incorporating local search heuristics*. *Operations Research Letters* 11(4):219–224.

Min H, 1989 *The multiple vehicle routing problem with simultaneous delivery and pick-up points*. *Transportation Research Part A: General* 23(5):377–386, URL `http://dx.doi.org/https://doi.org/10.1016/0191-2607(89)90085-X`.

Monaci M, Toth P, 2006 *A set-covering-based heuristic approach for bin-packing problems*. *INFORMS Journal on Computing* 18(1):71–85.

Montané A, Galvão R, 2006 *A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service*. *Computers & Operations Research* 33(3):595–619, URL `http://dx.doi.org/https://doi.org/10.1016/j.cor.2004.07.009`.

Nagata Y, Kobayashi S, 2013 *A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem*. *INFORMS Journal on Computing* 25(2):346–363.

Oesterle J, Bauernhansl T, 2016 *Exact method for the vehicle routing problem with mixed linehaul and backhaul customers, heterogeneous fleet, time window and manufacturing capacity*. *Procedia CIRP* 41:573–578, URL `http://dx.doi.org/https://doi.org/10.1016/j.procir.2015.12.040`, research and Innovation in Manufacturing: Key Enabling Technologies for the Factories of the Future – Proceedings of the 48th CIRP Conference on Manufacturing Systems.

Olgun B, Koç Ç, Altıparmak F, 2021 *A hyper heuristic for the green vehicle routing problem with simultaneous pickup and delivery*. *Computers & Industrial Engineering* 153:107010, URL `http://dx.doi.org/https://doi.org/10.1016/j.cie.2020.107010`.

Öztaş T, Tuş A, 2022 *A hybrid metaheuristic algorithm based on iterated local search for vehicle routing problem with simultaneous pickup and delivery*. *Expert Systems with Applications* 202:117401, URL `http://dx.doi.org/https://doi.org/10.1016/j.eswa.2022.117401`.

Park H, Son D, Koo B, Jeong B, 2021 *Waiting strategy for the vehicle routing problem with simultaneous pickup and delivery using genetic algorithm*. *Expert Systems with Applications* 165:113959, URL `http://dx.doi.org/https://doi.org/10.1016/j.eswa.2020.113959`.

PassMark® Software, 2020 *Professional cpu benchmarks*. URL `https://www.passmark.com/index.html`, visited on 2022-06-03.

Pecin D, Pessoa A, Poggi M, Uchoa E, 2017 *Improved branch-cut-and-price for capacitated vehicle routing*. *Mathematical Programming Computation* 9(1):61–100.

Pecin D, Pessoa A, Poggi M, Uchoa E, 2023 *CVRPLIB - Updates*. `http://vrp.atd-lab.inf.puc-rio.br/index.php/en/updates`.

Peng G, Vansteenwegen P, Liu X, Xing L, Kong X, 2018 *An iterated local search algorithm for agile earth observation satellite scheduling problem*. *2018 SpaceOps Conference*, 2311.

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

Pessoa A, Sadykov R, Uchoa E, Vanderbeck F, 2021 *VRPSolver*. `https://vrpsolver.math.u-bordeaux.fr/`.

Pillac V, Gendreau M, Guéret C, Medaglia AL, 2013 *A review of dynamic vehicle routing problems*. *European Journal of Operational Research* 225(1):1–11.

Polat O, 2017 *A parallel variable neighborhood search for the vehicle routing problem with divisible deliveries and pickups*. *Computers & Operations Research* 85:71–86, URL `http://dx.doi.org/https://doi.org/10.1016/j.cor.2017.03.009`.

Polat O, Kalayci CB, Kulak O, Günther HO, 2015 *A perturbation based variable neighborhood search heuristic for solving the vehicle routing problem with simultaneous pickup and delivery with time limit*. *European Journal of Operational Research* 242(2):369–382, URL `http://dx.doi.org/https://doi.org/10.1016/j.ejor.2014.10.010`.

Queiroga E, Frota Y, Sadykov R, Subramanian A, Uchoa E, Vidal T, 2020 *On the exact solution of vehicle routing problems with backhauls*. *European Journal of Operational Research* 287(1):76–89, URL `http://dx.doi.org/https://doi.org/10.1016/j.ejor.2020.04.047`.

Queiroga E, Sadykov R, Uchoa E, 2021 *A POPMUSIC matheuristic for the capacitated vehicle routing problem*. *Computers & Operations Research* 136:105475.

Rochat Y, Taillard ÉD, 1995 *Probabilistic diversification and intensification in local search for vehicle routing*. *Journal of heuristics* 1(1):147–167.

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

Rothberg E, 2007 *An evolutionary algorithm for polishing mixed integer programming solutions*. *INFORMS Journal on Computing* 19(4):534–541.

Ryan DM, Hjorring C, Glover F, 1993 *Extensions of the petal method for vehicle routeing*. *Journal of the Operational Research Society* 44(3):289–296.

Salhi S, Nagy G, 1999 *A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling*. *Journal of the Operational Research Society* 50(10):1034–1042, URL `http://dx.doi.org/https://doi.org/10.1057/palgrave.jors.2600808`.

Santana JC, Guerhardt F, Franzini C, Ho LL, Júnior SRR, Cânovas G, Berssaneti F, 2021 *Refurbishing and recycling of cell phones as a sustainable process of reverse logistics: A case study in brazil*. *Journal of Cleaner Production* 283.

Santos M, Amorin P, Marques A, Carvalho A, Póvoa A, 2020 *The vehicle routing problem with backhauls towards a sustainability perspective: a review*. *TOP* 28:358—-401, URL `http://dx.doi.org/https://doi.org/10.1007/s11750-019-00534-0`.

Schneider M, Schwahn F, Vigo D, 2017 *Designing granular solution methods for routing problems with time windows*. *European Journal of Operational Research* 263(2):493–509, URL `http://dx.doi.org/10.1016/j.ejor.2017.04.059`.

Shafiee RE, Ghomi SF, Sajadieh M, 2021 *Reverse logistics network design for product reuse, remanufacturing, recycling and refurbishing under uncertainty*. *Journal of Manufacturing Systems* 60:473–486.

Simsir F, Ekmekci D, 2019 *A metaheuristic solution approach to capacitied vehicle routing and network optimization*. *Engineering Science and Technology, an International Journal* 22(3):727–735, URL `http://dx.doi.org/https://doi.org/10.1016/j.jestch.2019.01.002`.

Skarupke M, 2016 *I wrote a faster sorting algorithm*. `https://probablydance.com/2016/12/27/i-wrote-a-faster-sorting-algorithm`, accessed: 2022-07-24.

Subramanian A, Uchoa E, Ochi LS, 2013 *A hybrid algorithm for a class of vehicle routing problems*. *Computers & Operations Research* 40(10):2519–2531.

Taillard ÉD, Helsgaun K, 2019 *POPMUSIC for the travelling salesman problem*. *European Journal of Operational Research* 272(2):420–429.

Tangpattanakul P, Jozefowiez N, Lopez P, 2015 *A multi-objective local search heuristic for scheduling earth observations taken by an agile satellite*. *European Journal of Operational Research* 245(2):542–554.

Taniguchi E, Heijden RVD, 2000 *An evaluation methodology for city logistics*. *Transport Reviews* 20(1):65–90.

Taniguchi E, Thompson R, Yamada T, van Duin R, 2001 *City logistic—network modelling and intelligent transport systems* (Pergamon, Amsterdam).

Toth P, Vigo D, 2003 *The granular tabu search and its application to the vehicle-routing problem*. *INFORMS Journal on Computing* 15(4):333–346, URL http://dx.doi.org/10.1287/ijoc.15.4.333.24890.

Toth P, Vigo D, 2014 *Vehicle routing: problems, methods, and applications* (SIAM, Philadelphia, PA), URL https://epubs.siam.org/doi/abs/10.1137/1.9781611973594.

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

Vasquez M, Hao JK, 2001 *A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite*. *Computational optimization and applications* 20(2):137–157.

Vidal T, Crainic TG, Gendreau M, Lahrichi N, Rei W, 2012 *A hybrid genetic algorithm for multidepot and periodic vehicle routing problems*. *Operations Research* 60(3):611–624.

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

Wang J, Demeulemeester E, Qiu D, 2016 *A pure proactive scheduling algorithm for multiple earth observation satellites under uncertainties of clouds*. *Computers & Operations Research* 74:1–13.

Wang J, Demeulemeester E, Qiu D, Liu J, 2015 *Exact and inexact scheduling algorithms for multiple earth observation satellites under uncertainties of clouds*. *Available at SSRN 2634934* .

Wassan N, Nagy G, Ahmadi S, 2008 *A heuristic method for the vehicle routing problem with mixed deliveries and pickups*. *Journal of Scheduling* 11(2):149—-161.

Wilcoxon F, 1945 *Individual comparisons by ranking methods*. *Biometrics Bulletin* 1(6):80–83, URL http://www.jstor.org/stable/3001968.

Wolsey LA, Nemhauser GL, 1999 *Integer and combinatorial optimization*, volume 55 (John Wiley & Sons).

Zachariadis EE, Kiranoudis CT, 2010 *A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem*. *Comput. Oper. Res.* 37(12):2089–2105, URL http://dx.doi.org/10.1016/j.cor.2010.02.009.

Zachariadis EE, Tarantilis CD, Kiranoudis CT, 2010 *An adaptive memory methodology for the vehicle routing problem with simultaneous pick-ups and deliveries*. *European Journal of Operational Research* 202(2):401–411, URL http://dx.doi.org/https://doi.org/10.1016/j.ejor.2009.05.015.

Zhang D, Guo L, Cai B, Sun N, Wang Q, 2013 *A hybrid discrete particle swarm optimization for satellite scheduling problem*. *IEEE conference anthology*, 1–5.