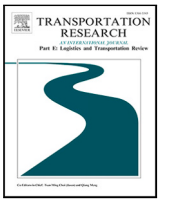
Transportation Research Part E 186 (2024) 103550

Contents lists available at ScienceDirect

Transportation Research Part E

journal homepage: www.elsevier.com/locate/tre



An efficient heuristic for very large-scale vehicle routing problems with simultaneous pickup and delivery

Francesco Cavaliere [a], Luca Accorsi [a], Demetrio Laganà [b,*], Roberto Musmanno [b], Daniele Vigo [a,c]

[a] DEI "G. Marconi", University of Bologna, Viale del Risorgimento 2, 40136 Bologna, Italy
[b] DIMEG, University of Calabria, Via Pietro Bucci, 87036 Rende (CS), Italy
[c] CIRI-ICT, University of Bologna, via Quinto Bucci, 336, 47521 Cesena, Italy

ARTICLE INFO

Keywords:
Vehicle routing problem with pickup and delivery
Metaheuristics
Acceleration techniques
Large-scale instances

ABSTRACT

The paper discusses two variants of the Capacitated Vehicle Routing Problem (CVRP), in which each customer to be visited can require both pickup and delivery, or only either pickup and delivery but not both. 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). Both VRPSPD and VRPMPD are particularly relevant in practical logistics distribution scenarios, especially when dealing with a substantial number of pickup and delivery locations. The aim of this work is to provide an algorithm which extends, through a variety of specializations, the FILO framework, originally proposed and specifically designed for the CVRP. This variant, called FSPD, has been developed to accomplish two objectives: first, being competitive with the state-of-the-art algorithms for the VRPSPD and VRPMPD; 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 and detailed analysis of the algorithm components conducted in this paper demonstrate the successful achievement of both objectives.

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 interest 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

* Corresponding author.
E-mail addresses: f.cavaliere@unibo.it (F. Cavaliere), luca.accorsi4@unibo.it (L. Accorsi), demetrio.lagana@unical.it (D. Laganà), roberto.musmanno@unical.it (R. Musmanno), daniele.vigo@unibo.it (D. Vigo).

https://doi.org/10.1016/j.tre.2024.103550
Received 3 October 2023; Received in revised form 26 February 2024; Accepted 22 April 2024
Available online 8 May 2024

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 et al., 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 et al., 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 zones which are assigned to one or more vehicles or on suitable micro-zones which may be combined or swapped between vehicles. In addition, pickup and delivery requests are often served by separate routes. Such an approach allows for starting package sorting in advance with respect to the complete knowledge of the demand to be served and greatly simplifies the routing design and management. Furthermore, it allows for implicitly enforcing 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. In this context, the capability of solving large-scale routing problems with pickup and delivery may represent an important opportunity to evaluate and possibly redesign service zones and shift towards operations models based on larger areas. For these reasons, we examine two specific variants of the CVRP, in which each customer to be visited can require both pickup and delivery, or only either pickup or delivery but not both. These problems are referred to in the scientific literature as the Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD) and the Vehicle Routing Problem with Mixed Pickup and Delivery (VRPMPD), respectively. 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 aim is to develop solution methods capable of solving problem instances of both the VRPSPD and the VRPMPD within a short computing time. These instances involve several thousands of customers, making them one or two orders of magnitude larger than those solved by existing algorithms in the literature. 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. A detailed component analysis of the proposed algorithm shows that these features allow for 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 2 describes the problems and the corresponding complexity. Section 3 provides a comprehensive literature review of the most recent work on VRPSPD and VRPMPD. In Section 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 5 presents the details of the experimental design and the results of our computational study. Finally, Section 7 provides some conclusions and discusses future work.

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 et al. (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, that is, $p_i > 0$ and $q_i = 0$, or $p_i = 0$ and $q_i > 0$. 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. 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 et al. (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 et al. (2020) show that the best solutions are obtained by Subramanian et al. (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 et al. (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 et al. (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 min 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 et al. (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 12 h 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 et al. (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 et al. (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 et al. (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 et al. (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 et al. (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 et al. (2013) and of Subramanian et al. (2013) is performed. The algorithm is able to obtain the best gap of +0.00% which is the same as Subramanian et al. (2013). These results are reached within an average computing time of 2.46 min.

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 et al. (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 et al. (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 et al. (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.

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 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 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 4.2.1) 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.

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 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 1. 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 et al. (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.

Algorithm 1: 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
```

The main *core optimization iteration* (Lines 7–18) is based on the well-known iterated local search (ILS) paradigm (see Lourenço et al. (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 4.1.1. 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.

4.1.1. 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).

4.1.2. 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 et al. (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).

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 1) 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 1). 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.

4.2.1. 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 $D_i = d_i$.

4.2.2. 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 4.2.1, 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 4.2.1, 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.

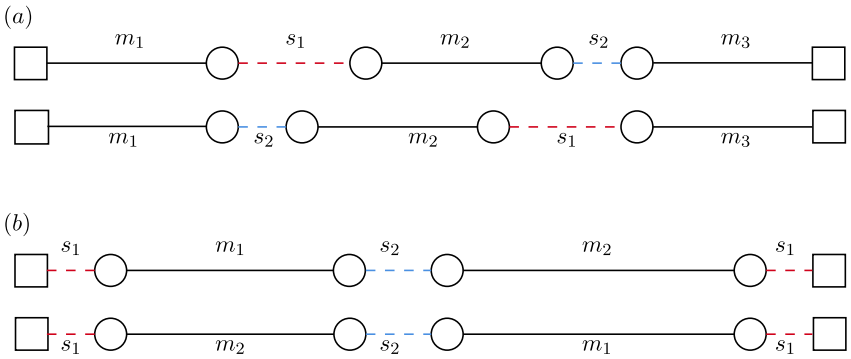


Fig. 1. Illustration of segment concatenation associated with a swap move, in the case the depot is not contained in the segments to be swapped (*a*) or is contained in one of them (*b*).

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.

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 Fig. 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 Fig. 1(*b*).

4.2.3. 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 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).

4.2.4. 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 6.2.1.

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 2 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 6.2.1 and 6.2.2.

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 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 Fig. 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 $\textsc{IsInsertFeasible}$ function, which computes the maximum load

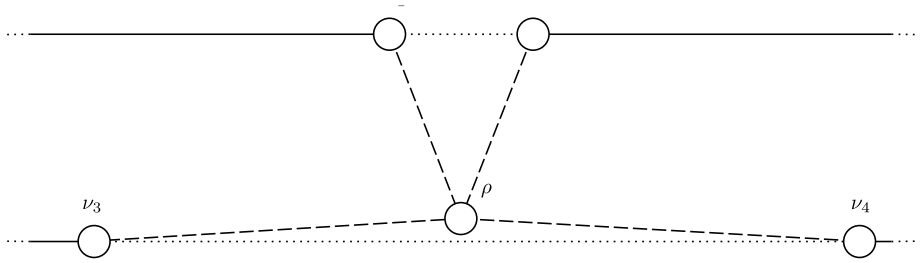


Fig. 2. Illustration of different insertion options. Although v_1 and v_2 are the closest customers to ρ, inserting ρ in this position would introduce a considerable detour. On the other hand, even if v_3 and v_4 are farther from ρ, the insertion cost of ρ between these two customers is smaller.

of the route using the involved segments resources as described in Section 4.2.1. 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.

Algorithm 2: 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 π_ρ, σ_ρ ∈ S between which ρ will be inserted
1  function FindBestInsert(S, ρ)
2  begin
       // Initialize candidate previous and successor nodes to the depot
3      π_ρ ← σ_ρ ← d;
4      c* ← 2 · D_{d,ρ};
5      i ← 0;
6      for v ∈ N⁻_ρ do
7          if i ≥ N_R then                                          // Early exit criterion
8              return π_ρ, σ_ρ;
9          end
10         if v ∉ S then
11             continue;
12         end
13         i ← i + 1;
14         π_v ← Pred(v, S);
15         σ_v ← Succ(v, S);
           // Recall that, since N⁻_ρ is sorted, distances less than D_{ρ,v} have already been checked
16         if D_{π_v,ρ} > D_{ρ,v} or (D_{π_v,ρ} = D_{ρ,v} and π_v > v) then
17             c' ← D_{π_v,ρ} + D_{ρ,v} − D_{π_v,v};
18             if c' < c* and IsInsertFeasible(ρ, S, π_v, v) then
19                 π_ρ, σ_ρ, c* ← π_v, v, c';
20             end
21         end
22         if D_{ρ,σ_v} > D_{v,ρ} or (D_{ρ,σ_v} = D_{v,ρ} and v > σ_v) then
23             c'' ← D_{v,ρ} + D_{ρ,σ_v} − D_{v,σ_v};
24             if c'' < c* and IsInsertFeasible(ρ, S, v, σ_v) then
25                 π_ρ, σ_ρ, c* ← v, σ_v, c'';
26             end
27         end
28     end
29     return π_ρ, σ_ρ;
30 end
```

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.

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 16 GB 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.

5.2. Parameter tuning

The parameter tuning process was initialized with values directly taken from the original FILO approach. Considering the small testbed outlined in Section 6, the average single-core running time to test one of the many possible configurations is approximately 24 h. For this reason, we adopted a sequential tuning strategy that considers only one parameter at a time, avoiding a computationally demanding yet statistically robust tuning approach. 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 6.1),
- the way to consider the insertion positions considered in the new recreate phase (Section 6.2.1),
- the value of the pruning factor for the new recreate phase (Section 6.2.2)

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 5.3.1 and 5.3.2 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 large neighborhood search and path relinking of Hof and Schneider (2019).
- ILS-RVND-SP: the ILS heuristic of Subramanian et al. (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 et al. (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 et al. (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 1
The CPU models used by algorithms from the literature and their single thread PassMark scores.

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 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.

5.3.1. 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.

5.3.1.1. 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 et al. (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.

Table 2 reports the results on the first seven instances only, whereas Table 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.

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 Figs. 3 and 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.

5.3.1.2. 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 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

Table 2
Results on the first seven instances of Salhi and Nagy (1999) CMTX and CMTY datasets.

Algorithm	X			Y		
	Avg	Time[a]	Time	Avg	Time[a]	Time
ALNS-PR	0.000[a]	–	105.611	0.009[a]	–	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[a]	–	100.532	0.121[a]	–	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[a]	4.557	–	0.111[a]	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

[a] Average gaps are actually the best gap obtained along several runs.

Table 3
Results on the complete (Salhi and Nagy, 1999) CMTX and CMTY datasets.

Algorithm	X			Y		
	Avg	Time[a]	Time	Avg	Time[a]	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

[a] Average gaps are actually the best gap obtained along several runs.

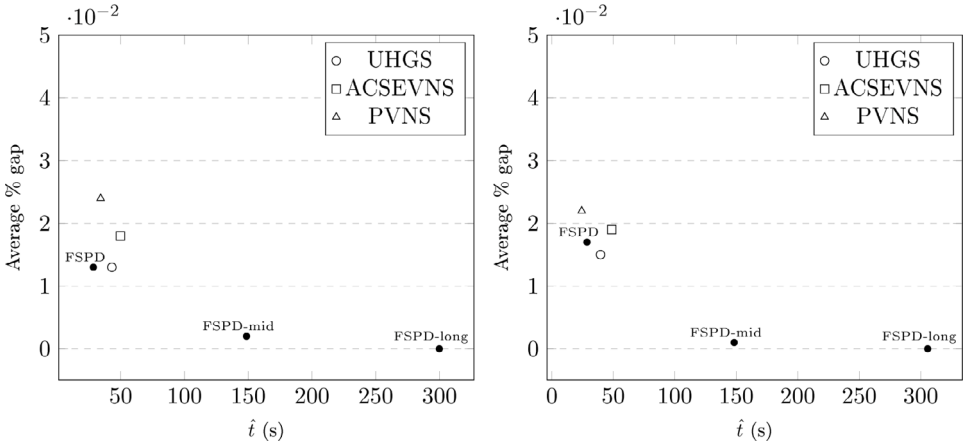


Fig. 3. Illustration of the results on the first seven instances of Salhi and Nagy (1999) X (left) and Y (right) datasets.

literature instances of the VRP with time windows. The aggregate results of the comparison between FSPD and its competitors are reported in Table 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.

5.3.2. 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 et al. (1979), by defining every second, fourth, or tenth customer of

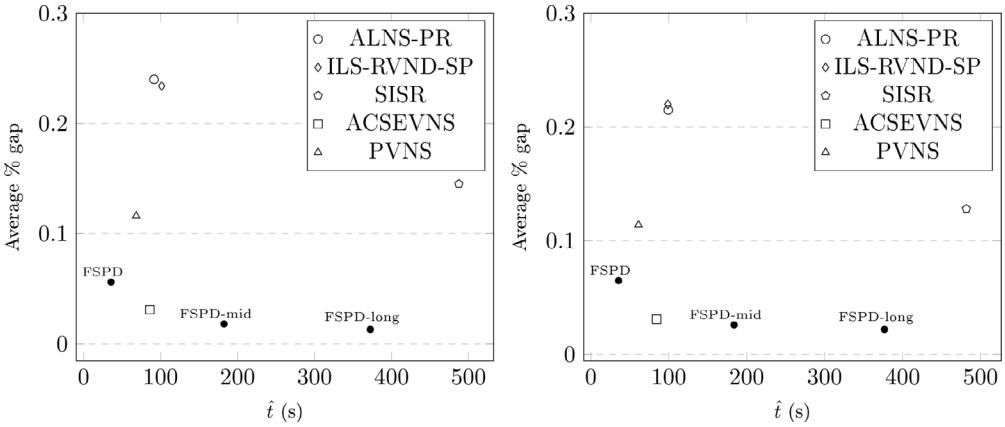


Fig. 4. Illustration of the results on the complete (Salhi and Nagy, 1999) X (left) and Y (right) datasets.

Table 4
Results on the Dethloff (2001) and Montané and Galvão (2006) datasets.

Algorithm	Montane			Dethloff		
	Avg	Time[a]	Time	Avg	Time[a]	Time
ALNS-PR	0.374	–	856.708	0.000[a]	–	9.167
ILS-RVND-SP	0.077	–	1933.605	–	–	–
h_PSO	–	–	–	0.000[a]	–	1.598
ACSEVNS	–	–	–	0.011	–	2.926
ILS-RVND-TA	0.890	–	452.206	0.084	–	7.425
VLBR	0.469[a]	21.573	–	0.000[a]	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

[a] Average gaps are actually the best gap obtained along several runs.

Table 5
Results on the first seven instances of Salhi and Nagy (1999) CMTH, CMTQ, and CMTT datasets.

Algorithm	H			Q			T		
	Avg	Time[a]	Time	Avg	Time[a]	Time	Avg	Time[a]	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[a]	–	90.211	0.215[a]	–	101.235	0.174[a]	–	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

[a] Average gaps are actually the best gap obtained along several runs.

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 demand. The three resulting datasets are denoted as CMTH, CMTQ, and CMTT, respectively. Results for these datasets are reported in Table 5 for the first seven instances without maximum distance constraint, and in Table 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.

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 the existing algorithms, including FSPD, prove able to find

Table 6
Results on the complete (Salhi and Nagy, 1999) CMTH, CMTQ, and CMTT datasets.

Algorithm	H			Q			T		
	Avg	Time[a]	Time	Avg	Time[a]	Time	Avg	Time[a]	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

[a] Average gaps are actually the best gap obtained along several runs.

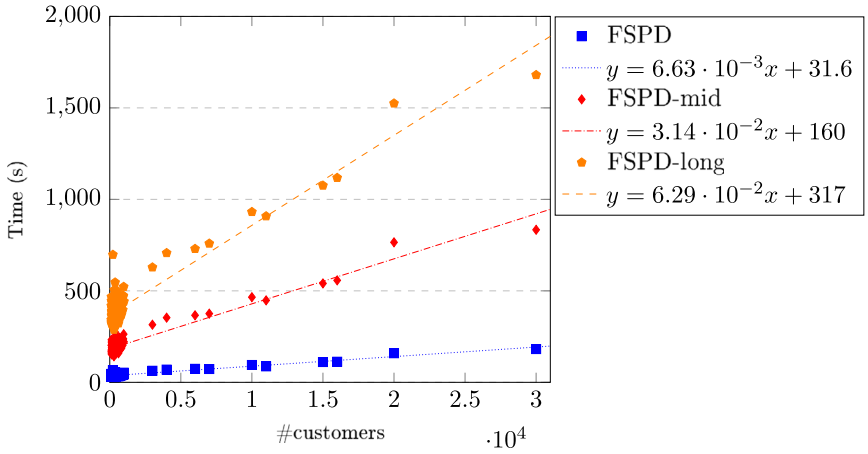


Fig. 5. Computing time required by different FSPD versions as a function of problem size for X and XXL VRPSPD problem instances.

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 1000 customers and the XXL instances proposed by Arnold et al. (2019), including ten instances ranging from 3000 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 Fig. 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.

5.4.1. VRPSPD x and XXL benchmark instances

In Table 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 5.3.1. The same procedure has been applied to the XXL instances of Arnold et al. (2019) obtaining the VRPSPD instances sets XXLX and XXLY for which the results are given in Table 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

Table 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 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

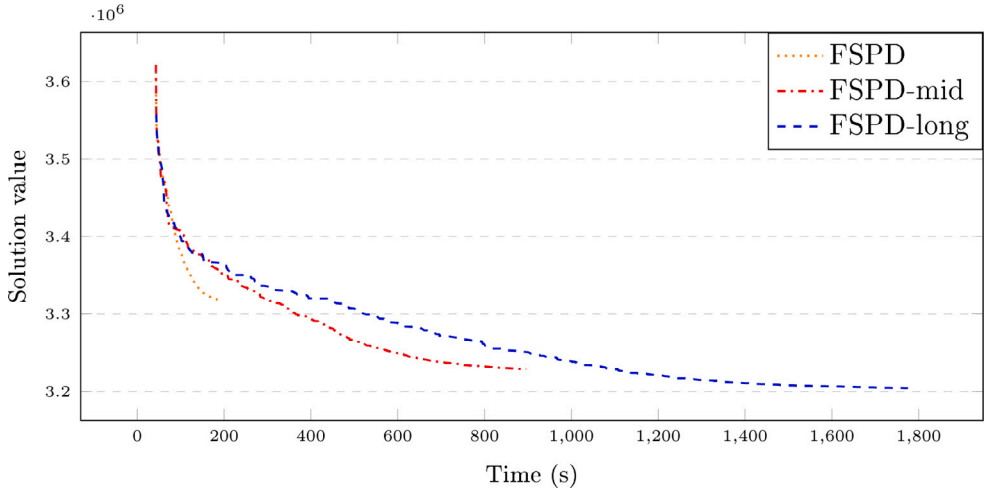


Fig. 6. Typical evolution over time of the best solution found for XXL instances. The data refers to instance Flanders2X with seed 0. The different convergence speed of the algorithms is due to the simulated annealing criteria whose annealing schedule is based on the total number of iterations while the initial and final temperatures are kept fixed.

benefit in terms of gap reduction. The average time for X instances is below 0.5, 2.5, and 5 min for FSPD, FSPD-mid, and FSPD-long, 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 Fig. 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 min for FSPD, FSPD-mid, and FSPD-long, respectively.

5.4.2. VRPMPD benchmark instances

As for the VRPSPD large-scale instances, six new datasets have been generated using the same approach described in Section 5.3.2. 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 9.

Similarly, starting from the XXL instances for CVRP proposed by Arnold et al. (2019) we generated datasets XXLH, XXLQ, and XXLT, for which the results are presented in Table 10.

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

Table 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

Table 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

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.

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 4.2.3. This particular aspect solely affects the 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 et al. (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.

We tested the three variants with recreate-effort values of 50 and 100, which display the best tradeoffs between quality and computing time as shown in Section 6.2.2. The results of our testing are reported in Table 11 and in Fig. 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.

Table 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	10 001	114.597	94.322	126.418	164.403	123.085	153.062
Ghent2	11 001	257.955	86.567	120.883	455.457	110.374	142.928
Brussels1	15 001	126.798	110.382	124.407	192.600	139.499	152.215
Brussels2	16 001	244.423	111.574	123.324	424.538	136.827	148.378
Flanders1	20 001	174.658	158.814	172.003	262.721	190.944	205.542
Flanders2	30 001	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**

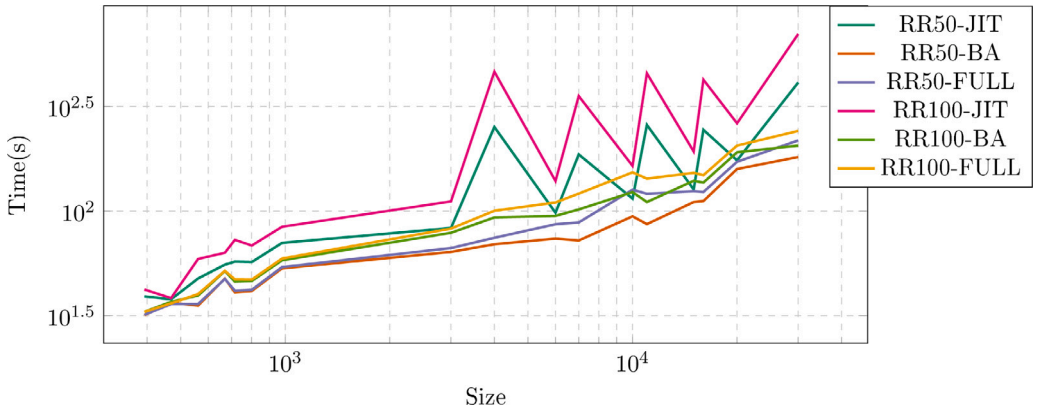


Fig. 7. Plots of the average computing time with respect to instance size for the three REF preprocessing techniques.

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 4.2.4.

6.2.1. 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 4.2.4, 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.

Table 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	10 001	1.696	1.440	1.441	1.440	1.595	1.424	1.428	1.424
Ghent2	11 001	2.924	2.274	2.292	2.274	2.647	2.198	2.201	2.198
Brussels1	15 001	2.439	2.044	2.030	2.044	2.279	2.047	2.050	2.047
Brussels2	16 001	3.469	2.752	2.753	2.752	3.111	2.572	2.591	2.572
Flanders1	20 001	1.860	1.149	1.154	1.149	1.619	1.212	1.213	1.212
Flanders2	30 001	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 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	10 001	78.463	526.760	94.322	93.810	86.429	1009.430	123.085	121.309
Ghent2	11 001	85.059	116.448	86.567	87.132	93.868	253.690	110.374	111.445
Brussels1	15 001	99.824	356.427	110.382	109.748	108.559	784.853	139.499	138.058
Brussels2	16 001	109.082	142.812	111.574	111.889	117.931	379.514	136.827	137.285
Flanders1	20 001	144.216	518.407	158.814	157.883	153.589	1027.148	190.944	189.886
Flanders2	30 001	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**

- 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 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. Tables 14 and 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. Fig. 8 shows how the quality is affected by different neighbor counting methodologies while Fig. 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 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

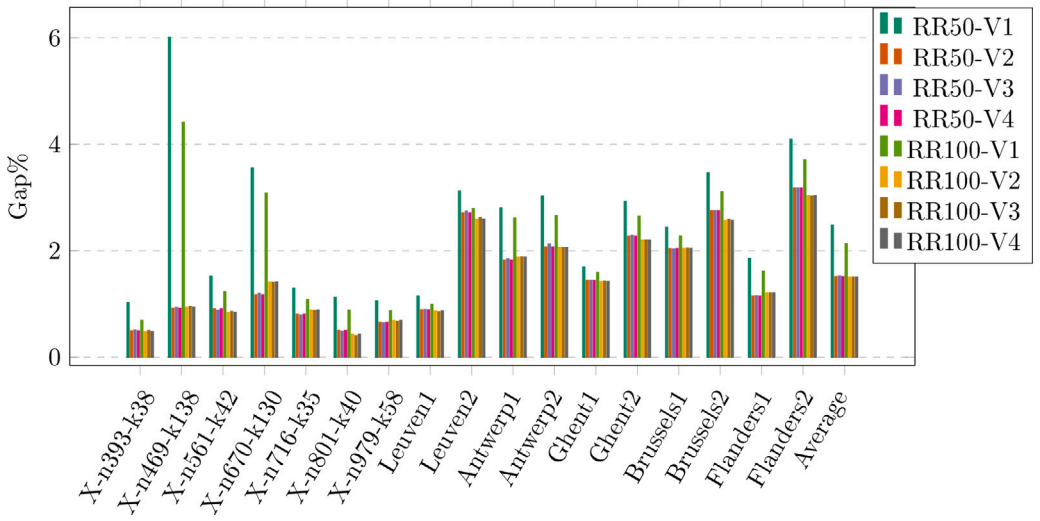


Fig. 8. Plots of the average gaps sorted by size for the 4 recreate variants with recreate effort of 50 and 100. For each instance, the values obtained for the 5 H, Q, T, X, and Y are averaged together.

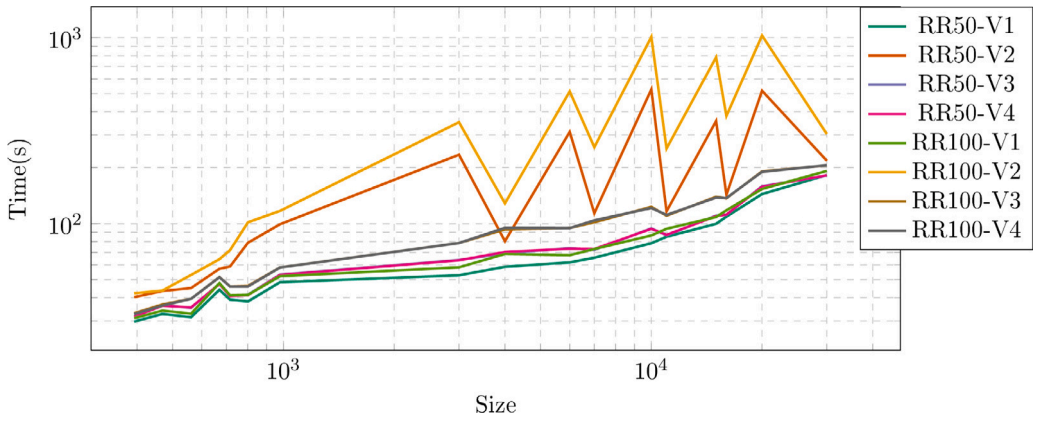


Fig. 9. Plots of the average computing time with respect to instance size for the 4 recreate variants with recreate effort of 50 and 100. For each instance, the values obtained for the 5 H, Q, T, X, and Y are averaged together.

to be particularly effective in improving the quality of the first version with V3 (Table 12) and in accelerating the second version with V4 (Table 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.

Table 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	10 001	1.551	1.466	1.433	1.440	1.424	1.500	1.497	1.569
Ghent2	11 001	2.930	2.749	2.459	2.274	2.198	2.165	2.187	2.165
Brussels1	15 001	2.251	2.157	2.041	2.044	2.047	2.107	2.131	2.235
Brussels2	16 001	3.481	3.238	2.946	2.752	2.572	2.525	2.575	2.590
Flanders1	20 001	1.282	1.179	1.122	1.149	1.212	1.287	1.328	1.434
Flanders2	30 001	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**

Table 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	10 001	66.414	68.478	77.513	93.558	121.560	171.436	213.405	235.539
Ghent2	11 001	66.160	69.183	76.095	87.107	111.546	194.489	324.823	504.103
Brussels1	15 001	85.665	88.211	96.026	110.744	138.808	206.171	282.118	387.810
Brussels2	16 001	91.481	94.439	101.356	112.080	137.055	218.398	356.726	759.884
Flanders1	20 001	131.096	133.115	141.788	158.230	190.755	278.504	405.867	679.633
Flanders2	30 001	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**

6.2.2. 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).

Tables 14 and 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 Figs. 10 and 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.

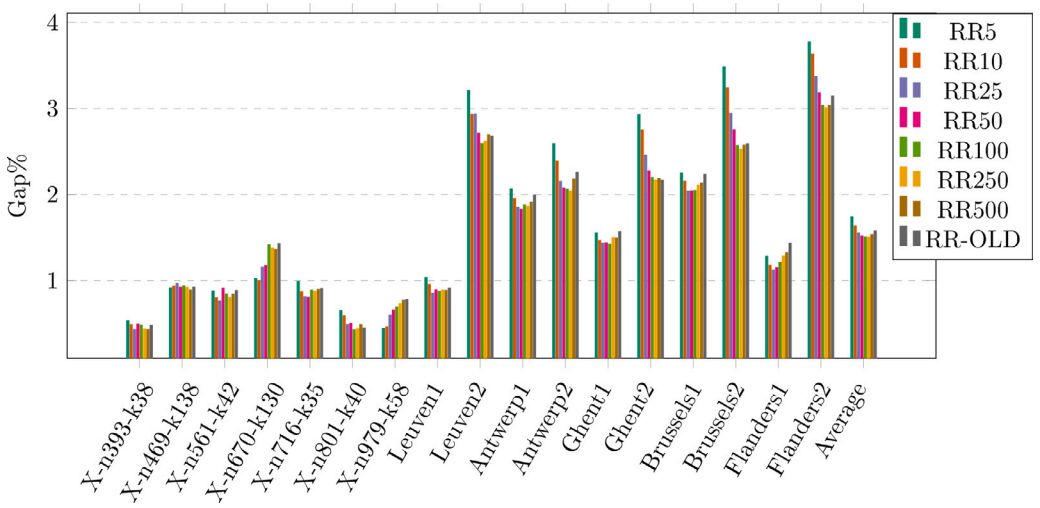


Fig. 10. Plots of the average gaps sorted by size for the different levels of recreate effort and FILO's original recreate technique.

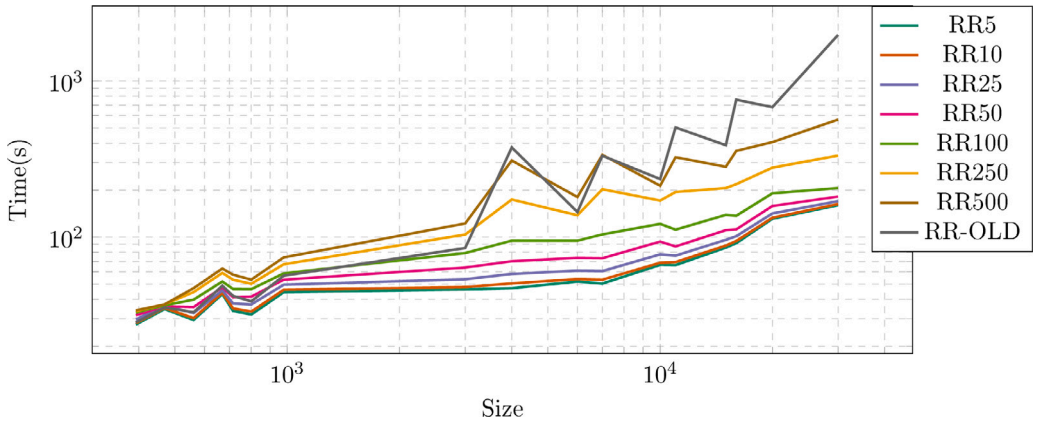


Fig. 11. Plots of the average computing time with respect to instance size for the different levels of recreate effort and FILO's original recreate technique.

Table 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

6.2.3. 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 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.

We observed remarkable robustness in our approach, as shown in Table 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,

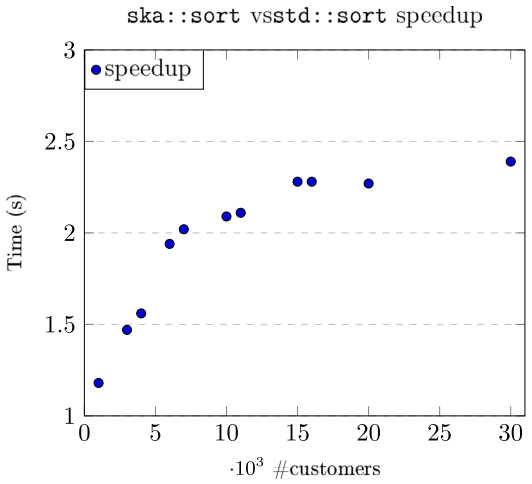


Fig. 12. Speedup obtained substituting the C++ standard library sort algorithm, with Malte Skarupke's radix-sort implementation.

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.

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.

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 et al. (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. Fig. 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 1000 customers, and each XXL instance proposed by Arnold et al. (2019), including up to 30,000 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.

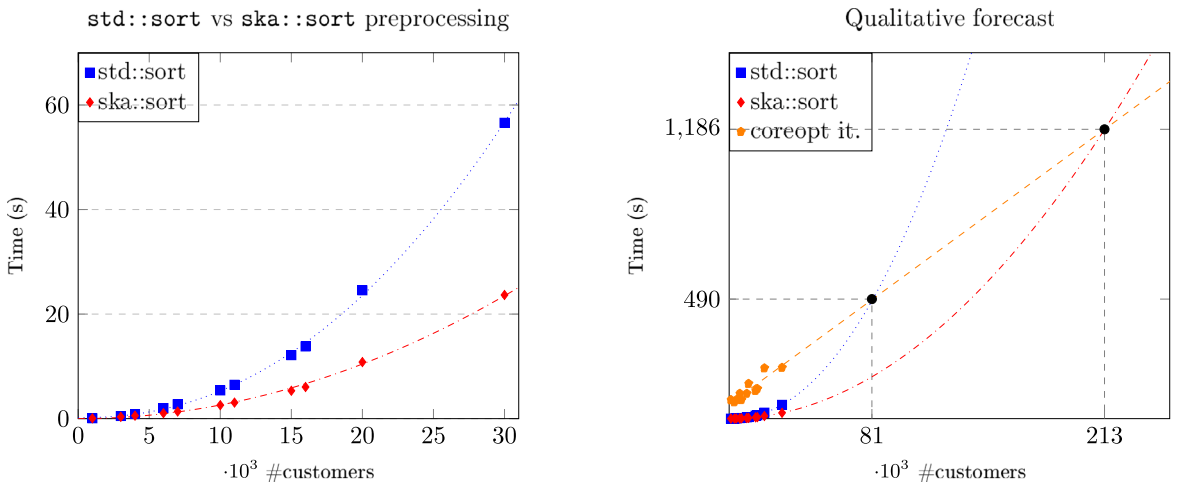


Fig. 13. On the left, time comparison between preprocessing based on C++ standard library sort algorithm (blue) and Malte Skarupke's radix-sort implementation (red). For both series, the trend-line is also reported. On the left, the same plot at a different scale with the addition of the time needed 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. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

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 Fig. 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.

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 and a detailed analysis of its components shows that each of them positively contributes to the overall efficiency and effectiveness of the approach. Remarkably, results show that FSPD reaches good quality solutions with 100,000 core optimization iterations while remaining below a computing time of 0.5 min.

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.

Table A.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**

CRediT authorship contribution statement

Francesco Cavaliere: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Luca Accorsi:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Conceptualization. **Demetrio Laganà:** Writing – review & editing, Writing – original draft, Validation, Methodology, Conceptualization. **Roberto Musmanno:** Writing – review & editing, Writing – original draft, Methodology, Conceptualization. **Daniele Vigo:** Writing – review & editing, Writing – original draft, Validation, Supervision, Project administration, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work of Francesco Cavaliere, Luca Accorsi and Daniele Vigo was partially funded by the U.S. Air Force Office of Scientific Research [Award FA8655-21-1-7046], the EU Project H2020 TUPLES (Grant id No. 101070149) and by the University of Calabria through Project Grant CIG8929018ABE.

Appendix. Complete results

A.1. VRPSPD instances

See Tables A.17–A.24.

A.2. VRPMPD instances

See Tables A.25–A.33.

Table A.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 A.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 A.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	11 047.19	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 A.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	19 781.0	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	17 156.0	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	11 813.0	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	10 238.0	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	10 120.0	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	40 033.0	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	18 933.0	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	11 182.0	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	11 924.0	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	31 022.0	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	17 226.0	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	12 041.0	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	11 699.0	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	14 173.0	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	31 714.0	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	29 553.0	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	18 461.0	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	17 194.0	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	13 287.0	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	32 656.0	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	36 513.0	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	14 985.0	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	20 609.0	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	83 037.0	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	29 633.0	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	19 563.0	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	15 284.0	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	18 359.0	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	51 806.0	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	23 839.0	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	26 275.0	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	14 339.0	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	18 246.0	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	49 863.0	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	26 257.0	0.366	0.713	30.311	26.658	0.000	0.517	158.582	114.461	0.263	0.487	309.083	192.907

(*continued on next page*)

Table A.21 (*continued*).

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
X-n275-k28	275	16 937.0	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	22 688.0	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	14 443.0	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	66 904.0	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	34 845.0	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	24 601.0	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	16 113.0	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	18 142.0	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	67 716.0	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	50 180.0	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	22 567.0	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	20 292.0	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	21 492.0	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	95 486.0	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	31 239.0	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	18 536.0	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	31 898.0	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	18 246.0	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	101 172.0	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	42 851.0	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	29 201.0	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	54 341.0	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	15 043.0	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	72 085.0	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	46 224.0	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	27 125.0	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	35 679.0	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	20 957.0	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	135 415.0	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	60 589.0	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	48 961.0	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	51 691.0	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	19 879.0	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	95 946.0	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	64 843.0	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	55 523.0	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	32 607.0	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	33 826.0	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	126 539.0	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	72 328.0	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	43 443.0	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	43 999.0	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	41 913.0	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	78 413.0	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	97 671.0	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	50 534.0	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	54 279.0	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	35 640.0	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	98 507.0	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	50 990.0	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	71 818.0	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	46 580.0	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	47 741.0	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	111 809.0	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	122 876.0	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	60 399.0	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	66 901.0	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	37 758.0	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	217 951.0	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	93 219.0	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	59 054.0	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	97 862.0	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	49 219.0	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 A.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	19 781.0	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	17 162.0	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	11 813.0	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	10 238.0	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	10 120.0	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	40 033.0	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	18 933.0	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	11 182.0	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	11 924.0	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	30 970.0	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	17 226.0	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	12 041.0	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	11 699.0	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	14 173.0	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	31 722.0	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	29 545.0	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	18 450.0	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	17 194.0	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	13 287.0	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	32 592.0	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	36 511.0	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	14 985.0	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	20 574.0	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	83 029.0	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	29 691.0	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	19 586.0	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	15 284.0	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	18 359.0	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	51 832.0	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	23 841.0	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	26 267.0	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	14 339.0	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	18 248.0	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	50 000.0	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	26 287.0	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	16 949.0	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	22 688.0	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	14 444.0	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	67 020.0	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	34 745.0	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	24 562.0	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	16 117.0	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	18 143.0	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	67 781.0	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	50 243.0	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	22 579.0	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	20 319.0	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	21 525.0	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	95 361.0	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	31 209.0	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	18 518.0	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	31 924.0	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	18 244.0	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	101 104.0	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	42 821.0	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	29 209.0	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	54 339.0	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	15 043.0	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	72 128.0	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	46 248.0	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	27 049.0	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	35 737.0	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	20 964.0	0.091	0.279	39.121	33.407	0.005	0.117	189.499	164.562	0.000	0.077	385.905	335.240

(*continued on next page*)

Table A.22 (*continued*).

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
X-n469-k138	469	135 368.0	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	60 581.0	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	48 961.0	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	51 689.0	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	19 889.0	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	95 980.0	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	64 904.0	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	55 522.0	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	32 583.0	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	33 846.0	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	126 581.0	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	72 296.0	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	43 463.0	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	43 946.0	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	41 906.0	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	78 386.0	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	97 628.0	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	50 523.0	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	54 293.0	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	35 634.0	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	98 482.0	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	50 983.0	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	71 805.0	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	46 551.0	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	47 729.0	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	111 646.0	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	123 100.0	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	60 370.0	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	66 926.0	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	37 836.0	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	217 877.0	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	93 117.0	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	58 998.0	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	97 882.0	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	49 013.0	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**

Table A.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	137 432.0	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	82 194.0	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	346 563.0	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	214 177.0	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	10 001	365 418.0	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	11 001	214 337.0	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	15 001	376 294.0	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	16 001	279 692.0	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	20 001	5 217 403.0	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	30 001	3 201 225.0	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 A.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	137 644.0	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	82 346.0	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	347 231.0	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	214 035.0	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	10 001	365 724.0	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	11 001	215 520.0	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	15 001	375 939.0	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	16 001	279 479.0	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	20 001	5 221 438.0	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	30 001	3 197 711.0	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 A.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 A.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 A.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 A.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	19 265.0	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	14 873.0	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	11 153.0	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	10 156.0	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	33 107.0	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	18 054.0	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	10 834.0	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	11 750.0	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	27 230.0	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	14 699.0	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	11 077.0	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	11 175.0	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	13 965.0	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	27 443.0	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	28 961.0	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	15 584.0	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	16 029.0	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	10 400.0	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	28 575.0	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	33 129.0	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	14 271.0	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	19 416.0	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	63 552.0	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	25 194.0	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	17 212.0	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	14 374.0	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	17 854.0	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	47 216.0	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	22 842.0	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	23 880.0	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	13 587.0	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	17 989.0	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	43 351.0	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	22 557.0	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	13 957.0	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	21 920.0	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	13 914.0	0.122	0.474	34.991	21.179	0.007	0.216	178.567	84.929	0.000	0.155	361.437	180.127

(*continued on next page*)

Table A.28 (*continued*).

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
X-n289-k60	289	53 645.0	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	29 439.0	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	22 601.0	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	15 311.0	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	17 909.0	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	52 922.0	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	42 704.0	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	20 292.0	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	19 004.0	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	20 404.0	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	78 262.0	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	26 616.0	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	17 065.0	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	30 994.0	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	15 534.0	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	79 505.0	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	39 670.0	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	24 675.0	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	37 559.0	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	13 762.0	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	62 929.0	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	40 564.0	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	23 932.0	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	34 134.0	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	16 389.0	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	121 262.0	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	50 929.0	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	39 991.0	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	38 142.0	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	18 863.0	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	87 859.0	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	53 430.0	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	49 629.0	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	28 190.0	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	28 756.0	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	104 695.0	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	62 516.0	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	37 394.0	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	35 778.0	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	38 426.0	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	58 522.0	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	86 862.0	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	42 293.0	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	47 773.0	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	26 624.0	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	78 945.0	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	45 591.0	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	67 064.0	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	44 079.0	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	43 900.0	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	88 466.0	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	106 845.0	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	52 058.0	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	55 683.0	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	34 905.0	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	176 175.0	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	80 655.0	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	50 863.0	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	67 077.0	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	44 761.0	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 A.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	21 901.0	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	20 203.0	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	12 855.0	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	11 208.0	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	11 540.0	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	43 331.0	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	21 898.0	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	12 106.0	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	13 424.0	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	34 428.0	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	17 004.0	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	13 543.0	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	12 186.0	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	17 258.0	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	36 146.0	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	35 442.0	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	20 123.0	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	19 828.0	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	13 620.0	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	34 301.0	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	44 628.0	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	16 328.0	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	24 736.0	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	89 853.0	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	32 722.0	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	21 070.0	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	16 458.0	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	21 940.0	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	63 944.0	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	29 446.0	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	30 356.0	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	16 000.0	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	21 715.0	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	56 216.0	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	28 147.0	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	17 468.0	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	27 553.0	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	16 677.0	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	71 464.0	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	36 682.0	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	27 344.0	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	17 957.0	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	21 418.0	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	73 302.0	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	61 527.0	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	24 881.0	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	23 159.0	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	25 273.0	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	106 481.0	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	33 844.0	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	20 558.0	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	40 159.0	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	18 738.0	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	111 347.0	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	50 170.0	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	30 847.0	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	51 626.0	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	16 569.0	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	84 151.0	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	52 307.0	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	29 742.0	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	44 048.0	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	19 890.0	0.045	0.295	37.743	34.515	0.055	0.144	192.685	172.872	0.000	0.113	387.184	330.957

(*continued on next page*)

Table A.29 (*continued*).

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
X-n469-k138	469	167 977.0	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	69 091.0	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	51 820.0	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	52 852.0	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	21 482.0	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	118 435.0	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	72 872.0	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	67 134.0	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	34 480.0	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	39 936.0	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	146 421.0	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	83 343.0	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	47 640.0	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	48 432.0	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	50 006.0	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	81 739.0	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	108 912.0	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	54 616.0	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	63 946.0	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	33 751.0	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	105 784.0	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	59 460.0	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	88 150.0	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	58 120.0	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	57 194.0	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	120 766.0	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	148 464.0	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	69 722.0	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	75 152.0	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	44 105.0	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	247 445.0	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	104 488.0	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	66 802.0	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	91 727.0	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	57 512.0	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 A.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	25 090.0	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	23 069.0	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	14 214.0	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	12 211.0	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	12 854.0	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	49 986.0	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	26 103.0	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	10 147.0	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	13 052.0	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	14 802.0	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	38 826.0	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	19 166.0	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	15 561.0	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	13 478.0	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	19 275.0	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	41 802.0	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	44 695.0	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	23 481.0	0.000	0.067	48.881	32.746	0.000	0.007	212.311	120.385	0.000	0.003	436.969	241.181

(*continued on next page*)

Table A.30 (*continued*).

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
X-n186-k15	186	22 118.0	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	15 345.0	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	40 119.0	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	53 699.0	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	18 405.0	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	28 480.0	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	10 147.0	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	106 528.0	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	37 176.0	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	23 853.0	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	18 037.0	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	24 974.0	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	75 119.0	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	34 957.0	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	35 525.0	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	17 585.0	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	24 572.0	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	67 782.0	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	32 171.0	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	19 644.0	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	31 515.0	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	18 905.0	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	87 470.0	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	43 375.0	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	31 758.0	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	20 240.0	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	24 367.0	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	85 629.0	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	71 661.0	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	28 085.0	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	25 753.0	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	28 460.0	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	125 720.0	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	38 520.0	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	23 684.0	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	46 945.0	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	21 654.0	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	134 666.0	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	59 344.0	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	35 186.0	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	61 902.0	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	18 625.0	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	99 301.0	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	60 716.0	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	33 453.0	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	51 028.0	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	22 510.0	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	202 649.0	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	81 819.0	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	60 730.0	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	62 729.0	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	23 056.0	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	138 782.0	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	85 175.0	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	79 026.0	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	39 267.0	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	46 237.0	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	172 982.0	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	99 152.0	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	54 298.0	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	57 242.0	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	58 278.0	0.182	0.408	40.681	38.793	0.014	0.166	204.895	193.096	0.000	0.104	406.613	370.436

(*continued on next page*)

Table A.30 (*continued*).

Instance	Size	BKS	FSPD				FSPD-mid				FSPD-long			
			Best	Avg	Time*	Time	Best	Avg	Time*	Time	Best	Avg	Time*	Time
X-n655-k131	655	96 769.0	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	133 617.0	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	62 677.0	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	74 837.0	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	39 434.0	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	123 851.0	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	71 545.0	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	103 657.0	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	66 085.0	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	67 122.0	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	144 690.0	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	175 138.0	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	81 204.0	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	90 507.0	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	49 904.0	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	298 667.0	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	120 742.0	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	78 057.0	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	108 570.0	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	66 711.0	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 A.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	114 189.0	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	76 187.0	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	277 391.0	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	185 142.0	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	10 001	271 224.0	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	11 001	163 972.0	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	15 001	302 630.0	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	16 001	222 184.0	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	20 001	4 070 908.0	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	30 001	2 692 883.0	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 A.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	151 052.0	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	92 568.0	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	372 610.0	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	234 127.0	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	10 001	365 274.0	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	11 001	209 515.0	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	15 001	395 568.0	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	16 001	282 942.0	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	20 001	5 580 410.0	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	30 001	3 513 440.0	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 A.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	177 105.0	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	103 576.0	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	434 312.0	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	270 063.0	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	10 001	429 480.0	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	11 001	238 958.0	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	15 001	461 696.0	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	16 001	321 980.0	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	20 001	6 599 087.0	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	30 001	4 053 751.0	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**

References

Accorsi, L., Vigo, D., 2021. A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems. Transp. Sci. 55 (4), 832–856. http://dx.doi.org/10.1287/trsc.2021.1059.

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.

Arenas, I., Sánchez, A., Armando, C., Solano, L., Medina, L., 2017. CVRPTW model applied to the collection of food donations. In: Proceedings of the International Conference on Industrial Engineering and Operations Management. Bogotá D.C., pp. 1307 – 1314.

Arnold, F., Gendreau, M., Sörensen, K., 2019. Efficiently solving very large-scale routing problems. Comput. Oper. Res. 107, 32–42. http://dx.doi.org/10.1016/j.cor.2019.03.006.

Beek, O., Raa, B., Dullaert, W., Vigo, D., 2018. An efficient implementation of a static move descriptor-based local search heuristic. Comput. Oper. Res. 94, 1–10. http://dx.doi.org/10.1016/j.cor.2018.01.006.

Bentley, J.L., 1990. K-d trees for semidynamic point sets. In: Proceedings of the Sixth Annual Symposium on Computational Geometry. pp. 187–197.

Christiaens, J., Vanden Berghe, G., 2020. Slack induction by string removals for vehicle routing problems. Transp. Sci. 54 (2), 417–433. http://dx.doi.org/10.1287/trsc.2019.0914.

Christofides, N., Mingozzi, A., Toth, P., 1979. The Vehicle Routing Problem, vol. 1, Wiley Interscience, pp. 315–338.

Clarke, G., Wright, J.W., 1964. Scheduling of vehicles from a central depot to a number of delivery points. Oper. Res. 12 (4), 568–581.

Delgado-Antequera, L., Laguna, M., Pacheco, J., Caballero, R., 2020. A bi-objective solution approach to a real-world waste collection problem. J. Oper. Res. Soc. 71 (2), 18 – 194. http://dx.doi.org/10.1080/01605682.2018.1545520.

Desaulniers, G., Desrosiers, J., Solomon, M.M., Soumis, F., Villeneuve, D., et al., 1998. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In: Fleet Management and Logistics. Springer, pp. 57–93. http://dx.doi.org/10.1007/978-1-4615-5755-5_3.

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

Gajpal, Y., Abad, P., 2009. Multi-ant colony system (MACS) for a vehicle routing problem with backhauls. European J. Oper. Res. 196 (1), 102–117. http://dx.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. Comput. Ind. Eng. 65 (1), 39–53. http://dx.doi.org/10.1016/j.cie.2012.01.005.

Hamzadayı, A., Baykasoğlu, A., Akpınar, Ş., 2020. Solving combinatorial optimization problems with single seekers society algorithm. Knowl.-Based Syst. 201–202, 106036. http://dx.doi.org/10.1016/j.knosys.2020.106036.

Helsgaun, K., 2000. An effective implementation of the Lin–Kernighan traveling salesman heuristic. European J. Oper. Res. 126 (1), 106–130. http://dx.doi.org/10.1016/S0377-2217(99)00284-2.

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. http://dx.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. Comput. Oper. Res. 115, 104858. http://dx.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. 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 J. Comput. 20 (2), 270–287.

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

Kindervater, G.A., Savelsbergh, M.W., 1997. 10. Vehicle routing: handling edge exchanges. In: Local Search in Combinatorial Optimization. John Wiley & Son, pp. 337–360. http://dx.doi.org/10.1515/9780691187563-013.

Koç, Ç., Laporte, G., 2018. Vehicle routing with backhauls: Review and research perspectives. Comput. Oper. Res. 91, 79–91. http://dx.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. Comput. Oper. Res. 122, 104987. http://dx.doi.org/10.1016/j.cor.2020.104987.

Lourenço, H.R., Martin, O.C., Stützle, T., 2003. Iterated local search. In: Handbook of Metaheuristics. Springer US, pp. 320–353.

Min, H., 1989. The multiple vehicle routing problem with simultaneous delivery and pick-up points. Transp. Res. A 23 (5), 377–386. http://dx.doi.org/10.1016/0191-2607(89)90085-X.

Montané, A., Galvão, R., 2006. A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. Comput. Oper. Res. 33 (3), 595–619. http://dx.doi.org/10.1016/j.cor.2004.07.009.

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. http://dx.doi.org/10.1016/j.procir.2015.12.040.

Olgun, B., Koç, Ç., Altıparmak, F., 2021. A hyper heuristic for the green vehicle routing problem with simultaneous pickup and delivery. Comput. Ind. Eng. 153, 107010. http://dx.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 Syst. Appl. 202, 117401. http://dx.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 Syst. Appl. 165, 113959. http://dx.doi.org/10.1016/j.eswa.2020.113959.

PassMark® Software, 2020. Professional CPU benchmarks. URL: https://www.passmark.com/index.html.

Polat, O., 2017. A parallel variable neighborhood search for the vehicle routing problem with divisible deliveries and pickups. Comput. Oper. Res. 85, 71–86. http://dx.doi.org/10.1016/j.cor.2017.03.009.

Polat, O., Kalayci, C.B., Kulak, O., Günther, H.-O., 2015. A perturbation based variable neighborhood search heuristic for solving the vehicle routing problem with simultaneous pickup and delivery with time limit. European J. Oper. Res. 242 (2), 369–382. http://dx.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 J. Oper. Res. 287 (1), 76–89. http://dx.doi.org/10.1016/j.ejor.2020.04.047.

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

Salhi, S., Nagy, G., 1999. A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. J. Oper. Res. Soc. 50 (10), 1034–1042. http://dx.doi.org/10.1057/palgrave.jors.2600808.

Santana, J.C., Guerhardt, F., Franzini, C., Ho, L.L., Júnior, S.R.R., Cânovas, G., Kenji Yamamura, C., Vanalle, R.M., Berssaneti, F., 2021. Refurbishing and recycling of cell phones as a sustainable process of reverse logistics: A case study in Brazil. J. Clean. Prod. 283, 124585. http://dx.doi.org/10.1016/j.jclepro.2020.124585.

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. http://dx.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 J. Oper. Res. 263 (2), 493–509. http://dx.doi.org/10.1016/j.ejor.2017.04.059.

Shafiee, R.E., Ghomi, S.F., Sajadieh, M., 2021. Reverse logistics network design for product reuse, remanufacturing, recycling and refurbishing under uncertainty. J. Manuf. Syst. 60, 473–486. http://dx.doi.org/10.1016/j.jmsy.2021.06.012.

Simsir, F., Ekmekci, D., 2019. A metaheuristic solution approach to capacitied vehicle routing and network optimization. Eng. Sci. Technol. Int. J. 22 (3), 727–735. http://dx.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.

Subramanian, A., Uchoa, E., Ochi, L.S., 2013. A hybrid algorithm for a class of vehicle routing problems. Comput. OR 40 (10), 2519–2531. http://dx.doi.org/10.1016/j.cor.2013.01.013.

Taniguchi, E., Heijden, R.V.D., 2000. An evaluation methodology for city logistics. Transp. Rev. 20 (1), 65–90. http://dx.doi.org/10.1080/014416400295347.

Taniguchi, E., Thompson, R., Yamada, T., van Duin, R., 2001. City Logistic – Network Modelling and Intelligent Transport Systems. Pergamon, Amsterdam, http://dx.doi.org/10.1108/9780585473840.

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

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., Subramanian, A., 2017. New benchmark instances for the capacitated vehicle routing problem. European J. Oper. Res. 257 (3), 845–858. http://dx.doi.org/10.1016/j.ejor.2016.08.012.

Vidal, T., Crainic, T., Gendreau, M., Prins, C., 2014. A unified solution framework for multi-attribute vehicle routing problems. European J. Oper. Res. 234 (3), 658–673. http://dx.doi.org/10.1016/j.ejor.2013.09.045.

Wassan, N., Nagy, G., Ahmadi, S., 2008. A heuristic method for the vehicle routing problem with mixed deliveries and pickups. J. Sched. 11 (2), 149 – 161. http://dx.doi.org/10.1007/s10951-008-0055-y.

Wilcoxon, F., 1945. Individual comparisons by ranking methods. Biom. Bull. 1 (6), 80–83.

Zachariadis, E.E., Kiranoudis, C.T., 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.

Zachariadis, E.E., Tarantilis, C.D., Kiranoudis, C.T., 2010. An adaptive memory methodology for the vehicle routing problem with simultaneous pick-ups and deliveries. European J. Oper. Res. 202 (2), 401–411. http://dx.doi.org/10.1016/j.ejor.2009.05.015.