SN Computer Science (2024) 5:225
https://doi.org/10.1007/s42979-023-02540-3

ORIGINAL RESEARCH

An Optimization-Based Decision Support System for Multi-trip Vehicle Routing Problems

Mirko Cavecchia[1] · Thiago Alves de Queiroz[2] · Manuel Iori[1] · Riccardo Lancellotti[3] · Giorgio Zucchi[4]

Received: 30 September 2023 / Accepted: 1 December 2023

Abstract

Decision support systems (DSS) are used daily to make complex and hard decisions. Developing a DSS is not an easy task and may require combining different approaches to reach accurate and timely responses. In this paper, we present a DSS based on a micro-service architecture that we developed to handle a variant of the vehicle routing problem. The DSS has been implemented for a service company operating in the field of pharmaceutical distribution, and it helps decision-makers define the routes that different types of vehicles need to perform during the day to serve the customers' demands. The underlying optimization problem assumes that a vehicle can perform multiple routes daily and is constrained to operate within a given time horizon. Customers are characterized by hard time windows on the delivery times. The proposed DSS first handles geo-referencing and distance calculation tasks. Then, it invokes a two-step optimization approach in which vehicle routes are generated and combined to reduce the number of vehicles used. For the latter task, we propose and evaluate four solution methods: two greedy heuristics, a metaheuristic, and a mathematical model. All the methods are applied to solve real and randomly generated instances, showing that the metaheuristic algorithm is superior to the others in terms of solution quality and computing time. The company had a very positive feedback on the proposed DSS and is now using it to support its daily operations.

Keywords Decision support system · Micro-service, Multi-trip vehicle routing problem · Iterated local search · Mathematical optimization problem

Introduction

The vehicle routing problem (VRP) is a difficult combinatorial optimization problem that has gained considerable attention from researchers, given its practical applications in different fields such as logistics and transportation. The main objective of the VRP is to find a set of routes for a fleet of vehicles that minimizes the total cost required to serve the demands of a set of geographically dispersed customers. Due to its NP-hard nature, advanced heuristic and metaheuristic algorithms have been proposed to find approximate but good-quality solutions of large-size VRP instances within limited computing times [1–3].

This paper addresses a variant of the VRP known as the multi-trip vehicle routing problem with time windows

This article is part of the topical collection "Recent Trends on Enterprise Information Systems" guest edited by Joaquim Filipe, Michał Śmiałek, Alexander Brodsky and Slimane Hammoudi.

✉ Mirko Cavecchia
mirko.cavecchia@unimore.it

Thiago Alves de Queiroz
taq@ufcat.edu.br

Manuel Iori
manuel.iori@unimore.it

Riccardo Lancellotti
riccardo.lancellotti@unimore.it

Giorgio Zucchi
giorgio.zucchi@coopservice.it

[1] Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, 42122 Reggio Emilia, Italy

[2] Institute of Mathematics and Technology, Federal University of Catalão, 75704-020, Catalão-GO, Brazil

[3] Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, 41125 Modena, Italy

[4] R &D department, Coopservice S.coop.p.A., 42122 Reggio Emilia, Italy

Published online: 20 January 2024

(MTVRPTW), which arises from a real-world application at Coopservice Soc.coop.p.A., an Italian large service provider company. In this problem, the routes to be determined must respect time window constraints, which impose each vehicle to serve a customer within specified time intervals. Besides that, vehicles with different capacities can depart from different depots, and each vehicle can perform multiple routes. Each route must start and end at the same depot. The objective is to minimize the number of vehicles used to fulfill the customers' demands.

To tackle the MTVRPTW, we propose a decision support system (DSS) designed to assist the company in its operations. The DSS is a web application composed of different micro-services. The micro-service architecture guarantees scalability with respect to the PC workload (i.e., new instances of the micro-services can be added to cope with traffic surges) and ensures interoperability with other systems that can be easily interfaced with the DSS. The orchestration of the micro-services has a single entry point where the user can enter the addresses of depots and customers. The addresses are then geo-referenced, and the distance between each couple of addresses is computed. Next, the DSS generates a set of feasible routes using the metaheuristic approach defined in [4]. However, since the vehicles can perform more than one route daily, the routes generated by the metaheuristic need to be aggregated to minimize the fleet size. To this end, we consider two scenarios in accordance with the company. In the first scenario, the starting time of each route is fixed, and we propose a greedy algorithm that sorts routes based on their departure time, assigning them to vehicles while respecting the total driving time. In the second, more flexible scenario, the starting time of each route can be moved between the earliest and latest possible starting times imposed by the time windows of the customers visited in the route. In this case, we extend the greedy algorithm to assign routes to vehicles as early as possible, and we propose an iterated local search (ILS) metaheuristic. The ILS tries to improve the greedy solution by applying a perturbation step followed by a local search based on swap and insertion movements. Additionally, we propose a mixed integer linear programming (MILP) model. All these algorithms have been evaluated on real instances provided by the industrial partner, and the ILS obtained the best overall results.

To the best of our knowledge, the DSS represents a significant innovation for the reference case study provided by the industrial partner and for many other similar applications. The micro-service-based architecture is a state-of-the-art technology in the area of DSS, offering benefits in terms of scalability and interoperability. Furthermore, using multiple algorithms, which can be flexibly combined, guarantees the support for evaluating approaches suitable for different scenarios. Finally, the open architecture based on micro-services simplifies the extension of the core framework so that new algorithms suitable for other specific scenarios can be easily added.

A preliminary version of the present research has been published in [5]. This paper is a clear step ahead, providing more algorithms added to the framework and outlining a broader performance evaluation based on several real and randomly-created data sets.

The remainder of this paper is structured as follows. The next section provides an overview of the existing literature in routing problems and related decision science. The subsequent section outlines the DSS architecture followed by which a formal description of the optimization problem is provided. Then the algorithms developed to solve the MTVRPTW are detailed. The penultimate section provides a computational evaluation of the DSS concerning different case studies. Finally, concluding remarks are given.

Literature Review

This section discusses the existing literature concerning the VRP and its variants, with a specific emphasis on the MTVRPTW. Additionally, it explores the domain of DSSs and their relevance within the research context.

The VRP may consider additional characteristics and constraints, called attributes, resulting in multi-attribute VRPs. In their survey, [6] reviewed more than sixty heuristic and metaheuristic algorithms, identifying the core components of each one and examining fifteen notable problems. When multiple attributes are combined together, we obtain the so-called rich VRPs. A recent survey for rich variants was proposed by [7], who presented a literature review on hybrid methods and problems found, e.g., in food and newspaper distributions. The authors emphasized the importance of cooperative methods that combine exact and approximate algorithms and the proposal of benchmark problems, including real-life instances. For a recent, concise review of VRP and its variants, we refer to [8], while [9] reviewed the variants with profits, split deliveries, multiple commodities over time, and integrated VRPs, like location-routing problems, multi-echelon routing problems, and the routing problems with loading constraints.

The VRPTW, an extension of the classical VRP, introduces temporal constraints on customer service. Each customer must be visited within a predefined time window. This additional constraint adds complexity to the problem, making it even more challenging. Over time, starting from [10], researchers have proposed various exact and heuristic methods to tackle the VRPTW [11, 12]. The applications of the VRPTW span across various domains, including food delivery [13], electric vehicle recharging [14], and pharmaceutical product delivery [4]. In addition, [15] addressed a combined problem involving VRPTW and scheduling of trucks and drivers, whereas [16] solved a school bus routing problem using an ILS and a MILP model.

Another well-established VRP variant is the MTVRPTW, where vehicles can undertake multiple routes in a single day, with each route starting and ending at the depot. This flexibility enables better adherence to customer time windows and can lead to more efficient solutions. The MTVRPTW has been an area of significant research interest, and various algorithms have been recently proposed to address it effectively [9, 17]. Many solutions and methods for the MTVRPTW can be found in the literature. [18] solved a variant related to the dial-a-ride problem, proposing hybrid bee colony algorithms and an adaptive large neighborhood search (ALNS). The algorithms were tested on new instances, and the hybrid algorithms obtained the best results. [19] solved an MTVRPTW variant arising in the meal delivery logistics, proposing a MILP model and two heuristics, namely an ILS and an ALNS. [20] handled the multi-trip time-dependent vehicle routing problem with time windows by proposing a MILP model and a hybrid ALNS. The proposed heuristics obtained good results on a set of new instances proposed by the authors. Recently, [21] solved a multi-trip multi-distribution center VRP with lower-bound capacity constraints to minimize the number of used vehicles and maximize the number of served customers. The authors considered greedy algorithms based on savings, insertions, and sweeps to handle the lower-bound capacity constraints. An ALNS improves the solutions obtained with the greedy algorithms.

For what concerns DSSs, their application spans from business and finance to healthcare and up to education. Several criteria can be used to categorize DSSs based on their scope or the provided functionalities. A possible classification proposed in [22] divides DSSs into communications-driven, data-driven, document-driven, knowledge-driven, and model-driven. The proposed approach in this paper can be seen as a model-driven DSS as it focuses on capturing a business process that is described as the composition of multiple basic functions.

The literature on DSSs presents a large corpus investigating whether micro-service or cloud-based architectures are viable options for managing services. For example, [23] proposed a DSS to evaluate whether an application is suitable for cloud deployment. Similarly, [24] focus on micro-service-based deployments, proposing a tool to evaluate the benefits of architectural re-design following the micro-service paradigm. With respect to this topic, our study embraces the vision of a highly dynamic description of the business process, where data is not a critical asset but rather a commodity. Within this context, the choice of a micro-service architecture for deploying the DSS is motivated, as our application fits the critical parameters that make micro-services appealing, as described by [25].

Finally, we notice that our approach also fits the area of spatial-based DSS, which has been explored by [26]. The proposed DSS fits the classic DSS area with optimization based on geo-referenced data. However, even if the core problem is classic, creating an agile, flexible, and scalable system takes advantage of the most recent advances in distributed systems and cloud computing.

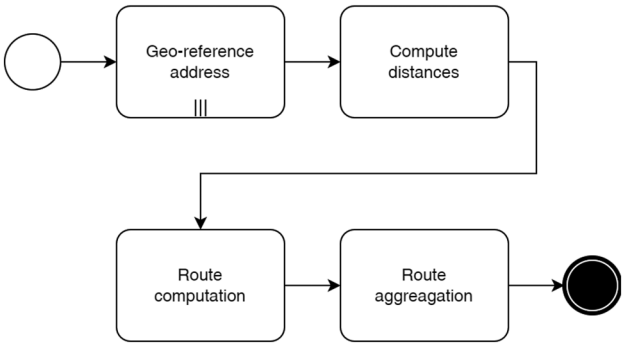


Fig. 1 BPM description of the business process

Decision Support System Architecture

We can define a DSS as an information system based on computers supporting decision-making. In this section, we outline the business process used in the proposed DSS and discuss how to implement it.

Business Process Overview

To better explain the characteristics of the proposed DSS, we start with a short description of the main tasks required to solve an MTVRPTW.

Figure 1 shows the main tasks of the considered business process according to a business process modeling (BPM) notation. The input (i.e., the white circle in the figure) is a list of points of interests (customers and depots) in the form of geo-referenced addresses (as shown in the leftmost box in the graph). The list of coordinates is further processed to create a distance matrix where each matrix element is the distance between two points (the distance can be expressed in terms of time or as a physical distance (i.e., km), depending on the metric considered for the problem). The matrix is then used as an input for the MTVRPTW, which is solved using a two-step approach. The set of optimized routes is given as output (i.e., the black circle in the figure).

The two tasks of geo-referencing and distance calculation require complex operations and rely on external services (such as APIs provided by third-party micro-services). For example, ArcGIS provides an API with Python bindings for most of its operations, as documented in [27]. However, external services typically limit the number of invocations per unit of time.

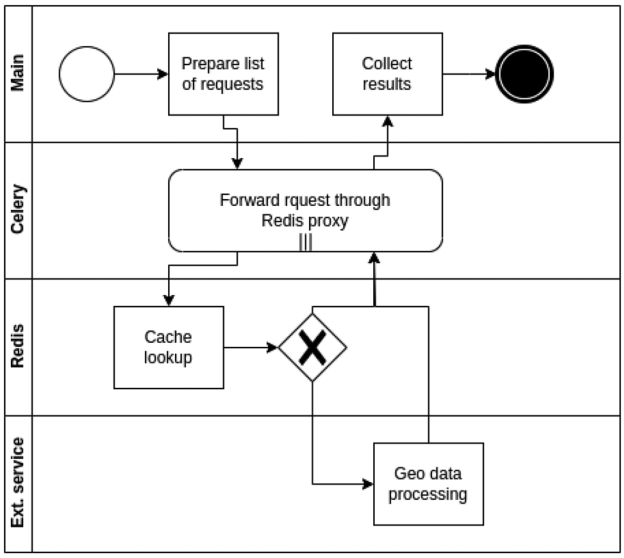


Fig. 2 Address geo-referencing and distance computation

This constraint requires a suitable architectural solution, as shown in Fig. 2. Considering geo-referencing the waypoints or computing distances, the requests are inserted in a queue of requests handled by Celery.[1] Each Celery Task ID identifies a list of requests. The Task ID can be used to track the state of a Task (i.e., how many requests have already been completed). A further element is inserted between Celery and the external APIs, a cache implemented through Redis.[2] Redis stores resolutions of requests and can be used to improve the system's scalability by reducing the time to interrogate external servers and avoiding wait time due to the invocation rate limitations.

Service Definition

Focusing on the services for the vehicle routing application, the services are described using the OpenAPI specification v2.0. For space reasons, we provide only a summary of the services.

The first service, which is geo-referencing a list of addresses, can be described as the following set of micro-services:

- Submission of a list of addresses for geo-referencing: for the sake of interoperability with other tasks of the company, the list of supported input formats also includes `.xls` files. The output of the input is a handle that includes the ID of the geo-referencing task that runs asynchronously concerning the end of the submission.
- Status of a task: the input is the handle with the task ID provided by the previous service, and the output is the number of resolved addresses up to now. The user interface (UI) can invoke this service to provide feedback to the user on the progress of a task.
- Download of coordinates lists: this service returns data only after the task is completed; otherwise, an error code is returned. The data can be returned as `JSON` and as an `.xls` file. The first type of output is used to display a map with the results of the geo-referencing (using a mash-application that is based on Open Street Maps APIs), while the `.xls` file is used as the input of the subsequent task of computing a distance matrix between every couple of points.

The API for the second task of the business process described in Fig. 1 is similar to the previous task. The main difference is that the main input is a `.xls` file with the coordinates of the geo-referenced point previously obtained, while the output is another `.xls` file containing the distance matrix. Like the previous task, the execution is asynchronous with a submission micro-service that returns a handle used to check the progress of the computation.

Finally, the last task shown in Fig. 1 is the resolution of the optimization problem. The solution is a two-step process, but the main API implementation masks this double step under two separate APIs:

- Problem submission: the API is used to submit a problem definition consisting of a distance matrix and an expected workload. The output is a handle to access the results of the algorithm invocation.
- Solution download: the handle provided in the problem submission API is used to access the data containing the problem solution. The output can be either a `JSON` data structure or a `.xls` file. The first is used for data visualization on a web UI, while the latter can be downloaded for interaction with the other tasks of the company.

Technologies

As the goal is to provide support in logistic optimization tasks that must be carried out by people who are not experts in using computer systems, the proposed algorithms are integrated into an intuitive and user-friendly web-based UI. The UI leverages the modular architecture of the software provided as a micro-services suite.

The micro-service approach to software development prescribes that software should be split into several independent building blocks that are loosely coupled. This approach made software development extremely agile, with each

[1] https://docs.celeryq.dev/.

[2] https://redis.io/.

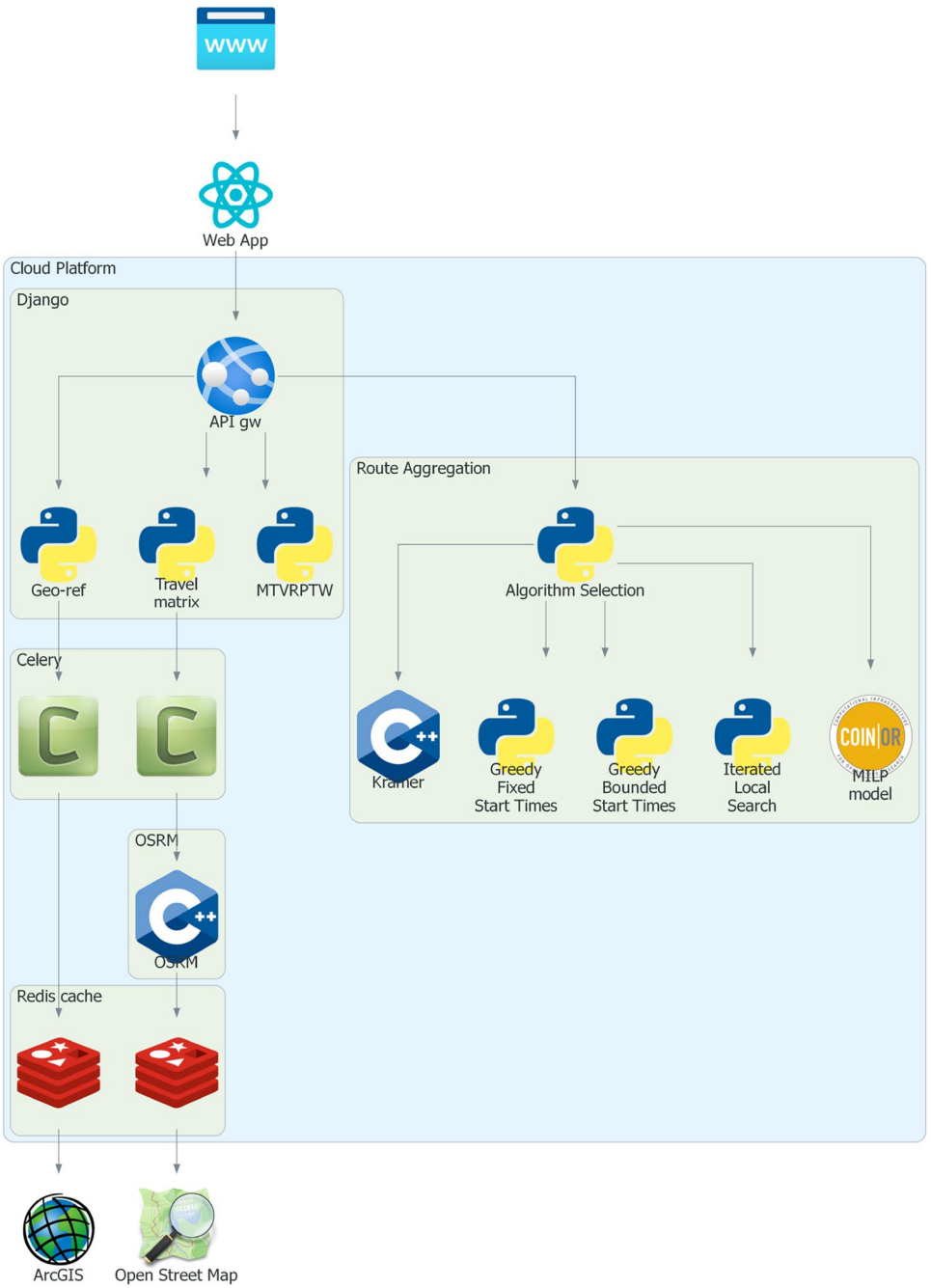


Fig. 3 Web interface architecture

service choosing its own technologies, scalability policies, and life cycle. This is a clear step ahead over monolithic architectures regarding scalability, maintainability, and fault tolerance [28].

The global DSS architecture comprises two main parts: a backend and a frontend. The backend handles HTTPS requests, performs computations, and stores data, providing the previously described micro-service interfaces to the system. The backend is developed using a model-view-controller paradigm [29] provided by the web framework Django. The frontend runs on the user web browser, is written in Javascript, and is based on the React framework to provide a simple interface for the end-user.

A key technology selected for the deployment of the DSS backend is the container management engine Docker,[3] which is also used in the testing and development phases of the micro-services. Each task and the related micro-services are implemented as isolated images (containers) that can be created, replicated, and destroyed using a simple set of command line tools.

[3] https://docs.docker.com/.

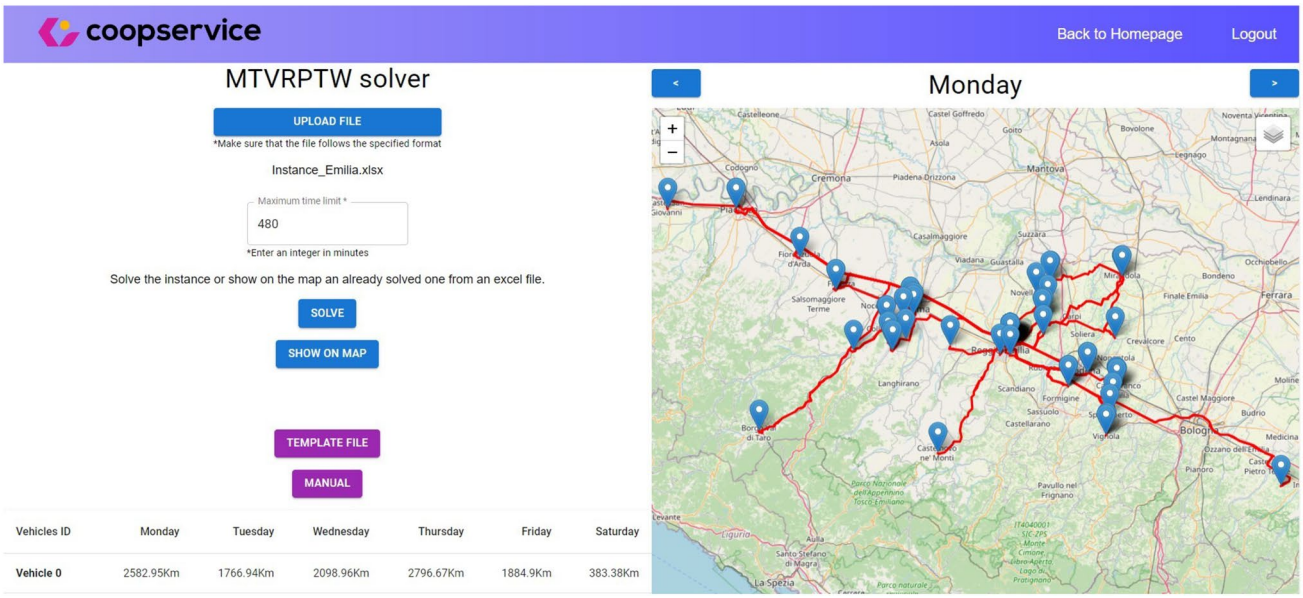


Fig. 4 A screenshot of the MTVRPTW module

The backend deployment scheme consists of several containers outlined by green boxes in Fig. 3:

- *Route Aggregation* contains the optimization algorithms, with the two parts of the solution approach (described in "Proposed methodology") implemented in C++ and Python;
- *OSRM*[4] is a C++ routing service designed to interrogate Open Street Map APIs;
- *Celery* is an asynchronous task queue manager used to automate multiple requests to different endpoints without the risk of overloading the external services;
- *Redis Cache* is a service that manages the sending, receiving, and queuing of messages with Celery;
- *Django* manages the API endpoints and provides the micro-service APIs.

The logical components of the software are also detailed in Fig. 3. These components can be detailed as follows:

- Geo-reference: this module is responsible for the first task of the BPM model shown in Fig. 1, that is to retrieve the coordinates of a list of addresses;
- Travel matrix generation: this module is in charge of the second step in Fig. 1, that is to create a distance and time matrix from the list of coordinates;
- MTVRPTW solver: this module is in charge of the problem solution. The problem is detailed in "Formal MTVRPTW description", while the solution approach used to solve it is described in "Proposed methodology".

As an example of the web UI we also provide a screenshot in Fig. 4. We observe a graphical representation of the computation output (in the form of a map with waypoints and routes) and a tabular representation of the solver output (with schedules and travel distances) that can be downloaded as an `.xls` file. Several buttons are used to upload the problem definition, start the solver execution, plot results, or download them.

Formal MTVRPTW Description

We define the MTVRPTW on a directed graph $G = (N, A)$ with a set of nodes N and a set of arcs $A = \{(i,j) : i,j \in N, i \neq j\}$. The set of nodes N is divided into depots (D) and customers (C). Each arc $(i,j) \in A$ is associated with a traveling time t_{ij}. Additionally, each node $i \in N$ is associated with a specific time window $[e_i, l_i]$, where e_i represents the earliest arrival time and l_i represents the latest arrival time. If a vehicle arrives before e_i, it must wait. In addition, a vehicle cannot arrive after l_i.

Multiple-day deliveries may be required for each customer. Let P denote the days in which the planning is required. Each customer $i \in C$ is characterized by a demand q_{ip} on the day $p \in P$ and a service time s_i. The vehicle fleet, represented by the set V, is heterogeneous and is divided into different types. Vehicles of the same type are defined by K_v, and all vehicles $k \in K_v$ are identical in loading capacity and

[4] https://project-osrm.org/.

permissible routes (e.g., mountainous arcs can be traveled only by the smallest vehicles).

A feasible solution for the problem must satisfy the following constraints: each route is associated with a unique depot and respects the vehicle capacity; a vehicle can perform multiple routes in a day, and each route needs to start and end at the same depot; each customer is assigned to exactly one route, and the total demand must be accomplished during a single visit within the time window. Additionally, the sum of the durations of the routes assigned to each vehicle cannot exceed $T = 480$ minutes per day. Furthermore, before starting another route, the vehicle requires a fixed loading time of $\Delta = 30$ minutes, which is included in the overall time limit T.

The objective of the problem is to find a set of routes that satisfy the aforementioned constraints while minimizing the number of used vehicles. All vehicles can operate on any day in set P. The problem is solved separately for each day $p \in P$, and the solution for one day does not depend on the solutions for other days.

Proposed Methodology

In this section, we present the two-phase decomposition approach that we implemented to solve the MTVRPTW. According to [30], this kind of strategy has allowed us to obtain high-quality results for VRPs with multiple trips. In the first phase of our approach, we solve an instance of the VRPTW. The objective is to obtain a set R of routes satisfying the customers' demands and other operational constraints. The set R is then used as input for the second phase. The objective in this phase is to minimize the number of vehicles when solving an instance of the MTVRPTW. In other words, "Solving the VRPTW" presents the first phase, while "Solving the MTVRPTW" discusses the algorithms that we have proposed for the second phase.

Solving the VRPTW

To solve the VRPTW, we use the algorithm proposed by [4]. They solved a real-world distribution case study in Coopservice, considering multiple depots, a heterogeneous fleet of vehicles, flexible time windows, periodic demands, incompatibilities between vehicles and customers, a maximum duration for the routes, and a maximum number of customers per route. The authors proposed a multi-start ILS using several neighborhood operators. The first step in the multi-start ILS is to obtain an initial solution from a constructive heuristic. The constructive heuristic creates routes by adding customers to the closest depot by inserting customers one at a time in the route that generates the lowest cost. This heuristic allows time window violations, but they are penalized when calculating the objective function. The next step is to improve the initial solution using local search. It comprises a randomized variable neighborhood descent and contains neighborhoods based on inter- and intra-route movements. Besides that, perturbation procedures are applied to the solution to escape from local optima. They modify the routes using random swap movements and customer relocations.

We use this algorithm to obtain a set R of routes used as input for the second phase. Since our goal is to find the minimum number of used vehicles, we aim to combine the routes of R. In other words, depending on the customers' time windows, a vehicle can return to the depot and perform another route, reducing the number of vehicles needed. Therefore, we propose four algorithms, explained in the next section, to attain this objective.

Solving the MTVRPTW

The algorithms we develop aim at merging the routes in R to generate giant routes (i.e., routes with multiple trips). Next, we assign these giant routes to appropriate vehicles, aiming to minimize the number of used vehicles. The first approach assumes the initial routes have a fixed starting time. Further elaboration regarding this algorithm is provided in "Fixed starting time Greedy Algorithm (FSTG)".

Considering that the customers' time windows are large enough, our next approaches optimize the starting time of each route in R while still respecting the time windows. The objective is to determine the best starting time for each route. In this context, for each vehicle type $v \in V$, each depot $d \in D$, and each day $p \in P$, we can aggregate the routes on the same day from the same depot and the same vehicle type.

We calculate the starting time of each route with the forward-time slack procedure introduced by [31]. Let $r \in R$ be a route, and N_r denote the sequence of nodes visited by the route r. For every node $i \in N_r$, we define ST_i as the earliest feasible starting time, WT_i as the cumulative idle time, and FT_i as the partial forward slack time. For the starting time of route r, the initial values are set as follows: $ST_1 = e_1$, $WT_1 = 0$, and $FT_1 = l_1 - e_1$. For successive nodes $i \in N_r$, the calculations are carried out as follows:

$$ST_i = \max(ST_{i-1} + t_{i-1,i} + s_i; e_i + s_i), \tag{1}$$

$$WT_i = WT_{i-1} + (ST_i - ST_{i-1} - t_{i-1,i} - s_{i-1}), \tag{2}$$

$$FT_i = \min(FT_{i-1}; l_i - ST_i + WT_i). \tag{3}$$

In addition, we also compute the latest starting time LT_i for each node $i \in N_r$. The calculation starts from the last node $n \in N_r$, initializing $LT_n = l_n$, and then proceeds in reverse order towards the first node of the route, as outlined below:

$$LT_{i-1} = \min(LT_i - t_{i-1,i} - s_i; l_{i-1}). \quad (4)$$

After all, the earliest and latest starting times for route r are determined by the following expressions:

$$\bar{e}_r = e_1 + \min(FT_n; WT_n), \quad (5)$$

$$\bar{l}_r = LT_1. \quad (6)$$

We generalize the fixed starting time greedy algorithm by incorporating the earliest and latest starting times for the routes in R, as elaborated in "Bounded starting time Greedy Algorithm (BSTG)". We also take advantage of that by implementing an ILS, outlined in "Iterated local search (ILS)". Furthermore, these improvements are also considered in a MILP problem presented in "Mathematical model".

Fixed Starting Time Greedy Algorithm (FSTG)

To solve the problem variant where the starting time of each route is fixed, we have developed the greedy algorithm outlined in Algorithm 1. The input is the set R of routes resulting from the previous step. Each route $r \in R$ is assigned to a day $p_r \in P$, to a depot $d_r \in D$, and to a vehicle of type $k_r \in K_v \subseteq V$. The objective is to assign every day the routes in R to the minimum number of vehicles.

The algorithm starts with an empty set σ, representing the set of used vehicles associated with a specific day p, a depot d, and a vehicle of type k (i.e., σ_{pdk}). Let us denote by $R_{pdk} \subseteq R$ the subset of routes for which $p_r = p$, $d_r = d$, and $k_r = k$. Set R_{pdk} is created in line 6, and then the routes it contains are sorted according to non-decreasing starting time (breaking ties by non-increasing duration) in line 7. This approach ensures that routes with similar starting times but longer duration are prioritized. Following this order, each route is assigned to an available vehicle belonging to σ_{pdk} in line 10. The assignment is feasible if the vehicle v exists and can perform the route concerning its starting time and duration, besides respecting the overall driving time T in day p and the time window of each customer. Otherwise, a new vehicle of type k is initialized to perform the route $r \in R_{pdk}$. The algorithm ends when all routes in R are assigned to a vehicle v (line 16).

Algorithm 1 Greedy Algorithm

```
1: procedure GREEDY(R)
2:     σ ← ∅
3:     for each day p ∈ P do
4:         for each depot d ∈ D do
5:             for each vehicle type k ∈ K_v : K_v ⊆ V do
6:                 Let R_pdk ⊆ R be the set of routes having p_r = p, d_r = d, k_r = k
7:                 Sort the routes in R_pdk by non-decreasing starting time
8:                 for each r ∈ R_pdk do
9:                     if there exists a vehicle of type k to serve r then
10:                        σ_pdk ← r
11:                    else
12:                        Initialize a new vehicle of type k to serve r and include r in σ_pdk
13:                    end if
14:                end for
15:                if all routes in R have been assigned then
16:                    return σ
17:                end if
18:            end for
19:        end for
20:    end for
21: end procedure
```

Table 1 Model decision variables

Variable	Description
x_{rkv}	Binary variable: 1 if route $r \in R$ is assigned to vehicle $k \in K_v$ of type $v \in V$, 0 otherwise
y_{kv}	Binary variable: 1 if vehicle $k \in K_v$ of type $v \in V$ is used, 0 otherwise
z_{rskv}	Binary variable: 1 if route $r \in R$ precedes route $s \in R$, both assigned to vehicle $k \in K_v$ of type $v \in V$, 0 otherwise
α_{kv}	Continuous variable: starting time of vehicle $k \in K_v$ of type $v \in V$
β_{kv}	Continuous variable: ending time of vehicle $k \in K_v$ of type $v \in V$
t_{rkv}	Continuous variable: starting time of route $r \in R$ assigned to vehicle $k \in K_v$ of type $v \in V$

The advantage of using the fixed starting times is that the resulting subproblem is easy, and, indeed, Algorithm 1 can solve it to proven optimality (i.e., it finds the minimum number of vehicles). The proof derives from the interval coloring problem, which is known to be polynomially solvable by the greedy algorithm (see, e.g., [32]). In our case, intervals are the fixed starting time routes, and the colors are the vehicles. The disadvantage is that we lose flexibility in the way routes can be merged and thus drastically reduce the solution space. In the next section, we overcome this drawback by allowing the route starting times to be modified.

Bounded Starting Time Greedy Algorithm (BSTG)

This section introduces an enhancement to Algorithm 1 that considers a bounded starting time for each route in R. In contrast to the fixed starting time strategy, this new version incorporates flexibility by moving the starting time within an interval defined by the earliest and the latest starting time. These bounds are computed using equations (5) and (6). In this way, the BSTG can optimize the merging of the routes by adjusting their starting times according to the defined earliest-to-latest interval.

Formally, for each day $p \in P$ and for each route $r \in R$, we compute $\bar{e}_r$ and $\bar{l}_r$ using equations (5) and (6), respectively. Then, in Algorithm 1, line 7 is changed to have the routes in $R_{pdk} \subseteq R$ sorted in a non-decreasing order based on $\bar{e}_r$. Consequently, routes are scheduled to begin as early as possible, respecting the defined earliest-to-latest interval.

Iterated Local Search (ILS)

The ILS is a metaheuristic optimization approach that integrates local search techniques with iterative improvement strategies. It aims to refine a given initial solution by iteratively applying perturbation and local search algorithms. The perturbation phase introduces randomness into the current solution to escape from local optima and expand the exploration of the solution space using local search algorithms [33].

Algorithm 2 presents the ILS we implemented. It consists of different phases: initialization, perturbation, local exploration, and acceptance. All these phases contribute to the iterative improvement process of the algorithm, allowing it to escape from local optima solutions and converge to better solutions. The algorithm receives in input ω, the maximum number of iterations without improvements. The initial solution is generated by the bounded starting time greedy algorithm described in "Bounded starting time Greedy Algorithm (BSTG)". Then, the solution is improved through the local search procedure described in Algorithm 3. The main loop in lines 6-16 of Algorithm 2 continues until ω is eventually reached. The loop begins by applying the perturbation procedure, detailed in Algorithm 4, followed by the local search procedure. After that, if the cost of the new solution (i.e., the number of vehicles used) improves the incumbent solution value, the incumbent solution is updated and the loop restarts from scratch (because we reached an improvement). In line 15, the current solution is updated with the new one if the new one uses fewer vehicles.

Algorithm 2 Iterated Local Search Algorithm

```
1:  procedure ILS(ω)
2:      i ← 0
3:      x_0 ← ConstructiveHeuristic
4:      x_current ← LocalSearch(x_0)
5:      x_best ← x_current
6:      while i < ω do
7:          x' ← Perturbation(x_current)
8:          x'' ← LocalSearch(x')
9:          if cost(x'') < cost(x_best) then
10:             x_best ← x''
11:             i ← 0
12:         else
13:             i ← i + 1
14:         end if
15:         x_current ← Acceptance(x_current, x'')
16:     end while
17:     return x_best
18: end procedure
```

Algorithm 3 Local Search Procedure

```
1: procedure LocalSearch(x)
2:     for each pair of routes r1 and r2 in x do
3:         x_new ← swap routes r1 and r2
4:         if cost(x_new) < cost(x) then
5:             x ← x_new
6:         end if
7:     end for
8:     for each pair of routes r1 and r2 in x do
9:         x_new ← insert route r1 before route r2
10:        if cost(x_new) < cost(x) then
11:            x ← x_new
12:        end if
13:    end for
14:    return x
15: end procedure
```

In the local search (Algorithm 3), we start by applying swap movements and, next, insertion movements. The swap procedure involves exchanging pairs of routes, while the insertion procedure involves selecting and inserting a route before another route. We apply these movements on all combinations of two routes. As soon as an improvement is found, if any, the solution is updated, and the search continues. On the other hand, the perturbation function (Algorithm 4) is used to change (drastically) a given solution by inserting a random subset of routes in random positions of the solution. It starts by selecting k, i.e., the number of subsets of perturbed routes. Next, each subset is defined randomly in lines 4-6 and inserted in a random solution position in line 7. These steps are repeated $k-1$ times.

Algorithm 4 Perturbation Procedure

```
1: procedure Perturbation(x)
2:     k ← random integer number in [1, |x|)
3:     for i ← 2 to k + 1 do
4:         start ← random integer number in the range of |x| − i
5:         end ← start + i
6:         subset_routes ← x[start : end]
7:         Insert subset_routes into x at a random position
8:     end for
9:     return x
10: end procedure
```

Mathematical Model

We also propose a MILP model to combine the routes $r \in R$ to minimize the daily number of used vehicles. The model has three sets of binary decision variables and three sets of continuous decision variables, as described in Table 1. The parameter T_r represents the total duration of route $r \in R$. Let N_r be the sequence of nodes visited by the route r, we obtain T_r as the sum of the traveling times to visit the nodes in N_r, and the service time at each node in N_r. In addition, we need to sum the fixed loading time Δ to T_r. Let M be a large number.

The resultant model is presented in (7)–(22) below. It is executed independently for each day $p \in P$, and so it only considers the routes $R_p \subseteq R$ of day p.

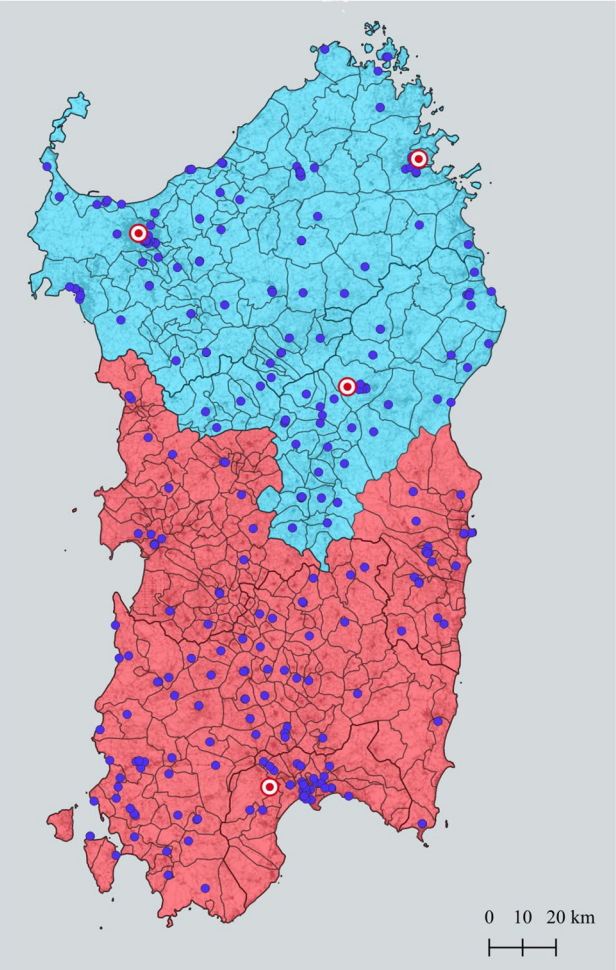


Fig. 5 Coopservice data of the Sardinia region

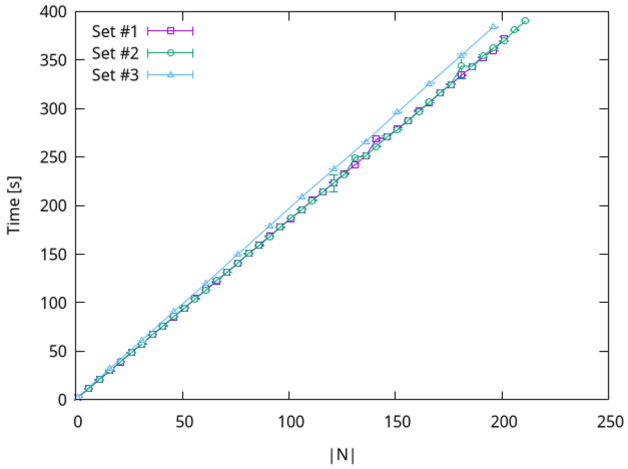
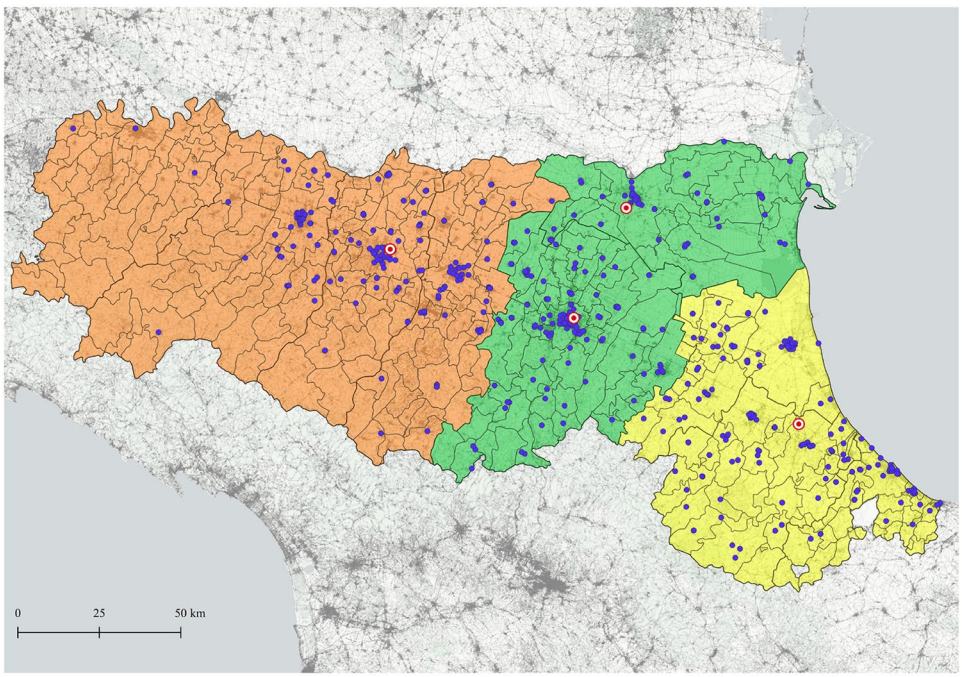


Fig. 7 Computing times of executing the geo-location services

Fig. 6 Coopservice data of the Emilia Romagna region



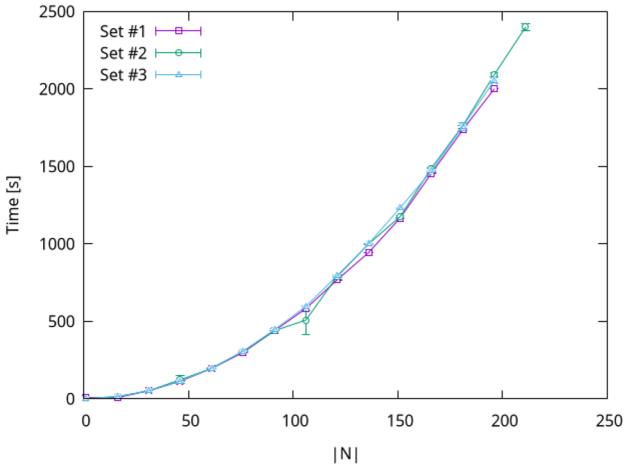


Fig. 8 Computing times of executing the distance-computation services

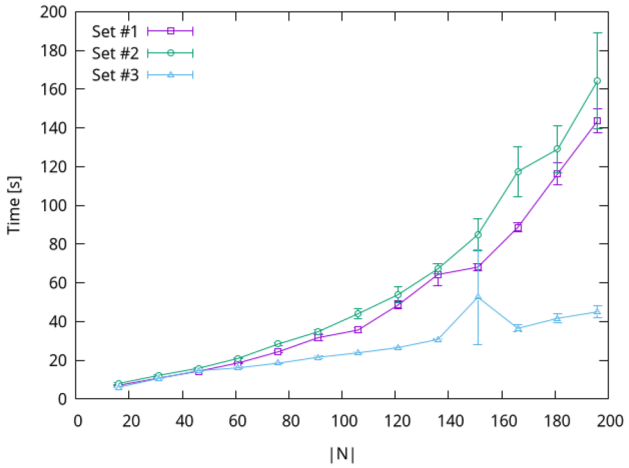


Fig. 9 Computing times of executing the routing services

$$\min Z_p = \sum_{v \in V} \sum_{k \in K_v} y_{kv} \tag{7}$$

$$x_{rkv} \leq y_{kv}, \quad \forall r \in R_p, \forall v \in V, \forall k \in K_v \tag{8}$$

$$\sum_{v \in V} \sum_{k \in K_v} x_{rkv} = 1, \quad \forall r \in R_p \tag{9}$$

$$z_{rskv} + z_{srkv} \geq x_{rkv} + x_{skv} - 1, \quad \forall v \in V, \forall k \in K_v, \forall r, s \in R_p : r \neq s \tag{10}$$

$$z_{rskv} + z_{srkv} \leq 1, \quad \forall v \in V, \forall k \in K_v, \forall r, s \in R_p : r \neq s \tag{11}$$

$$t_{rkv} + (T_r + \Delta) \leq t_{skv} + M(1 - z_{rskv}), \quad \forall v \in V, \forall k \in K_v, \forall r, s \in R_p : r \neq s \tag{12}$$

$$\bar{e}_r \leq t_{rkv} \leq \bar{l}_r, \quad \forall r \in R_p, \forall v \in V, \forall k \in K_v \tag{13}$$

$$\alpha_{kv} \leq t_{rkv} + M(1 - x_{rkv}), \quad \forall r \in R_p, \forall v \in V, \forall k \in K_v \tag{14}$$

$$\beta_{kv} \geq t_{rkv} + (T_r + \Delta) - M(1 - x_{rkv}), \quad \forall r \in R_p, \forall v \in V, \forall k \in K_v \tag{15}$$

$$\beta_{kv} - \alpha_{kv} \leq T + \Delta, \quad \forall v \in V, \forall k \in K_v \tag{16}$$

$$\sum_{r \in R_p} (T_r + \Delta) x_{rkv} \leq T + \Delta, \quad \forall v \in V, \forall k \in K_v \tag{17}$$

$$x_{rkv} \in \{0, 1\}, \quad \forall r \in R_p, \forall v \in V, \forall k \in K_v \tag{18}$$

$$y_{kv} \in \{0, 1\}, \quad \forall v \in V, \forall k \in K_v \tag{19}$$

$$z_{rskv} \in \{0, 1\}, \quad \forall r, s \in R_p : r \neq s, \forall v \in V, \forall k \in K_v \tag{20}$$

$$t_{rkv} \geq 0, \quad \forall r \in R_p, \forall v \in V, \forall k \in K_v \tag{21}$$

$$\alpha_{kv} \geq 0, \beta_{kv} \geq 0, \quad \forall v \in V, \forall k \in K_v \tag{22}$$

The objective function (7) asks to minimize the number of used vehicles on the day p. Constraints (8) impose that each route r is assigned to a given vehicle k if and only if k performs that route. Constraints (9) ensure that all routes are served by a vehicle. Constraints (10) and (11) impose the precedence between routes performed by the same vehicle. In (10), if two routes are performed by the same vehicle, one must precede the other. Instead, in (11), the first route precedes the second, or the second route precedes the first one. Constraints (12) impose that a route starts after the ending of another route if both are served by the same vehicle, respecting the total duration T_r and the fixed loading time Δ. Constraints (13) ensure that the starting time of each route is between the earliest and latest starting time, computed using equations (5) and (6), respectively. Constraints (14), (15), and (16) guarantee that the multiple routes a vehicle performs are executed within the maximum vehicle working time T. Note that in constraints (16), we add an additional fixed loading time Δ because it is not considered in the first route. That is why in constraints (15), we assume that all routes (including the first one performed by the vehicle) have a fixed loading time. In addition, constraint (17) are valid inequalities to ensure that the total duration of all the routes performed by each vehicle respects the maximum vehicle working time. Lastly, constraints (18)-(22) define the variables domain.

Table 2 Computational results on the 20 groups (120 random instances)

Instance	$\|D\|$	$\|C\|$	Kramer	FSTG		BSTG		ILS		Model	
			$\sum \|R_p\|$	$\sum Z_p$	$\sum t(s)$	$\sum Z_p$	$\sum t(s)$	$\sum Z_p$	$\sum t(s)$	$\sum Z_p$	$\sum t(s)$
Group-01	1	30	18	18	0.013	16	0.013	16	0.730	16	0.198
Group-02	1	60	23	23	0.013	23	0.013	23	1.358	23	0.273
Group-03	1	90	38	38	0.018	35	0.017	35	3.661	35	0.737
Group-04	1	120	45	45	0.017	42	0.017	42	6.049	42	1.103
Group-05	1	150	49	49	0.017	49	0.017	49	6.932	49	1.285
Group-06	2	30	13	13	0.016	13	0.015	13	0.142	13	0.173
Group-07	2	60	28	28	0.023	27	0.023	27	0.735	27	0.477
Group-08	2	90	31	31	0.020	29	0.019	29	1.183	29	0.564
Group-09	2	120	42	42	0.024	39	0.024	38	2.111	38	1.163
Group-10	2	150	49	49	0.025	46	0.026	46	3.145	46	1.592
Group-11	3	30	15	15	0.019	14	0.019	14	0.050	14	0.236
Group-12	3	60	29	29	0.029	27	0.028	27	0.464	27	0.771
Group-13	3	90	32	32	0.030	30	0.030	30	0.459	30	0.828
Group-14	3	120	39	39	0.029	36	0.029	36	1.150	36	1.274
Group-15	3	150	49	49	0.034	46	0.033	46	1.689	46	2.006
Group-16	1	30	18	18	0.013	16	0.012	16	0.823	16	0.184
Group-17	1	60	29	29	0.014	25	0.013	25	2.371	25	0.380
Group-18	1	90	39	39	0.015	38	0.014	38	4.702	38	0.748
Group-19	1	120	43	43	0.018	41	0.017	41	4.610	41	0.903
Group-20	1	150	55	55	0.019	50	0.018	50	8.148	50	2.357
Average			34	34	0.020	32	0.020	32	2.526	32	0.863

Computational Results

To evaluate the performance of the considered system, we deployed the DSS on a virtual machine (VM) in a private cloud environment. The VM has 16GB of RAM and 16 virtual cores. The underlying virtualization environment is based on XCP-NG (Xen-based) 8.2 and runs on a four Intel Xeon Gold 6252N CPU with a clock of 2.30 GHz. The guest is a VM running Microsoft Windows 11 Home 64-bits.

The algorithms in "Solving the VRPTW" were coded in C++, and those in "Solving the MTVRPTW" in Python 3.10. Model (7)–(22) was solved with the help of Coin-OR,[5] version 2.10.3. A time limit of 600 s was imposed on the model to solve each instance. For the ILS, the maximum number ω of iterations without improvements was set to 10 after the outcome of preliminary experiments.

The instances have been obtained from the operations planned by Coopservice, in compliance with data privacy regulations, in the Sardinia and Emilia Romagna regions in Italy. The Sardinia region is divided into two areas, as illustrated in Fig. 5: North Sardinia, with 152 customers, 3 depots, and 2 vehicle types; and South Sardinia, with 154 customers, 1 depot, and 2 vehicle types. The Emilia Romagna region, instead, is divided into three areas, as illustrated in Fig. 6: West Emilia Romagna, with 206 customers, 1 depot, and 1 vehicle type; Central Emilia Romagna, with 210 customers, 2 depots, and 1 vehicle type; and East Emilia Romagna, with 201 customers, 1 depot, and 1 vehicle type. In the figures, the blue points represent the customers, while the red ones represent the depots.

In the following, we first perform scalability analyses, considering instances obtained from subsets with different numbers of nodes in N (i.e., customers and depots) from the Emilia Romagna region. Next, we present the computational results obtained with the proposed algorithms on the given instances.

Scalability Analysis of the Micro-services

To evaluate the impact of the problem size on the speed of the different micro-services, we consider the several steps carried out by the DSS, from geo-referencing the points to solving the routing problem, as described in "Decision support system architecture". To this aim, we generate random instances by selecting subsets of nodes from the real-world data representing the Emilia Romagna region. Each resulting instance is solved five times, and the average computing times in seconds are reported to estimate the increase in the computational effort.

In the first experiment, shown in Fig. 7, we evaluate the execution time of geo-referencing points of interest varying the instance size. We first generate three sets: Set #1 (East

[5] https://www.coin-or.org/documentation.html.

Table 3 Computational results on the 28 real instances

Instance	p	Kramer	FSTG		BSTG		ILS		Model	
		$\|R_p\|$	Z_p	$t(s)$	Z_p	$t(s)$	Z_p	$t(s)$	Z_p	$t(s)$
North Sardinia	1	9	9	0.005	7	0.005	7	0.911	7	0.344
North Sardinia	2	9	9	0.007	9	0.007	9	0.476	9	0.319
North Sardinia	3	8	8	0.007	8	0.006	8	0.328	8	0.269
North Sardinia	4	9	9	0.006	9	0.006	9	0.610	9	0.331
North Sardinia	5	10	10	0.007	9	0.006	9	0.664	9	0.473
North Sardinia	6	3	3	0.004	3	0.004	3	0.002	3	0.038
Total		48	48	0.036	45	0.034	45	2.991	45	1.774
South Sardinia	1	10	10	0.003	9	0.003	9	1.359	9	0.337
South Sardinia	2	11	11	0.003	11	0.003	11	1.781	11	0.404
South Sardinia	3	11	11	0.003	8	0.003	8	1.771	8	0.440
South Sardinia	4	11	11	0.004	9	0.004	9	1.772	9	0.428
South Sardinia	5	10	10	0.003	10	0.003	10	1.347	10	0.315
South Sardinia	6	5	5	0.003	5	0.003	5	0.382	5	0.070
Total		58	58	0.019	52	0.019	52	8.412	52	1.994
West Emilia Romagna	1	22	22	0.003	21	0.003	21	23.808	21	TL
West Emilia Romagna	2	22	21	0.003	19	0.003	17	23.933	17	TL
West Emilia Romagna	3	22	21	0.003	21	0.003	21	23.897	21	TL
West Emilia Romagna	4	26	24	0.003	23	0.003	23	35.838	-	TL
West Emilia Romagna	5	24	21	0.003	19	0.003	18	30.266	19	TL
West Emilia Romagna	6	3	3	0.002	3	0.002	3	0.199	3	0.071
Total		119	112	0.017	106	0.017	103	137.941	-	3000.071
Central Emilia Romagna	1	58	58	0.007	20	0.006	20	143.140	-	TL
Central Emilia Romagna	2	10	10	0.004	5	0.004	4	2.192	4	1.158
Central Emilia Romagna	3	58	58	0.007	20	0.006	20	129.670	-	TL
Central Emilia Romagna	4	10	10	0.003	4	0.003	4	2.178	4	0.815
Central Emilia Romagna	5	58	58	0.007	21	0.006	20	131.501	-	TL
Total		194	194	0.028	70	0.025	68	408.681	-	1801.973
East Emilia Romagna	1	54	54	0.005	24	0.005	23	245.443	29	TL
East Emilia Romagna	2	7	7	0.002	4	0.002	4	1.577	4	0.311
East Emilia Romagna	3	54	54	0.005	24	0.005	23	244.933	32	TL
East Emilia Romagna	4	7	7	0.002	4	0.002	4	1.575	4	0.560
East Emilia Romagna	5	55	55	0.005	23	0.005	23	252.420	29	TL
Total		177	177	0.019	79	0.019	77	745.948	98	1800.871

Emilia Romagna) contains 41 instances with number $|N|$ of nodes in the range [1, 201]; Set #2 (Central Emilia Romagna) contains 43 instances with $|N|$ in the range [1, 211]; and Set #3 (West Emilia Romagna) contains 14 instances with $|N|$ in the range [1, 196]. We can observe a quasi-linear increase in the running time as the size of the instance grows. The performance here is impacted by the running time of the external ArcGIS service used for geo-referencing. Due to the nature of the service, where a single key is used to access the external service, multi-threaded and parallel execution of the service do not provide a significant gain (the invocation rate for a given key is limited).

The second experiment concerns the performance and scalability of the distance matrix computation. To this aim, we consider the output of the previous step, and we feed the geo-referenced points into the distance computation micro-service. For each problem size, we record the average computing time required to obtain the distance matrix. Figure 8 shows the micro-service computing time (in seconds) as a function of the number of points. The distance matrix contains a value for each couple of points. This explains the quadratic shape of the running time curve as a function of the instance size. Due to the frequency of external service invocations, we also observe that the distance computation matrix can require a time in the order of tens of minutes. This observation confirms the validity of our design choice to return a job ID for subsequent polling rather than returning directly the distance at the end of the computation: the TCP protocol (used to interact with the micro-service instance) would return a timeout error before

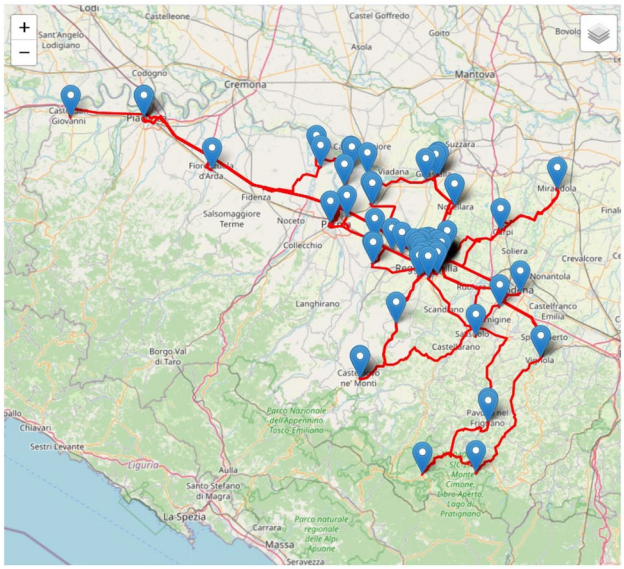


Fig. 10 Illustrative example of a solution found for the West Emilia Romagna area

completing the service whenever the execution time of the micro-service exceeds 120 s. A second observation from this experiment is the critical impact of caching on performance: whenever a distance is already available in the cache, there is no need to contact the external service.

The third experiment concerns the computing time required to solve the VRPTW with the algorithm proposed by [4], as discussed in "Solving the VRPTW". We evaluate the scalability of the micro-service as a function of the instance size. The results are given in Fig. 9, where we observe a non-linear increase in the running time as a function of the number of points to visit, which is compatible with the NP-hard nature of the vehicle routing problem. However, the limited size of the considered problem instances makes it feasible to provide a heuristic solution in a reasonable time. The computational effort of the algorithms in "Solving the MTVRPTW" is reported in the next section.

Computational Results of the MTVRPTW Algorithms

With the algorithms proposed in "Solving the MTVRPTW", we aim to minimize the number of vehicles used to generate a weekly schedule to serve the customers. The activities planned in the areas of North Sardinia, South Sardinia, and West Emilia Romagna span over six working days, whereas those planned in Central Emilia Romagna and East Emilia Romagna last five working days. Consequently, we have a total of 28 real instances, one per day and area. To obtain a very extensive validation of the algorithms, we have also created 120 random instances that contain subsets of customers from the 120 working days (grouped in 20 working weeks of 6 days each) in the Sardinia region. Each instance is solved five times, and we report, in the next tables, the best solution found among these runs and the average computing time.

The 120 random instances are divided into two classes:

- Class 1 is obtained from the North Sardinia area, considering a number of depots randomly selected in the set $\{1, 2, 3\}$, and a number of customers randomly selected in the set $\{30, 60, 90, 120, 150\}$. Customers are randomly selected from the original set in the real instance. Two types of vehicles are available. The instances of this class correspond to Group-01 until Group-15.
- Class 2 is generated similarly to Class 1, but it refers to the South Sardinia area and is characterized by a single depot. The instances of this class correspond to Group-16 until Group-20.

In Table 2, we present the results obtained with the proposed algorithms on the 120 randomly generated instances, organized in 20 groups. Columns $|D|$ and $|C|$ report the number of depots and total customers in the group (week), respectively. Column $|R_p|$ represents the number of routes generated by solving the VRPTW with the algorithm of [4]. Columns Z_p and $t(s)$ contain the number of used vehicles per day and the average computing times in seconds over the five runs, obtained from solving the MTVRPTW with the proposed algorithms: FSTG, BSTG, ILS, and mathematical model. The results comprise the entire week (i.e., a group having six days, where the best solution for each day is considered). Each line then shows the total values as the sum of the best solutions per day over the five runs.

By analyzing the results in Table 2, we can notice that no improvement is obtained in three cases: Group-02, Group-05, and Group-06, all related to North Sardinia. In contrast, for all other cases, the proposed algorithms, except the FSTG, obtain a reduction in the number of vehicles used. Notably, the BSTG achieves an average reduction of 6.14% in the number of used vehicles compared to the solution obtained with the algorithm of [4], while the ILS and the mathematical model reach an average reduction of 6.29%.

From Table 2, we observe that the ILS and the mathematical model have the same solutions. To evaluate how competitive they are, we now present the results for the 28 real instances, which are harder to solve. These results are reported in Table 3 for each day p and each area. For some instances, the imposed time limit has been reached (TL), and the mathematical model found no feasible solution (entry '-'). In the last line of each table, we summarize the total values comprising the entire week for each area.

Table 3 shows that the ILS obtains the best overall results, especially when the mathematical model cannot find a solution within the given time limit. It means that the model does not scale well as the size of instances grows. For example, the ILS returns 77 vehicles for the East Emilia Romagna, while the mathematical model gives 98 vehicles. In the West and Central Emilia Romagna, the mathematical model cannot find any feasible solution on days $p = 4$ and $p = 1, 3, 5$, respectively, while the ILS achieves the best total number of vehicles, followed by the BSTG with 3 and 2 vehicles more, respectively. The BSTG obtains the same results concerning the ILS for the North and South Sardinia. On the other hand, the FSTG always returns the same number of vehicles as obtained with the algorithm of [4], except for the West Emilia Romagna.

The ILS can reduce by 6.25%, 10.34%, 13.45%, 64.95%, and 56.50% the number of vehicles concerning the algorithm of [4] for North Sardinia, South Sardinia, West Emilia Romagna, Central Emilia Romagna, and East Emilia Romagna, respectively, considering the entire week. It is important to consider that, in Italy, the cost of a large vehicle (120 tons or more) can easily exceed 50,000 euros. Therefore, even if the number of route reductions may appear small in some instances, they represent significant cost savings for Coopservice. Concerning the computing times, they are negligible for FSTG and BSTG. The ILS requires more time (260.80 s, on average per week), but generally, it is much faster than the mathematical model (1321.34 s, on average, per week).

Besides that, the proposed DSS allows the decision-maker to visualize the generated routes on a map. Figure 10 illustrates an example of the solutions found in the West Emilia Romagna. The simplified UI allows the decision-makers to easily check the solution and its feasibility in daily planning.

Conclusions and Future Works

This paper presents the development of a model-driven decision support system (DSS) designed to help decision-makers tackle complex logistic decisions. The proposed DSS encompasses specialized modules to address the multi-trip vehicle routing problem with time windows (i.e., MTVRPTW). It considers a set of agile micro-services capable of geo-referencing points and performing distance calculations. The data produced is then used as input to the vehicle routing problem, which in turn is solved by a two-phase approach. In the first stage, we use a solution method proposed in the literature to find a solution to the vehicle routing problem with time windows. Next, four proposed methods are used to obtain a solution to the MTVRPTW: two greedy heuristics, an ILS metaheuristic, and a MILP model. The overall DSS architecture has the flexibility to allow changing the micro-services already implemented as well as to add new services and even solution methods.

The DSS is validated on real and randomly generated instances, considering different numbers of depots, customers, and vehicle types. Results indicate the superior performance of the ILS, which can outperform on the real instances the algorithm of [4] by 30.30%, the fixed starting time greedy algorithm by 29.22%, the bounded starting time greedy algorithm by 1.64%, and the mathematical model by 4.50%, on average. All these improvements significantly benefit the company that is using the DSS in terms of operational costs. Besides that, the computing times of the ILS are compatible with daily use.

Concerning future works, it would be interesting to add new micro-services in the DSS to, e.g., handle other logistic problems faced by the company in activities such as pick-up and delivery and car patrolling (see, e.g., [34]). Another interesting direction would be to investigate the impact of new constraints on the problem, e.g., labor constraints or maximum CO_2 emissions. In this regard, exploring the adoption of electric vehicles instead of fuel vehicles, so as to minimize emissions while meeting additional operational constraints (see, e.g., [35]), offers a very interesting perspective for future research. It would also be interesting to apply the proposed DSS to handle problems related to reverse logistics. To this aim, one could try to adapt the current algorithms, especially the ILS, by including new perturbation procedures and local search operators.

Acknowledgements The authors thank the financial support provided by the National Council for Scientific and Technological Development (CNPq) [grant numbers 311185/2020-7, 405369/2021-2, 408722/2023-1, and 315555/2023-8], the State of Goiás Research Foundation (FAPEG), the National Recovery and Resilience Plan (NRRP), Mission 04 Component 2 Investment 1.5-NextGenerationEU, Call for tender n. 3277 dated 30/12/2021, Award Number: 0001052 dated 23/06/2022, and Coopservice Soc.coop.p.A. Besides that, Coopservice has provided the authors with relevant information, feedback, and data for the numerical experiments.

Declarations

Conflict of interest No potential conflict of interest was reported by the authors.

References

1. Golden B, Raghavan S, Wasil E (eds.): The Vehicle Routing Problem: Latest Advances and New Challenges. Operations Research/Computer Science Interfaces, vol. 43. Springer, Boston, MA 2008; https://doi.org/10.1007/978-0-387-77778-8

2. Toth P, Vigo D. Vehicle Routing: Problems, Methods, and Applications. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA 2014; https://doi.org/10.1137/1.9781611973594
3. Uchoa E, Pecin D, Pessoa A, Poggi M, Vidal T, Subramanian A. New benchmark instances for the capacitated vehicle routing problem. European Journal of Operational Research. 2017;257(3):845–58. https://doi.org/10.1016/j.ejor.2016.08.012.
4. Kramer R, Cordeau J-F, Iori M. Rich vehicle routing with auxiliary depots and anticipated deliveries: An application to pharmaceutical distribution. Transportation Research Part E: Logistics and Transportation Review. 2019;129:162–74. https://doi.org/10.1016/j.tre.2019.07.012.
5. Cavecchia M, De Queiroz TA, Iori M, Lancellotti R, Zucchi G. A Decision Support System for Multi-Trip Vehicle Routing Problems. In: Proceedings of the 25th International Conference on Enterprise Information Systems, vol. 1, pp. 335–343. SCITEPRESS - Science and Technology Publications, Prague, Czech Republic 2023; https://doi.org/10.5220/0011806600003467
6. Vidal T, Crainic TG, Gendreau M, Prins C. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. European Journal of Operational Research. 2013;231(1):1–21. https://doi.org/10.1016/j.ejor.2013.02.053.
7. Goel RK, Bansal RS. Hybrid algorithms for rich vehicle routing problems: A survey. In: Nalepa, J. (ed.) Smart Delivery Systems. Intelligent Data-Centric Systems, 2020;157–184 Chap. 5. https://doi.org/10.1016/B978-0-12-815715-2.00011-7
8. Vidal T, Laporte G, Matl P. A concise guide to existing and emerging vehicle routing problem variants. European Journal of Operational Research. 2020;286(2):401–16. https://doi.org/10.1016/j.ejor.2019.10.010.
9. Mor A, Speranza MG. Vehicle routing problems over time: a survey. Annals of Operations Research. 2022;314(1):255–75. https://doi.org/10.1007/s10479-021-04488-0.
10. Desrochers M, Desrosiers J, Solomon M. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. Operations Research. 1992;40(2):342–54. https://doi.org/10.1287/opre.40.2.342.
11. Desaulniers G, Madsen O, Ropke S. The Vehicle Routing Problem with Time Windows. In: Toth, P., Vigo, D. (eds.) Vehicle Routing vol. 18, 2014;119–159. Chap. 5. https://doi.org/10.1137/1.9781611973594.ch5
12. Vidal T, Crainic TG, Gendreau M, Prins C. A unified solution framework for multi-attribute vehicle routing problems. European Journal of Operational Research. 2014;234(3):658–73. https://doi.org/10.1016/j.ejor.2013.09.045.
13. Amorim P, Parragh SN, Sperandio F, Almada-Lobo B. A rich vehicle routing problem dealing with perishable food: a case study. TOP. 2014;22(2):489–508. https://doi.org/10.1007/s11750-012-0266-4.
14. Keskin M, Çatay B. A matheuristic method for the electric vehicle routing problem with time windows and fast chargers. Computers & Operations Research. 2018;100:172–88. https://doi.org/10.1016/j.cor.2018.06.019.
15. Mendes N, Iori M. A Decision Support System for a Multi-trip Vehicle Routing Problem with Trucks and Drivers Scheduling. In: Proceedings of the 22nd International Conference on Enterprise Information Systems, pp. 339–349. SCITEPRESS - Science and Technology Publications, Prague, Czech Republic 2020; https://doi.org/10.5220/0009364403390349
16. Sciortino M, Lewis R, Thompson J. A School Bus Routing Heuristic Algorithm Allowing Heterogeneous Fleets and Bus Stop Selection. SN Computer Science. 2022;4(1):74. https://doi.org/10.1007/s42979-022-01466-6.
17. Cattaruzza D, Absi N, Feillet D. Vehicle routing problems with multiple trips. Annals of Operations Research. 2018;271(1):127–59. https://doi.org/10.1007/s10479-018-2988-7.
18. Masmoudi MA, Hosny M, Braekers K, Dammak A. Three effective metaheuristics to solve the multi-depot multi-trip heterogeneous dial-a-ride problem. Transportation Research Part E: Logistics and Transportation Review. 2016;96:60–80. https://doi.org/10.1016/j.tre.2016.10.002.
19. Wang Z. Delivering meals for multiple suppliers: Exclusive or sharing logistics service. Transportation Research Part E: Logistics and Transportation Review. 2018;118:496–512. https://doi.org/10.1016/j.tre.2018.09.001.
20. Pan B, Zhang Z, Lim A. Multi-trip time-dependent vehicle routing problem with time windows. European Journal of Operational Research. 2021;291(1):218–31. https://doi.org/10.1016/j.ejor.2020.09.022.
21. Nguyen VS, Pham QD, Nguyen TH, Bui QT. Modeling and solving a multi-trip multi-distribution center vehicle routing problem with lower-bound capacity constraints. Computers & Industrial Engineering. 2022;172: 108597. https://doi.org/10.1016/j.cie.2022.108597.
22. Power DJ, Sharda R. Decision Support Systems. In: Nof, S.Y. (ed.) Springer Handbook of Automation, pp. 1539–1548. Springer, Berlin, Heidelberg 2009; https://doi.org/10.1007/978-3-540-78831-7_87
23. Farshidi S, Jansen S, Jong R, Brinkkemper S. A decision support system for cloud service provider selection problem in software producing organizations. In: 2018 IEEE 20th Conference on Business Informatics (CBI), 2018;01:139–148 https://doi.org/10.1109/CBI.2018.00024
24. Christoforou A, Garriga M, Andreou AS, Baresi L. Supporting the Decision of Migrating to Microservices Through Multi-layer Fuzzy Cognitive Maps. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) Service-Oriented Computing, pp. 471–480. Springer, Cham 2017; https://doi.org/10.1007/978-3-319-69035-3_34
25. Di Francesco P, Malavolta I, Lago P. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In: 2017 IEEE International Conference on Software Architecture (ICSA), 2017;21–30: https://doi.org/10.1109/ICSA.2017.24
26. Keenan PB, Jankowski P. Spatial Decision Support Systems: Three decades on. Decision Support Systems. 2019;116:64–76. https://doi.org/10.1016/j.dss.2018.10.010.
27. ESRI: ArcGIS API for Python. https://developers.arcgis.com/python/api-reference/arcgis.gis.toc.html Accessed 2023-08-08
28. Taibi D, Lenarduzzi V, Pahl C. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. IEEE Cloud Computing. 2017;4(5):22–32. https://doi.org/10.1109/MCC.2017.4250931.
29. Hunt J. Applying the Model-View-Controller Pattern. In: Hunt, J. (ed.) Guide to the Unified Process Featuring UML, Java and Design Patterns, pp. 235–252. Springer, London 2003; https://doi.org/10.1007/1-85233-856-3_14
30. Santini A, Schneider M, Vidal T, Vigo D. Decomposition strategies for vehicle routing heuristics. INFORMS Journal on Computing. 2023;35(3):543–59. https://doi.org/10.1287/ijoc.2023.1288.
31. Savelsbergh MWP. The Vehicle Routing Problem with Time Windows: Minimizing Route Duration. ORSA Journal on

Computing. 1992;4(2):146–54. https://doi.org/10.1287/ijoc.4.2.146.
32. Cormen TH, Leiserson CE, Rivest RL, Stein C, editors. Introduction to Algorithms. 2nd ed. Cambridge, Mass: MIT Press and McGraw-Hill; 2001.
33. Lourenço HR, Martin OC, Stützle T. Iterated Local Search: Framework and Applications. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management Science, pp. 129–168. Springer, Cham 2019; https://doi.org/10.1007/978-3-319-91086-4_5
34. Vidigal Corrêa VH, Dong H, Iori M, Santos AG, Yagiura M, Zucchi G. An iterated local search for a multi-period orienteering problem arising in a car patrolling application. Networks. 2023. https://doi.org/10.1002/net.22187.
35. Biswas S, Behera S, Choudhury NBD. A Brief Review on the Barriers of Electric Vehicle Adoption and Present Scenario in India. In: Goyal, S.K., Palwalia, D.K., Tiwari, R., Gupta, Y. (eds.) Flexible Electronics for Electric Vehicles. Lecture Notes in Electrical Engineering, Springer, Singapore 2024;539–550 https://doi.org/10.1007/978-981-99-4795-9_51