European Journal of Operational Research 329 (2026) 87–95

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A multi-start local search matheuristic for the capacitated arc routing problem with irregular services

Demetrio Laganà [a],*, Paolo Paronuzzi [b]

[a] Department of Mechanical, Energy and Management Engineering (DIMEG), University of Calabria, Italy
[b] Department of Electrical, Electronic and Information Engineering (DEI), University of Bologna, Italy

ARTICLE INFO

Keywords:
Capacitated periodic arc routing
Matheuristic
Column generation
Local search

ABSTRACT

This paper investigates the periodic capacitated arc routing problem with irregular services, a challenging combinatorial optimization problem that arises in various real-world scenarios, such as waste collection and road maintenance. This problem is defined over a mixed graph and asks for scheduling a fleet of capacitated vehicles to meet the demands associated with a set of required links, while minimizing the routing costs over a given planning horizon. Each required link has its own service plan, indicating the frequency with which its demand must be met over the planning horizon. To solve this problem, we propose a novel matheuristic approach that combines a route-based mathematical formulation with a multi-start local search framework. The matheuristic incorporates two distinct local search strategies, which are crucial for improving solution quality, as revealed by the computational experiments. Additional experiments further confirm the effectiveness of the proposed matheuristic, comparing its performance against an exact method and a heuristic algorithm from the literature, solving the uncapacitated version of the problem. Finally, we analyze the impact of introducing the capacity constraint by comparing the solutions obtained in the two cases.

1. Introduction

The study of operational issues in the field of distribution logistics includes the formulation of combinatorial optimization problems, where the objective is to minimize the costs incurred due to the service provided by vehicles. In many applications, the service is performed when a vehicle reaches a designated physical location. These locations can be effectively represented through nodes within a network, and the result is the definition and solution of a node routing problem. Conversely, in numerous other applications, the service is performed when traversing specific *required* links connecting two locations and the result is the definition and solution of an Arc Routing Problem (ARP). Solving an ARP generally consists of finding a tour (or a set of tours) with total minimum cost traversing the set of required links of a network at least once. Real applications of $ARPs$ occur in several contexts, such as mail delivery, snow removal, meter reading and street cleaning, and ARP has been addressed in the scientific literature in many deterministic and uncertain variants (Corberán et al., 2021; De Maio et al., 2021; Mourão & Pinto, 2017).

The most basic version of ARP is the Chinese postman problem, which asks to find a minimum-cost tour that traverses all the links of the network. The rural postman problem generalizes the former one, as only a subset of links are required. In the Capacitated ARP ($CARP$), the required links are associated with demands which must be satisfied by a fleet of vehicles with limited capacity; thus, in this case, serving a link implies the consumption of the vehicle's capacity. When the required links must be served repeatedly over a planning horizon, a periodic ARP can be defined; in this setting, each required link is associated with a frequency representing the number of times it must be served during the planning horizon. This horizon is represented as a set of periods, with each period typically constituting a day. Furthermore, when the service of each required link is required at a different frequency within subsets of consecutive periods that constitute a partition of the planning horizon, services are said to be *irregular*.

In all these cases, a directed arc of a graph can represent a one-way service over a link of the network, while an undirected edge or a couple of opposite directed arcs can represent a two-way service. The latter specific representation depends on the context of the application: a two-way service is represented by two opposite arcs if the service on each side has to be performed separately in the two directions. Conversely, when the service needs to be performed only once, no matter which side, it is modeled as an edge. A graph with both types of links is called *mixed*.

This paper investigates for the first time the Periodic Capacitated Arc Routing Problem with Irregular Services ($PCARP–IS$), defined

* Corresponding author.
E-mail addresses: demetrio.lagana@unical.it (D. Laganà), paolo.paronuzzi@unibo.it (P. Paronuzzi).

https://doi.org/10.1016/j.ejor.2025.07.032
Received 3 March 2025; Accepted 14 July 2025
Available online 13 August 2025

on a mixed graph. For each period within a planning horizon, the $PCARP–IS$ aims to schedule a fleet of capacitated vehicles to meet the demands of all required links, accounting for irregular services and minimizing total routing costs.

The primary application of the $PCARP–IS$ is in the context of garbage collection, where collection frequency varies based on the volume of waste produced along the streets throughout the week. Thus, garbage collection is typically carried out periodically, with different roads requiring different service frequencies depending on their importance within the network and the amount of waste they produce.

The frequency of visits also depends on the day of the week or the week itself, as observed by Monroy et al. (2013). For instance, certain roads may require more frequent servicing on weekends due to increased waste volumes, whereas fewer visits may be needed during the workweek. Conversely, other roads may require more attention on weekdays than on weekends. To accommodate this variability, the time horizon is partitioned into discrete subsets of varying sizes, with the number of visits adapting accordingly. Moreover, if capacity is considered as a temporal limitation and a demand represents a time requirement, the $PCARP–IS$ can be employed to study road maintenance problems, where the service on each road is provided by teams of workers with limited temporal availability according to their contractual obligations.

When relaxing capacity constraints, the $PCARP–IS$ reduces to the Periodic Rural Postman Problem with Irregular Services ($PRPP–IS$). This problem was introduced by Benavent et al. (2019) and may be used for modeling further applications. To illustrate, Marzolf et al. (2006) presents a case study of road network monitoring, with reference to the Quebec Ministry of Transportation. The activity is conducted on a daily basis by patrol trucks, with the objective of maintaining the safety and viability of the road network through the rapid detection of incidents that occur on it.

Contributions. This work contributes to the literature by introducing and addressing the $PCARP–IS$. We provide a new matheuristic approach, which is built upon a route-based Integer Linear Program (ILP) formulation of the $PCARP–IS$ and the framework of the multi-start local search algorithm. The matheuristic incorporates two distinct local search strategies. Computational experiments are conducted in two phases. First, the matheuristic is tuned and evaluated on a set of $PCARP–IS$ instances generated by adapting existing benchmarks for the $PRPP–IS$. This phase demonstrates the effectiveness of the local search strategies. Subsequently, the performance of the tuned matheuristic is compared against an exact method and a heuristic algorithm from the literature on the original $PRPP–IS$ instances. Finally, a comparative analysis is performed between the solutions obtained for the $PCARP–IS$ and the $PRPP–IS$, providing insights on the impact of vehicle capacity constraints on problem solutions.

Outline. The paper is organized as follows. Section 2 formally introduces the $PCARP–IS$. In Section 3, we position our work with respect to the pertinent literature. We describe a mathematical formulation for the $PCARP–IS$ in Section 4. In Section 5 we detail the matheuristic designed for the $PCARP–IS$, and in Section 6 we outline the data sets and present our computational experiments. We conclude with a summary in Section 7.

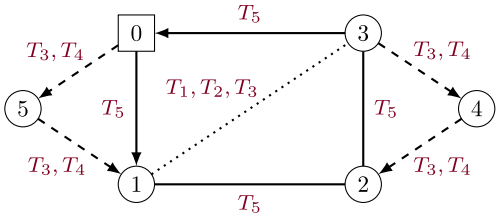
2. Problem definition

The $PCARP–IS$ is defined on a mixed graph $G = (V, A, E)$, where V is the set of the vertices, including a depot 0, A is the set of arcs, E is the set of edges, and $A \cup E$ is the set of links. A link $l \in A \cup E$ with a positive demand $d_l > 0$ is named *required*. A tour is a circuit defined as an ordered sequence of oriented links, starting from the depot, in which the same edge or arc may appear more than once within the sequence. Let $\bar{A} = \{a \in A : d_a > 0\} \subseteq A$ denote the set of required arcs, $\bar{E} = \{e \in E : d_e > 0\} \subseteq E$ denote the set of required edges, and $L = \bar{A} \cup \bar{E}$ denote the set of required links. The demands must be satisfied over a finite and discrete time horizon H ($h = |H|$) by means of an unlimited fleet of homogeneous vehicles, each one having capacity Q, that is located in the depot. In this work, both terms *serve* a link and *visit* a link (along with their related forms and derivations) refer to the act of fulfilling the demand associated with that required link. Thus, a vehicle can serve a link if it is associated with a tour containing that required link. The sum of the demands of the links visited by the same vehicle in some period $t \in H$ must respect its capacity.

Every required link has its own service plan that identifies the number of times it must be served during the time horizon. More precisely, for each required link $l \in L$, we define P_l as a partition of H into subsets of consecutive periods, and we define f_T^l as the *frequency*, i.e., the number of visits in $T \in P_l$ required by the link l (where $f_T^l \leq |T|$, since each required link can be visited at most once a period). If a link l requires a service (or not) in a single period t, this can be modeled by setting $T = \{t\} \in P_l$ and $f_T^l = 1$ (or 0). We assume that the demand d_l is constant across all periods in which link l is serviced. Hence, the total demand associated with link l over the planning horizon is $d_l \cdot \sum_{T \in P_l} f_T^l$. Finally, let $c_l \geq 0$ be the cost that a vehicle incurs when traversing link $l \in A \cup E$ (serving a link does not involve any additional cost).

The goal of the $PCARP–IS$ is to schedule tours of vehicles in each period of the time horizon to satisfy the service plan of all the required links, while respecting the vehicles' capacity and minimizing the total routing cost.

Figure below illustrates an instance of $PCARP–IS$ consisting of 6 vertices, where vertex 0 represents the depot. The set of links includes arcs: $l_1 = (0,1)$, $l_2 = (0,5)$, $l_3 = (3,0)$, $l_4 = (3,4)$, $l_5 = (4,2)$, and $l_6 = (5,1)$, as well as edges: $l_7 = (1,2)$, $l_8 = (1,3)$, and $l_9 = (2,3)$. To reflect a realistic scenario, we assume the planning horizon corresponds to a week, i.e., $H = \{\text{Mon}, \text{Tue}, \text{Wed}, \text{Thu}, \text{Fri}, \text{Sat}, \text{Sun}\}$. Then, we define $T_1 = \{\text{Mon}, \text{Tue}\}$, $T_2 = \{\text{Wed}, \text{Thu}\}$, $T_3 = \{\text{Fri}, \text{Sat}, \text{Sun}\}$, $T_4 = \{\text{Mon}, \text{Tue}, \text{Wed}, \text{Thu}\}$, $T_5 = \{\text{Mon}, \text{Tue}, \text{Wed}, \text{Thu}, \text{Fri}, \text{Sat}, \text{Sun}\}$ and we consider: $P_{l_1} = P_{l_3} = P_{l_7} = P_{l_9} = \{T_5\}$, $P_{l_2} = P_{l_4} = P_{l_5} = P_{l_6} = \{T_3, T_4\}$, and $P_{l_8} = \{T_1, T_2, T_3\}$. The values of vehicle capacity, demands, and service frequencies can be arbitrarily defined, as they are not relevant for the purpose of this illustrative example.



It is worth noting that the problem definition allows for service assignments that may fall on consecutive days across consecutive subsets of periods. More explicitly, if the time horizon is one month and a link must be visited once a week, the problem definition considers feasible visiting it both on Sunday and Monday. While this eventuality may be acceptable also in some practical settings, in contexts where a minimum time interval between services is required, this can be enforced through additional constraints or post-processing. However, in the considered framework, it is still possible to enforce approximate spacing by restricting the days in which a link can be served, for instance, by allowing exactly one visit in the first part of the week and none in the second.

3. Literature review

In light of the practical relevance of the $PCARP–IS$, a summary of the literature on periodic $ARPs$ is presented. Ghiani et al. (2005) study

the periodic undirected rural postman problem, in which every required edge must be served a specified number of times over a planning time horizon in a manner that ensures equal intervals between service days. The authors design a local search heuristic to solve the problem. A similar problem is studied by Triki (2017) who restricts the time horizon to two working days but considers different service patterns for the edges: some edges require daily servicing, some require servicing on one of two days, and some do not require servicing at all. The author proposes a construction heuristic based on a cluster-first-route-second approach to address the problem. The Periodic $CARP$ ($PCARP$) has been introduced by Chu et al. (2004), who define the problem on a undirected graph where required edges must be served a given number of times within a specified time horizon. The same authors extend their research (Chu et al., 2005) proposing valid linear programming models and several heuristic and metaheuristic algorithms; a scatter search method is the best performing one. Lacomme et al. (2005) present several versions of the $PCARP$ and consider realistic networks. They introduce a memetic algorithm. Later, Mei et al. (2011) also design a memetic algorithm that outperformed the previous one. More recently, Vidal (2017) proposed a hybrid genetic algorithm that includes extended neighborhoods to solve the $PCARP$ and other capacitated $ARPs$. The experiments presented in that paper demonstrate that this versatile algorithm outperforms most of the other approaches mentioned so far and can be considered the state-of-the-art for this class of problems.

The $PCARP$'s primary objective is to minimize routing costs over time. However, in the literature, objectives other than cost minimization have been considered. Chu et al. (2006) highlight that the $PCARP$ is a strategic issue requiring long-term planning, so companies may prioritize fleet size over total travel cost. They consider separately these two objectives in their greedy heuristic and a scatter search method. Khosravi et al. (2015) apply $PCARP$ to the waste collection in mobile disposal sites. They also take into account the minimization of number of vehicles in their objective, using an aggregated and weighted function. Chen and Hao (2018) propose a two-phase hybrid local search algorithm that hierarchically considers the two objectives. The initial phase optimizes the fleet size, while the subsequent phase optimizes routing costs using the resulting number of vehicles from the first phase. Monroy et al. (2013) introduce a first version of the $PCARP$ with irregular services. Besides considering only directed graphs, their variant mainly differs from ours in that required arcs are associated with both a demand, consuming the vehicle capacity, and a profit. Traversing a required arc earns the associated profit, and the objective is to maximize total profit, instead of minimizing routing costs. The authors present a mathematical model and propose a two-stage heuristic algorithm. Riquelme-Rodriguez, Gamache, and Langevin (2014), Riquelme-Rodriguez, Langevin, and Gamache (2014) investigate the $PCARP$ with inventory constraints, where required links represent customers and material is delivered to them so that each one reaches a desired inventory level. (Huang & Lin, 2014) address a variant of the $PCARP$ in which the graph contains a refill point, i.e., a node where the vehicle's capacity can be recovered. The authors study a case related to tree watering in Kaohsiung (Taiwan) and use an ant colony optimization algorithm. Ant colony optimization is also proposed by Kanso (2020) for the multi-depot $PCARP$. Kiilerich and Wohlk (2018) present the semi-periodic $PCARP$, where required links must be served at regular intervals. The authors also deal with other variants of the $CARP$ and present test cases using real data on waste collection in five areas in Denmark. As mentioned in the introduction, Benavent et al. (2019) introduced the $PRPP–IS$ that correspond to a $PCARP–IS$ in which demands and capacities are not considered. Thus, a single vehicle is enough to serve all required links. They propose an exact approach based on a mathematical formulation along with several families of valid inequalities. Later, Benavent et al. (2023) present an hybrid algorithm that combines heuristic methods and mathematical programming to solve the same problem.

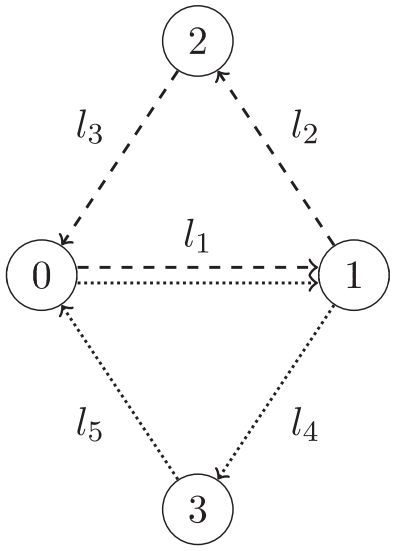


Fig. 1. An example to illustrate the meaning of variables used by formulation **RT**.

This paper addresses the $PCARP–IS$ using a matheuristic approach that integrates a mathematical model into a multi-start local search. Similar matheuristic schemes have been proved to be effective also in periodic node routing problems; see Archetti and Speranza (2014) for a survey on matheuristic algorithms in vehicle routing problems and Laganà et al. (2024), Vadseth et al. (2021), as examples of recent works on a periodic vehicle routing problem embedded in an integrated logistics problem.

4. A mathematical formulation

It is important to remark that in the $PCARP–IS$ a link can either be traversed and served, or traversed but not served. Clearly, in the latter case, the demand of the link does not consume the vehicle's capacity. We propose a mathematical formulation that considers so-called *selective* routes in which some of the traversed links are *visitable*, i.e., may be served (with the model deciding which ones), while the remaining traversed links (if any) are not considered for service. Thus, a selective route r is uniquely defined by a pair $\{\pi(r), L(r)\}$, where $\pi(r)$ is the minimum-cost tour that includes $L(r)$ and $L(r) \subseteq \pi(r)$ is its set of visitable links. We indicate by c_r the cost of a selective route, defined as $c_r = \sum_{l \in \pi(r)} c_l$,. Hence, two different selective routes may have the same cost and traversing the same links, but differ in the set of visitable links. An example of this situation is shown in Fig. 1, suppose $l_1 = (0,1)$ with $d_{l_1} = 4$, $l_2 = (1,2)$ with $d_{l_2} = 5$, $l_3 = (2,0)$ with $d_{l_3} = 6$, and $Q = 10$. A selective route r, such that $\pi(r) = \{l_1, l_2, l_3\}$ and $L(r) = \{l_1, l_2\}$, differs from the selective route r', such that $\pi(r') = \{l_1, l_2, l_3\}$ and $L(r') = \{l_3\}$. In addition, we say that $L(r)$ is maximal with respect to the inclusion of visitable links if including the demand of any other link belonging to $\pi(r)$ would exceed the vehicle's capacity. In the above example, $L(r)$ is maximal, while $L(r')$ is not. Let us also suppose that $l_4 = (1,3)$ and $l_5 = (3,0)$ are not required, and consider a selective route ρ such that $\pi(\rho) = \{l_1, l_4, l_5\}$. Since the total demand of the links in $\pi(\rho)$ does not exceed the vehicle capacity, then all the required links are included in the corresponding maximal $L(\rho) = \{l_1\}$. Note that, if $c_\rho < c_r$, then there exist instances of $PCARP–IS$ whose optimal solution must include the tour associated with ρ (for example, when l_1 must be served on different period than l_2 and l_3).

Let us define R as the set of all selective routes r such that $L(r)$ is maximal with respect to the inclusion of visitable links. We define binary variables y_{rt} taking value 1 if the selective route $r \in R$ is used in period $t \in H$, and assignment variables x_{lt} taking value 1 if the required

link $l \in L$ is served in period $t \in H$. We formulate the $PCARP\text{–}IS$ as follows:

$$\textbf{(RT)}\ \min \quad \sum_{t\in H}\sum_{r\in R} c_r y_{rt} \tag{1}$$

$$\text{s.t.} \quad \sum_{t\in T} x_{lt} = f_T^l \qquad \forall l \in L, \forall T \in P_l, \tag{2}$$

$$x_{lt} \le \sum_{r\in R: l\in L(r)} y_{rt} \qquad \forall l \in L, \forall t \in H, \tag{3}$$

$$y_{rt} \in \{0,1\} \qquad \forall r \in R,\ \forall t \in H, \tag{4}$$

$$x_{lt} \in \{0,1\} \qquad \forall l \in L,\ \forall t \in H. \tag{5}$$

The objective function (1) minimizes the total routing cost; constraints (2) guarantee the respect of the service plan of each required link; constraints (3) impose that at least one selective route r whose set $L(r)$ contains the required link $l \in L$ must be used in period $t \in H$, if l is served in t; finally, (4)–(5) ensure that the variables are binary.

Formulation **RT** allows the capacity constraints to be integrated directly into the definition of the y_{rt} variables, while maintaining flexibility in deciding which traversed link to serve in each period through the x_{lt} variables. These variables also ensure that each required link is visited at most once per period. Specifically, if a link l is served in a given period t, i.e. $x_{lt} = 1$, and more than one y_{rt} variable such that $l \in L(r)$ is activated in the same period t, then we can arbitrarily decide which of the corresponding vehicles actually visits l. We refer again to Fig. 1 to illustrate a simple example of this situation. Suppose $h = 1$, all arcs are required with $d_l = 1$ and the vehicles' capacity is $Q = 3$. Then, the optimal solution comprises the activation of selective routes r (dashed in the figure), such that $\pi(r) = L(r) = \{l_1, l_2, l_3\}$, and ρ (dotted in the figure), such that $\pi(\rho) = L(\rho) = \{l_1, l_4, l_5\}$, as well as the service of the required arc $l_1 = (0,1)$ through any of these selective routes.

In summary, the mathematical formulation **RT** exploits the implicit fulfillment of capacity constraints within the selective route definition to decouple service selection from tour selection. If compared to a formulation where the capacity constraint is not embedded in the route definition, this approach reduces the model's symmetry. In fact, allowing that all the required links traversed by a route can be visited, without considering the capacity in the variable definition, would entail the introduction of an additional index l to the y_{rt} variables, to determine which route r serves a link l in a given period t. In the example depicted in Fig. 1, the optimal solution of model **RT** is unique, while the formulation based on three-index variables y_{lrt} has two equivalent solutions, differing in which route serves the common link.

5. A matheuristic for the RT formulation

Formulation **RT** provides an exact model of the $PCARP\text{–}IS$, though the size of the model does not grow polynomially with the input size due to the exponential number of variables involved. In order to manage this exponential number of variables, we propose an iterative heuristic algorithm. At each iteration, the algorithm operates on a limited subset of y_{rt} variables, focusing on those that are part of the best solution found in the previous iteration, as well as additional variables introduced to improve the incumbent solution. This process defines a *neighborhood* around the current solution. The neighborhood includes all feasible solutions that can be generated by recombining the variables in the current best solution with the remaining variables.

More specifically, the proposed approach, described in Algorithm 1, begins with an initialization phase where a starting pool of variables is generated. Following this, the algorithm enters a local search loop, where (i) the formulation **RT** is constructed using the current subset of variables and solved via an ILP solver; then, (ii) the subset is updated by discarding variables associated with selective routes that do not appear in the current incumbent solution and generating new ones based on selective routes that appear in the current incumbent solution. This process is iterated until no further improvement is observed. Then, the initialization phase is repeated, and the local search loop restarts, until arbitrary stopping criteria are met.

This strategy gives rise to a matheuristic algorithm which we refer to as the Multi-start Local Search Matheuristic, abbreviated as `MLSM` in the following.

Algorithm 1 Multi-start Local Search Matheuristic.

```
while not stopping criteria do
    Initialization: generate an initial pool of y_rt variables;
    repeat
        solve model RT;
        update the pool of y_rt variables;
    until solution not improved
end while
```

5.1. Initialization

In this section, we present the procedure used to build an initial pool of y_{rt} variables. The basic idea is to solve the problem at hand into two distinct phases: first, assigning the required links to periods in which they are served; second, deciding how to visit the required links, i.e., which tours to use in each period.

5.1.1. First phase

We denote by $L(t)$ the set of required links assigned to period t and, initially, we set $L(t) = \emptyset$ for every period $t \in H$. The assignment process is described as follows: for each link $l \in L$ and for each $T \in P_l$ such that $f_T^l > 0$, we define $\bar{T}$ as the largest subset of T such that, for every period $t \in \bar{T}$, the conditions $L(t) \neq \emptyset$ and $l \notin L(t)$ are satisfied. We note that $\bar{T}$ may be empty if no period in T satisfies the above conditions. With the goal of grouping the required links into the fewest possible number of periods, the assignment of link l is carried out according to these rules:

1. If $|\bar{T}| = f_T^l$, then assign link l to all sets $L(t)$ such that $t \in \bar{T}$.
2. If $|\bar{T}| < f_T^l$, then assign link l to all sets $L(t)$ such that $t \in \bar{T}$, and additionally to $f_T^l - |\bar{T}|$ other periods randomly selected from T.
3. If $|\bar{T}| > f_T^l$, then assign link l to f_T^l periods randomly selected from $\bar{T}$.

The procedure is also randomized by shuffling the order in which the required links are scanned. Thus, running this procedure at different times is likely to produce different assignments. This guarantees a different starting solution for each iteration of `MLSM`.

We illustrate the assignment process with a concise example. Consider three links: l_1, with $P_{l_1} = \{\{1,2,3\},\{4\}\}$ and $f_T^{l_1} = 2$ and 0, respectively; l_2, with $P_{l_2} = \{\{1\},\{2,3,4\}\}$ and $f_T^{l_2} = 0$ and 2, respectively; and l_3, with $P_{l_3} = \{\{1\},\{2,3\},\{4\}\}$ and $f_T^{l_3} = 0$, 1 and 0, respectively. Initially, $L(t) = \emptyset$ for all $t \in \{1,2,3,4\}$. Then, the assignment proceeds sequentially as follows.

Iter 1. For l_1 and $T = \{1,2,3\}$, we apply rule 2, as $\bar{T} = \emptyset$ and hence satisfies $|\bar{T}| < f_T^{l_1} = 2$; l_1 is assigned to two random periods in $\{1,2,3\}$, say $\{1,2\}$, yielding $L(1) = \{l_1\}$ and $L(2) = \{l_1\}$.

Iter 2. For l_2 and $T = \{2,3,4\}$, we apply rule 2 again, as $\bar{T} = \{2\}$ and hence satisfies $|\bar{T}| < f_T^{l_2} = 2$; l_2 is assigned to period 2 plus one random period from $\{3,4\}$, say period 3, yielding $L(2) = \{l_1, l_2\}$ and $L(3) = \{l_2\}$.

Iter 3. For l_3 and $T = \{2,3\}$, we apply rule 3, as $\bar{T} = \{2,3\}$ and hence satisfies $|\bar{T}| > f_T^{l_3} = 1$; one period from $\{2,3\}$ is randomly selected, say period 2, resulting in $L(2) = \{l_1, l_2, l_3\}$.

The final assignment is $L(1) = \{l_1\}$, $L(2) = \{l_1, l_2, l_3\}$, $L(3) = \{l_2\}$ and $L(4) = \emptyset$.

5.1.2. Second phase

We recall that the $CARP$ is a $PCARP\text{–}IS$ in which $h = 1$ and, therefore, required links must be served once. Thus, in the second phase of the procedure, we solve a $CARP$ instance for each period t such that $L(t) \neq \emptyset$, using an *ad-hoc* solution algorithm. Then, from the obtained solution we derive an initial pool of y_{rt} variables. Specifically, for each vehicle used in the $CARP$ solution we define a variable y_{rt} such that $\pi(r)$ corresponds to the vehicle's tour and $L(r)$ contains all the links visited by the vehicle. In addition, we include in $L(r)$ other required links randomly selected among those that are traversed (but not visited) by the vehicle, until capacity is saturated; in this way, we ensure that set $L(r)$ is maximal with respect to the inclusion of visitable links. Solving model **RT** with the pool of variables obtained from this initialization phase guarantees the finding of a feasible solution for the $PCARP\text{–}IS$.

5.2. Local search

After the initialization step, we iteratively repeat a phase in which model **RT** is solved and the pool of variables is updated. This phase works as a local search in which new columns y_{rt} are generated at each iteration. Given a feasible solution $(\bar{\mathbf{y}}, \bar{\mathbf{x}})$, we remove from the model all the variables y_{rt} such that $\bar{y}_{rt} = 0$, while we keep all the variables y_{rt} such that $\bar{y}_{rt} = 1$; this ensures that the feasible solution from the previous iteration is always maintained. Then, we use two different strategies to generate new columns.

5.2.1. First local search strategy

Algorithm 2 reports the pseudo-code of the first local search strategy, named *Tour Mixing*.

The basic idea of the first strategy is define new variables y_{rt} by mixing the vehicle tours used in the different periods of the current feasible solution returned by model **RT**. We recall that a variable y_{rt} is associated with a selective route $r = \{\pi(r), L(r)\}$ and a period t. In the procedure outlined below, the period t of the new variable is clear from the context. As for the selective route r, the focus is on the definition of the set $\pi(r)$, while the associated set $L(r)$ always consists of required links randomly selected from $\pi(r)$.

In addition, we define a path $[l_1, l_2]$ as an ordered and oriented subset of links, starting from link l_1 and ending in link l_2 (both extreme links are included in the path), and we also define the function $sp(l_1, l_2)$ returning the shortest path from the source node of l_1 to the destination node of l_2; thus, $sp(l_1, l_2)$ does not necessarily contain l_1 and l_2.

Given a tour $\pi(r)$ that is used by some vehicle in the current feasible solution and aiming to find tours that are different from it, we consider path $[l_1, l_2] \subset \pi(r)$ whose cost $c_{l_1 l_2}$ is strictly higher than the cost of $sp(l_1, l_2)$. Limiting our focus to this kind of paths, for each period t, for each selective route r such that $\bar{y}_{rt} = 1$ and for each link $l_1 \in \pi(r)$, we generate the first, say, at most γ variables $y_{r't}$ with $\pi(r') = \pi(r) \setminus [l_1, l_2] \cup sp(l_1, l_2)$, where γ is a parameter to be tuned, and l_2 are subsequent links to l_1 in the sequence defined by $\pi(r)$ (line 4). Then, for all the remaining periods $t' \neq t$, we generate one variable $y_{r't'}$ for each of the removed path by including the same path into the tour $\pi(r'')$ of a selective route r'', such that $\bar{y}_{r''t'} = 1$. If more than one such selective route exists, we choose the one with the lowest insertion cost, computed without changing the order of the links in the path nor in $\pi(r'')$. This insertion may require additional links to be used, so $\pi(r') \subseteq \pi(r'') \cup [l_1, l_2]$ (line 9). If some period has no active vehicle, we generate the variable associated with the cheapest tour visiting $[l_1, l_2]$ only, by computing $\pi(r')$ as the shortest path from the depot to the source node of l_1 and the shortest path from the destination node of l_2 to the depot (line 11).

In Algorithm 2, Function $c(\rho, [l_1, l_2])$ returns the lowest insertion cost of path $[l_1, l_2]$ into the sequence $\pi(\rho)$ of a selective route ρ. We remark that r' is reused at different points in Algorithm 1 to denote newly generated routes.

Algorithm 2 Tour Mixing.

```
1: for t ∈ T and r ∈ R : ȳ_rt = 1 do
2:     for l_1 ∈ π(r) do
3:         for at most γ paths [l_1, l_2] ⊂ π(r) : sp(l_1, l_2) < c_{l_1 l_2} do
4:             generate y_{r't} with π(r') = π(r) \ [l_1, l_2] ∪ sp(l_1, l_2);
5:             for t' ∈ T : t' ≠ t do
6:                 define Ω = {ρ ∈ R\{r} : ȳ_{ρt'} = 1}
7:                 if Ω ≠ ∅ then
8:                     r'' = argmin {c(ρ, [l_1, l_2]) : ρ ∈ Ω};
9:                     generate the cheapest y_{r't'} with π(r') ⊆ π(r'') ∪
   [l_1, l_2];
10:                else
11:                    generate the cheapest y_{r't'} such that [l_1, l_2] ⊆ π(r')
12:                end if
13:            end for
14:        end for
15:    end for
16: end for
```

At each iteration, this procedure generates a set of variables y_{rt}, whose number mainly depends on parameter γ. Increasing γ is equivalent to enlarging the size of the local search neighborhood we are considering, enhancing the chances to improve the incumbent solution at the current iteration. On the other hand, a larger value of γ also increases the number of variables y_{rt}, resulting in a formulation **RT** that generally requires more computing time to be solved.

5.2.2. Second local search strategy

In the second strategy, we solve a $CARP$ for each period $t \in T$, by using the assignment provided by $\bar{x}_{lt}$ to define the corresponding instance. Similarly to the initialization phase, for each vehicle used in the $CARP$ solution we define a variable y_{rt} such that $\pi(r)$ corresponds to the vehicle's tour and $L(r)$ contains all the links visited by the vehicle. Additionally, to ensure set $L(r)$ is maximal, we include in $L(r)$ other required links randomly selected among those that are traversed (but not visited) by the vehicle, until capacity is saturated.

This strategy is much more time consuming than the first one. For this reason, we only use this strategy when the first one does not improve the objective function.

6. Computational experiments

The first part of this section focuses on confirming the effectiveness of the various components of MLSM.

As mentioned in the introduction, this is the first work addressing the $PCARP\text{–}IS$, so no direct comparison with other algorithms is possible. The only feasible comparison is with approaches that solve a problem closely related to $PCARP\text{–}IS$. Therefore, in the second part we assess our method on the $PRPP\text{–}IS$, that is the uncapacitated (single-vehicle) version of the problem. We compare MLSM against both the exact approach proposed by Benavent et al. (2019) for the $PRPP\text{–}IS$, and the heuristic algorithm proposed by Benavent et al. (2023) for the $PRPP\text{–}IS$. The computational results show that our method is competitive with both of them in producing high-quality solutions.

We conclude this section by comparing the solutions obtained when solving the two versions of the problem, investigating the impact of the capacity constraint in term of routing cost and other relevant characteristics.

Instance benchmark. We derive the instances used in our experiments from those proposed by Benavent et al. (2019) for the $PRPP\text{–}IS$. Specifically, we generate a new benchmark by assigning a random integer demand, uniformly distributed between 1 and 100, to each required link. We set the value Q of the capacity equal to $\lfloor \sum_{l \in L} \sum_{T \in P_l} f_T^l d_l / h \rfloor$.

These new capacitated instances are used in the first part of our computational study to show the effectiveness of our matheuristic approach. For the uncapacitated (single-vehicle) version of the problem, assessed in the second part of this section, we use the original instances without modifications. All instances are defined on a time horizon of 7 periods, denoted by $H = \{1, 2, 3, 4, 5, 6, 7\}$. The following subsets of periods are defined: $T_1 = \{1, 2\}$, $T_2 = \{3, 4\}$, $T_3 = \{5, 6, 7\}$, $T_4 = \{1, 2, 3, 4\}$, and $T_5 = \{1, 2, 3, 4, 5, 6, 7\}$.

Based on instance features, both benchmarks are partitioned into four classes of instances:

1. `pc_m` – all links in the graph are required, with P_l equal to $\{T_1, T_2, T_3\}$, $\{T_3, T_4\}$, or $\{T_5\}$;
2. `pc_s` – all links in the graph are required, with P_l equal to $\{T_3, T_4\}$ or $\{T_5\}$;
3. `pr_m` – some links in the graph are required, with P_l equal to $\{T_1, T_2, T_3\}$, $\{T_3, T_4\}$, or $\{T_5\}$;
4. `pr_s` – some links in the graph are required, with P_l equal to $\{T_3, T_4\}$ or $\{T_5\}$.

Each class contains 34 instances, resulting in a total of 136 instances. The number of vertices varies between 24 and 50, the number of links between 37 and 150, and the total number of services (i.e., the sum of the frequencies) ranges from 43 to 349.

For further details on how the instances were generated, we refer to Benavent et al. (2019). The new benchmark of capacitated instances and the comprehensive computational results are accessible at https://github.com/paoloparonuzzi/PCARP-IS.git.

Computational setting. Our algorithm is implemented in C++ and uses the Gurobi Optimizer version 11 to solve the ILP model **RT** in single-threaded mode. All the remaining Gurobi parameters are left to their default values. The experiments are performed on a computer equipped with an AMD Ryzen 3960X processor clocked at 3.8 GHz, 128 GB RAM, running Linux.

When solving a $CARP$ instance, we use the hybrid genetic algorithm (HGS-CARP) of Vidal (2017), stopped after 1000 iterations without improvement in the best solution.

Algorithm `MLSM` checks the stopping criteria at each restart. It halts if either the number of restarts without improvements in the incumbent solution exceeds 50 or the elapsed time exceeds 30 min.

6.1. Tuning and evaluation of the algorithm

A way to assess the effectiveness of an algorithm component is by defining an alternative configuration of the algorithm in which the component is removed. However, removing only the first local search strategy of `MLSM` would be meaningless, given that the second strategy requires a solution different from the initial one to be effective. Thus, we also include in this analysis the computational results for tuning parameter γ introduced in Section 5.2.1, which controls the neighborhood size considered at each iteration. Preliminary experiments found $\gamma = 20$ to be the best value of this parameter for `MLSM`; in our analysis, we additionally report the results obtained with $\gamma = 1$ (configuration `MLSM-`γ1) and $\gamma = 50$ (configuration `MLSM-`γ50).

Configuration `NoF2` is a variant in which only the second local search strategy is deactivated (see Section 5.2.2).

Finally, configuration `VID` is a version in which the local search phase is completely deactivated, resulting in an algorithm that is heavily based on the approach implemented by Vidal (2017).

This integrated analysis allows to evaluate both the best setting of γ and the impact of each strategy.

Table 1 reports, for each instance class (one per row) and for each considered algorithm configuration, the number of times that a specific configuration finds the best solution among all configurations (columns Best), the average percentage gap (columns Gap) and the average runtime in seconds (columns Time). The percentage gap is computed as $100\times(\bar{z}-z^*)/z^*$, where $\bar{z}$ is the solution value found by the configuration and z^* is the best solution found by any of the configurations. The final row of the table summarizes the overall performance by reporting the total count of best solutions found by each configuration across all instance classes and the average percentage gap and runtime over all instances.

The results show that `MLSM` achieves the best performance across all classes of instances, with a total of 90 best solutions found and an average percentage gap of 0.15. However, this performance comes at the cost of a relatively large average runtime of 711.4 s.

The configuration with $\gamma = 1$ finds only 40 best solutions and has an average percentage gap of 0.64, indicating that small neighborhood sizes may limit solution quality, although this configuration has a shorter average runtime (568.5 s). On the other hand, the configuration with $\gamma = 50$ finds 70 best solutions (average percentage gap of 0.26) with a moderate increase in runtime (768.6 s) compared to `MLSM`. The `NoF2` variant, which deactivates the second local search strategy, finds 43 best solutions with only a moderate decrease in runtime (694.5 s) compared to `MLSM`, indicating that this strategy plays a role in improving solution quality. Similarly, `VID`, which deactivates the local search phase entirely, achieves the fastest average runtime (399.1 s) but identifies only 24 best solutions and has an average percentage gap of 1.24, suggesting that the lack of local search limits its effectiveness.

In summary, `MLSM` is the most effective configuration in terms of solution quality, highlighting the importance of both the local search strategies in reaching good quality solutions.

6.2. Comparison with state-of-the-art algorithms on the $PRPP–IS$

In this section, we compare our algorithm against both the exact approach proposed by Benavent et al. (2019), which we refer to as `B19`, and the heuristic algorithm proposed by Benavent et al. (2023), which we refer to as `B23`. In order to ensure consistency in the comparison, we re-executed the tests for both `B19` and `B23` on the same machine used for our algorithm. As CPLEX was used as solver, we used the more recent version 22.1 of IBM CPLEX, replacing version 12.6. We run CPLEX in single-thread mode.

The exact approach `B19` is based on a mathematical formulation with an exponential number of constraints. It is solved using a tailored branch-and-cut algorithm that adds several families of valid inequalities. The heuristic algorithm `B23` relies on a two-phase hybrid algorithm that combines heuristics and a so-called fragment-based model. This model operates as the final step to produce a single feasible solution. Both these algorithms are limited to the $PRPP–IS$ and cannot be easily extended to handle capacity constraints. In contrast, our matheuristic approach addresses the capacitated version of the problem and, although it has been specifically designed and optimized for this variant, it also performs well on the uncapacitated version, as we will show in our experiments. Moreover, if one would exploit multiple threads, our algorithm is naturally parallelizable, as each restart operates independently and could be executed on separate threads.

Fig. 2 compares the overall performance of the algorithms. The x-axis represents the average execution time in seconds, while there are two y-axes: the left y-axis represents the average percentage gap, and the right y-axis represents the count of the best solutions found by an algorithm. The percentage gap is computed as $100 \times (\bar{z} - z^*)/z^*$, where $\bar{z}$ is the solution value found by the algorithm and z^* is the best solution found by any of the three algorithms. The best solutions count is defined as the total count of times each algorithm achieved the best solution across instances. The average percentage gap is represented by a dot for each algorithm. The best solutions count is represented by vertical bars. The labels next to each dot identify the algorithm associated with each pair of gap and best solutions values.

We observe that `MLSM` achieves the smallest gap of 0.25%, indicating its effectiveness in finding high-quality solutions. This performance

Table 1
Comparison of the performance obtained by different configurations solving the *PCARP–IS*.

	MLSM			MLSM-γ1			MLSM-γ50			NoF2			VID		
	Best	Gap	Time	Best	Gap	Time	Best	Gap	Time	Best	Gap	Time	Best	Gap	Time
pc_m	23	0.13	1366.2	7	0.81	916.1	14	0.39	1441.0	7	0.52	1385.7	1	1.71	587.4
pc_s	21	0.22	701.5	12	0.83	620.4	14	0.31	788.6	9	0.60	703.3	10	1.47	472.9
pr_m	20	0.17	504.6	7	0.49	468.0	22	0.15	568.9	11	0.45	437.3	5	0.89	300.7
pr_s	26	0.10	273.4	14	0.43	269.5	20	0.18	276.1	16	0.39	251.7	8	0.89	235.3
	90	0.15	711.4	40	0.64	568.5	70	0.26	768.6	43	0.49	694.5	24	1.24	399.1

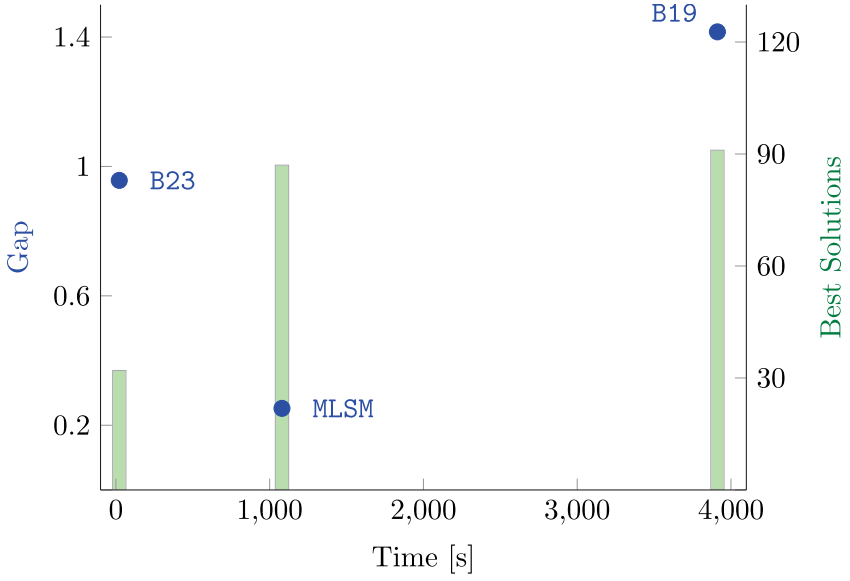


Fig. 2. Comparison of algorithm performance: Gap and Best solutions over Time.

is achieved with an average runtime of slightly more than 1000 s, showcasing a balance between computational effort and solution quality. B23 achieves a gap near to 1, which is significantly larger than that of MLSM. However, this comes with the advantage of an extremely fast runtime of few seconds. B19 exhibits the largest gap (larger than 1.4%), and it also requires the longest runtime at almost 4000 s.

When considering the number of best solutions, MLSM achieves 87 best solutions, which is close to the largest value of 91 obtained by B19. Meanwhile, B23 achieves only 32 best solutions, the lowest among the three algorithms.

The large gap observed for B19, despite the algorithm finds a large number of best solutions, is primarily due to the many instances where the solver reaches the time limit with a very large remaining gap, with the worst-case gap reaching 21.11%. In contrast, the maximum gaps for MLSM and B23 are much more contained, at 2.25% and 6.72%, respectively.

Table 2 provides a comparison of the performance of the three algorithms across the considered instance classes. It reports the average percentage gap, number of best solutions, and average computing time, for each algorithm and each instance class.

For MLSM, the Table also includes the number of restarts performed by the algorithm (column RS), and the execution time of the longest complete iteration (column Time*), from the initialization step until a local optimum is found. The last row of the table summarizes the averages (and the total, in case of column Best) across all instance classes.

Analyzing the results by instance class, we observe that MLSM consistently achieves lowest gap values compared to its competitors, with the best gap recorded at 0.16% for the pr_m class. Additionally, the algorithm finds a substantial number of best solutions, particularly with 24 best solutions in the pr_m class, indicating its efficiency in solving this type of instances. B23 exhibits a larger average gap, with a maximum of 1.29% in the pc_s class and a minimum of 0.6% in the pr_m class. For this class, it also exhibits the shortest average computing time of 5.4 s. B19 achieves the largest gap values. They are always larger than 1, reaching up to 1.94 in the pc_s class. Despite finding a number of best solutions that is slightly larger than that of MLSM for pc_m and pr_s classes, we can state that B19 is overall outperformed by MLSM, which always runs faster and finds, on average, better solutions. Finally, we examine columns RS and Time* for MLSM: ideally, with a number of threads equal to RS, the algorithm would achieve the runtime in Time*. This potential reduction suggests that, as parallel processing resources increase, the algorithm's total runtime could be significantly optimized, nearing B23 in terms of time efficiency.

Overall, MLSM emerges as a strong competitor, balancing solution quality and runtime effectively, and the results show that it outperforms B19, achieving better solution quality, on average, while requiring a significantly smaller computational time. In contrast, B23 prioritizes speed, making it suitable for scenarios where fast execution is essential.

6.2.1. Impact of vehicle capacity

Since MLSM is the only available algorithm solving both versions of the problem, we computed the percentage gap as $100\times(\bar{z}-z^*)/z^*$, where $\bar{z}$ represents the solution value found by MLSM for the *PCARP–IS*, and z^* is the best known solution value for the corresponding *PRPP–IS* instance.

Table 3 presents the average percentage gap between solving the problem with and without capacity constraints. The columns display the results for the four considered instance classes, while the last column reports the overall average percentage gap across all instance classes.

In addition to the average computed across all instances (row Gap), we also report the average gap computed only over the subset of

Table 2
Comparison of the performance obtained by different algorithms solving the $PRPP–IS$.

	MLSM					B23			B19		
	Gap	Best	Time	RS	Time*	Gap	Best	Time	Gap	Best	Time
pc_m	0.36	19	1689.3	32	484.6	1.09	6	44.8	1.23	20	4899.1
pc_s	0.23	23	1234.9	59	101.4	1.29	7	24.4	1.94	21	4490.2
pr_m	0.16	24	816.6	67	25.7	0.60	8	5.4	1.42	24	3311.7
pr_s	0.25	21	579.3	77	13.2	0.86	11	13.5	1.07	26	2946.5
	0.25	87	1080.0	59	156.2	0.96	32	22.0	1.42	91	3911.9

Table 3
Average percentage gaps between capacitated and uncapacitated cases.

	pc_m	pc_s	pr_m	pr_s	Overall
Gap	3.65	6.73	6.73	11.03	7.04
Gap*	(11) 7.93	(13) 10.86	(19) 9.54	(21) 13.75	(64) 10.91

Table 4
Comparison of the solution characteristics between different problem configurations.

	# Tours	Tour length	Saturation	Cost
$PRPP–IS$	6.26	38.8	1.34	842
$PCARP–IS$	9.40	24.8	0.77	888
$PCARP–IS(2d)$	15.78	17.2	0.89	1011
$PCARP–IS(f^+)$	16.82	24.6	0.75	1571

instances for which z^* is proven to be optimal for the corresponding $PRPP–IS$ instance (row Gap*) and we report in parenthesis the sizes of these subsets. This allows for a comparison with respect to a valid dual bound.

The results presented in the table indicate that introducing capacity constraints significantly impacts solution cost across all instance classes. The smallest gap is observed for the pc_m instances, with an average gap of 3.65%, suggesting that these instances are less sensitive to the addition of capacity constraints. Conversely, the pr_s instances exhibit the largest average gap of 11.03%. Overall, the average percentage gap across all classes is 7.04%, emphasizing a consistent increase of the solution cost when capacity constraints are included.

As expected, when the comparison is restricted to instances with proven optimal values for z^*, the gaps increase (e.g., from 7.04% to 10.91% overall). However, the increase remains moderate across all instance classes, further supporting the robustness of our method.

Table 4 presents the average values of key solution characteristics for different problem configurations, considering only the solutions obtained by MLSM. The configurations include the uncapacitated case ($PRPP–IS$), the capacitated case ($PCARP–IS$), and two variants considering modified instances: $PCARP–IS(2d)$ with doubled demands and $PCARP–IS(f^+)$ with non-zero frequencies f_T^l increased by 1, if $f_T^l + 1 \leq |T|$; both variants keep the same values of vehicles capacity of the original instances. It is important to note that, for all configurations, the number of tours is computed as the number of times a vehicle returns to the depot. This definition allows for the possibility of multiple tours in the same period, as the considered graphs are not complete. The tour length is computed as the number of links in the tour. The saturation is computed as the ratio of total demand to the product of vehicle capacity and the number of tours. Thus, for the capacitated cases, this value reflects how efficiently the available capacity is used, while for the $PRPP–IS$ the saturation metric is still computed for comparison purposes and can exceed 1 due to the absence of capacity limits. Column Cost reports the average value of the objective function.

The results show a significant impact of capacity constraints and input variations on the solution structure. Firstly, the number of tours increases by around 50% when capacity constraints are introduced, from an average of 6.26 in the uncapacitated case to 9.4 in the standard capacitated case. This increase is expected as capacity constraints force the algorithm to split longer routes into shorter ones. Indeed, the average tour length considerably decreases from 38.8 in the uncapacitated case to 24.8 in the capacitated case, as the limited vehicle capacity in the $PCARP–IS$ results in fewer links per tour compared to the $PRPP–IS$. Regarding saturation, for the uncapacitated case, the value of 1.34 indicates that, without capacity constraints, vehicles would be significantly overloaded. In the standard capacitated case, the saturation level of 0.77 suggests that vehicles are used well below their maximum capacity. The two additional configurations reveal interesting patterns. Both lead to a considerable increase in the number of tours used (15.78 and 16.82, respectively) compared to $PCARP–IS$. As expected, the tour length in $PCARP–IS(2d)$ considerably decreases (17.2), due to the larger values of each single demand, while in $PCARP–IS(f^+)$ it remains almost unchanged compared to $PCARP–IS$. On the other hand, configuration (f^+) results in a much higher average cost (1571 vs. 1011) than $PCARP–IS(2d)$. This outcome is explained by the fact that the demand increase affects vehicle capacity saturation differently in the two configurations: it remains roughly constant (0.75) in $PCARP–IS(f^+)$, but rises to 0.89 in $PCARP–IS(2d)$. This behavior likely depends on the more efficient capacity utilization achieved in some routes and periods under the $(2d)$ configuration.

7. Conclusion

In this study, we introduced a matheuristic approach to tackle the capacitated version of the periodic arc routing problem with irregular services, an extension of the $PCARP$ that has not been previously addressed in the literature. This extension brings the problem formulation closer to real-world applications, although further constraints, such as avoiding consecutive-day services, could be considered in future studies.

Our approach combines mathematical optimization techniques with local search strategies, designed to efficiently explore the solution space and provide high-quality solutions within a reasonable computational time.

Extensive computational experiments revealed that all the local search strategies we introduced play a crucial role in optimizing solution quality. Furthermore, the analysis indicates that the proposed approach is competitive even in contexts where capacity constraints are less critical, as proved by the comparison against state-of-the-art methods for the uncapacitated case. These results underscore the robustness of the approach.

Further research could investigate the potential for enhancements to the local search phases, such as the introduction of additional diversification mechanisms, as well as the possibility of extending the approach to other variants of capacitated arc routing problems. The promising outcomes of this study suggest that our matheuristic framework has significant potential for further development and adaptation to related routing challenges.

CRediT authorship contribution statement

Demetrio Laganà: Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Paolo Paronuzzi:** Writing – review & editing, Writing – original draft, Validation, Software, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

Acknowledgments

This research was partially funded by the University of Calabria, through project grant CIG8929018ABE. The authors thank Enrico Malaguti, Michele Monaci and Roberto Musmanno for their valuable comments and suggestions. The authors also thank Antonio Punzo for his contributions to the development of the software.

References

Archetti, C., & Speranza, M. (2014). A survey on matheuristics for routing problems. *EURO Journal on Computational Optimization*, *2*(4), 223–246.

Benavent, E., Corberán, A., Laganà, D., & Vocaturo, F. (2019). The periodic rural postman problem with irregular services on mixed graphs. *European Journal of Operational Research*, *276*(3), 826–839.

Benavent, E., Corberán, A., Laganà, D., & Vocaturo, F. (2023). A two-phase hybrid algorithm for the periodic rural postman problem with irregular services on mixed graphs. *European Journal of Operational Research*, *307*(1), 64–81.

Chen, Y., & Hao, J. (2018). Two phased hybrid local search for the periodic capacitated arc routing problem. *European Journal of Operational Research*, *264*(1), 55–65.

Chu, F., Labadi, N., & Prins, C. (2004). The periodic capacitated arc routing problem linear programming model, metaheuristic and lower bounds. *Journal of Systems Science and Systems Engineering*, *13*, 423–435.

Chu, F., Labadi, N., & Prins, C. (2005). Heuristics for the periodic capacitated arc routing problem. *Journal of Intelligent Manufacturing*, *16*, 243–251.

Chu, F., Labadi, N., & Prins, C. (2006). A scatter search for the periodic capacitated arc routing problem. *European Journal of Operational Research*, *169*(2), 586–605.

Corberán, A., Eglese, R., Hasle, G., Plana, I., & Sanchis, J. (2021). Arc routing problems: A review of the past, present, and future. *Networks*, *77*(1), 88–115.

De Maio, A., Laganà, D., Musmanno, R., & Vocaturo, F. (2021). Arc routing under uncertainty: Introduction and literature review. *Computers & Operations Research*, *135*, Article 105442.

Ghiani, G., Musmanno, R., Paletta, G., & Triki, C. (2005). A heuristic for the periodic rural postman problem. *Computers & Operations Research*, *32*(2), 219–228.

Huang, S., & Lin, T. (2014). Using ant colony optimization to solve periodic arc routing problem with refill points. *Journal of Industrial and Production Engineering*, *31*(7), 441–451.

Kanso, B. (2020). Hybrid ant colony algorithm for the multi-depot periodic open capacitated arc routing problem. *Int. J. Artif. Intell. Appl*, *11*(1), 53–64.

Khosravi, P., Alinaghian, M., Sajadi, S., & Babaee Tirkolaee, E. (2015). The periodic capacitated arc routing problem with mobile disposal sites specified for waste collection. *Journal of Applied Research on Industrial Engineering*, *2*(1), 64–76.

Kiilerich, L., & Wohlk, S. (2018). New large-scale data instances for CARP and new variations of CARP. *INFOR. Information Systems and Operational Research*, *56*(1), 1–32.

Lacomme, P., Prins, C., & Ramdane-Cherif, W. (2005). Evolutionary algorithms for periodic arc routing problems. *European Journal of Operational Research*, *165*(2), 535–553.

Laganà, D., Malaguti, E., Monaci, M., Musmanno, R., & Paronuzzi, P. (2024). An iterated local search matheuristic approach for the multi-vehicle inventory routing problem. *Computers & Operations Research*, *169*, Article 106717.

Marzolf, F., Trepanier, M., & Langevin, A. (2006). Road network monitoring: algorithms and a case study. *Computers & Operations Research*, *33*(12), 3494–3507.

Mei, Y., Tang, K., & Yao, X. (2011). A memetic algorithm for periodic capacitated arc routing problem. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *41*(6), 1654–1667.

Monroy, I., Amaya, C., & Langevin, A. (2013). The periodic capacitated arc routing problem with irregular services. *Discrete Applied Mathematics*, *161*(4–5), 691–701.

Mourão, M., & Pinto, L. (2017). An updated annotated bibliography on arc routing problems. *Networks*, *70*(3), 144–194.

Riquelme-Rodriguez, J., Gamache, M., & Langevin, A. (2014). Periodic capacitated arc-routing problem with inventory constraints. *Journal of the Operational Research Society*, *65*(12), 1840–1852.

Riquelme-Rodriguez, J., Langevin, A., & Gamache, M. (2014). Adaptive large neighborhood search for the periodic capacitated arc routing problem with inventory constraints. *Networks*, *64*(2), 125–139.

Triki, C. (2017). Solving the periodic edge routing problem in the municipal waste collection. *Asia-Pacific Journal of Operational Research*, *34*(03), Article 1740015.

Vadseth, S., Andersson, H., & Stahlhane, M. (2021). An iterative matheuristic for the inventory routing problem. *Computers & Operations Research*, *131*, Article 105262.

Vidal, T. (2017). Node, edge, arc routing and turn penalties: Multiple problems—one neighborhood extension. *Operations Research*, *65*(4), 992–1010.