Expert Systems With Applications 230 (2023) 120513

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Heuristics for the Profitable Close-Enough Arc Routing Problem

Miguel Reula [a,*], Rafael Martí [b]

[a] University of Valencia, Department of Mathematics Teaching, Avda. Tarongers 4, 46022 Valencia, Spain
[b] University of Valencia, Department of Statistics and Operations Research, Doctor Moliner 50, 46100 Burjassot, Spain

ARTICLE INFO

Keywords:
Arc routing
Close-enough
Profits
Metaheuristic
Logistics

ABSTRACT

In this paper we solve a practical variant of an arc routing problem. We target the close enough model in which clients can be served from relatively close arcs. This variant, known as the profitable close-enough arc routing problem, models real situations, such as inventory management or automated meter reading. We propose a heuristic based on the variable neighborhood search methodology to maximize the sum of profits of the clients served (penalized with the distance traveled). Our method incorporates efficient search strategies to speed up the optimization process, as required in practical applications. We present extensive experimentation over a benchmark of previously reported instances. Specifically, we first set the key search parameters of our method, and then compare it with the state-of-the-art heuristics for this problem. Our heuristic outperforms the previous algorithms published for this problem, as confirmed by the statistical analysis, which permits to draw significant conclusions.

1. Introduction

Arc Routing Problems (ARPs) typically deal with traversing a set of connecting edges (in the undirected case) or connecting arcs (in the directed one) at the minimum possible cost. We may find many types of objectives and constraints modeling different applications, from garbage collection to meter reading, and constitute nowadays a well-established field in combinatorial optimization.

As Corberán and Laporte (2014) described in their reference book in vehicle routing, "*the study of modern arc routing truly started in 1960 with the first publication on the Chinese Postman Problem*". Over the years, arc routing has evolved into a relevant research area, which might include real-life characteristics such as multiple criteria, soft constraints, or stochastic travel times, just to name a few.

We can find different variants of ARPs according to the characteristics of the network (*directed* or *undirected*), vehicles (*homogeneous/heterogeneous fleet*), depot (*single, multiple or mobile*), demand, and objective. These two later elements contain many subcategories, reflecting the interest of researchers and practitioners on these topics. In particular, among the most important ones in the literature are the inclusion of *time windows*, *time-dependent service cost*, or *multiple objectives*, including *service* and *labor cost*, *length*, and *duration* of routes. In addition, with the emergence of new technologies in the automotive industry, many other variants have emerged in recent years. For example, many of these variants and including new particularities of the Electric Vehicles are considered in Lu, Chen, Hao, and He (2020) and in Fateme Attar, Mohammadi, Reza Pasandideh, and Naderi (2022), or the utilization of drones in a Routing Problems in Kuo, Lu, Lai, and Mara (2022)

In this paper, we target an interesting variant of the classic arc routing problem. We consider problems with profits, which deal with situations where clients have an associated profit that is collected when servicing them. They basically consist of designing one or more routes providing service to some chosen clients, in such a way that a certain objective is optimized. This objective is usually defined in terms of the cost (distance or travel time) or/and the profit. A routing problem with profits is called *profitable* when its objective is to maximize the difference between the collected profit and the traveling cost (Feillet, Dejax, & Gendreau, 2005).

In arc routing problems, the service is typically performed while the vehicle traverses an arc of the graph. However, in the last few years, advances in new technologies, such as radio-frequency identification (RFID), permit to perform some tasks remotely; and therefore, the concept of providing a service needs to be redefined. In the case of ARP models, we need to introduce the concept that while a vehicle, traverses an arc, it may provide service not only to this specific arc, but also to other elements not present in this arc. For example, RFID makes it possible to remotely collect consumption data, such as gas, electricity, or water from their meters without being in their specific location (house or facility). The meter sends a signal with the consumption information, which is read by the receiver if it is close-enough (i.e., less than a certain distance threshold given by technological specifications). Thus, the operator only has to enter the meter's coverage area to perform the

* Corresponding author.
E-mail addresses: miguel.reula@uv.es (M. Reula), rafael.marti@uv.es (R. Martí).

https://doi.org/10.1016/j.eswa.2023.120513
Received 19 May 2022; Received in revised form 20 February 2023; Accepted 17 May 2023
Available online 23 May 2023

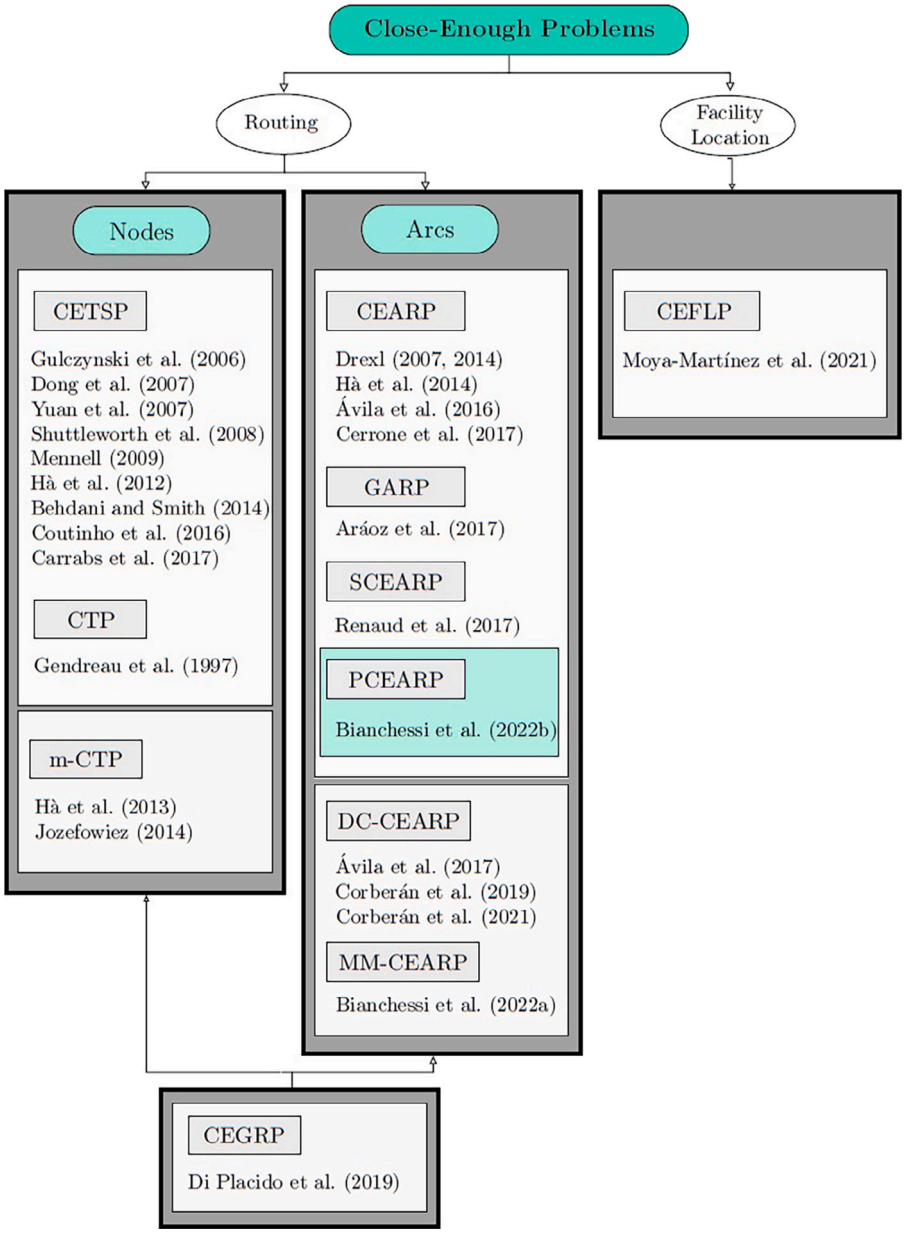

Fig. 1. Close-enough problems.
ROUTING: - Nodes - *CETSP:* (Behdani & Smith, 2014; Carrabs, Cerrone, Cerulli, & Gaudioso, 2017; Coutinho, Subramanian, do Nascimento, & Pessoa, 2016; Dong, Yang, & Chen, 2007; Gulczynski et al., 2006; Hà, Bostel, Langevin, & Rousseau, 2012; Mennell, 2009; Shuttleworth, Golden, Smith, & Wasil, 2008; Yuan, Orlowska, & Sadiq, 2007) ; *CTP:* (Gendreau, Laporte, & Semet, 1997) ; *m-CTP:* (Hà, Bostel, Langevin, & Rousseau, 2013; Jozefowiez, 2014) **- Arcs -** *CEARP:* (Ávila, Corberán, Plana, & Sanchis, 2016; Cerrone, Cerulli, Golden, & Pentangelo, 2017; Drexl, 2007, 2014; Hà et al., 2014) ; *GARP:* (Aráoz, Fernández, & Franquesa, 2017) ; *SCEARP:* (Renaud, Absi, & Feillet, 2017) ; *PCEARP:* (Bianchessi, Corberán, Plana, Reula, & Sanchis, 2022b) ; *DC-CEARP:* (Ávila, Corberán, Plana, & Sanchis, 2017; Corberán, Plana, Reula, & Sanchis, 2019, 2021) ; *MM-CEARP:* (Bianchessi, Corberán, Plana, Reula, & Sanchis, 2022a) **Arcs and Nodes.** *CEGRP:* (Di Placido, Russo, Capobianco, & Cerrone, 2019) **FACILITY LOCATION.** *CEFLP:* (Moya-Martínez, Landete, & Monge, 2021).

service, without the need to physically visit all the clients (Gulczynski, Heath, & Price, 2006). Applications in the context of meter reading over a street network can be found in Hà, Bostel, Langevin, and Rousseau (2014), in which clients (meters in these applications) are not necessarily nodes of the network, and are served when a close-enough street is traversed.

The problems described above can be grouped under the generic term "close-enough models". As a matter of fact, we can find this type of situation, in which the service is provided remotely, not only in ARPs, but also in node routing problems, and even in location problems. Fig. 1 shows a classification of close-enough models in these three families: node-routing problems (Nodes), ARPs (Arcs), and facility location. In the first family we can find the Close-Enough Traveling salesman Problem (CETSP), and the Covering Tour Problem (CTP), which is an undirected version of the CETSP. As can be seen in this figure, we can find many ARPs variants with close-enough models. Specifically, we have identified the following six problems, where the first four of them consider 1 vehicle, and the last two are built based on several vehicles:

- CEARP — Close-Enough Arc Routing Problem
- GARP — Generalized Arc Routing Problem
- SCEARP — Stochastic Close-Enough Arc Routing Problem
- PCEARP — Profitable Close-Enough Arc Routing Problem
- DC-CEARP — Distance-Constrained Close-Enough ARP
- MM-CEARP — Min–Max Close-Enough Arc Routing Problem

The third category displayed in Fig. 1 is devoted to close-enough facility location models, and we have only identified one contribution in this family. This figure also includes the associated references for a complete picture of the area.

In this paper we consider the Profitable Close Enough Arc Routing Problem (PCEARP) in which the required arcs are grouped into families associated with clients, and to service them it is enough to traverse one of their associated arcs. Note that in this problem not all clients have to be served, but the model has to select those with a relatively large profit. It must be noted that routing problems traditionally consider that clients are located either in the nodes or the arcs of the underlying graph. The PCEARP, proposed in Bianchessi et al. (2022b), introduces more flexibility to model the new requirements associated with providing a remote service; however, as will be shown, it translates into more complexity in terms of its resolution. In particular, clients are not located in arcs or nodes, but can be served from certain arcs. This is why we consider families of arcs associated to clients, indicating that traversing any arc in the family provides the service to that client. As the PCEARP generalizes both the CEARP (when the profits associated with all the clients are very large) and the PRPP (when each required arc defines a client), it is an NP-hard problem covering different practical applications.

A PCEARP real-life application is found in inventory management. Large companies competing in the global market usually work with many products and references. In fact, in these companies, inventory management is becoming a complicated and costly task, making very difficult to keep track the number of articles of each reference in each warehouse, thus causing the so-called shrinkage (loss of items in retail stores). This is not only the case of private companies, but also of the public institutions. According to Fadel Adib, Assistant Professor of Media Arts and Sciences, "Between 2003 and 2011, the U.S. Army lost track of $5.8 billion of supplies among its warehouses. Similarly, the U.S. National Retail Federation reported that shrinkage averaged around $45.2 billion in 2016".

The development of RFID tags revolutionized supply chain management a few years ago (Chen, Wang, Xie, & Qi, 2014) by allowing warehouse managers to log inventory much more efficiently, keeping track of the real stock when products enter and leave the warehouse. However, since part of the process was still manual, it resulted in time-consuming operations. This means that mismatches were often overlooked until they were exposed by client requests. In order to make this task easier, MIT researchers have developed a device that allows aerial drones to read RFID tags from tens of meters away and identify the tags location with accuracy (with an average error lower than 20 centimeters). In this way, to check the stock of each product, the drone only needs to be close enough to the product location (Duric, Jovanovic, & Sibalija, 2018). If we consider the products as clients and assign a profit to those products for which we are interested in knowing the stock, we would be referring to a PCEARP in which the drones move through the aisles of the warehouse. In this way, we can state that the model considered in this paper is applied to solve an important problem in supply chain, which constitutes a critical pillar in modern economy.

2. Problem definition and formulation

Given a strongly connected digraph $G = (V, A)$, where V is the set of vertices, and A the set of arcs, we consider $d_{ij} \geq 0$ as the distance (or length) of arc $(i, j) \in A$. Without loss of generality, vertex 1 denotes the depot. As described in the introduction, in this problem we consider an additional set of clients $\mathbb{H}$, which is not necessarily located in the vertices or arcs of the graph. As a matter of fact, each client $c \in \mathbb{H}$ receives service if any of its associated arcs is traversed. Let $H_c \subseteq A$ be this set of associated arcs to c. We also define the profit $p_c \geq 0$ of client c. This profit is collected (only once) if the client is serviced. Note that the subsets H_c are not necessarily disjoints.

The Profitable Close-Enough Arc Routing Problem (PCEARP) consists in finding a route on G that starts and ends at the depot. This route, usually called tour or "closed walk", constitutes a solution of the PCEARP. Its objective is to maximize the difference between the sum of the profits collected in the route, and its total length.

In arc routing problems, the concept of required arcs is usually straightforward and implies that a feasible solution has to traverse it. As mentioned above, we use here a broad scope of this concept, and we called required to those arcs that provide service to one or more clients. In line with this, we define the set of required arcs $A_R = \cup_{c \in \mathbb{H}} H_c$. Note that a feasible solution may or may not traverse these arcs; but if it traverses them, the associated profits are collected. Symmetrically, non-required arcs, employed to connect the required arcs in the tour, are in the set $A_{NR} = A \setminus A_R$.

Given a set of vertices $S \subset V$, we define its out-cutset $\delta^+(S)$ as the set of arcs from S to $V \setminus S$. In mathematical terms: $\delta^+(S) = \{(i, j) \in A : i \in S,\ j \in V \setminus S\}$. Similarly, we define the in-cutset of S, $\delta^-(S)$, as the set of arcs from $V \setminus S$ to S: $\delta^-(S) = \{(i, j) \in A : i \in V \setminus S,\ j \in S\}$.

The PCEARP can be formulated in mathematical terms by representing a tour on G with an integer vector $x = (x_{ij}) \in \mathbb{Z}^{|A|}$ where, for each arc $(i, j) \in A$, x_{ij} is equal to the number of times that it is traversed. A solution to the PCEARP is thus represented by (x, z), where x is the integer vector associated to the tour, and $z = (z_c) \in \mathbb{Z}^{|\mathbb{H}|}$ is the binary vector associated with the clients, in which:

$$z_c = \begin{cases} 1, & \text{if the client } c \text{ is serviced,} \\ 0, & \text{otherwise.} \end{cases}$$

The PCEARP is modeled in Bianchessi et al. (2022b) as an integer linear program based on the (x, z) variables. In particular, the objective function and constraints are computed as follows:

$$\text{Maximize} \sum_{c \in \mathbb{H}} p_c z_c - \sum_{(i,j) \in A} d_{ij} x_{ij}$$

s.t.:

$$x(\delta^+(i)) = x(\delta^-(i)) \qquad \forall\, i \in V \tag{1}$$

$$x(\delta^+(S)) \geq z_c - x(H_c \cap A(V \setminus S)) \qquad \forall S \subseteq V \setminus \{1\}, \forall c \in \mathbb{H} \tag{2}$$

$$x(H_c) \geq z_c \qquad \forall c \in \mathbb{H} \tag{3}$$

$$x_{ij} \in \mathbb{Z}^+ \qquad \forall (i, j) \in A \tag{4}$$

$$z_c \in \{0, 1\} \qquad \forall c \in \mathbb{H}, \tag{5}$$

In this formulation, for a set of arcs $F \subset A$, $x(F) = \sum_{(i,j) \in F} x_{ij}$. Constraints (1) are the typical symmetry conditions of arc routing problems to guarantee that vertices are balanced (i.e., their in-degree match their out-degree). Constraints (2) ensure that the routes are connected either if clients are serviced or not. Inequalities (3) imply that, if a client c is served, at least an arc in H_c is traversed. Finally, (4)–(5) include variables definition.

As described in Bianchessi et al. (2022b), this is a partial formulation of the PCEARP in the sense that it may produce solutions disconnected from the depot (with the so-called subtours satisfying (1)–(5)). However, the authors proved that when maximizing the objective function, the model is indirectly penalizing the existence of subtours, and then its optimal solution is always a connected tour (i.e, a closed route without subtours).

It is important to note that we are facing here a typical situation when modeling complex combinatorial problems; although we know an explicit formulation of the problem, we cannot use it directly on a solver. This is because the number of constraints is too large to be included (it actually reflects the exponential growth of the model in terms of the size of the data). Specifically, Constraints (2) are defined in terms of the number of vertices of any subset S of V, which translates into $2^{|V|}$, which is too large even for medium size instances. This is why (Bianchessi et al., 2022b) proposed a branch and cut algorithm to efficiently solve these instances. We will use their results when assessing the performance of our heuristic.

In the next section, we define the PCEARP, and formulate it in mathematical terms. The main contributions of this paper are: (a) to review previous heuristics for the PCEARP (Section 3), (b) to propose a new heuristic based on the Variable Neighborhood Search for the PCEARP (Section 4), which includes new search strategies for an efficient exploration of the solutions space, and (c) to perform an empirical comparison on previously reported instances to assess the efficiency of the proposed heuristic with respect to both previous heuristics and optimal solutions. The paper finishes with the associated conclusions.

3. Previous heuristics

The PCEARP was recently introduced in the context of a branch-and-price algorithm for a related problem, the min–max close-enough arc routing problem. This variant based on profits, was identified by Bianchessi et al. (2022a) in the pricing problem that has to be solved as a part of their exact method. The authors proposed a GRASP heuristic (Martí, Pardalos, & Resende, 2018) to speed up the column generation applied in the nodes of the search tree in the branch-and-price algorithm. It must be noted, that their GRASP (GRASP_BP) solves the minimization problem involved in the pricing part, so we adapted it here to target the PCEARP, which is equivalent but modeled as a maximization problem. Note also that it exhibits short computational times, as required by an algorithm that is repeatedly applied. A short description of the method follows.

GRASP_BP starts by computing, for each required arc $a \in A_R$, the profit $p(a)$ of the clients served by the tour going from the depot to the arc (i.e., the sum of its individual profits), and coming back to the depot, minus its total distance. In other words, it computes the objective function value of this tour under construction. The method builds a set $\mathcal{A}_{ini}$ with the arcs with a relatively large $p(a)$ value to initiate a route. In particular, this set contains the $min\{50, \frac{|A_R|}{2}\}$ arcs with the largest p-value. GRASP_BP is applied a number of iterations equal to the cardinality of $\mathcal{A}_{ini}$, and at each iteration, a route is initialized by selecting an arc in $\mathcal{A}_{ini}$ (without replacement).

After the initialization, we have a partial route r with an objective function value $f(r)$ that computes the sum of the profits of clients served, minus the distance traveled. A GRASP_BP iteration tries to first complete it by adding more arcs and serving more clients, and then to improve the resulting route by replacing some arcs in it. Let $A_R(r)$ be the set of required arcs in route r. Note that r may contain non-required arcs (i.e., arcs that do not serve any client), that are necessary to connect the required arcs. These non required arcs are typically computed as the shortest paths joining the required arcs.

Construction phase At each step of the GRASP_BP construction, the method computes, for each required arc a not in the route ($a \in A_R \setminus A_R(r)$), its value ψ_a, as the change in the objective function if a is inserted in route r (in the best possible position). Note that, if a is added to r, to compute the objective function value of the new route r', we have to update in $f(r)$ both the sum of profits of the route, and the sum of distances. It is expected that new clients are now served and therefore the sum of profits will probably increase, but at a price that comes from the increment in the total distance to visit them (to traverse the arcs serving them). In mathematical terms, $f(r') = f(r) + \psi_a$.

As it is customary in GRASP, a restricted candidate list (RCL) is built with the arcs with a good evaluation, and the method randomly selects one of them. Let ψ_{max} and ψ_{min} be the maximum and minimum respectively of the ψ_a values for all the required arcs not in the current route. Then,

$$RCL = \{a \in A_R \setminus A_R(r) : \psi_a \geq \alpha(\psi_{max} - \psi_{min}) + \psi_{min}\},$$

where the parameter α balances the greediness and the randomization in the selection from RCL. Specifically, if $\alpha = 0$ the method is completely random since all the required arcs (not in the route) are in RCL. On the other hand, if $\alpha = 1$ the method is entirely greedy since only the best arcs (those with maximum ψ_a) are in RCL. The authors applied this method with $\alpha = 0.9$ to reflect the strategy of considering a small randomization component.

The selected arc is added to the route, and its value updated. A GRASP_BP construction performs iterations, adding arcs to the partial route, until no further improvement is possible, and at that point, the resulting route is submitted to the improvement method.

Improvement phase The method applies a post-processing procedure to improve each constructed solution by exploring its neighborhood. In particular, it consists of a destroy-and-repair method (Corberán et al., 2019) that first removes some arcs from the route, and then add new arcs to improve the resulting solution. It is indeed more complex than the standard local search usually applied in GRASP, in an effort to obtain improved outcomes.

In the Destroy step of the algorithm, a small number of required arcs, η, are randomly selected and removed from the route. The authors proposed to alternate η between 1 and 3 in consecutive iterations. Then, in the Repair step, the best required arc $a \in A_R \setminus A_R(r)$, is included in it, if it improves its value. Further required arcs are added to the solution as long as they improve it. An improvement iteration finishes when no further improvement is possible adding extra arcs. Then, it resorts to the destroy step. The improvement phase alternates between both steps, destroy and repair, until a cutoff point is reached. The authors recommended to establish this limit as a function of the problem size. Specifically, they set it as $m_{LS} = \min\{\frac{L}{2}, \frac{|A_R(r)|}{2}\}$, where L is the number of clients and $|A_R(r)|$ the number of required arcs in the route.

As mentioned above, this GRASP_BP was originally proposed in the context of a branch-and-price method, and was applied for a very short computing time (2 s), in line with its role to operate as a subroutine. Since we are considering it now as solver itself, we will apply it in our computational testing for a longer running time (60 s), similar to the other heuristics considered.

Algorithm 1 GRASP_BP Algorithm

```
1: function GRASP(t_l, α)
2:     it ← 1
3:     it_LS ← 1
4:     A_ini ← GenerateSet ()
5:     while t < t_l & it ≤ |A_ini| do
6:         a ← A_ini(i)
7:         r ← ConstructionPhase (a, α)
8:         η ← 1
9:         while η ≤ 3 do
10:            while it_LS < m_LS do
11:                r' ← ImprovementPhase (r, η)
12:                if f(r') > f(r) then
13:                    f(r) ← f(r')
14:                    it_LS ← 1
15:                else
16:                    it_LS ← it_LS + 1
17:            η ← η + 1
18:        it ← it + 1
19: return r
```

Another heuristic procedure for the PCEARP has been proposed in Bianchessi et al. (2022b), where a detailed study of the problem is conducted. In that paper, the authors provide an iterative algorithm that combines a constructive heuristic and a local search heuristic. This method can be considered an extension of the previous heuristic, in which they improve the construction strategies to obtain a high-quality bound in their branch-and-cut algorithm. As a matter of fact, this improved heuristic permits to identify many optimal solutions to the exact procedure in relatively short running times for small instances.

Similarly to the method described above, in each iteration the constructive algorithm is first applied to construct a solution. Then, each solution (route) can be improved by applying the local search algorithm. After applying both phases, construction and improvement, iteratively, the output of the method (final solution) is the route associated with the best profit among those calculated. The authors call it the iterative algorithm, but considering that it is a greedy randomized adaptive search procedure, we will refer it as GRASP_IT.

It is very interesting to note that both methods apply a greedy randomized strategy in the construction process. However, in the case of the previous GRASP_BP the method first evaluates all the potential candidates, creating the so-called Restricted Candidate List with the good elements, from which one of them is randomly selected. On the other hand, the GRASP_IT described in what follows alternates the order in which these two strategies, greedy and random, are applied. In particular, it first selects some elements completely at random (performing a random sampling in the candidate list), and then, after evaluating them, select the best one.

Algorithm 2 GRASP_IT Algorithm

```
1: function GRASP(t_l, it_max, m_C, m_LS)
2:     it ← 1
3:     A_ini ← GenerateSet ()
4:     A ← A_ini
5:     while t < t_l & it ≤ it_max do
6:         for a ← 1 to |A| do
7:             r ← ConstructionPhase (a, m_C)
8:             it_LS ← 1
9:             while it_LS < m_LS do
10:                r' ← ImprovementPhase (r, η)
11:                if f(r') > f(r) then
12:                    f(r) ← f(r')
13:                    it_LS ← 1
14:                else
15:                    it_LS ← it_LS + 1
16:            it ← it + 1
17:        A_ini2 ← GenerateSet ()
18:        A ← A_ini2;
19: return r
```

To introduce new strategies based on clients, we will use additional notation to the one we have presented above. Given a route r, in addition to the ordered set of required arcs $A_R(r)$, the set of clients that are served by this route is specified with $C(r)$. Given $a \in A_R$, the set C_a will be the set of clients served when a is traversed. Each iteration starts by computing the list of required arcs $\mathcal{A}_{ini}$ in a similar way than the GRASP_BP described above. There are however, some improvements over the previous algorithm. Specifically, for each client $c \in \mathbb{H}$, the arc $a = (i, j) \in H_c$ with minimum distance from the depot is added to $\mathcal{A}_{ini}$. In other words, instead of scanning the list of arcs in search for a good element (as the GRASP_BP does), the method explores the clients and from them, examines the arcs.

Once a route is initialized with an arc $a_i \in \mathcal{A}_{ini}$, then all the clients in C_{a_i} are included in $C(r)$. It is next checked if the required arcs traversed from the depot to a_i and from a_i to the depot (if any) can service other unassigned clients. If that is the case, both the arcs and the clients are inserted in $A_R(r)$ and $C(r)$, respectively. In line with the objective function, the method looks in each route r for a subset of clients/arcs with high profits (in order to add them to r). This is done by first sampling clients at random, and then selecting the best of them. In particular, $\lfloor(|\mathbb{H}| - |C(r)|)/4\rfloor$ clients not included in $C(r)$ are randomly selected, and the client $\bar{c}$ and the arc $\bar{a}$ producing the best change in the objective value are selected and $A_R(r)$ and $C(r)$.

Each constructed solution is improved with the local search heuristic based on the destroy-and-repair mechanism proposed by Bianchessi et al. (2022a) in GRASP. Once all the routes associated with the arcs in $\mathcal{A}_{ini}$ have been studied, we can perform further iterations if the stopping criterion is not met, by applying a different arc selection rule to initialize routes. Specifically, for each arc $a \in \mathcal{A}_{ini}$, a client c is randomly selected from the $\max\{10, |\mathbb{H}|/50\}$ clients closest to a in $\mathbb{H} \setminus C_a$. The arc $\bar{a} \in H_c$, which produces the best value of the objective function when added to the route initialized with a, is selected to define, together with a, the new initial route r. In this way, it is built a new list $\mathcal{A}_{ini2}$, where each component is a pair of required arcs, to initialize the routes.

The GRASP_IT (Algorithm 2) is executed for a maximum number of $it_{max} = 100 \cdot |\mathbb{H}|$ iterations and $t_l = 60$ s. The maximum number of iterations without improvement has been set to $m_C = \min\{|\mathbb{H}|/4, 25\}$ for the constructive heuristic and to $m_{LS} = \min\{|\mathbb{H}|/2, 50\}$ for the local search heuristic. The number of consecutive arcs removed while applying the destroy-and-repair mechanism has been set to $\eta = \min\{|A'_R|, 5\}$.

4. A variable neighborhood search heuristic

In this paper we consider the Variable Neighborhood Search (VNS); a metaheuristic based on exploring several neighborhoods during the solution search to overcome the limitations of local optimality. VNS was introduced by Mladenović and Hansen (1997) and, since then, this methodology has continuously evolved resulting in several extensions and variants. See Hansen, Mladenović, Todosijević, and Hanafi (2016) for a reference text.

As stated by Mladenovic and Hansen, VNS is based on three principles:

- A local minimum with respect to one neighborhood is not necessarily so with another.
- A global minimum is a local minimum with respect to all possible neighborhood structures.
- For many problems local minima with respect to one or several neighborhoods are relatively close to each other.

The second principle is true for all the optimization problems. Principles 1 and 3 may or may not hold but they are usually true in many combinatorial optimization problems.

We propose a MultiStart-VNS (MS-VNS) to solve the Profitable Close-Enough Arc Routing Problem (PCEARP). VNS combines deterministic and random changes of neighborhood structures in order to find a balance between diversification and intensification. Incorporating VNS in a multi-start scheme allows us to perform several independent iterations to further improve the diversification ability of the procedure. Algorithm 4 shows the associated pseudo-code. We first start with the definition of the four neighborhoods in our method, and then present the description of the algorithm.

4.1. Neighborhoods definition

Swaps are the primary mechanism in our neighborhood definition. In the neighborhood N_{1-1}, as it is shown in Fig. 2, we exchange a required arc in the solution with another required arc not present in the solution. We apply the so-called first strategy, in which we perform the first improvement move and skip the examination of the rest of the neighborhood.

We consider the set of required arcs $A_R(r)$ in route r as an ordered set, reflecting the order of the route that starting from the depot traverses each arc in the set. It is assumed that the route follows the shortest path from the final vertex of any required arc in the sequence to the initial vertex of the next one. Starting with the first arc in $A_R(r)$, a_1, our exploration of N_{1-1} tries to remove it from the route, and to replace it with another required arc. In particular, the method randomly selects a customer $c \in \mathbb{H}$ not served by r, and explores all the arcs providing service to c (i.e, the arcs $b \in H_c$). The exchange of a_1 with b, evaluates the insertion of b in the best possible position of route

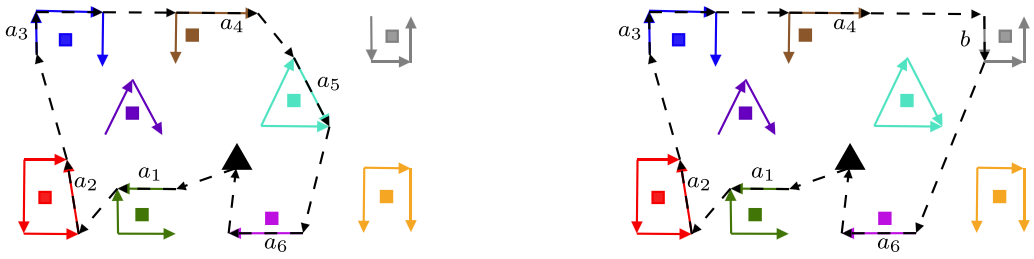

Fig. 2. Neighborhood change - N_{1-1}.

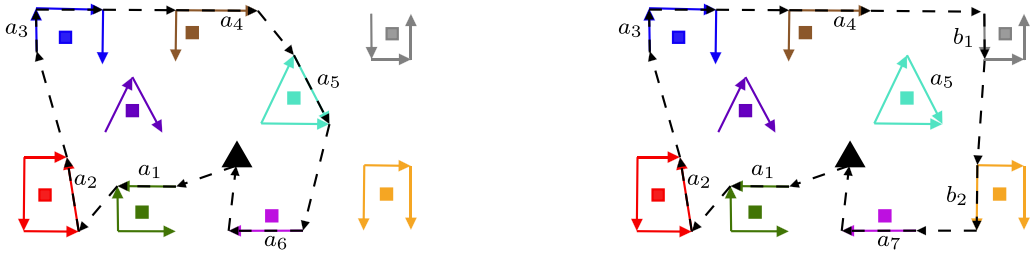

Fig. 3. Neighborhood change - N_{1-2}.

r. Note, that this evaluation also computes the shortest paths necessary to reconnect route r. The method considers all the arcs $b \in H_c$ and if any of them results in a better route, the best move is performed and the route is updated with both required and non-required arcs. Fig. 2 illustrates this move. It is worth mentioning that when including the shortest path to reconnect the route, it may happen that a non-served client is served now, and therefore, the evaluation has to accurately reflect it. If all the arcs $b \in H_c$ has been explored and no improving exchange has been found, the method considers the next required arc a_2 in $A_R(r)$ and applies the same process again with another randomly selected client. The neighborhood exploration continues, as long as improvement swaps are performed, scanning $A_R(r)$ in order consecutively; and it stops after a full examination of $A_R(r)$ without improvement.

In the neighborhood N_{2-1}, we exchange two required arcs in route r with one required arc not present in the solution. Specifically, we start by considering to swap a_1 and a_2 in $A_R(r)$ with an arc $b \in H_c$ for a randomly selected customer c not served in the current route r. The method follows the same procedure described above, evaluating the insertion of the arcs $b \in H_c$ in the best possible position, and finally performing the best associated move, if it improves the solution. After that, it considers the next pair of arcs in the ordered set $A_R(r)$, namely a_2 and a_3 and proceeds in the same fashion. The local search based on this neighborhood performs swaps of 2 required arcs with 1 required arc not in the solution as long as any improvement is found.

From the description of the two neighborhoods above, N_{1-1} and N_{2-1}, we can easily understand how operate our two other neighborhoods, N_{1-2} and N_{2-2}. In particular, in N_{1-2} (Fig. 3) we consider swaps of 1 required arc with 2 required arcs not in the solution, while in N_{2-2} we exchange two arcs with two arcs. Search strategies are implemented in line with the methods above, scanning the ordered set $A_R(r)$ in search for an improving move, and selecting at random two non-served clients to insert in the solution one required arc associated with each of them.

A criticism to the explorations proposed above is that they rely on a random selection of the client not served in the current route to identify the required arc or arcs to be inserted. We have also considered an alternative selection mechanism based on a deterministic criterion. In particular, for each required arc in the route, we compute its closets required arcs not present in the route. In this way, when we evaluate a swap, its closest arc is the first candidate to try to replace it. This can be easily generalize to the four neighborhoods described, since instead of considering the closest arc, we can consider the closest and second closest arcs.

We cannot establish before hand which of both alternative designs is better. The **random client** selection, clearly favors the diversification in the exploration, but at a cost of considering many alternatives, with the high computational cost involved. On the other hand, the **deterministic arc** selection of the closest required arc reduces the exploration of alternatives, but clearly performs a narrow exploration losing diversification power. We therefore will test both designs in our computational experience.

4.2. VND algorithm

The VND algorithm is based on a systematic change of the neighborhood structure. In particular, when the local search does not obtain an improved solution in the neighborhood of the current one, then, it resorts to another neighborhood. To do it in a systematic way, we first order the four neighborhoods described above for our routing problem. As it is customary in this methodology, we order them from the simplest to the most complex one, in order to reduce the computational effort of exploring them. Specifically, we consider the following order: N_{1-1}, N_{1-2}, N_{2-1}, and N_{2-2}. Additionally, in each of these neighborhoods we can perform a **random client** or a **deterministic arc** selection, resulting in $l_{max} = 8$ neighborhoods. Algorithm 3 shows the pseudo-code of the method.

Algorithm 3 Variable Neighborhood Descendent

```
1: function VND(r_0, l_max)
2:     r ← r_0;
3:     l ← 1;
4:     while l < l_max do
5:         r' ← FirstInNeighborhood (r, l)
6:         if f(r') > f(r) then
7:             r ← r'
8:             l ← 1;
9:         else
10:            l ← l + 1;
11:    return r
```

The VND procedure in Algorithm 3 starts with a solution r_0, that is stored as the current best, r, in step 2. In the main loop in steps 4–10 is where the method performs multiple iterations. Starting with the first neighborhood in the list, $l = 1$, the method applies the first improvement strategy and returns in step 5 solution r' in the neighborhood. If it improves upon the best so far, r, then in step 7 it is updated, $r = r'$, and in step 8 we resort to the first neighborhood in the list ($l = 1$) to continue the exploration; otherwise, we change the neighborhood in step 10 by making $l = l+1$. The method stops when no further neighborhood can be explored. At this stage, the final solution r cannot be improved with any of the neighborhoods considered, and therefore it is locally optimal in all of them.

4.3. MultiStart-VNS algorithm

The MultiStart-VNS method, MS-VNS, performs multiple calls to the VND from different initial solutions obtained with the application of the Shake method that perturbs the current solution.

Algorithm 4 MultiStart-VNS

1: **function** MS-VNS(k_{max}, l_{max})
2: $r_{best} \leftarrow \emptyset$
3: $t_{max} \leftarrow 60$
4: **while** $t \leq t_{\max}$ **do**
5: $r \leftarrow$ Construct()
6: $k \leftarrow 1$
7: **while** $k \leq k_{\max}$ **do**
8: $r' \leftarrow$ Shake(r, k)
9: $r'' \leftarrow$ VND(r', l_{max})
10: $k \leftarrow$ NeighborhoodChange(r, r'', k)
11: UpdateBest(r, r_{best}, t_{max})
return r_{best}

The MS-VNS algorithm takes as input parameters the time limit $t_{\max}$ and the largest neighborhood to be explored $k_{\max}$. The method first initializes the best solution found, r_{best}, as an empty set in step 2 of the algorithm. Both r_{best} and $t_{\max}$ will be updated during the search when the solution found so far is improved. At each iteration a solution r is generated with the constructive algorithm of the GRASP_IT method described in Section 3 (step 5). The solution is locally improved with the VND in step 9. It must be noted that the Shake method does not apply in its first call of the while loop as explained below.

To perform an efficient search, we define the maximum time to run our method t_{max} as a reactive parameter, updated as a function of the improvements found. As shown in Algorithm 4, initially we set a time limit of 60 s, which is the maximum time to run the algorithm. However, the UpdateBest(t, r_{best}, t_{max}) function updates t_{max} every time we find a feasible solution that improves the existing one, that is, when we improve the r_{best}. As it is customary in heuristic search, we consider the time to target, t_{target}, as the time until the best solution is found. When a better solution than the incumbent one is found, with the current time t_{target} we update $t_{max} = \lceil 10 \cdot t_{target} \rceil$, if $60 - \lceil t_{target} \rceil > \lceil 10 \cdot t_{target} \rceil$; otherwise, the maximum time remains as $t_{max} = 60$ s.

The Shake method has the objective of diversify the search by performing a random change (perturbation) in the current solution. In our routing problem, this method simply removes part of the route in the solution (and rebuilds it performing the necessary GRASP_IT iterations). To remove it, Shake randomly selects an arc in the route, and then removes a percentage $\%k$ of consecutive arcs from the total number of arcs in the route. In its first call, $k = 0\%$, meaning that no change is performed, and the constructed solution is directly submitted to the VND method described in the previous subsection. If the solution returned by VND in step 8, r'', does not improve the best solution r_{best}, then the NeighborhoodChange() method in step 9, increases k by making $k = 10\%$. In this way, we will change a larger part of the solution in the next iteration of the while loop. Following in this fashion, when the VND method is able to improve upon the best solution, the NeighborhoodChange() restores $k = 0\%$; otherwise, it increases k in steps of 10% up to the maximum percentage of change considered with $k_{max} = 50\%$.

In the outer for loop in Algorithm 4, we can see how the MS-VNS generates an initial solution in step 4 to start a complex search that alternates shaking and improvement in the inner while loop (steps 6–9). As mentioned, the Shake method has a diversification role, and the VND an intensification one, and their alternation provides an efficient and economical way to explore the solution space. Note that, on top of that, we superimpose a multi-start framework (outer while loop) that permits us to launch the search repeatedly from good initial solutions, obtaining in this way a final high-quality solution.

Table 1
Characteristics of the benchmark instances.

	#Inst	$\|V\|$	$\|A\|$		$\|A_R\|$		$\|A_{NR}\|$		$\|\mathbb{H}\|$	
			Min	Max	Min	Max	Min	Max	Min	Max
Albaida	72	116	259	305	124	172	109	162	18	33
Madrigueras	72	196	453	544	224	305	197	281	22	47
Random50	36	50	296	300	105	292	7	193	10	100
Random75	36	75	448	450	143	438	10	305	15	150
Random100	36	100	498	500	134	490	10	366	20	200
Random150	36	150	749	750	256	731	19	493	30	300
Random200	36	200	997	1000	321	972	27	679	40	400
Random300	36	300	1498	1500	502	1457	43	998	60	600
Random400	36	400	1999	2000	675	1936	63	1324	80	800

5. Computational experiments

In this section, we evaluate the performance of the proposed MS-VNS algorithm. We first present the testing environment including the instances, and then we conduct the analysis of the algorithm performance. Our experimentation is performed on a set of randomly generated instances previously reported for this problem, from which we consider a 10% of them to set up the key search parameters of our algorithm. In this way, by considering a fraction of the instances (the so-called training set) to empirically tune our algorithm, we prevent the over training of the method. We conduct a scientific testing analyzing the performance of the fastest neighborhood changes when embedded into the VND framework (shown in Algorithm 3). Finally, we compare our method described in Algorithm 4 with the state-of-the-art methods for the PCEARP: GRASP_BP and GRASP_IT and with the optimal solutions size permitting.

All the algorithms described in this paper have been implemented in the C++ programming language. For a fair comparison, all the tests have been run on virtual machines running on an OpenStack virtualisation platform supported by several blade servers, each with two 18-core Intel Xeon Gold 5220 processors running at 2.2 GHz and 384 GBytes of RAM. Each virtual machine has four virtual processors and sixteen GBytes of RAM.

5.1. Benchmark instances

For testing purposes, 396 PCEARP benchmark instances from the literature are used. The instance were generated by Bianchessi et al. (2022b) by extending CEARP instances for the Profitable CEARP. They are divided into two groups: *Albaida* and *Madrigueras*, with graphs representing street networks of two Spanish towns, and *Random*, which corresponds to randomly generated instances with up to 400 vertices. For each CEARP instance, the authors defined three intervals for the profit, generating three scenarios. Specifically, the profits were defined depending on the percentage of clients serviced, on average, in the optimal around 60%, 80%, and 90% of the total number of clients. Table 1 shows the main characteristics of the sets of instances. These instances and the best solutions found for all the sets can be found at *http://www.uv.es/corberan/instancias.htm* in the class PCEARP.

Table 2
Tuning VND Training set instances.

	Net profit	# Improv	GAP(%)
VND_R	1193.46	611	0.554
VND_0	1132.03	425	0.568
VND_1	1147.49	481	0.566
VND_2	1074.16	232	0.595
VND_3	1158.84	499	0.558
VND_D	1126.63	385	0.572

5.2. Algorithm tuning

In order to analyze the best combination of procedures and set the parameters value, we consider training set with a subset of instance with the 10% of the total number in the benchmark set. To do that, we randomly select 30 instances in which there are at least one instance of each set. It means, that the training set contains Albaida, Madrigueras and Random instances with the three intervals of profit.

The first experiment is devoted to study the combination of neighborhoods assuming the order: N_{1-1}, N_{1-2}, N_{2-1}, and N_{2-2}. As the complexity of the associated movements increases when the number of changes increases, our aim is to define the combination in which it is more efficient to be executed within the algorithm. For this purpose, several combinations of the neighborhood changes have been tested in a VND algorithm by applying the two insertion strategies (random and deterministic). Here we constructed an initial feasible solution and then applied the VND algorithm to try to improve it. To denote each neighborhood changes we will use the following notation. A random neighborhood change in which we remove i required arcs from the route and then insert other j containing any unassigned clients, is N^R_{i-j}. If instead of using the **random client** selection, we use the **deterministic arc** one, we refer to it as N^D_{i-j}. The following are the different combinations that we have considered:

- VND_R: N^R_{1-1}, N^R_{1-2}, N^R_{2-1}, and N^R_{2-2} (all random)
- VND_0: N^R_{1-1}, N^D_{1-2}, N^D_{2-1}, and N^R_{2-2}
- VND_1: N^D_{1-1}, N^R_{1-2}, N^R_{2-1}, and N^D_{2-2}
- VND_2: N^R_{1-1}, N^D_{1-2}, N^R_{2-1}, and N^D_{2-2}
- VND_3: N^D_{1-1}, N^R_{1-2}, N^D_{2-1}, and N^R_{2-2}
- VND_D: N^D_{1-1}, N^D_{1-2}, N^D_{2-1}, and N^D_{2-2} (all deterministic)

In order to avoid the randomization procedure leading to miss-leading conclusions, for each of the 30 instances of the training set, we run the method 10 times and calculate the average obtained. Table 2, shows the average results of each VND variant. Column "Net Profit" contains the average of the objective function obtained, i.e., the average of the net profit, and column "# Improv". the total number of improvements achieved. Additionally, column "GAP(%)" shows the average of the relative percentage deviation of the solutions. For each instance i, GAP_i is calculated as $\frac{BKS_i - f_i}{BKS_i} \cdot 100$, where f_i denotes the objective function of the solution found by the method, and BKS_i the Best Known Solution found for the instance. Note that, as the computation time of the VND is extremely short, a few milliseconds, we do not consider it a measure that provides relevant information in this case.

Given the results in Table 2, the combination that provides the best performance is VND_R, since it yields the largest number of improvements and, consequently, the best solutions on average. Although the results obtained by VND_3 were very similar, it is obvious that this is the best combination. It can be concluded because if this is achieved in a single iteration, the advantage of the random procedures is that when we repeat the procedure many times, it increases the solutions space explored. On the other case, the deterministic variant converges many times to the same movement, and the local search remains more static. Therefore, with this combination of neighborhood changes, we define the structure of the MultiStart-VNS, that it is $l_{max} = 4$ and the order N^R_{1-1}, N^R_{1-2}, N^R_{2-1}, and N^R_{2-2}.

5.2.1. Comparison with previous methods

In this section, we compare the MultiStart-VNS with the previous algorithms in the literature; namely GRASP_IT and GRASP_BP. In particular, we embed the best version of the VND, the VND_R, in the MS-VNS framework shown in Algorithm 4. In order to perform a fair comparison under the same conditions, all the tests have been performed on the same machine, described above, and with the same maximum computing time of 60 s. To evaluate the stochastic nature of the methods, we replicate them 30 times on each instance (running with different initial random seeds), and report the best, worst, and average values over the 30 runs. The comparison has been performed on the 396 instances on our test-bed taken from the PCEARP literature.

Table 3 shows the results obtained with the two previous heuristics together with our method, MS-VNS, grouped by data-set. Each row in this table shows the average results on each data-set, where "#Inst". is the number of instances per set. For each algorithm, the column f_{max} reports the average objective function value of the best solution found in the 30 replications (the one with the maximum objective function value). Similarly, f_{min} and f_{avg} represent the minimum (worst) and average values respectively over the 30 replications on each instance. The values in the table are average values of these statistics on each data set. In each of the 30 replication the method is run for 60 s. The last row in the table summarizes the results over the entire data set.

Results in Table 3 show that MS-VNS systematically obtains better solutions than the previous methods, GRASP_IT and GRASP_BP. In particular, the average value of its 30 replications over the entire set of instances is 1740.3, while the average values of GRASP_IT and GRASP_BP are 1643.3 and 1341.4 respectively. In fact, the average worst solution obtained with MS-VNS in its 30 replications, 1656.2, is better than these average values of the previous methods. This table also shows that GRASP_BP systematically obtains worse solutions than MS-VNS and GRASP_IT. The average value of the best solution obtained with GRASP_BP on its best run, 1519.9, is worse than the average values of the worst solution of MS-VNS (1656.2) and GRASP_IT (1586.2).

To complement the information above on the robustness of the methods, we compute the standard deviation of the 30 values obtained in the replications on each instance. The average standard deviations over the entire set of instances are 40.5, 50.3, and 86.9 for the MS-VNS, GRASP_IT, and GRASP_BP respectively, which are quite small compared with the average values shown in Table 3 of 1740.3, 1643.3, and 1341.4 respectively. This indicates that our proposal is robust across replications and outperforms the previous methods in terms of both quality and robustness. Considering that GRASP_IT consistently dominates GRASP_BP, which on the other hand is to be expected, since it was designed to feed a complex branch and price method with a relatively good solution, from now on, we compare our MS-VNS heuristic with the previous GRASP_IT that performs relatively well.

In our second final experiment, we test the ability of our method, MS-VNS, and the best previous method, GRASP_IT, to match the best known solution for each instance in the test-bed. Table 4 reports, for each data set, the number of instances in which each method reaches the best known solution, "#BKS", the average relative percentage deviation of each solution from the best known one, "GAP(%)", and the average running time (in seconds), "CPU(s)".

The experimental results shown in Tables 3 and 4 across the 9 benchmark data-sets proved that the proposed method outperforms the previous heuristic, GRASP_IT. Specifically, MS-VNS is able to obtain 287 best solutions out of the 396 instances in the experimentation, while GRASP_IT only reaches 204 best known solutions. Considering the average gap w.r.t the best solution known, MS-VNS also exhibits better performance than GRASP_IT, since it obtains a remarkable 1.72% compared with the 4.04% of the previous method. These good results of our method can be explained by the use of multiple neighborhoods, which permits it to escape from inferior locally optimal solutions.

We complement the information reported in Table 4 with the bar-chart shown in Fig. 4. This figure represents the average percentage

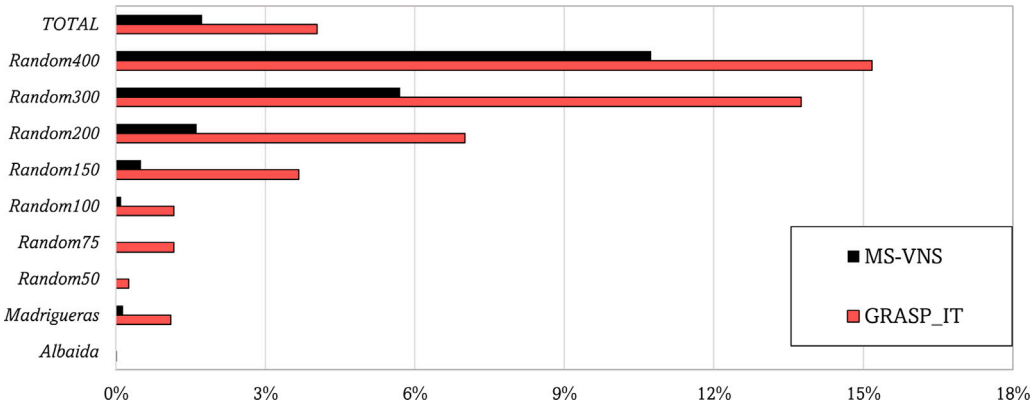

Fig. 4. Percentage GAP to best known of MS-VNS and GRASP_IT.

Table 3
Comparison of MS-VNS, with the two previous heuristics on 30 runs of 60 s per instance.

	#Inst.	MS-VNS			GRASP_IT			GRASP_BP		
		f_{min}	f_{avg}	f_{max}	f_{min}	f_{avg}	f_{max}	f_{min}	f_{avg}	f_{max}
Albaida	72	840.5	851.3	853.8	836.7	848.1	853.7	635.4	715.3	769.1
Madrigueras	72	1186.2	1201.6	1209.1	1125.8	1163.8	1195.2	668.4	829.8	987.3
Random50	36	1008.1	1024.4	1026.0	1014.0	1018.9	1024.5	822.4	901.5	980.3
Random75	36	1436.4	1470.5	1483.8	1451.1	1463.7	1475.0	1181.2	1296.0	1406.9
Random100	36	1595.1	1679.6	1696.7	1595.0	1629.2	1666.7	1122.3	1286.8	1488.5
Random150	36	2158.4	2245.8	2307.0	2077.9	2141.9	2224.3	1425.4	1653.8	1947.1
Random200	36	2304.8	2451.3	2551.6	2166.5	2262.7	2397.8	1699.6	1894.1	2144.8
Random300	36	2340.1	2554.6	2718.5	2093.0	2231.6	2400.1	1525.3	1746.2	2014.0
Random400	36	3321.6	3611.6	3828.5	3125.4	3304.7	3523.7	2602.0	2887.3	3224.4
		1656.2	**1740.3**	**1794.3**	**1586.2**	**1643.3**	**1710.0**	**1180.5**	**1341.4**	**1519.9**

Table 4
Comparison of MS-VNS and GRASP_IT w.r.t best known solutions.

	#Inst.	MS-VNS		GRASP_IT	
		GAP(%)	# BestSol	GAP(%)	# BestSol
Albaida	72	0.00	72	0.01	71
Madrigueras	72	0.15	69	1.10	50
Random50	36	0.01	35	0.26	29
Random75	36	0.00	36	1.16	22
Random100	36	0.10	33	1.16	17
Random150	36	0.50	20	3.67	7
Random200	36	1.62	12	7.01	5
Random300	36	5.71	7	13.76	3
Random400	36	10.74	3	15.17	0
		1.72	**287**	**4.04**	**204**

Table 5
MS-VNS results on a 1800 s run w.r.t 362 optimal solutions.

	#Inst.	GAP(%)	t_{targ}(s)	% Opt
Albaida	72	0.00	0.1	100.0
Madrigueras	72	0.15	18.1	95.8
Random50	36	0.01	1.1	97.2
Random75	36	0.00	7.3	100.0
Random100	36	0.10	29.8	91.7
Random150	35	0.50	109.7	57.1
Random200	35	1.58	249.5	34.3
Random300	23	5.47	142.3	30.4
Random400	17	12.88	262.8	17.6
	362	**1.19**	**63.5**	**79.3**

deviation of the best solutions obtained with each method to the best known solutions in each instance set. In this way, we can observe two effects. The first one is that our MS-VNS heuristic consistently obtains better solutions than the previous GRASP_IT heuristic, since in all sets it has lower deviation values. The second effect is to conclude, from the observation of the average deviations, the difficulty of the instances. Considering that, for example in the *Random400* set both heuristics have deviations larger than 10%, we can say that these instances are challenging for heuristic methods. On the other hand, considering that for example in the *Albaida* set both methods exhibit very small deviations, lower than 1%, we can say that this set is easy to solve for them, and do not seem to pose a challenge for modern heuristics.

We consider now a challenging experiment to test the ability of our heuristic MS-VNS to match the optimal solutions. The branch and cut of Bianchessi et al. (2022b) is able to obtain most of the optimal solutions in the data set within 1 h of computing time (362 out of the 396 instances). In this experiment, we run our method for a total of 1800 s on each instance, and we compute the time that it takes to reach the optimum, when it is able to match it. Table 5 reports, for each data-set, the number of instances, "#Inst"., the average relative percentage deviation of each instance not optimally solved, "GAP(%)", the average time in seconds taken by MS-VNS to reach the optimal solution, t_{targ}(s), and the percentage of instances optimally solved with the heuristic, "% Opt".

Results in Table 5 show that MS-VNS is able to match most of the optimal solutions in small and medium size instances ($|V| \leq 100$). However, in some of the large instances, it has difficulties to match the optimal solutions known. For example, we can see that in the *Random*75 and *Random*100 data sets, it is able to match 100% and 91.7% optimal solutions respectively, while in the *Random*200 and *Random*300 these percentages drop to 34.3% and 30.4% respectively. We can conclude then that although our heuristic performs much better than the existing heuristics, there is still room for improvement with respect to the optimal values. Note however, that in the cases in which MS-VNS does not match the optimal value, it obtains a high-quality solution with a value very close to the optimal one, as shown in the GAP average of 1.19%. Additionally, note that although the method is run for 1800 s on each instance, it is able to quickly match the optimal value in many cases, as reported in the average time to target value of $t_{targ} = 63.5$ s.

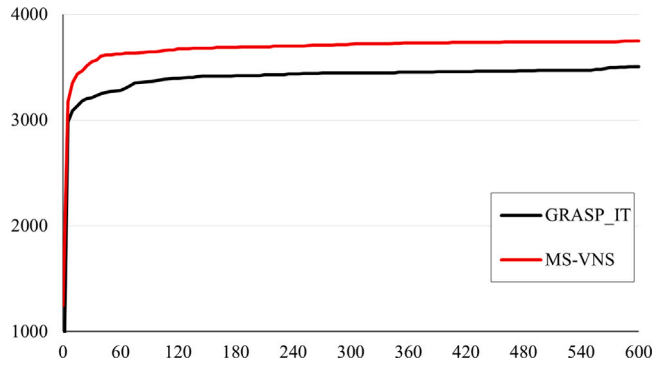

Fig. 5. Search Profile of MS-VNS and GRASP_IT over 600 s.

We conclude our experimentation with a search profile to analyze the evolution of the best solution found with the methods over time. Fig. 5 shows on the x-axis the run time in seconds, and in the y-axis the objective function value. It depicts the profiles (the value of the best solution found so far) of MS-VNS and GRASP_IT on a long run of 600 s. We consider the most difficult instances in our data sets (*Random400*) in this experiment, and report the average values of the best solutions found over time. This diagram clearly shows that both methods are able to improve their incumbent solution in the first 60 s, and after that they basically stagnate, and only present a marginal improvement. During the entire time considered, MS-VNS consistently obtained better solutions than GRASP_IT, thus confirming the results shown above. We observed similar profiles for the rest of the instances, and we may conclude that the time limit of 60 s provides a good time horizon to compare the heuristics methods.

6. Conclusions

In this paper, we study a recently proposed variant in arc routing, the profitable close-enough arc routing problem. In spite of its practical importance, this variant has been ignored in the area of routing and distribution. We propose an efficient heuristic based on the variable neighborhood search methodology.

A mathematical model and an exact method were already proposed for this problem, but the literature lacked of efficient heuristics for it, as required by logistic expert systems to deal with daily operations in the supply chain. We study different neighborhoods and disclose the best combination of them, in order to design our solving heuristic.

An extensive experimentation shows that the proposed method is able to obtain very good solutions, optimal in many cases, in very short running times. Additionally, it outperforms the two previous heuristics identified in the scientific literature, obtaining better solutions in shorter running times. There are however, some difficult instances for which our method is not able to match the optimal solution. We hope that this study triggers the interest of researchers on this applied problem and continue the research in this line.

CRediT authorship contribution statement

Miguel Reula: Conceptualization, Software, Validation, Data curation, Writing – original draft, Formal analysis, Visualization, Investigation. **Rafael Martí:** Methodology, Writing – original draft, Writing – review & editing, Supervision, Visualization, Investigation, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Miguel Reula reports financial support was provided by Spanish Ministerio de Ciencia, Innovación y Universidades (MICIU).

Data availability

I have shared the link to my data repository in the article.

Acknowledgments

The authors have been supported by grant PID2021-125709OB-C21 funded by MCIN/AEI/10.13039/501100011033 and by ERDF A way of making Europe. They also received financial support from the Generalitat Valenciana, Spain (code CIAICO/2021/224). We thank the European Union, the Spanish government (Ministerio de Ciencia e Innovación) and the Valencian government, Spain for their financial support.

References

Aráoz, J., Fernández, E., & Franquesa, C. (2017). The generalized arc routing problem. *TOP*, *25*, 497–525.

Ávila, T., Corberán, Á., Plana, I., & Sanchis, J. M. (2016). A new branch-and-cut algorithm for the generalized directed rural postman problem. *Transportation Science*, *50*, 750–761.

Ávila, T., Corberán, Á., Plana, I., & Sanchis, J. M. (2017). Formulations and exact algorithms for the distance-constrained generalized directed rural postman problem. *EURO Journal on Computational Optimization*, *5*, 339–365.

Behdani, B., & Smith, J. (2014). An integer-programming-based approach to the close-enough traveling salesman problem. *INFORMS Journal on Computing*, *26*, 415–432.

Bianchessi, N., Corberán, Á., Plana, I., Reula, M., & Sanchis, J. (2022a). The min-max close-enough arc routing problem. *European Journal of Operational Research*, *300*(3), 837–851.

Bianchessi, N., Corberán, Á., Plana, I., Reula, M., & Sanchis, J. (2022b). The profitable close-enough arc routing problem. *Computers & Operations Research*, *140*, Article 105653.

Carrabs, F., Cerrone, C., Cerulli, R., & Gaudioso, M. (2017). A novel discretization scheme for the close enough traveling salesman problem. *Computers & Operations Research*, *78*, 163–171.

Cerrone, C., Cerulli, R., Golden, B., & Pentangelo, R. (2017). A flow formulation for the close-enough arc routing problem. In A. Sforza, & C. Sterle (Eds.), *Springer proceedings in mathematics & statistics*: *vol. 217*, *Optimization and decision science: Methodologies and applications* (pp. 539–546). Springer.

Chen, S., Wang, H., Xie, Y., & Qi, C. (2014). Mean-risk analysis of radio frequency identification technology in supply chain with inventory misplacement: Risk-sharing and coordination. *Omega*, *46*, 86–103.

Corberán, Á., & Laporte, G. (2014). *Arc routing: Problems, methods, and applications*. Philadelphia: MOS-SIAM Series on Optimization.

Corberán, Á., Plana, I., Reula, M., & Sanchis, J. (2019). A matheuristic for the distance-constrained close-enough arc routing problem. *TOP*, *27*, 312–326.

Corberán, Á., Plana, I., Reula, M., & Sanchis, J. (2021). On the distance-constrained close enough arc routing problem. *European Journal of Operational Research*, *291*(1), 32–51.

Coutinho, W., Subramanian, A., do Nascimento, R., & Pessoa, A. (2016). A branch-and-bound algorithm for the close enough traveling salesman problem. *INFORMS Journal on Computing*, *28*, 752–765.

Di Placido, A., Russo, D., Capobianco, G., & Cerrone, C. (2019). Close-enough generalized routing problem.

Dong, J., Yang, N., & Chen, M. (2007). Heuristic approaches for a TSP variant: The automatic meter reading shortest tour problem. In *Extending the horizons: Advances in computing, optimization, and decision technologies* (pp. 145–163). Springer.

Drexl, M. (2007). *On some generalized routing problems* (Ph.D. thesis), Rheinisch-Westfälische Technische Hochschule, Aachen University.

Drexl, M. (2014). On the generalized directed rural postman problem. *Journal of the Operational Research Society*, *65*, 1143–1154.

Duric, J. S., Jovanovic, S. Z., & Sibalija, T. (2018). Improving the efficiency of the warehouse storage process with the use of drones. *Advanced Quality*, *46*, 46–51.

Fateme Attar, S., Mohammadi, M., Reza Pasandideh, S. H., & Naderi, B. (2022). Formulation and exact algorithms for electric vehicle production routing problem. *Expert Systems with Applications*, Article 117292.

Feillet, D., Dejax, P., & Gendreau, M. (2005). Traveling salesman problems with profits. *Transportation Science*, (39), 188–205.

Gendreau, M., Laporte, G., & Semet, F. (1997). The covering tour problem. *Operations Research*, *45*(4), 568–576.

Gulczynski, D., Heath, J., & Price, C. (2006). The close enough traveling salesman problem: A discussion of several heuristics. In *Operations research/computer science interfaces series*: *vol. 36*, *Perspectives in operations research* (pp. 217–283). Springer.

Hà, M.-H., Bostel, N., Langevin, A., & Rousseau, L.-M. (2012). An exact algorithm for close enough traveling salesman problem. In *Proceedings of the 1st international conference on operations research and enterprise systems* (pp. 233–238).

Hà, M.-H., Bostel, N., Langevin, A., & Rousseau, L.-M. (2013). An exact algorithm and a metaheuristic for the multi-vehicle covering tour problem with a constraint on the number of vertices. *European Journal of Operational Research*, *226*(2), 211–220.

Hà, M.-H., Bostel, N., Langevin, A., & Rousseau, L.-M. (2014). Solving the close enough arc routing problem. *Networks*, *63*, 107–118.

Hansen, P., Mladenović, N., Todosijević, R., & Hanafi, S. (2016). Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, *5*(3), 423–454.

Jozefowiez, N. (2014). A branch-and-price algorithm for the multivehicle covering tour problem. *Networks*, *64*(3), 160–168.

Kuo, R., Lu, S.-H., Lai, P.-Y., & Mara, S. T. W. (2022). Vehicle routing problem with drones considering time windows. *Expert Systems with Applications*, *191*, Article 116264.

Lu, J., Chen, Y., Hao, J.-K., & He, R. (2020). The time-dependent electric vehicle routing problem: Model and solution. *Expert Systems with Applications*, *161*, Article 113593.

Martí, R., Pardalos, P., & Resende, M. (2018). *Handbook of heuristics*. Switzerland: Springer Nature.

Mennell, W. (2009). *Heuristics for solving three routing problems: Close-enough traveling salesman problem, close-enough vehicle routing problem, sequence-dependent team orienteering problem* (Ph.D. thesis), College Park: University of Maryland.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100.

Moya-Martínez, A., Landete, M., & Monge, J. F. (2021). Close-enough facility location. *Mathematics*, *9*(6).

Renaud, A., Absi, N., & Feillet, D. (2017). The stochastic close-enough arc routing problem. *Networks*, *69*, 205–221.

Shuttleworth, R., Golden, B., Smith, S., & Wasil, E. (2008). Advances in meter reading: Heuristic solution of the close enough traveling salesman problem over a street network. In B. Golden, S. Raghavan, & E. Wasil (Eds.), *The vehicle routing problem: Lastest advances and new challenges* (pp. 487–501). Springer.

Yuan, B., Orlowska, M., & Sadiq, S. (2007). On the optimal robot routing problem in wireless sensor networks. *IEEE Transactions on Knowledge and Data Engineering*, *19*(9), 1252–1261.