Contents lists available at ScienceDirect

Knowledge-Based Systems

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

A modified single-objective genetic algorithm for solving the rural postman problem with load-dependent costs

David De Santis [a,b], Mercedes Landete [b], Xavier Cabezas [a,c], José María Sanchis [d], Juanjo Peiró [e,*]

[a] Facultad de Ciencias Naturales y Matemáticas, Escuela Superior Politécnica del Litoral, Guayaquil, Ecuador
[b] Centro de Investigación Operativa, Universidad Miguel Hernández, Elche, Spain
[c] Centro de Estudios e Investigaciones Estadísticas, Escuela Superior Politécnica del Litoral, Guayaquil, Ecuador
[d] Departamento de Matemática Aplicada, Universitat Politècnica de València, Spain
[e] Departament d'Estadística i Investigació Operativa. Facultat de Ciències Matemàtiques, Universitat de València, Spain

ARTICLE INFO

Keywords:
CPP-LC
Genetic algorithm
Rural postman problem
Transportation
Logistics

ABSTRACT

This study addresses the rural postman problem with load-dependent costs, a variant of the arc routing problem where the traversal cost of an edge depends on its length and the vehicle's load. The objective is to find a minimum-cost tour that services all required edges, a problem of particular importance when the demand weight is significant compared to the vehicle's curb weight. We present an integer linear programming model for the problem and propose a heuristic algorithm based on bio-inspired methodologies to efficiently obtain near-optimal solutions within short computing times. The effectiveness of the approach is demonstrated through computational experiments on benchmark instances, and the results highlight the practicality of the proposed methods.

1. Introduction

Arc routing problems (ARPs) nowadays constitute a well-established field of hard-to-solve combinatorial optimization problems with many applications. In these problems, a service is demanded on the arcs and edges of a given network. Usually, the objective is to find a route along the arcs and edges that meets a series of constraints, among which is that the total cost of traversing them is minimal.

This family of problems has been extensively studied over the last sixty years due to their real-life applications and the technical challenges they entail. Examples where specific segments of a street network require service include street cleaning, snow plowing, salt spreading, road marking, as well as mapping and path inspection. Other applications arising in cutting machines, plotters and printers are also naturally formulated as arc routing problems. In other types of delivery or pickup applications, such as postal service, newspaper delivery, garbage collection and meter reading, although the demand is located at points, appropriate models can also be developed as arc routing problems.

The study of ARPs can be traced back to the 18th century, when Euler [1] solved the well-known *Königsberg bridge* problem. This problem concerned finding a closed walk that traversed each of the seven bridges of the city of Königsberg without crossing any bridge more than once. Although the case studied by Euler was not an optimization problem, since the distance traveled in the closed walk was not considered, his work laid the foundation for later advancements. Much later, the Chinese mathematician Mei-Ko [2] presented what is now known as the Chinese postman problem (CPP), which aims at finding the shortest closed walk (tour) for a mailman, starting and ending at a post office, by traversing each street segment of a network. Nowadays, the CPP is defined as the problem of finding a minimum cost tour that traverses all the edges of a given undirected graph with a known cost associated with the traversal of each edge of the graph.

A first way to generalize the CPP relates to the type of connections between the vertices in the underlying graph in which the problem is defined: edges, arcs, or a mix of both. An edge is a link that can be traversed in both directions with the same cost. An arc represents a link that, with an associated cost, can only be traversed in one direction. A graph in which all the links are arcs is called *directed*, and the CPP defined on it is called a directed CPP, whereas if the CPP is defined on an undirected graph it is called an undirected CPP. Edmonds and Johnson [3] proved that both the undirected and the directed versions of the CPP can be solved in polynomial time. The *mixed* CPP, defined

* Corresponding author.
E-mail addresses: dade@espol.edu.ec (D. De Santis), landete@umh.es (M. Landete), joxacabe@espol.edu.ec (X. Cabezas), jmsanchis@mat.upv.es (J.M. Sanchis), juanjo.peiro@uv.es (J. Peiró).

https://doi.org/10.1016/j.knosys.2025.113146
Received 25 September 2024; Received in revised form 15 January 2025; Accepted 5 February 2025
Available online 14 February 2025

on a mixed graph that contains both edges and arcs, is $\mathcal{NP}$-hard, as shown in Papadimitriou [4]. A *windy* graph is an undirected graph with two traversal costs associated with each edge, one in each direction of travel. The CPP on a windy graph, proposed by Minieka [5] and named the windy postman problem (WPP), is also $\mathcal{NP}$-hard. Note that if the two costs of an edge are equal in a windy graph, it can be considered an undirected edge, while if one of its costs is infinite, it can be considered an arc. Therefore, windy graphs generalize undirected, directed and mixed graphs.

A second natural generalization of the CPP is the rural postman problem (RPP) introduced by Orloff [6]. Here, only a subset of links, called the set of *required* links, need to be serviced (traversed), while the remaining links (called *unrequired*) do not, though they are available for deadheading in order to obtain a feasible tour. Thus, the (undirected) RPP is defined as the problem of finding a minimum cost tour that traverses a given subset of the edges of an undirected graph with a cost associated with the traversal of each edge. Lenstra and Rinnooy Kan [7] showed that the RPP is $\mathcal{NP}$-hard. In fact, all its versions, the directed, undirected, mixed, and windy RPPs, are known to be $\mathcal{NP}$-hard. Since then, a large number of generalizations of the RPP (and, therefore, of the CPP) have been proposed and defined to model real-life applications more precisely and to express different objectives. Summaries of the current state of knowledge can be found in Corberán and Laporte [8], Corberán et al. [9], and Mourão and Pinto [10]. Some examples of these generalizations are:

- The addition of time windows for the completion of services, an objective related to profit or benefit maximization (see Malandraki and Daskin [11]).
- The association of a demand with each required edge while declaring a maximum capacity for the vehicle to satisfy demands, resulting in the capacitated version of the arc routing problem, which is a multi-vehicle ARP introduced by Golden and Wong [12].
- The repetition of a service in some periods of time (e.g., days) on a time horizon (e.g., weeks or months), which is called the periodic RPP (see Benavent et al. [13]).
- The assumption that the vehicle is not required to reach the exact point where the customer is located but only needs to pass *close enough* to that point (see Reula and Martí [14]).

In recent years, new generalizations of the CPP have emerged in response to environmental concerns, particularly the need to account for CO_2 emissions in routing problems. Unlike traditional formulations, these models incorporate variable costs that depend on both the edge length and the vehicle's load. Vehicle fuel consumption, a key factor in emissions, is influenced by three primary variables: distance traveled, vehicle weight (including load), and speed. By addressing these variables, load-dependent costs offer a practical approximation of real-world transportation challenges.

These factors were first considered by Zachariadis et al. [15] for the load-dependent vehicle routing problem (LDVRP), in which the objective function is determined as the product of the distance traveled and the gross weight that is transported over this distance, and in Corberán et al. [16] for the CPP with load-dependent costs (CPP-LC), in which the cost of traversing an edge is a variable determined as the length of the edge multiplied by the total weight of the vehicle at the moment it is traversed. As Corberán et al. [16] say, that cost is a good approximation of the amount of pollution emitted by a vehicle traveling at constant speed. In that study, the authors proposed two formulations: one based on arcs, which is seen as the natural representation of the problem, and another based on nodes, where the arcs are separated by adding additional end nodes. They also proposed various metaheuristic approaches and revealed the extraordinary difficulty of solving the CPP-LC.

Our contribution

In this study, we generalize the above-mentioned CPP-LC model with the rural postman problem with load-dependent costs (RPP-LC). As in the CPP-LC, the cost of traversing an edge will be defined as the length of the edge multiplied by the load carried by the vehicle at the time it traverses the edge, although not all edges will be required to be traversed, since only a subset of required edges will contain those with a positive demand. The vehicle will leave the depot with a load equal to the sum of all demands. Along the tour, when a required edge is serviced, the vehicle load will decrease by an amount equal to the edge demand. The goal will be to find a tour that services all the required edges with a minimum total cost.

To the best of our knowledge, this is the first model and solution method proposed for the RPP-LC. We model the problem using an integer linear formulation and tackle it by means of a metaheuristic algorithm, following the philosophy of bio-inspired methodologies to obtain near-optimal feasible solutions in short computing times. We test the applicability of the formulation and the quality of the algorithmic proposal using a set of well-known benchmark instances for related problems by running several computational experiments, whose results we also discuss. We finish the paper with some conclusions.

2. Problem statement and mathematical model formulation

Let $G = (V, E)$ be a connected graph with a set of nodes $V = \{1, 2, \ldots, n\}$ and a set of edges E defined by m pairs of nodes $\{i, j\}$, with $i, j \in V, i \neq j$. An edge $e = \{i, j\}$ represents a link between nodes i and j, can be traveled in both directions, and has a length $d_e \geq 0$. Let $E_s \subseteq E$ be a subset of edges, known as the set of *required* edges, which must be traversed for service that involves a significant change in weight in the vehicle (e.g., spreading salt on roads...). For each $e \in E_s$, let $w_e \geq 0$ denote its demand, that is, the units of commodity (e.g., kilograms of salt, liters of water) to be provided on edge e. We call Q the sum of the demands of edges in E_s, i.e., $Q = \sum_{e \in E_s} w_e$.

A vehicle with curb weight W (i.e., its weight when unloaded), departs from a designated *depot* carrying an initial load equal to the total demand Q. The vehicle travels along the edges of the graph to deliver the demands for service on the required edges. It may also need to traverse some non-required edges to define a tour that ends at the depot. The first time a required edge is traversed, we will consider it serviced, that is, w_e units of commodity have been unloaded from the vehicle. Both the required edges after being serviced and the non-required edges can be traversed in deadheading mode (no commodity is unloaded) any number of times.

The cost of traversing an edge $e \in E$ is the result of multiplying d_e by the weight of the vehicle while traversing e. The objective of the RPP-LC is to find a minimum-cost tour that starts and ends at the depot and traverses and services all the required edges (an RPP-LC tour).

Since the weight of the vehicle varies while the vehicle performs a given RPP-LC tour, the cost of traversing an edge is also variable. For each edge $e \in E$, let q_e denote the load in the vehicle at the moment of traversing e in a given RPP-LC tour. If e is traversed in deadheading mode, q_e can be computed as the sum of the demand of all edges that have not yet been visited, including the current edge e, which we denote by $E_s \setminus E_e$, where E_e is the set of edges visited before e, i.e., $q_e = \sum_{u \in E_s \setminus E_e} w_u$. If edge $e = \{i, j\} \in E_s$ is traversed while serving it, the load at node i is $\sum_{u \in E_s \setminus E_e} w_u$ and the load at node j is $\sum_{u \in E_s \setminus E_e} w_u - w_e$. If we assume that the unloading is regular along the edge, the average load of the vehicle at the moment of servicing e is $q_e = \sum_{u \in E_s \setminus E_e} w_u - \frac{w_e}{2}$. The cost of traversing an edge e is computed as $d_e(W + q_e)$. Accordingly, the total cost of a given RPP-LC tour is computed as $\sum_{e \in E} d_e(W + q_e)$.

2.1. Illustrative example

Consider the RPP-LC instance depicted in Fig. 1, with seven nodes and 11 edges of equal length $d_e = 1$. There are three required edges, $\{2, 4\}$, $\{3, 6\}$, and $\{5, 7\}$, drawn in bold with their corresponding demand w_e.

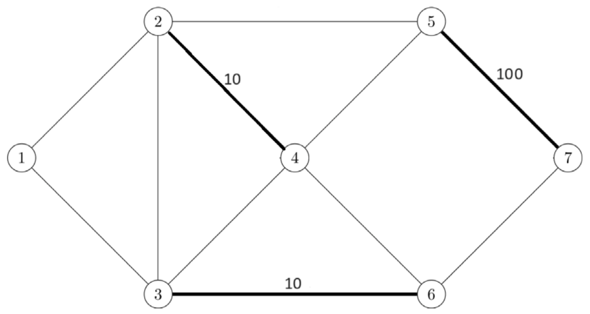

Fig. 1. An example of a graph with seven nodes.

A feasible RPP-LC tour for this instance is $1 \to 2 \to 4 \to 5 \to 7 \to 6 \to 3 \to 1$: it begins and ends at the depot, is connected, and traverses the three required edges. This tour, with length 7, would be an optimal tour of the pure RPP, that is, without considering the weight and load of the vehicle.

In order to calculate the real RPP-LC cost of this tour, let us assume, for example, that $W = 0$, and compute q_e and the cost $d_e(W+q_e)$ for each edge traversed. We obtain a total cost of $\sum_{e\in E} d_e(W+q_e) = 420$, according to the following table:

Traversed edge e	Load q_e	Cost $= d_e(W+q_e)$
$\{1,2\}$	120	$1(0+120) = 120$
$\{2,4\}$	$120 - \frac{10}{2} = 115$	$1(0+115) = 115$
$\{4,5\}$	110	$1(0+110) = 110$
$\{5,7\}$	$110 - \frac{100}{2} = 60$	$1(0+60) = 60$
$\{7,6\}$	10	$1(0+10) = 10$
$\{6,3\}$	$10 - \frac{10}{2} = 5$	$1(0+5) = 5$
$\{3,1\}$	0	$1(0+0) = 0$
Total cost of the tour		420

This tour is not optimal for the RPP-LC in this instance, since the large demand of the edge $\{5,7\}$, $w_{\{5,7\}} = 100$, makes it desirable to service that edge as soon as possible. A second tour, $1 \to 2 \to 5 \to 7 \to 5 \to 2 \to 4 \to 6 \to 3 \to 1$, has a cost of $\sum_{e\in E} d_e(W+q_e) = 380$, according to:

Traversed edge e	Load q_e	Cost $= d_e(W+q_e)$
$\{1,2\}$	120	$1(0+120) = 120$
$\{2,5\}$	120	$1(0+120) = 120$
$\{5,7\}$	$120 - \frac{100}{2} = 70$	$1(0+70) = 70$
$\{7,5\}$	20	$1(0+20) = 20$
$\{5,2\}$	20	$1(0+20) = 20$
$\{2,4\}$	$20 - \frac{10}{2} = 15$	$1(0+15) = 15$
$\{4,6\}$	10	$1(0+10) = 10$
$\{6,3\}$	$10 - \frac{10}{2} = 5$	$1(0+5) = 5$
$\{3,1\}$	0	$1(0+0) = 0$
Total cost of the tour		380

This tour, despite being longer than the previous one (it has length 9 instead of 7), has a lower RPP-LC cost (380 instead of 420). This example encourages us to formulate and look for solution techniques to specifically target the RPP-LC.

If we now assume that $W > 0$, the RPP-LC tour cost increases by W times the total distance traveled. Thus, the cost of the first tour increases to $420 + 7W$ while the cost of the second tour increases to $380 + 9W$. If, for example, we fix $W = 10$, the cost of the first tour becomes 490 and that of the second is 470. So, when W becomes large enough ($W > 20$ in our illustrative example), the first tour comes to be better than the second. In general, RPP-LC instances with large values for the curb weight compared with the demand values have the same optimal tour as the pure RPP. Hence, the study we carry out next makes sense for applications in which the demand weight is relatively large compared to the curb weight.

2.2. A formulation

To formulate the RPP-LC, we consider the arc routing formulation of Corberán et al. [16] as a basis for our modeling framework. This selection enables us to state our problem in a very natural way compared to their node routing formulation, which needs some additional (non-natural) mathematical artifacts based on transformations of graphs and replications of nodes.

We start by noting that, given that the cost of traversing an edge will depend on the load carried in the vehicle when the edge is traversed, and that this load will change depending on when a required edge is serviced, $|E_s| + 1$ periods will be considered. We will denote them by the index $k \in \{0,1,2,\dots,K\}$, with $K = |E_s|$. Period $k = 0$, starting at the depot, consists of all deadheadings (if any) needed to reach the first edge serviced. Each of the next periods $k = 1,\dots,K$ consists of an edge that is serviced at the beginning of the period and all deadheadings (if any) needed to reach the edge serviced at the beginning of the next period (or the depot, in period K). For example, in the illustrative instance shown in Fig. 1 with the feasible tour $1 \to 2 \to 5 \to 7 \to 5 \to 2 \to 4 \to 6 \to 3 \to 1$, we have $K = |E_s| = 3$, and the edges of the corresponding four periods are:

- Period 0: $1 \to 2 \to 5$.
- Period 1: $5 \to 7 \to 5 \to 2$.
- Period 2: $2 \to 4 \to 6$.
- Period 3: $6 \to 3 \to 1$.

We now define the following sets of decision variables:

- For the service of each required edge $\{i,j\} \in E_s$ and each period $k \in \{1,\dots,K\}$, we use a binary variable y_{ij}^k with the following meaning:

$$y_{ij}^k = \begin{cases} 1, & \text{if edge } \{i,j\} \in E_s \text{ is serviced from } i \text{ to } j \text{ in} \\ & \text{period } k, \\ 0, & \text{otherwise} \end{cases}$$

  Note that in period $k = 0$ no edge is served, so $y_{ij}^0 = 0$ for every edge $\{i,j\} \in E_s$ is not needed.
- For the deadheading of each edge $\{i,j\} \in E$ and each period $k \in \{0,1,\dots,K\}$, we use a binary variable x_{ij}^k where

$$x_{ij}^k = \begin{cases} 1, & \text{if edge } \{i,j\} \in E \text{ is deadheaded from } i \text{ to } j \text{ in period } k, \\ 0, & \text{otherwise.} \end{cases}$$

- For the load in the vehicle at the beginning of period $k \in \{1,\dots,K\}$, we use a continuous variable $f_k > 0$. Note that $f_1 = Q$ and that we can assume $f_{K+1} = 0$.

The objective function of the problem aims at minimizing the total cost of an RPP-LC tour, which can be decomposed into the service cost in periods $k = 1,\dots,K$, and the deadheading cost in periods $k = 0,1,\dots,K$ as

$$\sum_{k=1}^{K} \sum_{e=(i,j)\in E_s} d_e\left(W + f_k - \frac{w_e}{2}\right)\left(y_{ij}^k + y_{ji}^k\right) + \sum_{k=0}^{K} \sum_{e=(i,j)\in E} \left(W + f_{k+1}\right)\left(x_{ij}^k + x_{ji}^k\right).$$

This objective function is not linear due to $f_k\,(y_{ij}^k + y_{ji}^k)$ and $f_{k+1}\,(x_{ij}^k + x_{ji}^k)$, which are products of variables. It can be converted to a linear one by means of the classical linearization of the product of a binary variable multiplied by a bounded variable. To do so, suppose we know some bounds on the f_k such that $L_k \le f_k \le U_k$ for all $k = 1,\dots,K$. Obviously, $L = 0$ and $U = Q$ are bounds for each f_k, but tighter ones can be calculated if desired (see Corberán et al. [16]).

By defining the following sets of auxiliary variables t_e^k and z_e^k as

- $t_e^k = f_k(y_{ij}^k + y_{ji}^k), \quad \forall e = (i,j) \in E_s, \forall k = 1,\dots,K$, and
- $z_e^k = f_{k+1}(x_{ij}^k + x_{ji}^k), \quad \forall e = (i,j) \in E, \forall k = 0,1,\dots,K$,

and by adding constraints

$$t_e^k \leq f_k + L_k\left(y_{ij}^k + y_{ji}^k - 1\right), \quad \forall e=(i,j)\in E, \forall k=1,\dots,K, \tag{1}$$

$$t_e^k \geq f_k + U_k\left(y_{ij}^k + y_{ji}^k - 1\right), \quad \forall e=(i,j)\in E, \forall k=1,\dots,K, \tag{2}$$

$$t_e^k \leq U_k\left(y_{ij}^k + y_{ji}^k\right), \quad \forall e=(i,j)\in E, \forall k=1,\dots,K, \tag{3}$$

$$t_e^k \geq L_k\left(y_{ij}^k + y_{ji}^k\right), \quad \forall e=(i,j)\in E, \forall k=1,\dots,K, \tag{4}$$

$$z_e^k \leq f_{k+1} + L_{k+1}\left(x_{ij}^k + x_{ji}^k - 1\right), \quad \forall e=(i,j)\in E, \forall k=0,1,\dots,K-1, \tag{5}$$

$$z_e^k \geq f_{k+1} + U_{k+1}\left(x_{ij}^k + x_{ji}^k - 1\right), \quad \forall e=(i,j)\in E, \forall k=0,1,\dots,K-1, \tag{6}$$

$$z_e^k \leq U_{k+1}\left(x_{ij}^k + x_{ji}^k\right), \quad \forall e=(i,j)\in E, \forall k=0,1,\dots,K-1, \tag{7}$$

$$z_e^k \geq L_{k+1}\left(x_{ij}^k + x_{ji}^k\right), \quad \forall e=(i,j)\in E, \forall k=0,1,\dots,K-1, \tag{8}$$

to the formulation of the problem, the non-linear objective function can be linearized as

$$\sum_{k=1}^{K}\sum_{e=(i,j)\in E_s}\left(d_e\left(W-\frac{w_e}{2}\right)(y_{ij}^k+y_{ji}^k)+d_e t_e^k\right) + \sum_{k=0}^{K}\sum_{e=(i,j)\in E}\left(d_e W(x_{ij}^k+x_{ji}^k)+d_e z_e^k\right). \tag{9}$$

Moreover, in the formulation of the problem we will use additional notation:

- $y^k(\delta^+(i)) = \sum_{e\in E_s} y_{ij}^k$ and, similarly, $y^k(\delta^-(i)) = \sum_{e\in E} y_{ji}^k$. Note that some of these sets could be empty if node i is not incident with required edges.
- $x^k(\delta^+(i)) = \sum_{e\in E} x_{ij}^k$ and, similarly, $x^k(\delta^-(i)) = \sum_{e\in E} x_{ji}^k$.

Then, the RPP-LC can be formulated with the following integer linear programming problem:

Min (9)

s.t.:

Constraints (1)–(8), and

$$\sum_{k=1}^{K}(y_{ij}^k + y_{ji}^k) = 1, \quad \forall e\in E_s, \tag{10}$$

$$\sum_{(i,j)\in E_s}(y_{ij}^k + y_{ji}^k) = 1, \quad \forall k=1,\dots,K, \tag{11}$$

$$f_{k+1} = f_k - \sum_{e=(i,j)\in E_s} w_e(y_{ij}^k + y_{ji}^k), \quad \forall k=1,\dots,K, \tag{12}$$

$$f_1 = Q, \quad f_{K+1} = 0, \tag{13}$$

$$y^k(\delta^-(i)) + x^k(\delta^-(i)) = y^{k+1}(\delta^+(i)) + x^k(\delta^+(i)), \quad \forall i\in V, \forall k=1,\dots,K-1, \tag{14}$$

$$x^0(\delta^-(i)) = y^1(\delta^+(i)) + x^0(\delta^+(i)), \quad \forall i\in V\setminus\{1\}, \tag{15}$$

$$y^K(\delta^-(i)) + x^K(\delta^-(i)) = x^K(\delta^+(i)), \quad \forall i\in V\setminus\{1\}, \tag{16}$$

$$x^0(\delta^+(1)) + y^1(\delta^+(1)) = y^K(\delta^-(1)) + x^K(\delta^-(1)) = 1, \tag{17}$$

$$x_{ij}^k \leq \sum_{\ell=1}^{k-1}(y_{ij}^\ell + y_{ji}^\ell) + y_{ji}^k, \quad \forall (i,j)\in E_s, \forall k=1,\dots,K, \tag{18}$$

$$x_{ij}^k + x_{ji}^k \leq 1, \quad \forall (i,j)\in E, \forall k=0,\dots,K, \tag{19}$$

$$x_{ij}^k, x_{ji}^k, y_{ij}^k, y_{ji}^k \in \{0,1\}, \quad \forall (i,j)\in E, \forall k=2,\dots,K, \tag{20}$$

$$L_k \leq f_k \leq U_k, \quad \forall k=2,\dots,K. \tag{21}$$

In this model, constraints (10) and (11), respectively, guarantee that each required edge is serviced in one period, and that only one required edge is serviced at each period. Constraints (12) assure that when a required edge $e\in E_s$ is serviced, the vehicle load is decreased by w_e units. Symmetry conditions on the vertices are implied by constraints (14)–(17). We do so by imposing in (14) that every time the tour enters a node i, it has to leave it, for all the periods except the first and last; (15) and (16) are the corresponding equations for periods 0 and K in nodes other than the depot, and (17) are for periods 0 and K at the depot. Variables x and y are related through constraints (18), which state that a required edge can be deadheaded only if serviced in a previous period or in the same period but traversed in the opposite direction. Constraints (19) allow a non-required edge to be deadheaded only once in a given period.

We note that constraints (18) and (19) are not needed to find the optimal CPP-LC tour. Their removal would allow feasible solutions in which some edges could be deadheaded before being served, although, as has been said, these solutions would never be optimal. Nevertheless, these constraints have proved to be useful from a computational point of view and so we include them. We also note that this formulation extends and generalizes the arc routing formulation of Corberán et al. [16] for the CPP to the case of the RPP.

This formulation can be provided to an integer linear program solver for a specific instance in order to obtain optimal solutions.

3. A genetic algorithm

Metaheuristic algorithms (see, e.g., Gendreau and Potvin [17] and Martí et al. [18], as well as the references therein) have become a very popular family of solution methods for hard optimization problems because, very frequently, they are capable of finding high-quality solutions in short computing times. Thus, they can be employed as an alternative approach to find close-to-optimal solutions for the RPP-LC when exact methods, based on searching the solution space described by formulations, turn out to be impractical.

In this section, we present an adaptation of the classical metaheuristic methodology known as Genetic Search. Genetic algorithms (GAs) were first proposed by Holland [19] as population-based searching frameworks to find optimal solutions for mathematical problems by simulating the natural evolution process, inspired by biological organisms. Nowadays, they constitute a well-established search framework that can be applied to deal with a wide range of problems in statistics, machine learning [20], combinatorial optimization and other domains.

Various types of genetic algorithms can be found in the literature, classified by various factors:

- Concerning the type of information used to encode solution characteristics, we find *binary-coded* GAs, in which characteristics are coded in binary strings comprising boolean variables, *permutation-coded* GAs, in which a string of natural numbers represents a useful sequence (permutation), and *real-coded* GAs, where real numbers associated with problems are used to store information.
- Regarding the number of objectives that need to be optimized, we can find *single-* and *multi-objective* GAs. In the former, we wish to find the best solution for a unique specific objective. In the latter, we face problems with several objectives and we do not necessarily have an optimal solution that optimizes all the objective functions simultaneously. Often, objectives conflict with each other, and the optimal solutions of some objectives usually conflict with the optimality of other objectives. Therefore, we have to choose some trade-off or achieve a certain balance of objectives.

We refer the reader to Deb et al. [21] for specific implementation details of a well established and powerful methodology, known as NGSA-II, to solve multi-objective optimization problems with GAs.

- Regarding the simultaneity of searches that the algorithm carries out in a run, we find *serial* (also known as *sequential*) and *parallel* GAs. Parallel algorithms are parallel implementations of GAs. For instance, when creating a population of solutions, a GA can instruct multiple processors to create their own populations and merge them later in a common space. Such parallelism may bring advantages, e.g., parallel searches with information exchange between multiple searches are often better than serial computation. We refer the reader to Rivera [22] and Konfrst [23] for more detailed information on parallel GAs, and to Akopov et al. [24] and Zhong et al. [25] for insights of advanced implementations on their multiagent versions.

We note that the above-mentioned types of GAs can also have subtypes, for example those that use different selection and mutation operators as genetic operators. In addition, the flexibility of GAs allows us to hybridize some of their elements and produce *hybrid* genetic algorithms (see, e.g., [26] for details on experiences with hybridization).

Next, we explain how we represent a feasible solution of the problem. Then, we explain the main structure of the heuristic algorithm, and finally, we provide details of each of its ingredients used in this solving technique.

3.1. Solution representation

Let us recall that a feasible solution of the RPP-LC is a tour that begins and ends at the depot and traverses, among others, all the required edges. If we assign a unique identifier, for example the natural numbers from 1 to $|E_s|$, to each required edge in such a way that each identifier represents a unique edge of E_s, any permutation $\mathtt{x}$ of $\{1,\ldots,|E_s|\}$, which can be stored using an array of integers, provides an ordered sequence of required edges that a vehicle can visit, in that order, in a feasible tour for the RPP-LC.

Once we obtain a particular $\mathtt{x}$, to define the direction in which any edge $\{i,j\} \in E_s$, for $i < j$, will be traversed, we can use two identifiers: a 1 if we traverse it from i to j, and a 0 otherwise. This information can be saved in a vector of directions, say $\mathtt{d}$, of size $|E_s|$, which can be efficiently stored in a boolean array.

The remaining information that completes a solution is the sequence of non-required edges to be traversed along the tour. Although, once we know $\mathtt{x}$ and $\mathtt{d}$, we can always compute — inline — the shortest path between any pair of required edges to obtain the shortest sequence of non-required edges, for our genetic algorithm proposal it will be helpful to pre-compute some information:

- A matrix, say D_{sp}, with the shortest distance between each pair of nodes in N, where N is the set with all nodes that are incident with the edges in E_s plus the depot (if it is not already included in N).
- A list S_{p} of the shortest path between each pair of nodes in N. This information will speed up the construction of the complete tour, given $\mathtt{x}$ and $\mathtt{d}$.

D_{sp} and S_{p} will be used, as proposed by Corberán et al. [16], in a dynamic programming method that guarantees an optimal sequence of $\mathtt{d}$ from a given $\mathtt{x}$. Hence, for us, a feasible solution $\mathtt{s}$ of an instance of the RPP-LC can be represented by $\mathtt{s} = (\mathtt{x}, \mathtt{d})$ and completed with the information pre-computed and saved in D_{sp} and S_{p}.

We will see later that this representation is very useful for the application of the heuristic methodology chosen.

3.2. Adaptation of the genetic algorithm framework

In essence, a GA consists of methods to generate, maintain, and transform a set of solutions (called *chromosomes*), which change (*evolve*) over successive iterations (*generations*) using three operators: selection, crossover, and mutation. The initial step of the GA consists of obtaining an initial set (a *population*) of solutions. Each solution in this population is evaluated, based on certain criteria, and accordingly a *fitness* value is assigned to it. Then, a selection mechanism is invoked to choose relatively good (*fit*) solutions to be part of a special subset (*generation*) in which the crossover (*reproduction*) process will be applied. That process creates new solutions — a new population — by recombining characteristics of existing solutions, through a crossover from the current generation. Mimicking nature and its processes, random changes (*mutations*) may occur at any point in some solutions, as a measure to prevent premature convergence of the method. From the resulting population, one can re-apply the process to successfully generate a new generation and continue with the search process. This search stops when a stopping criterion is met, which is usually related to the time spent in the entire process, a maximum number of generations, or a convergence criterion.

Standard and advanced GA implementations have been developed for a wide variety of hard-to-solve combinatorial optimization problems. See, among others, some successful applications on related problems in the works by Domínguez-Casasola et al. [27], Felipe et al. [28], Liu and Zeng [29], and Romanuke [30] for solving the TSP, and also in the works by Felipe et al. [31], Sgarro and Grilli [32], Shen et al. [33], and Wang et al. [34] for solving the VRP.

For the RPP-LC, we next describe a detailed adaptation of the methodology, whose general structure is depicted in Algorithm 1. This procedure receives some user-defined parameters as input: the size of each population, $\mathtt{N}$, the percentage of the population that will be selected for crossover, p_c, the probability of mutation for a given solution, p_m, and the maximum computing time allowed for the entire procedure to be performed, $\mathtt{maxTime}$. As the methodology suggests, the framework starts by obtaining a population of solutions from scratch as a first step. We do so, in line 2 of Algorithm 1, by invoking the $\mathtt{InitialPopulation}$ method. Then, in a **while** loop, in which the process will be kept until a maximum computing time is reached, a block of sub-processes for evaluating solutions, selecting parents, crossover, improvement and mutation is performed to generate new offspring for a new population, which will replace the current population. Once the stopping criterion has been met, the procedure returns the best solution found during the whole process. Each of these sub-processes will be explained in detail below.

3.2.1. Obtaining $\mathtt{Pop}$, the initial population of solutions

The underlying idea of this sub-process, whose structure is depicted in Algorithm 2, is to construct $\mathtt{N}$ solutions from scratch. As we mentioned before, a solution $\mathtt{s}$ can be represented by $(\mathtt{x}, \mathtt{d})$. Hence, the procedure we have designed returns a duple of this kind.

In GAs, $\mathtt{Pop}$ is frequently generated by purely random methods. Such randomization has advantages, such as having a diverse initial population that prevents premature convergence. Accordingly, our procedure generates several $\mathtt{x}$ vectors by invoking, each time, a standard $\mathtt{shuffle}$ function that returns a random permutation of the required edges numbered from 1 to $|E_s|$. Note that this may render $\mathtt{x}$ vectors that were generated before. We can avoid such occurrences by checking the previous outcomes every time a new vector $\mathtt{x}$ is generated, at the expense of extra computing time. Since we wish to obtain a first set of solutions quickly, we decided not to impose any guaranteeing mechanism to check for previous occurrences.

Later, to obtain $\mathtt{d}$ for each $\mathtt{x}$, we apply the dynamic programming algorithm by Corberán et al. [16], which is able, on receiving $\mathtt{x}$ as input, to obtain the shortest paths between any pair of required edges and, hence, the best directions in which to traverse the required edges. During this shortest path calculation, the objective function $\mathtt{cost}$ is calculated using $\sum_{e\in E} d_e(W + q_e)$, and therefore it is not necessary to do this computation in a separate step.

At the end of each global iteration, $\mathtt{x}$, $\mathtt{d}$ and $\mathtt{cost}$ are placed, in rows, in an array of each corresponding type, say $\mathtt{X}$, $\mathtt{D}$ and $\mathtt{F}$. The output

Algorithm 1: GA proposal for the RPP-LC

```
   Input: N, $p_c$, $p_m$, maxTime.
1  Initiate a global variable elapsedTime $\leftarrow 0$.
2  Obtain Pop, a set of feasible solutions that constitute a
     population, by invoking the InitialPopulation function.

3  while elapsedTime $<$ maxTime do
4      Evaluate the fitness of each individual in Pop using the
         Fitness function.
5      Select a percentage $p_c$ of the population in Pop, using the
         roulette wheel function, and place them in a new set
         called Selected.
6      Activate an new empty set, NextPop, in which to store the
         individuals of the next population.
7      while |NextPop| $<$ N do
8          Select parent1 and parent2 from Selected using
             the roulette wheel function.
9          Perform the order crossover operator between
             parent1 and parent2 to create offspring child.
10         Apply the improvement method localSearch to
             child.
11         Generate rand $\leftarrow$ random number in the interval $(0, 1)$.
12         if rand $< p_m$ then
13             Perform a mutation on child using the relocation
                 operator.
14         Add child to NextPop.
15     Add the best individual from Pop to NextPop.
16     Replace Pop by NextPop.
17     Update elapsedTime.
   Output: The best individual from Pop.
```

Algorithm 2: InitialPopulation function

```
   Input: N.
1  for $i = 1$ to N do
2      Generate x $\leftarrow$ shuffle$(1, \dots, |E_s|)$.
3      Apply dynamic programming to obtain d and cost.
4      Store x, d and cost in row $i$ of X, D and F, respectively.
   Output: (X, D, F)
```

of the procedure is the triplet (X, D, F), which constitutes the initial Pop.

3.2.2. Evaluating the fitness of an individual

We calculate the fitness of an individual (x, d) as the cost of its objective function. However, we note that this measure could be adapted, if desired, to consider another characteristic of the problem.

3.2.3. Selection of individuals

Once we have obtained Pop, a percentage p_c of its solutions will be selected and placed in a special subset called Selected. The underlying idea of this sub-routine is that the most promising solutions (with relatively good fitness) are kept in it to be used later for generating new solutions with better fitness.

We make this selection random using an empirical distribution function that considers the fitness of each individual. This well-known mechanism, known as *roulette wheel* (see Hancock [35]), guarantees that individuals with good fitness are more likely to be chosen. Additionally, the best individual of Pop according to the fitness function, if not previously selected by roulette wheel, is also placed in Selected due to elitism criteria. The purpose of such criteria is to maintain the quality of the solutions generation after generation (see, e.g., Ahn and Ramakrishna [36]).

3.2.4. Crossover of solutions

The idea of the crossover process is that two individuals, parent1 and parent2, from Selected are randomly paired using the roulette wheel function to obtain a new individual that we call child.

We carry out the crossover function using the 2 cut-off technique by combining the x vectors from parent1 and parent2 to obtain a new x for child. The specific details of the implementation of this operator can be found in Algorithm 3. For brevity, we will refer to the corresponding x vectors in such solutions as parent1, parent2 and child.

Algorithm 3: Order crossover operator

```
   Input: parent1, parent2.
1  Activate child as a new empty individual.
2  Randomly generate two integer numbers $g_1, g_2 \in \{1, \dots, |E_s|\}$
     such that $g_1 < g_2$.
3  for int $i = g_1$ to $i = g_2$ do
4      child[i] $\leftarrow$ parent1[i]. //Copy parent1's attribute.
5  Declare an integer variable attributesTransferred and
     store in attributesTransferred $\leftarrow g_2 + 1$.
6  if attributesTransferred $> |E_s|$ then
7      attributesTransferred $\leftarrow 1$.
8  Declare two integer variables, posParent2 and posChild,
     and store in posParent2 $\leftarrow$ attributesTransferred
     and posChild $\leftarrow$ attributesTransferred.
9  while posParent2 $\neq g_2$ do
10     while parent2[posParent2] $\in$ child do
11         posParent2 + +.
12         if posParent2 $> |E_s|$ then
13             posParent2 $\leftarrow 1$.
14     child[posChild] $\leftarrow$ parent2[posParent2]. //Copy
         parent2's attribute.
15     posParent2 + +.
16     posChild + +.
17     if posChild $> |E_s|$ then
18         posChild $\leftarrow 1$.
   Output: child.
```

The idea of this operator is not only to copy the ordered sequence of a subset of required edges from parent1 directly to child, but to do so in exactly the same position where parent1 had them (lines 3 to 6 of the pseudo-code). The partial solution child is then completed by copying parent2's information in the remaining positions in a similar way, while also maintaining the same order as in parent2 (lines 11 to 20 in the pseudo-code). Once a child is obtained from the crossover, we apply the DP algorithm to obtain d and cost.

3.2.5. Improvement phase via local search

Any solution previously obtained by our methods can be used as the starting point of a procedure that aims at finding a local optimum with respect to some neighborhood structure. In the design of this improvement phase (see the details of the specific application in Algorithm 4), we have considered three well-known local improvement algorithms based on simple tour modifications:

- 2-Exchange: We select two different edges in the tour and exchange (swap) their positions.
- 1-OPT: We select one edge in the tour and place it in a different position in the tour by moving the other edges to the left and right of these new places.
- 2-OPT: We select two edges in the tour and place them in different positions in the tour, again by moving the other edges to the left and right of this new place.

An example of the application of these movements is provided in Fig. 2.

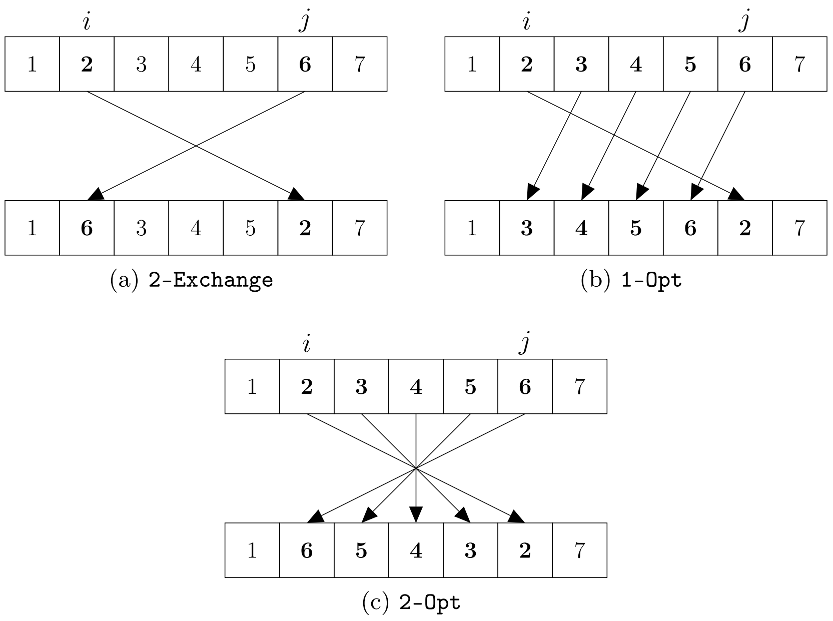

Fig. 2. Examples of the different applications of local search procedures.

For each s that is used as a starting point, we explore each of these three neighborhoods, and the next solution s_{best} is the one that maximizes the improvement in terms of objective function among those three. In our design, the algorithm repeatedly performs operations for as long as it reduces the cost of the solution, until no operation yields any further improvement, returning the best solution found so far as the output of the procedure.

Algorithm 4: `localSearch` procedure

```
Input: x, d, cost.
1  Let s_orig be the solution associated to the input (x, d, cost).
2  Activate a new empty solution s_best with cost $+\infty$.
3  Declare a boolean flag variable STOP ← FALSE.
4  while STOP == FALSE do
5      Change flag STOP ← TRUE.
6      for int $i = 1$ to $i < |E_s|$ do
7          for int $j = i + 1$ to $j = |E_s|$ do
8              // Explore the 1-OPT neighborhood:
9              Activate a temporary solution s_1-OPT and copy in it
                all the attributes of s_orig.
10             Apply 1-OPT$(i, j)$ to s_1-OPT.
11             if Cost of s_1-OPT < Cost of s_best then
12                 s_best ← s_1-OPT.
13             // Explore the 2-OPT neighborhood:
14             Activate a temporary solution s_2-OPT and copy in it
                all the attributes of s_orig.
15             Apply 2-OPT$(i, j)$ to s_2-OPT.
16             if Cost of s_2-OPT < Cost of s_best then
17                 s_best ← s_2-OPT.
18             // Explore the 2-EXCH neighborhood:
19             Activate a temporary solution s_2-EXCH and copy in
                it all the attributes of s_orig.
20             Apply 2-EXCH$(i, j)$ to s_2-EXCH.
21             if Cost of s_2-EXCH < Cost of s_best then
22                 s_best ← s_2-EXCH.
23             // Store the best solution found so far:
24             if Cost of s_best < Cost of s_orig then
25                 s_orig ← s_best.
26                 Change flag STOP ← FALSE.
Output: s_best
```

3.2.6. *Mutation procedure*

A set of feasible solutions, e.g. Pop, may eventually contain many solutions with very similar attributes. This drastically reduces the possibilities of creating new better solutions — through combination and selection — after several iterations. The purpose of a mutation procedure is to prevent such premature convergence. Typically, this is performed by applying small but random changes in some part of the solution at any stage of the solution process. This allows solutions to acquire different attributes that will be used in the following iterations.

We propose applying a random change subjected to a mutation process with probability $0 < p_m < 1$ for any child created after the improvement method via local search. If the event occurs, an edge is randomly selected in the x vector of child and placed in another position, the latter also being randomly selected.

4. Computational experiments

This section is devoted to reporting some computational experiments performed to complement the previous sections.

We start by providing details of the experimental setting. Then, we present the results of a comprehensive numerical study, performed using a set of instances, to investigate the computational efficacy of the formulation as well as its potential application to solve the RPP-CL. Following this, we present the results of several computational experiments that were conducted to assess the performance of the genetic algorithm, and we show the advantages of using the procedures proposed. The detailed information about these results is provided in several tables in Appendix.

4.1. Experimental design: technology employed and test instances

All the experiments we will report were performed on an AMD Ryzen 7-5700U processor with 1.8 GHz and 16 GB of RAM, running on the Windows 11 64-bit operating system. On the one hand, the mathematical formulation presented in Section 2.2 was implemented in Python 3 and solved with IBM ILOG CPLEX solver version 22.1 for integer programming models, with a time limit of 3600 s and the exploitation of multi-threading capabilities, that is, all cores (eight in our case) were available for use. Other than this, its default parameters are assumed unless otherwise stated. On the other hand, the GA was implemented in the Matlab programming language version R2016b, and it was run with the following parameters: $\mathtt{N} = 20$, $p_c = 0.7$, $p_m = 0.1$, and $\mathtt{maxTime} = 300$. Some of the chosen parameter values were selected based on some preliminary experimentation and observation. In particular, N was selected to allow the genetic algorithm to perform a small quantity of individuals in each global iteration in order to obtain more generations during the running time. However, p_c was selected according to related literature, and maxTime was selected considering we wanted to compete against the formulation in very short computing times.

Regarding the instances used in the experiments, Corberán et al. [9] generated three sets for the CPP-LC:

- Set E: 18 instances obtained from Eulerian graphs, with $|V| = 7, 10$ and 20, as well as $|E| = 12, 18$ and 32. For each graph, three different values of W were considered: $0, \frac{Q}{2}$ and $5Q$, with $Q = \sum_{e\in E} w_e$. For each of these nine combinations, they generated two instances, one with $w_e = d_e$ and another with randomly generated demands w_e.
- Set P: 18 instances derived from three RPP instances proposed by Christofides et al. [37], namely P1, P2 and P4, with $|V| = 11, 14$ and 17, as well as $|E| = 13, 32$ and 35. For each graph, they generated six instances, proceeding as before.
- Set H: 24 instances obtained from 12 RPP instances proposed by Hertz et al. [38], namely r1 to r8 (with $|V| \in [6, 14]$ and $|E| \in [11, 48]$), as well as d10, d11, g10 and g11 (with $|V| \in [18, 27]$ and $|E| \in [22, 33]$). For each graph, a proportional and a non-proportional instance was generated, all of them with $W = \frac{Q}{2}$.

We have made use of these instances as a basis for the generation of RPP-LC instances. To adapt them, only some of their edges need to be required. To do so, for each type of instance described above, we made a random selection of 25%, 50% and 75% of the edges to be required. In total, we obtained 180 instances, which have been placed in a Dropbox repository to be available for public access[1] and reproducibility.

4.2. Usability of the formulation

The first experiments we carried out aimed to explore whether the set of instances is solvable by using CPLEX when the formulation is provided with a time limit of 3600 s. Two main factors could affect the solvability of an instance, namely the density of required edges and the type of instance. Regarding the former, in Table 1 we summarize, for the three levels of density tested (25%, 50% and 75%), the number of instances that the formulation solved to optimality out of a total of 180. From this table, we can clearly see that the formulation could solve all instances tested with 25% density. We can also see that the higher the density values, the more difficult it is to optimally solve instances using the formulation. Using the results in Table 1, we performed a χ^2 statistical test to verify whether the difficulty of solving the instances is related to the different densities of the three groups tested. The resulting p-value of 0.002658 indicates, in our opinion, that such a statistical relationship exists.

Table 1
Number of instances solved to optimality for groups of different density.

Density	Solved	Not solved
25%	60	0
50%	48	12
75%	33	27

Table 2
Number of instances solved to optimality for different types of data.

Set	Solved	Not solved
E	51	3
P	38	16
H	52	20

Regarding the type of instances solved, we summarize in Table 2, for the three types of instances tested (namely, sets E, P and H), the number of instances that the formulation solved to optimality out of 180. From this table, we can see that 94.4% (51 out of 54) of the instances in set E were solved to optimality. That percentage diminishes to 70.3% and 72.2%, respectively, for sets P and H. Using the results in the table, we also performed a χ^2 statistical test to verify whether the difficulty of solving the instances is related to the type of instances tested. The resulting p-value of 0.00000001569 also indicates, in our opinion, that there is such a statistical relationship.

We are now interested in studying the CPU time needed to solve the instances that we were able to solve to optimality as well as the bounds obtained for those instances that were not. In Table 3 we show, for each instance and density tested, the CPU time, in seconds, needed to solve it. For the cases in which the solver could not solve an instance to optimality due to the limitation in computing time, the table reports a "t.l"., indicating that a time limit was reached. From this table, several interesting insights arise:

- Most instances with 25% density (first column) need less than a second to be solved. Only those in subset Hd need more CPU time.
- From the previous experiments, we already noticed that the higher the density, the more time the solver needed to guarantee optimality. When we look at the results of the instances with 50% density, we appreciate a change in the magnitude of CPU time needed in many subsets. See, for example, the results for subsets E20, Hg and P04: despite the fact that the solver could solve almost all instances in these subsets to optimality, the effort needed now is one or two orders of magnitude higher. In other subsets such as Hd, the change in density directly results in the impossibility of solving all their instances to optimality.

In Table 4 we report, only for the unsolved instances, the linear relaxation value and the best feasible solution value (labeled as LB and UB, respectively) that the solver had obtained on reaching the time limit. In this table, it is interesting to note that the differences when LB and UB are compared after an hour of computation are large for most instances, which means that these instances would have required a large additional amount of computing time to be solved to optimality.

In summary, Tables 3 and 4 indicate that for medium and high density instances it may be reasonable to resort to ad-hoc metaheuristic methods if we need high-quality feasible solutions in short computing times for the RPP-LC.

4.3. Solving the problem with the GA

In this section, we test the ability of our GA method to obtain high-quality solutions in short computing times.

[1] https://www.dropbox.com/scl/fo/pue18avllpzi0hmhnf28k/AISr3UC1hhrC7sZG_8hJn3U?rlkey=vbkl9ixs3cy762e8g1m7xqpkq&st=8u7ya19b&dl=0.

Table 3
CPU time (sec.) required to solve each instance.

Instance information			Density		
Set	Subset	Name	*25%*	*50%*	*75%*
E	E07	E07W0NP	0.52	0.48	0.93
		E07W0P	0.11	0.32	1.39
		E07W1000NP	0.41	0.31	1.57
		E07W100NP	0.54	0.67	0.97
		E07W200P	0.21	0.66	2.2
		E07W20P	0.13	0.47	1.17
	E10	E10W0NP	0.77	1.83	3,0
		E10W0P	0.75	1.23	20.73
		E10W1000NP	0.57	0.83	2.61
		E10W1000P	0.63	1.05	135.56
		E10W200NP	0.73	1.43	3.89
		E10W200P	0.55	1.86	6.45
	E20	E20W0NP	1.05	36.19	514.52
		E20W0P	2.03	104.73	t.l.
		E20W10000NP	1.51	4.94	1292.24
		E20W2000NP	0.89	20.79	703,0
		E20W2000P	0.69	21.46	t.l.
		E20W400P	0.92	130.38	t.l.
H	Hd	hertzd10_np	153.33	t.l.	t.l.
		hertzd10_p	3048.07	t.l.	t.l.
		hertzd11_np	118.92	t.l.	t.l.
		hertzd11_p	28.26	t.l.	t.l.
	Hg	hertzg10_np	0.79	10.28	99.98
		hertzg10_p	0.58	5.88	327.72
		hertzg11_np	1.31	795.77	t.l.
		hertzg11_p	2.3	45.75	3541.55
	Hr	hertzr1_np	0.1	0.16	0.89
		hertzr1_p	0.07	0.24	2.37
		hertzr2_np	0.1	0.29	2.13
		hertzr2_p	0.14	0.29	4.58
		hertzr3_np	0.13	0.27	1.76
		hertzr3_p	0.13	0.27	36,0
		hertzr4_np	1.42	t.l.	t.l.
		hertzr4_p	4.88	t.l.	t.l.
		hertzr5_np	0.54	7.06	187.72
		hertzr5_p	0.46	22.38	t.l.
		hertzr6_np	0.28	6.06	12.3
		hertzr6_p	0.47	22.11	t.l.
		hertzr7_np	9.33	t.l.	t.l.
		hertzr7_p	13.71	t.l.	t.l.
		hertzr8_np	0.5	3.56	3107.43
		hertzr8_p	0.47	13.56	t.l.
P	P01	P01W0NP	0.21	0.29	0.72
		P01W0P	0.16	0.36	2.58
		P01W300NP	0.09	0.31	0.38
		P01W300P	0.12	0.27	0.53
		P01W35P	0.11	0.41	0.62
		P01W65NP	0.11	0.31	0.66
	P02	P02W0NP	1.67	3425.58	t.l.
		P02W0P	1.56	t.l.	t.l.
		P02W2000NP	1.79	219.69	t.l.
		P02W2000P	1.99	t.l.	t.l.
		P02W250NP	1.75	832.01	t.l.
		P02W250P	1.3	t.l.	t.l.
	P04	P04W0NP	1.22	441.76	t.l.
		P04W0P	37.54	t.l.	t.l.
		P04W1000NP	1.06	909.93	t.l.
		P04W1000P	1.32	194.87	t.l.
		P04W130NP	0.79	797.47	t.l.
		P04W70P	3.31	1368.18	t.l.

Table 4
Best bounds for each instance on reaching the time limit.

Instance information			Density			
Set	Subset	Name	*50%*		*75%*	
			LB	UB	LB	UB
E	E20	E20W0P			208 237	238 647
		E20W2000P			1 668 044	1 906 916
		E20W400P			523 025	577 061
H	Hd	hertzd10_np	141 771	153 391	181 597	232 784
		hertzd10_p	109 056	130 926	175 565	222 270
		hertzd11_np	84 999	93 779	133 550	159 974
		hertzd11_p	69 654	83 318	89 120	130 507
	Hg	hertzg11_np			2793	3191
	Hr	hertzr4_np	1 665 032	1 839 675	2 785 578	3 196 238
		hertzr4_p	1 640 723	1 808 576	3 400 065	3 976 624
		hertzr5_p			302 663	317 609
		hertzr6_p			314 891	333 375
		hertzr7_np	1 282 996	1 492 385	2 121 829	2 356 577
		hertzr7_p	728 382	785 191	1 147 418	1 323 872
		hertzr8_p			669 534	744 845
P	P02	P02W0NP			33 458	40 192
		P02W0P	30 218	34 087	50 361	70 933
		P02W2000NP			746 983	819 375
		P02W2000P	759 816	768 073	882 226	945 387
		P02W250NP			137 491	148 675
		P02W250P	88 511	90 788	137 051	153 925
	P04	P04W0NP			7643	8608
		P04W0P	2107	2492	5012	7005
		P04W1000NP			110 944	120 885
		P04W1000P			110 283	116 538
		P04W130NP			19 370	21 064
		P04W70P			13 059	14 807

For this purpose, we first perform an experiment to see whether the GA is a good method to match the optimal solutions for the instances in the test-bed for which we know the optimal value. This information is collected in Table 5, in which we present the results for the objective value and CPU time in those instances for which we obtained the optimal value using the formulation. In this table, the first three columns contain the instance information. They are followed by three sets of columns, one for each density value (25%, 50% and 75%). For each density, the table includes three columns: the first provides the optimal value obtained with CPLEX; the second provides the GAP value measured as $100 \times \frac{UB_{GA}-UB_{CPLEX}}{UB_{CPLEX}}$; and the third provides the CPU time of the GA in seconds. These last columns showing the CPU time can be directly compared with the values of the CPU time needed by the formulation that are collected in Table 3. Some observations arise from this table:

- From the set of columns for 25% density, we can see that the GA was able to match the optimal value of all the instances tested (all with 0.0% of GAP). When looking at the CPU times, we can see that, in general, the GA method finishes in very short computing times. Hd was the only subset of instances that required more computing effort, but even for that set, the resulting CPU times compare favorably to the CPU times with CPLEX in Table 3.
- When looking at the results in the columns for 50% density, we first notice that the GA method matched all the optimal values (again, 0% GAP), and that it clearly outperforms CPLEX in terms of computing times.
- Regarding the results of the instances with 75% density, we see that the GA method matches all the optimal values except for four instances (those boldfaced in the table). For these four instances, the GAP is quite small. When looking at the CPU times, again, they compare favorably to the CPU time with CPLEX in Table 3.

Given that most of the instances for which we know the optimal value were solved by the GA in very low CPU times, we can say that the GA is able to obtain high-quality solutions in short computing times, and can be recommended as a very good alternative to the formulation deployed in CPLEX if one needs a reliable and fast solving method.

To supplement the information above on the robustness of the GA, we performed an experiment to compare the values of the best feasible solutions obtained by the GA and CPLEX when the latter could not solve the instance to optimality in the time limit provided. In Table 6 we report the results of that experiment. Concerning this comparison, we can see that the GA clearly outperforms CPLEX in terms of the values of the best solutions provided in very short computing times, since it

Table 5
Comparison of GA and CPLEX in optimally solved instances.

Instance information			Density								
Set	Subset	Name	*25%*			*50%*			*75%*		
			UB Cplex	Gap GA	CPU GA	UB Cplex	Gap GA	CPU GA	UB Cplex	Gap GA	CPU GA
E	E07	E07W0NP	391	0%	0.0	992.5	0%	0.03	1967.5	0%	0.09
		E07W0P	49.5	0%	0.0	176	0%	0.03	302	0%	0.08
		E07W1000NP	16 222	0%	0.0	21 775.5	0%	0.05	33 698.5	0%	0.06
		E07W100NP	1882	0%	0.0	2561	0%	0.03	4195.5	0%	0.07
		E07W200P	3276	0%	0.0	5202.5	0%	0.02	7616	0%	0.06
		E07W20P	469.5	0%	0.0	758.5	0%	0.03	1192	0%	0.08
	E10	E10W0NP	7542	0%	0.06	20 223.5	0%	0.72	38 212.5	0%	1.28
		E10W0P	12 255.5	0%	0.05	22 366	0%	0.07	52 045.5	0%	0.44
		E10W1000NP	195 921	0%	0.01	257 328	0%	0.06	400 306	0%	0.47
		E10W1000P	200 663	0%	0.01	299 568	0%	0.07	434 771.5	0%	0.54
		E10W200NP	46 662.5	0%	0.04	77 038.5	0%	0.08	122 444.5	0%	0.7
		E10W200P	45 088.5	0%	0.03	78 039	0%	0.08	113 452	0%	0.29
	E20	E20W0NP	185 987.5	0%	0.05	637 128	0%	5.53	1 225 498	**0.44%**	207.1
		E20W0P	42 582	0%	0.05	93 732.5	0%	0.35			
		E20W10000NP	4 717 132	0%	0.04	5 927 434.5	0%	0.69	8 529 676	**0.13%**	13.98
		E20W2000NP	1 188 121	0%	0.05	1 963 934	0%	1.45	2 766 971.5	0%	9.7
		E20W2000P	704 420.5	0%	0.05	1 280 905	0%	5.84			
		E20W400P	195 831	0%	0.08	395 670	0%	6.62			
H	Hd	hertzd10_np	68 847	0%	2.87						
		hertzd10_p	66 702.5	0%	23.16						
		hertzd11_np	51 172.5	0%	0.28						
		hertzd11_p	38 750	0%	1.51						
	Hg	hertzg10_np	964.5	0%	0.03	1402.5	0%	0.68	1789	0%	24.58
		hertzg10_p	560.5	0%	0.04	795	0%	0.74	1291	0%	9.43
		hertzg11_np	1147	0%	0.06	2219	0%	18.6			
		hertzg11_p	601.5	0%	0.08	937	0%	2.08	1403	**0.78%**	125.17
	Hr	hertzr1_np	89 436.5	0%	0.0	176 403.5	0%	0.03	280 360.5	0%	0.12
		hertzr1_p	165 216.5	0%	0.0	243 444	0%	0.02	335 500.5	0%	0.06
		hertzr2_np	118 772.5	0%	0.0	216 845	0%	0.03	295 413.5	0%	0.24
		hertzr2_p	112 946.5	0%	0.0	125 868	0%	0.04	234 076.5	0%	1.01
		hertzr3_np	79 245.5	0%	0.0	118 480	0%	0.03	209 564	0%	0.36
		hertzr3_p	84 804.5	0%	0.0	197 336	0%	0.03	319 300.5	0%	0.15
		hertzr4_np	902 738	0%	0.19						
		hertzr4_p	682 282	0%	0.1						
		hertzr5_np	103 262.5	0%	0.04	177 665	0%	0.37	296 577	0%	19.35
		hertzr5_p	99 600.5	0%	0.05	195 347.5	0%	0.59			
		hertzr6_np	139 532.5	0%	0.03	231 063.5	0%	0.11	307 200	0%	3.12
		hertzr6_p	128 476	0%	0.02	212 934.5	0%	0.6			
		hertzr7_np	601 730	0%	0.27						
		hertzr7_p	391 570	0%	3.12						
		hertzr8_np	194 362	0%	0.03	378 357.5	0%	1.68	623 000.5	**0.01%**	7.42
		hertzr8_p	222 149.5	0%	0.03	452 583.5	0%	0.49			
P	P01	P01W0NP	1067.5	0%	0.0	1850.5	0%	0.2	3982.5	0%	0.09
		P01W0P	855	0%	0.0	1567.5	0%	0.08	2108	0%	0.24
		P01W300NP	18 809	0%	0.0	20 492.5	0%	0.03	27 150.5	0%	0.16
		P01W300P	15 355.5	0%	0.01	22 623.5	0%	0.02	25 429.5	0%	0.09
		P01W35P	2180.5	0%	0.0	3420.5	0%	0.04	4913	0%	0.88
		P01W65NP	5566.5	0%	0.0	7046.5	0%	0.05	9062	0%	0.18
	P02	P02W0NP	6695	0%	0.06	22 759	0%	38.76			
		P02W0P	14 460	0%	0.08						
		P02W2000NP	288 112.5	0%	0.06	481 899.5	0%	3.28			
		P02W2000P	425 078	0%	0.06						
		P02W250NP	52 048.5	0%	0.06	90 402	0%	15.43			
		P02W250P	57 104.5	0%	0.17						
	P04	P04W0NP	1419	0%	0.06	3811	0%	78.01			
		P04W0P	1313.5	0%	0.09						
		P04W1000NP	45 333	0%	0.07	84 370	0%	16.18			
		P04W1000P	58 627.5	0%	0.13	83 371.5	0%	8.49			
		P04W130NP	6490.5	0%	0.06	14 779.5	0%	2.46			
		P04W70P	4202.5	0%	0.08	9061.5	0%	9.95			

is able to systematically improve on the bounds, in almost all cases, compared to the CPLEX results. In our opinion, this confirms the fact that the GA we designed is an attractive alternative solving method for the RPP-LC.

A comment on the stability of the obtained solutions

When comparing the best GA solutions with other algorithms, it is essential to demonstrate the stability of the solutions obtained with respect to the randomness applied in some parts of the algorithm. This is especially important when randomness plays an important role, as is the case for our proposed GA, to check whether the resulting solution outputs differ from each other.

Such stability can be analyzed by repeatedly running the GA with different random seeds. In particular, we performed an experiment in which each of the 180 instances was solved five times by our GA, using

Table 6
Comparison of best-known solution values with GA and CPLEX.

Instance information			Density							
Set	Subset	Name	*50%*				*75%*			
			UB Cplex	UB GA	Gap GA	CPU GA	UB Cplex	UB GA	Gap GA	CPU GA
E	E20	E20W0P					238 647	229 378	−3.88%	151.12
		E20W2000P					1 906 915.5	1 890 231.5	−0.87%	137.49
		E20W400P					577 061	562 138	−2.59%	225.03
H	Hd	hertzd10_np	153 390.5	150 122.5	−2.13%	194.59	232 783.5	207 615.5	−10.81%	222.28
		hertzd10_p	130 926	128 293	−2.01%	31.71	222 270	212 145	−4.56%	177.91
		hertzd11_np	93 778.5	92 028.5	−1.87%	129.6	159 974	149 599	−6.49%	301.03
		hertzd11_p	83 317.5	80 822.5	−2.99%	8.28	130 507	111 977	−14.20%	39.49
	Hg	hertzg11_np					3191	3040	−4.73%	45.5
	Hr	hertzr4_np	1 839 674.5	1 839 217.5	−0.02%	69.28	3 196 238	3 098 112	−3.07%	123.97
		hertzr4_p	1 808 576	1 808 576	0.00%	7.51	3 976 623.5	3 962 092.5	−0.37%	46.9
		hertzr5_p					317 609	317 420	−0.06%	92.27
		hertzr6_p					333 375	333 375	0.00%	0.74
		hertzr7_np	1 492 384.5	1 454 855.5	−2.51%	143.7	2 356 577	2 343 967	−0.54%	224.62
		hertzr7_p	785 191	784 432	−0.10%	8.38	1 323 871.5	1 303 384.5	−1.55%	301.36
		hertzr8_p					744 844.5	744 325.5	−0.07%	48.15
P	P02	P02W0NP					40 192	38 576	−4.02%	203.65
		P02W0P	34 087	34 087	0.00%	9.56	70 933	65 114	−8.20%	37.6
		P02W2000NP					819 375	811 188	−1.00%	184.05
		P02W2000P	768 073	768 061	0.00%	53.35	945 387	941 813	−0.38%	52.78
		P02W250NP					148 675	147 343	−0.90%	209.53
		P02W250P	90 787.5	90 787.5	0.00%	1.23	153 924.5	153 844.5	−0.05%	13.52
	P04	P04W0NP					8607.5	8611.5	0.05%	216.8
		P04W0P	2492	2492	0.00%	178.9	7004.5	6746.5	−3.68%	274.18
		P04W1000NP					120 884.5	119 843.5	−0.86%	225.75
		P04W1000P					116 538	116 517	−0.02%	98.86
		P04W130NP					21 063.5	20 769.5	−1.40%	30.96
		P04W70P					14 807	14 517	−1.96%	126.75

a different random seed generator each time in our GA code—in our case, setting the Matlab's $\mathtt{rng}(i)$ function with $i \in \{1,\ldots,5\}$ at the beginning of the code.

In 161 of the 180 instances, the GA produced completely stable optimal solutions, that is, it did not show any differences among the five solutions obtained. This indicates that in the set of instances that we are dealing with, in almost 90% of the cases the GA produced stable results. For the remaining 19 instances, the GA showed different optimal values in their repeated executions. In Table 7, we report some statistical summaries of the results obtained in these 19 instances. Specifically, we calculated the sample mean (column denoted by $\bar{x}$), the standard deviation (s), the minimum and maximum observation values (min and max, respectively) and the coefficient of variation (cv $= \frac{s}{\bar{x}}$) for the five runs of each instance. We use the coefficient of variation as a scale-free metric that provides a standardized way to compare variability between data sets in which the scales differ greatly among instances. A high cv value indicates a greater dispersion of data points around the mean, suggesting higher relative variability. Conversely, a low cv implies lower relative variability. In domains where accuracy is paramount, a cv < 0.1 may be considered acceptable. From Table 7, we can see that the cv values are very small in all of the instances reported, indicating that our GA is very stable and robust.

Comparison with Hexaly

The last point we wish to address concerns the justification of a specialized GA proposal to solve the RPP-LC rather than a straightforward application of a mathematical formulation to an off-the-shelf solver based on (meta) heuristic rules. This analysis would provide a more comprehensive assessment of the performance of the GA in the context of the task it is intended to accomplish: to obtain high-quality solutions in short computing times.

To perform such an analysis, we carried out an experiment that consisted of providing our formulation from Section 2.2 to the commercial solver Hexaly.[2] Hexaly is a well-known solver, previously marketed as LocalSolver, mainly based on heuristic and meta-heuristic searching strategies. Similarly to our GA method, Hexaly can be used as an alternative solving method for a given problem formulation and input data for which we do not require a guarantee of optimality. In our experiment, for each of the 180 instances, we let Hexaly use all cores of our CPU and set the time limit to 300 s.

In Table 8, we report a summary of the comparison of the results obtained by Hexaly and our GA, for which we also set the time limit to 300 s. In particular, for each of the two methods, we report the relative deviation of the best solution value obtained (UB) with respect to the best known bound (UB*), computed as $100 \times \frac{\text{UB}-\text{UB}^*}{\text{UB}^*}$, only for those instances with relative deviation greater than 0. The last row of the table reports the average relative deviation for both methods in each set of instances reported. From this table, several interesting findings arise:

- On the one hand, we can see that our GA method performs very well, since almost all relative deviations are 0%.
- On the other hand, we can see that Hexaly performs quite well when solving instances with 25% of density. However, for the instances with density 50% and 75%, the relative deviation dramatically increases, hence it underperforms compared to our GA proposal. Notice that, for some instances, Hexaly cannot obtain a feasible solution after the time limit provided, indicated with "t.l". in the table.

Therefore, these results provide convincing evidence of the superior performance of our proposed modified genetic single-objective algorithm, especially when compared to other off-the-shelf alternatives based on heuristic and metaheuristic strategies.

5. Conclusions

In this paper, we have introduced a variant of the Chinese postman problem that takes account of load-dependent cost, considering only a subset of the edges in the graph as required edges. A mathematical

[2] https://www.hexaly.com.

Table 7
Statistical summary of the 19 instances with variability among five repeated runs.

Instance information			Density									
Set	Subset	Name	*50%*					*75%*				
			$\bar{x}$	s	min	max	cv	$\bar{x}$	s	min	max	cv
E	E20	E20W2000NP						2 770 802	8565	2 766 972	2 786 125	0.0031
H	Hd	hertzd10_np						202 520	1021	201 084	203 891	0.0050
		hertzd10_p	128 067	97	128 005	128 233	0.0008	210 791	122	210 663	210 930	0.0006
		hertzd11_np						146 474	761	145 213	147 147	0.0052
		hertzd11_p						111 878	115	111 720	112 030	0.0010
	Hg	hertzg11_np						3013	15	3001	3040	0.0051
		hertzg11_p						1406	6	1403	1417	0.0045
	Hr	hertzr4_np	1 838 745	1056	1 836 857	1 839 218	0.0006					
		hertzr7_np	1 456 509	3697	1 454 856	1 463 122	0.0025	2 319 845	3704	2 317 310	2 325 477	0.0016
		hertzr7_p						1 303 444	172	1 303 295	1 303 678	0.0001
P	P02	P02W0NP						38 768	430	38 576	39 538	0.0111
		P02W0P						65 116	4	65 114	65 122	0.0001
		P02W250NP						147 019	74	146 943	147 132	0.0005
		P02W250P						132 442	6	132 436	132 447	0.0001
	P04	P04W0NP						8550	74	8471	8617	0.0087
		P04W0P	2493	3	2492	2498	0.0011	6720	7	6714	6731	0.0011
		P04W1000P						116 515	1	116 514	116 517	0.0001
		P04W130NP						20 675	28	20 663	20 726	0.0014
		P04W70P						14 523	8	14 517	14 531	0.0005

Table 8
Comparison of relative deviation values of GA and Hexaly.

Instance information			Density					
Set	Subset	Name	*25%*		*50%*		*75%*	
			GA	Hexaly	GA	Hexaly	GA	Hexaly
E	E20	E20W0NP					0.45%	0.00%
		E20W0P			0%	4.10%	0%	17.55%
		E20W10000NP					0%	11.95%
		E20W2000NP					0%	16.40%
		E20W2000P			0%	5.51%	0%	11.78%
		E20W400P			0%	3.07%	0%	29.01%
H	Hd	hertzd10_np	0%	2.22%	0%	42.82%	0%	t.l.
		hertzd10_p	0%	7.52%	0%	t.l.	0%	t.l.
		hertzd11_np	0%	1.69%	0%	10.91%	0%	t.l.
		hertzd11_p			0%	27.95%	0%	t.l.
	Hg	hertzg10_np					0%	7.83%
		hertzg10_p					0%	4.49%
		hertzg11_np			0%	2.21%	0%	5.62%
		hertzg11_p					0%	20.65%
	Hr	hertzr4_np			0%	11.86%	0%	50.13%
		hertzr4_p			0%	47.16%	0%	20.21%
		hertzr5_np					0%	1.56%
		hertzr5_p					0%	0.24%
		hertzr7_np			0%	30.75%	0%	t.l.
		hertzr7_p			0%	4.38%	0%	t.l.
		hertzr8_np					0%	2.30%
		hertzr8_p					0%	7.64%
P	P02	P02W0NP			0%	6.27%	0%	1.41%
		P02W0P			0%	0.34%	0%	3.41%
		P02W2000NP			0.01%	0.00%	0%	8.18%
		P02W2000P			0%	0.07%	0%	2.65%
		P02W250NP			0%	2.88%	0%	7.56%
		P02W250P	0%	0.01%	0%	0.99%	0%	29.86%
	P04	P04W0NP			0%	4.38%	0.19%	0.00%
		P04W0P			0%	1.44%	0%	1.93%
		P04W1000NP			0%	6.13%	0%	21.53%
		P04W1000P			0%	2.70%	0%	95.62%
		P04W130NP			0%	8.65%	0%	67.24%
		P04W70P			0%	5.69%	0%	29.99%
Average			0.00%	2.86%	0.00%	10.01%	0.02%	17.03%

model for the new variant has been formulated, adapting the case in which the edges are not connected with the depot directly. We have proposed a genetic algorithm to solve it, using a 2-cut points technique for crossover, a relocation method for mutation, elitist criteria, and three local search methods.

To test the formulation proposed and the genetic algorithm, a set of instances derived from well-known related problems has been created, considering densities of 25%, 50%, and 75% or the original graphs. The performance of the GA has been satisfactory, outperforming in terms of computing time almost all the results obtained by exact and heuristic methods based on the formulation, thus confirming that the GA we have designed is an attractive solving method.

CRediT authorship contribution statement

David De Santis: Writing – original draft, Software, Methodology, Conceptualization. **Mercedes Landete:** Writing – original draft, Visualization, Supervision, Methodology, Funding acquisition, Conceptualization. **Xavier Cabezas:** Writing – original draft, Visualization, Supervision, Software, Conceptualization. **José María Sanchis:** Writing – original draft, Visualization, Supervision, Methodology, Conceptualization. **Juanjo Peiró:** Writing – original draft, Visualization, Validation, Supervision, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

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

Acknowledgments

This work was supported by Gobierno de España through Ministerio de Ciencia, Innovación y Ministerio de Universidades and Fondo Europeo de Desarrollo Regional (FEDER), Spain under projects PID2021-122344NB-I00, and RED2018-102363-T, by Generalitat Valenciana, Spain under projects PROMETEO/ 2021/063 and CIGE/2022/57. The authors would like to thank the anonymous reviewers for their comments and suggestions, which have contributed to improving the content and readability of the article.

Appendix

See Tables 9–11

Table 9
25% density.

Instance	Exact method					GA		Exact method vs GA	
	UB_Exact	LB_Exact	GAP	Sol_status	T_Exact	UB_GA	T_GA	UB_GAP	T_GAP
P01W0NP	1067.50	1067.50	0.00	optimal	0.21	1067.50	0.00	0.00	0.20
P01W0P	855.00	855.00	0.00	optimal	0.16	855.00	0.00	0.00	0.16
P01W65NP	5566.50	5566.50	0.00	optimal	0.11	5566.50	0.00	0.00	0.11
P01W35P	2180.50	2180.50	0.00	optimal	0.11	2180.50	0.00	0.00	0.11
P01W300NP	18 809.00	18 809.00	0.00	optimal	0.09	18 809.00	0.00	0.00	0.09
P01W300P	15 355.50	15 355.50	0.00	optimal	0.12	15 355.50	0.01	0.00	0.10
P02W0NP	6695.00	6695.00	0.00	optimal	1.67	6695.00	0.06	0.00	1.61
P02W0P	14 460.00	14 460.00	0.00	optimal	1.56	14 460.00	0.08	0.00	1.48
P02W250NP	52 048.50	52 048.50	0.00	optimal	1.75	52 048.50	0.06	0.00	1.69
P02W250P	57 104.50	57 104.50	0.00	optimal	1.30	57 104.50	0.17	0.00	1.13
P02W2000NP	288 112.50	288 112.50	0.00	optimal	1.79	288 112.50	0.06	0.00	1.73
P02W2000P	425 078.00	425 078.00	0.00	optimal	1.99	425 078.00	0.06	0.00	1.93
P04W0NP	1419.00	1419.00	0.00	optimal	1.22	1419.00	0.06	0.00	1.16
P04W0P	1313.50	1313.50	0.00	optimal	37.54	1313.50	0.09	0.00	37.45
P04W70P	4202.50	4202.50	0.00	optimal	3.31	4202.50	0.08	0.00	3.24
P04W130NP	6490.50	6490.50	0.00	optimal	0.79	6490.50	0.06	0.00	0.73
P04W1000NP	45 333.00	45 333.00	0.00	optimal	1.06	45 333.00	0.07	0.00	0.99
P04W1000P	58 627.50	58 627.50	0.00	optimal	1.32	58 627.50	0.13	0.00	1.19
E07W0NP	391.00	391.00	0.00	optimal	0.52	391.00	0.00	0.00	0.52
E07W0P	49.50	49.50	0.00	optimal	0.11	49.50	0.00	0.00	0.11
E07W100NP	1882.00	1882.00	0.00	optimal	0.54	1882.00	0.00	0.00	0.54
E07W20P	469.50	469.50	0.00	optimal	0.13	469.50	0.00	0.00	0.13
E07W1000NP	16 222.00	16 222.00	0.00	optimal	0.41	16 222.00	0.00	0.00	0.41
E07W200P	3276.00	3276.00	0.00	optimal	0.21	3276.00	0.00	0.00	0.21
E10W0NP	7542.00	7542.00	0.00	optimal	0.77	7542.00	0.06	0.00	0.71
E10W0P	12 255.50	12 255.50	0.00	optimal	0.75	12 255.50	0.05	0.00	0.70
E10W200NP	46 662.50	46 662.50	0.00	optimal	0.73	46 662.50	0.04	0.00	0.70
E10W200P	45 088.50	45 087.50	1.00	optimal	0.55	45 088.50	0.03	0.00	0.52
E10W1000NP	195 921.00	195 921.00	0.00	optimal	0.57	195 921.00	0.01	0.00	0.56
E10W1000P	200 663.00	200 663.00	0.00	optimal	0.63	200 663.00	0.01	0.00	0.62
E20W0NP	185 987.50	185 987.50	0.00	optimal	1.05	185 987.50	0.05	0.00	1.00
E20W0P	42 582.00	42 582.00	0.00	optimal	2.03	42 582.00	0.05	0.00	1.98
E20W2000NP	1 188 121.00	1 188 121.00	0.00	optimal	0.89	1 188 121.00	0.05	0.00	0.84
E20W400P	195 831.00	195 831.00	0.00	optimal	0.92	195 831.00	0.08	0.00	0.84
E20W10000NP	4 717 132.00	4 717 132.00	0.00	optimal	1.51	4 717 132.00	0.04	0.00	1.47
E20W2000P	704 420.50	704 420.50	0.00	optimal	0.69	704 420.50	0.05	0.00	0.65
hertzr1_np	89 436.50	89 436.50	0.00	optimal	0.10	89 436.50	0.00	0.00	0.10
hertzr1_p	165 216.50	165 216.50	0.00	optimal	0.07	165 216.50	0.00	0.00	0.07
hertzr2_np	118 772.50	118 772.50	0.00	optimal	0.10	118 772.50	0.00	0.00	0.10
hertzr2_p	112 946.50	112 946.50	0.00	optimal	0.14	112 946.50	0.00	0.00	0.14
hertzr3_np	79 245.50	79 245.50	0.00	optimal	0.13	79 245.50	0.00	0.00	0.13
hertzr3_p	84 804.50	84 804.50	0.00	optimal	0.13	84 804.50	0.00	0.00	0.13
hertzr4_np	902 738.00	902 738.00	0.00	optimal	1.42	902 738.00	0.19	0.00	1.23
hertzr4_p	682 282.00	682 282.00	0.00	optimal	4.88	682 282.00	0.10	0.00	4.77
hertzr5_np	103 262.50	103 262.50	0.00	optimal	0.54	103 262.50	0.04	0.00	0.50

(*continued on next page*)

Table 9 (*continued*).

Instance	Exact method					GA		Exact method vs GA	
	UB_Exact	LB_Exact	GAP	Sol_status	T_Exact	UB_GA	T_GA	UB_GAP	T_GAP
hertzr5_p	99 600.50	99 600.50	0.00	optimal	0.46	99 600.50	0.05	0.00	0.42
hertzr6_np	139 532.50	139 532.50	0.00	optimal	0.28	139 532.50	0.03	0.00	0.25
hertzr6_p	128 476.00	128 476.00	0.00	optimal	0.47	128 476.00	0.02	0.00	0.45
hertzr7_np	601 730.00	601 730.00	0.00	optimal	9.33	601 730.00	0.27	0.00	9.06
hertzr7_p	391 570.00	391 570.00	0.00	optimal	13.71	391 570.00	3.12	0.00	10.60
hertzr8_np	194 362.00	194 362.00	0.00	optimal	0.50	194 362.00	0.03	0.00	0.48
hertzr8_p	222 149.50	222 149.50	0.00	optimal	0.47	222 149.50	0.03	0.00	0.43
hertzd10_np	68 847.00	68 847.00	0.00	optimal	153.33	68 847.00	2.87	0.00	150.46
hertzd10_p	66 702.50	66 702.50	0.00	optimal	3048.07	66 702.50	23.16	0.00	3024.91
hertzd11_np	51 172.50	51 172.50	0.00	optimal	118.92	51 172.50	0.28	0.00	118.64
hertzd11_p	38 750.00	38 750.00	0.00	optimal	28.26	38 750.00	1.51	0.00	26.75
hertzg10_np	964.50	964.50	0.00	optimal	0.79	964.50	0.03	0.00	0.76
hertzg10_p	560.50	560.50	0.00	optimal	0.58	560.50	0.04	0.00	0.54
hertzg11_np	1147.00	1147.00	0.00	optimal	1.31	1147.00	0.06	0.00	1.25
hertzg11_p	601.50	601.50	0.00	optimal	2.30	601.50	0.08	0.00	2.22

Table 10
50% density.

Instance	Exact method					GA		Exact method vs GA	
	UB_Exact	LB_Exact	GAP	Sol_status	T_Exact	UB_GA	T_GA	UB_GAP	T_GAP
P01W0NP	1850.50	1850.50	0.00	optimal	0.29	1850.50	0.20	0.00	0.09
P01W0P	1567.50	1567.50	0.00	optimal	0.36	1567.50	0.08	0.00	0.28
P01W65NP	7046.50	7046.50	0.00	optimal	0.31	7046.50	0.05	0.00	0.27
P01W35P	3420.50	3420.50	0.00	optimal	0.41	3420.50	0.04	0.00	0.37
P01W300NP	20 492.50	20 492.50	0.00	optimal	0.31	20 492.50	0.03	0.00	0.27
P01W300P	22 623.50	22 623.50	0.00	optimal	0.27	22 623.50	0.02	0.00	0.25
P02W0NP	22 759.00	22 759.00	0.00	optimal	3425.58	22 759.00	38.76	0.00	3386.81
P02W0P	34 087.00	30 218.11	3868.89	time_limit	3600.00	34 087.00	9.56	0.00	3590.44
P02W250NP	90 402.00	90 393.37	0.00	optimal	832.01	90 402.00	15.43	0.00	816.59
P02W250P	90 787.50	88 511.16	2276.34	time_limit	3601.80	90 787.50	1.23	0.00	3600.57
P02W2000NP	481 899.50	481 899.50	0.00	optimal	219.69	481 899.50	3.28	0.00	216.41
P02W2000P	768 073.00	759 815.92	8257.08	time_limit	3600.00	768 061.00	53.35	12.00	3546.65
P04W0NP	3811.00	3811.00	0.00	optimal	441.76	3811.00	78.01	0.00	363.76
P04W0P	2492.00	2106.67	385.33	time_limit	3600.75	2492.00	178.90	0.00	3421.85
P04W70P	9061.50	9061.50	0.00	optimal	1368.18	9061.50	9.95	0.00	1358.23
P04W130NP	14 779.50	14 779.50	0.00	optimal	797.47	14 779.50	2.46	0.00	795.01
P04W1000NP	84 370.00	84 370.00	0.00	optimal	909.93	84 370.00	16.18	0.00	893.75
P04W1000P	83 371.50	83 371.50	0.00	optimal	194.87	83 371.50	8.49	0.00	186.38
E07W0NP	992.50	992.50	0.00	optimal	0.48	992.50	0.03	0.00	0.45
E07W0P	176.00	176.00	0.00	optimal	0.32	176.00	0.03	0.00	0.29
E07W100NP	2561.00	2561.00	0.00	optimal	0.67	2561.00	0.03	0.00	0.65
E07W20P	758.50	758.50	0.00	optimal	0.47	758.50	0.03	0.00	0.44
E07W1000NP	21 775.50	21 775.50	0.00	optimal	0.31	21 775.50	0.05	0.00	0.26
E07W200P	5202.50	5202.50	0.00	optimal	0.66	5202.50	0.02	0.00	0.64
E10W0NP	20 223.50	20 223.50	0.00	optimal	1.83	20 223.50	0.72	0.00	1.11
E10W0P	22 366.00	22 366.00	0.00	optimal	1.23	22 366.00	0.07	0.00	1.16
E10W200NP	77 038.50	77 038.50	0.00	optimal	1.43	77 038.50	0.08	0.00	1.35
E10W200P	78 039.00	78 039.00	0.00	optimal	1.86	78 039.00	0.08	0.00	1.78
E10W1000NP	257 328.00	257 328.00	0.00	optimal	0.83	257 328.00	0.06	0.00	0.77
E10W1000P	299 568.00	299 568.00	0.00	optimal	1.05	299 568.00	0.07	0.00	0.97
E20W0NP	637 128.00	637 128.00	0.00	optimal	36.19	637 128.00	5.53	0.00	30.66
E20W0P	93 732.50	93 732.50	0.00	optimal	104.73	93 732.50	0.35	0.00	104.39
E20W2000NP	1 963 934.00	1 963 934.00	0.00	optimal	20.79	1 963 934.00	1.45	0.00	19.34
E20W400P	395 670.00	395 670.00	0.00	optimal	130.38	395 670.00	6.62	0.00	123.77
E20W10000NP	5 927 434.50	5 927 434.50	0.00	optimal	4.94	5 927 434.50	0.69	0.00	4.25
E20W2000P	1 280 905.00	1 280 905.00	0.00	optimal	21.46	1 280 905.00	5.84	0.00	15.62
hertzr1_np	176 403.50	176 403.50	0.00	optimal	0.16	176 403.50	0.03	0.00	0.13
hertzr1_p	243 444.00	243 444.00	0.00	optimal	0.24	243 444.00	0.02	0.00	0.21
hertzr2_np	216 845.00	216 845.00	0.00	optimal	0.29	216 845.00	0.03	0.00	0.26
hertzr2_p	125 868.00	125 868.00	0.00	optimal	0.29	125 868.00	0.04	0.00	0.25
hertzr3_np	118 480.00	118 480.00	0.00	optimal	0.27	118 480.00	0.03	0.00	0.24
hertzr3_p	197 336.00	197 336.00	0.00	optimal	0.27	197 336.00	0.03	0.00	0.24
hertzr4_np	1 839 674.50	1 665 032.18	174 642.32	time_limit	3600.00	1 839 217.50	69.28	457.00	3530.72

(*continued on next page*)

Table 10 (*continued*).

Instance	Exact method					GA		Exact method vs GA	
	UB_Exact	LB_Exact	GAP	Sol_status	T_Exact	UB_GA	T_GA	UB_GAP	T_GAP
hertzr4_p	1 808 576.00	1 640 722.89	167 853.11	time_limit	3600.00	1 808 576.00	7.51	0.00	3592.49
hertzr5_np	177 665.00	177 665.00	0.00	optimal	7.06	177 665.00	0.37	0.00	6.69
hertzr5_p	195 347.50	195 347.50	0.00	optimal	22.38	195 347.50	0.59	0.00	21.79
hertzr6_np	231 063.50	231 063.50	0.00	optimal	6.06	231 063.50	0.11	0.00	5.95
hertzr6_p	212 934.50	212 934.50	0.00	optimal	22.11	212 934.50	0.60	0.00	21.51
hertzr7_np	1 492 384.50	1 282 995.93	209 388.57	time_limit	3600.00	1 454 855.50	143.70	37 529.00	3456.30
hertzr7_p	785 191.00	728 382.31	56 808.69	time_limit	3600.00	784 432.00	8.38	759.00	3591.62
hertzr8_np	378 357.50	378 357.50	0.00	optimal	3.56	378 357.50	1.68	0.00	1.88
hertzr8_p	452 583.50	452 559.70	23.80	optimal	13.56	452 583.50	0.49	0.00	13.07
hertzd10_np	153 390.50	141 771.01	11 619.49	time_limit	3600.00	150 122.50	194.59	3268.00	3405.41
hertzd10_p	130 926.00	109 056.40	21 869.60	time_limit	3600.00	128 293.00	31.71	2633.00	3568.29
hertzd11_np	93 778.50	84 999.01	8779.49	time_limit	3600.00	92 028.50	129.60	1750.00	3470.40
hertzd11_p	83 317.50	69 654.01	13 663.49	time_limit	3600.00	80 822.50	8.28	2495.00	3591.72
hertzg10_np	1402.50	1402.41	0.09	optimal	10.28	1402.50	0.68	0.00	9.60
hertzg10_p	795.00	795.00	0.00	optimal	5.88	795.00	0.74	0.00	5.14
hertzg11_np	2219.00	2219.00	0.00	optimal	795.77	2219.00	18.60	0.00	777.17
hertzg11_p	937.00	937.00	0.00	optimal	45.75	937.00	2.08	0.00	43.67

Table 11
75% density.

Instance	Exact method					GA		Exact method vs GA	
	UB_Exact	LB_Exact	GAP	Sol_status	T_Exact	UB_GA	T_GA	UB_GAP	T_GAP
P01W0NP	3982.50	3982.50	0.00	optimal	0.72	3982.50	0.09	0.00	0.63
P01W0P	2108.00	2108.00	0.00	optimal	2.58	2108.00	0.24	0.00	2.34
P01W65NP	9062.00	9062.00	0.00	optimal	0.66	9062.00	0.18	0.00	0.48
P01W35P	4913.00	4913.00	0.00	optimal	0.62	4913.00	0.88	0.00	−0.26
P01W300NP	27 150.50	27 150.50	0.00	optimal	0.38	27 150.50	0.16	0.00	0.22
P01W300P	25 429.50	25 429.50	0.00	optimal	0.53	25 429.50	0.09	0.00	0.45
P02W0NP	40 192.00	33 457.65	6734.35	time_limit	3600.00	38 576.00	203.65	1616.00	3396.35
P02W0P	70 933.00	50 360.53	20 572.47	time_limit	3600.00	65 114.00	37.60	5819.00	3562.40
P02W250NP	148 675.00	137 491.16	11 183.84	time_limit	3600.00	147 343.00	209.53	1332.00	3390.47
P02W250P	153 924.50	137 050.85	16 873.65	time_limit	3613.38	153 844.50	13.52	80.00	3599.85
P02W2000NP	819 375.00	746 983.37	72 391.63	time_limit	3600.00	811 188.00	184.05	8187.00	3415.95
P02W2000P	945 387.00	882 226.07	63 160.93	time_limit	3600.00	941 813.00	52.78	3574.00	3547.22
P04W0NP	8607.50	7642.75	964.75	time_limit	3600.00	8611.50	216.80	−4.00	3383.20
P04W0P	7004.50	5012.23	1992.27	time_limit	3600.00	6746.50	274.18	258.00	3325.82
P04W70P	14 807.00	13 058.98	1748.02	time_limit	3600.00	14 517.00	126.75	290.00	3473.25
P04W130NP	21 063.50	19 370.13	1693.37	time_limit	3600.00	20 769.50	30.96	294.00	3569.04
P04W1000NP	120 884.50	110 944.43	9940.07	time_limit	3600.00	119 843.50	225.75	1041.00	3374.25
P04W1000P	116 538.00	110 282.61	6255.39	time_limit	3600.00	116 517.00	98.86	21.00	3501.14
E07W0NP	1967.50	1967.50	0.00	optimal	0.93	1967.50	0.09	0.00	0.84
E07W0P	302.00	302.00	0.00	optimal	1.39	302.00	0.08	0.00	1.30
E07W100NP	4195.50	4195.50	0.00	optimal	0.97	4195.50	0.07	0.00	0.90
E07W20P	1192.00	1192.00	0.00	optimal	1.17	1192.00	0.08	0.00	1.09
E07W1000NP	33 698.50	33 698.50	0.00	optimal	1.57	33 698.50	0.06	0.00	1.51
E07W200P	7616.00	7616.00	0.00	optimal	2.20	7616.00	0.06	0.00	2.14
E10W0NP	38 212.50	38 212.50	0.00	optimal	3.00	38 212.50	1.28	0.00	1.72
E10W0P	52 045.50	52 045.50	0.00	optimal	20.73	52 045.50	0.44	0.00	20.29
E10W200NP	122 444.50	122 444.50	0.00	optimal	3.89	122 444.50	0.70	0.00	3.19
E10W200P	113 452.00	113 452.00	0.00	optimal	6.45	113 452.00	0.29	0.00	6.17
E10W1000NP	400 306.00	400 306.00	0.00	optimal	2.61	400 306.00	0.47	0.00	2.14
E10W1000P	434 771.50	434 771.50	0.00	optimal	135.56	434 771.50	0.54	0.00	135.01
E20W0NP	1 225 498.00	1 225 498.00	0.00	optimal	514.52	1 230 961.00	207.10	−5463.00	307.42
E20W0P	238 647.00	208 236.84	30 410.16	time_limit	3600.00	229 378.00	151.12	9269.00	3448.88
E20W2000NP	2 766 971.50	2 766 971.50	0.00	optimal	703.00	2 766 971.50	9.70	0.00	693.30
E20W400P	577 061.00	523 024.97	54 036.03	time_limit	3600.00	562 138.00	225.03	14 923.00	3374.97
E20W10000NP	8 529 676.00	8 529 676.00	0.00	optimal	1292.24	8 541 314.00	13.98	−11638.00	1278.26
E20W2000P	1 906 915.50	1 668 043.76	238 871.74	time_limit	3600.00	1 890 231.50	137.49	16 684.00	3462.51
hertzr1_np	280 360.50	280 360.50	0.00	optimal	0.89	280 360.50	0.12	0.00	0.77
hertzr1_p	335 500.50	335 500.50	0.00	optimal	2.37	335 500.50	0.06	0.00	2.31
hertzr2_np	295 413.50	295 413.50	0.00	optimal	2.13	295 413.50	0.24	0.00	1.89
hertzr2_p	234 076.50	234 076.50	0.00	optimal	4.58	234 076.50	1.01	0.00	3.57
hertzr3_np	209 564.00	209 564.00	0.00	optimal	1.76	209 564.00	0.36	0.00	1.40
hertzr3_p	319 300.50	319 300.50	0.00	optimal	36.00	319 300.50	0.15	0.00	35.85
hertzr4_np	3 196 238.00	2 785 577.91	410 660.09	time_limit	3600.00	3 098 112.00	123.97	98 126.00	3476.03
hertzr4_p	3 976 623.50	3 400 065.37	576 558.13	time_limit	3600.00	3 962 092.50	46.90	14 531.00	3553.10
hertzr5_np	296 577.00	296 577.00	0.00	optimal	187.72	296 577.00	19.35	0.00	168.37
hertzr5_p	317 609.00	302 663.09	14 945.91	time_limit	3600.00	317 420.00	92.27	189.00	3507.73
hertzr6_np	307 200.00	307 200.00	0.00	optimal	12.30	307 200.00	3.12	0.00	9.19
hertzr6_p	333 375.00	314 891.06	18 483.94	time_limit	3600.00	333 375.00	0.74	0.00	3599.26

(*continued on next page*)

Table 11 (*continued*).

Instance	Exact method					GA		Exact method vs GA	
	UB_Exact	LB_Exact	GAP	Sol_status	T_Exact	UB_GA	T_GA	UB_GAP	T_GAP
hertzr7_np	2 356 577.00	2 121 828.85	234 748.15	time_limit	3600.00	2 343 967.00	224.62	12 610.00	3375.38
hertzr7_p	1 323 871.50	1 147 417.66	176 453.84	time_limit	3600.00	1 303 384.50	301.36	20 487.00	3298.64
hertzr8_np	623 000.50	623 000.50	0.00	optimal	3107.43	623 052.50	7.42	−52.00	3100.01
hertzr8_p	744 844.50	669 533.81	75 310.69	time_limit	3600.00	744 325.50	48.15	519.00	3551.85
hertzd10_np	232 783.50	181 596.55	51 186.95	time_limit	3600.00	207 615.50	222.28	25 168.00	3377.72
hertzd10_p	222 270.00	175 564.91	46 705.09	time_limit	3600.00	212 145.00	177.91	10 125.00	3422.09
hertzd11_np	159 974.00	133 549.96	26 424.04	time_limit	3600.00	149 599.00	301.03	10 375.00	3298.97
hertzd11_p	130 507.00	89 120.31	41 386.69	time_limit	3600.00	111 977.00	39.49	18 530.00	3560.51
hertzg10_np	1789.00	1789.00	0.00	optimal	99.98	1789.00	24.58	0.00	75.40
hertzg10_p	1291.00	1291.00	0.00	optimal	327.72	1291.00	9.43	0.00	318.29
hertzg11_np	3191.00	2793.06	397.94	time_limit	3600.00	3040.00	45.50	151.00	3554.50
hertzg11_p	1403.00	1403.00	0.00	optimal	3541.55	1414.00	125.17	−11.00	3416.38

Data availability

Data will be made available on request.

References

[1] L. Euler, Solutio problematis ad geometriam situs pertinentis, Comment. Acad. Sci. Imp. Petropolitanae 8 (1736) 128–140.
[2] K. Mei-Ko, Graphic programming using odd or even points, Chin. Math. 1 (1962) 273–277.
[3] J. Edmonds, E. Johnson, Matching, Euler tours and the Chinese postman problem, Math. Program. 5 (1973) 88–124.
[4] C. Papadimitriou, On the complexity of edge traversing, J. ACM 23 (3) (1976) 544–554.
[5] E. Minieka, The Chinese postman problem for mixed networks, Manag. Sci. 25 (7) (1979) 643–648.
[6] C. Orloff, A fundamental problem in vehicle routing, Netw. 4 (1974) 35–64.
[7] J.K. Lenstra, A.H.G. Rinnooy Kan, Complexity of vehicle routing and scheduling problems, Netw. 11 (1981) 221–227.
[8] Á. Corberán, G. Laporte, in: A. Corberán, G. Laporte (Eds.), Arc Routing, SIAM – Society for Industrial and Applied Mathematics, Philadelphia, PA, 2015.
[9] A. Corberán, R. Eglese, G. Hasle, I. Plana, J.M. Sanchis, Arc routing problems: A review of the past, present, and future, Netw. 77 (1) (2020) 88–115.
[10] M.C. Mourão, L.S. Pinto, An updated annotated bibliography on arc routing problems, Netw. 70 (2017) 144–194.
[11] C. Malandraki, M. Daskin, The maximum benefit Chinese postman problem and the maximum benefit traveling salesman problem, European J. Oper. Res. 65 (7) (1993) 218–234.
[12] B. Golden, R. Wong, Capacitated arc routing problems, Netw. 11 (1981) 305–315.
[13] E. Benavent, Á. Corberán, D. Laganá, F. Vocaturro, The periodic rural postman problem with irregular services on mixed graphs, European J. Oper. Res. 276 (2019) 826–839.
[14] M. Reula, R. Martí, Heuristics for the profitable close-enough arc routing problem, Expert Syst. Appl. 230 (2023) 120513.
[15] E.E. Zachariadis, C.D. Tarantilis, C.T. Kiranoudis, The load-dependent vehicle routing problem and its pick-up and delivery extension, Transp. Res. B: Methodol. 71 (2015) 158–181.
[16] Á. Corberán, G. Erdoğan, G. Laporte, I. Plana, J.M. Sanchis, The Chinese postman problem with load-dependent costs, Transp. Sci. 52 (2) (2018) 370–385.
[17] M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, Springer, Cham, Switzerland, 2019.
[18] R. Martí, M. Sevaux, K. Sörensen, 50 years of metaheuristics, To Appear. Eur. J. Oper. Res. (2024).
[19] J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.
[20] D.E. Goldberg, J.H. Holland, Genetic algorithms and machine learning, Mach. Learn. 3 (2) (1988) 195–199.
[21] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[22] W. Rivera, Scalable parallel genetic algorithms, Artif. Intell. Rev. 16 (2001) 153–168.
[23] Z. Konfrst, Parallel genetic algorithms: Advances, computing trends, applications and perspectives, in: International Parallel and Distributed Processing Symposium, Vol. 18, 2004, p. 162.
[24] A.S. Akopov, L.A. Beklaryan, M. Thakur, B.D. Verma, Parallel multi-agent real-coded genetic algorithm for large-scale black-box single-objective optimisation, Knowl.-Based Syst. 174 (2019) 103–122.
[25] W. Zhong, J. Liu, M. Xue, L. Jiao, A multiagent genetic algorithm for global numerical optimization, IEEE Trans. Syst. Man Cybern. B 34 (2) (2004) 1128–1141.
[26] C. Grosan, A. Abraham, Hybrid evolutionary algorithms: Methodologies, architectures, and reviews, in: A. Abraham, C. Grosan, H. Ishibuchi (Eds.), Hybrid Evolutionary Algorithms, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 1–17.
[27] S. Domínguez-Casasola, J.L. González-Velarde, Y.Á. Ríos-Solís, K.A. Reyes-Vega, The capacitated family traveling salesperson problem, Int. Trans. Oper. Res. 31 (2024) 2123–2153.
[28] A. Felipe, M.T. Ortuño, G. Tirado, New neighborhood structures for the double traveling salesman problem with multiple stacks, Top 17 (2009) 190–213.
[29] F. Liu, G. Zeng, Study of genetic algorithm with reinforcement learning to solve the TSP, Expert Syst. Appl. 36 (2009) 6995–7001.
[30] V. Romanuke, Deep clustering of the traveling salesman problem to parallelize its solution, Comput. Oper. Res. 165 (2024) 106548.
[31] A. Felipe, M.T. Ortuño, G. Righini, G. Tirado, A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges, Transp. Res. E: Logist. Transp. Rev. 71 (2014) 111–128.
[32] G.A. Sgarro, L. Grilli, Ant colony optimization for Chinese postman problem, Neural Comput. Appl. 36 (2024) 2901–2920.
[33] L. Shen, X. Xu, F. Shao, H. Shao, Y. Ge, A multi-objective optimization model for medical waste recycling network design under uncertainties, Transp. Res. E: Logist. Transp. Rev. 184 (2024) 103492.
[34] Y. Wang, S. Luo, J. Fan, L. Zhen, The multidepot vehicle routing problem with intelligent recycling prices and transportation resource sharing, Transp. Res. E: Logist. Transp. Rev. 185 (2024) 103503.
[35] P.J.B. Hancock, An empirical comparison of selection methods in evolutionary algorithms, Lecture Notes in Comput. Sci. 865 (1994) 80–94.
[36] C.W. Ahn, R.S. Ramakrishna, Elitism-based compact genetic algorithms, IEEE Trans. Evol. Comput. 7 (4) (2003) 367–385.
[37] N. Christofides, V. Campos, Á. Corberán, E. Mota, An Algorithm for the Rural Postman Problem, Technical Report, Imperial College London ICOR 81.5, 1981.
[38] A. Hertz, G. Laporte, P. Nanchen-Hugo, Improvement procedures for the undirected rural postman problem, INFORMS J. Comput. 11 (1999) 53–62.