Computers & Industrial Engineering 133 (2019) 207–219

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

Solving the open vehicle routing problem with capacity and distance constraints with a biased random key genetic algorithm

Efrain Ruiz[a,*], Valeria Soto-Mendoza[b], Alvaro Ernesto Ruiz Barbosa[c], Ricardo Reyes[d]

[a] *División de Estudios de Posgrado, Instituto Tecnológico de Saltillo, Blvd. Venustiano Carranza 2400, Saltillo, Coahuila 25280, Mexico*
[b] *Research Center on Applied Mathematics, Autonomous University of Coahuila, Campo Redondo S/N, Saltillo, Coahuila CP 25115, Mexico*
[c] *Colegio de Postgraduados, Campus Puebla, Km. 125.5 carretera federal, México-Puebla, Puebla, Puebla 72760, Mexico*
[d] *Departamento de Ingeniería Industrial, Instituto Tecnológico de Saltillo, Blvd. Venustiano Carranza 2400, Saltillo, Coahuila 25280, Mexico*

ARTICLE INFO

Keywords:
Optimization
Combinatorial optimization
Vehicle routing
Open vehicle routing problem
Biased random key genetic algorithm

ABSTRACT

This paper presents a biased random-key genetic algorithm designed to solve the open vehicle routing problem with capacity and distance constraints. Consider a depot from which vehicles depart to deliver goods demanded by clients. Every client is served by one vehicle that is part of a homogeneous fleet. The vehicles initiate their routes at the depot and finish them after servicing the last client without returning to the depot. The problem's objective function is to minimize the total distance traveled by the vehicles while respecting the capacity and maximum distance constraints. The potential applications in real life scenarios make this problem relevant among other vehicle routing problems. Three sets of benchmark problems from the literature are used to test the proposed algorithm. The obtained results show the algorithm's good performance since the best-known solutions for 16 of the 30 instances were improved.

1. Introduction

Supply chain management requires the logistics operations integration inside and outside of an organization to achieve an efficient flow of materials and goods. In some cases, the integration is made within the organization by considering more than one level (Cuda, Guastaroba, & Speranza, 2015). In other cases, the integration considers suppliers and clients outside of the organization (Gharaei & Pasandideh, 2017). The planning horizon has a considerable effect on the number of levels to be included in the supply chain; for long-term planning, usually two or more echelons must be integrated while for daily operations generally one echelon is considered. The fundamental problem of optimizing vehicle route design process arises in the context of these daily operations. This problem, known as the vehicle routing problem (VRP), was first introduced by Dantzig and Ramser (1959) and it is part of the logistics operations in almost any supply chain. VRP's numerous extensions and generalizations consider additional constraints that affect the route design process such as time windows (Solomon, 1987), pickup-and-delivery (Min, 1989), back-hauls (Jordan & Burns, 1984) and open routes (Sariklis & Powell, 2000). For the first three problems vehicles are required to return to the depot after they serve the last client, but that is not the case with the open vehicle routing problem (OVRP) where routes finish after servicing the last client. In fact, the difference between the OVRP and the classic VRP is depicted in Fig. 1. The OVRP, which was first proposed by Sariklis (1997), is relevant for companies without a vehicle fleet of its own that decide to hire or rent vehicles to make their deliveries (Tarantilis, Ioannou, Kiranoudis, & Prastacos, 2005). This problem is also addressed in school transportation route design where buses follow the same route to pick up and return students to designated spots. Moreover, other OVRP applications include an air express courier problem (Schrage, 1981) and the train services planning at the Channel tunnel (Fu, Eglese, & Li, 2005). The potential applications in real life scenarios make this problem relevant and pertinent to be studied.

In this paper, a biased random-key genetic algorithm (BRKGA) is used to solve the OVRP with capacity and maximum route length constraints. A fleet of homogeneous vehicles is considered. The proposed heuristic implements the predecessor assignment decoder proposed by Ruiz, Albareda-Sambola, Fernández, and Resende (2015) and includes a local search phase. The resulting algorithm is tested with three sets of benchmark instances taken from the literature. The algorithm improves the best-known solutions for six of the fourteen instances contained in the first set and two of the eight instances in the second set. In the third set, new upper bounds are established for all instances.

* Corresponding author.
E-mail addresses: eruiz@itsaltillo.edu.mx (E. Ruiz), vsoto@uadec.edu.mx (V. Soto-Mendoza), aruiz@colpos.mx (A.E. Ruiz Barbosa), jreyes@itsaltillo.edu.mx (R. Reyes).

https://doi.org/10.1016/j.cie.2019.05.002
Received 18 August 2018; Received in revised form 11 March 2019; Accepted 2 May 2019
Available online 06 May 2019

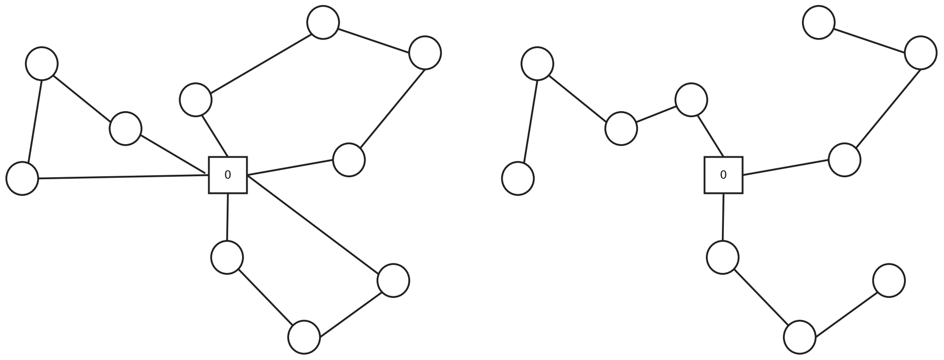

Fig. 1. Comparison between VRP and OVRP routes.

In Section 2 of this paper, a literature review is presented. The OVRP is described in Section 3 and a new formulation for the problem is introduced in Section 4. This is followed by Section 5 where the used algorithm is explained. Section 6 describes implementation details and the corresponding test instances. The computational results are presented in Section 7 and finally concluding remarks are reported in Section 8.

2. Literature review

The OVRP considers capacity and maximum distance constraints over the routes (vehicles), although certain test instances in the literature do not consider the latter constraints. Since the OVRP is of type NP-hard by the reduction of the Hamiltonian path problem (Syslo, Deo, & Kowalik, 1983), almost all previous research works have proposed heuristic methods. Many heuristics have been used to solve the OVRP benchmark instances. Variable neighborhood search was utilized by Fleszar, Osman, and Hindi (2009), and ant colony optimization-based metaheuristics were developed by Li, Tian, and Leung (2009), Brito, Martínez, Moreno, and Verdegay (2015) and Yousefikhoshbakht and Mahmoodi Darani (2019). Tabu search algorithms, where proposed by Fu et al. (2005) and Brandão (2004), while a record-to-record travel algorithm (ORTR) by Li, Golden, and Wasil (2007). Repoussis, Tarantilis, Bräysy, and Ioannou (2010), Marinakis and Marinaki (2012) and Maleki and Yousefikhoshbakht (2019) proposed hybrid evolutionary strategies and Marinakis and Marinaki (2014) proposed a bumblebee mating optimization algorithm. In Zachariadis and Kiranoudis (2010) a wide solutions neighborhood is used and the ILP improvement is the approach chosen by Salari, Toth, and Tramontani (2010). The works by Hosseinabadi, Kardgar, Shojafar, Shamshirband, and Abraham (2016) and Hosseinabadi, Vahidi, Balas, and Mirkamali, 2018 used a gravitational algorithm to solve the benchmark instances. Finally, Dutta, Barma, Kar, and De (2019) presented a genetic algorithm with local search. Recent papers study various OVRP extensions. The OVRP with time windows constraints is studied in Brandão (2018), Schopka and Kopfer (2016) and Niu, Yang, Chen, and Xiao (2018); all of them presenting a heuristic solution approach. The works by Liu, Jiang, and Geng (2014), Lalla-Ruiz, Expósito-Izquierdo, Taheripour, and Voß (2016) and Soto, Sevaux, Rossi, and Reinholz (2017) study the problem's multi-depot extension. Other extensions studied are the multi-objective OVRP (Sánchez-Oro, López-Sánchez, & Colmenar, 2017), the OVRP with cross-docking (Vincent, Jewpanya, & Redi, 2016), and the OVRP with demand uncertainty (Cao, Lai, & Yang, 2014; Kritzinger, Tricoire, Doerner, Hartl, & Stützle, 2017). The location-open vehicle routing problem is studied in Vincent and Lin (2015) and Dutta et al. (2019). The open vehicle routing problem with decoupling points is proposed by Atefi, Salari, Coelho, and Renaud (2018) Finally, the papers by Yu, Ding, and Zhu (2011), Bauer and Lysgaard (2015), Şevkli and Güler (2017) and Zuñiga and Mendoza (2018) consider real-life applications. Despite the problem's difficulty, exact methods have been developed like the branch-and-cut algorithm by Letchford, Lysgaard, and Eglese (2007).

It is important to mention that among the previous research works, the best results for instances without distance constraints are those by Zachariadis and Kiranoudis (2010). For instances including distance constraints the best results are those obtained by Marinakis and Marinaki (2014). Despite the numerous proposed solution algorithms, it is always important to propose new ones to find better solutions for the problem.

3. OVRP description

The OVRP can be defined over a complete graph $G = (V, A)$, in which $V = \{0, 1, \ldots, n\}$ is a set of vertices containing the depot 0 and the clients $V^+ = \{1, \ldots, n\} \subset V$. Every client v_i, $i = 1, \ldots, n$ has an associated demand d_i that must be delivered. Each arc $a = (i, j) \in A$ has an associated cost $c_a > 0$. A feasible solution for the OVRP contains a set of open routes $R \subset A$ with origin at v_0 that satisfy capacity and maximum distance constraints. The cost of such a set of open routes R, is given by

$$c(R) = \sum_{a=(i,j)\in R} c_a.$$

Any open route in the solution contains subsets that can be denoted as subroutes ($R_k \subseteq R$) with origin at vertex $k \in V$. A first client is a vertex k directly connected to the depot 0 in R. A particular case of subroutes includes those with origin at a first client of R, which will be denoted as s-routes. Therefore, an s-route ($R_k \subseteq R$) is a subroute with origin at the first client k. The set of clients that are part of a s-route R_k are denoted as $V(R_k) \subset V$. The total demand of a given s-route R_k is computed using

$$d(R_k) = \sum_{j\in V(R_k)} d_j$$

The total distance of such R_k is computed using

$$l(R_k) = \sum_{a=(i,j)\in V(R_k)} c_a$$

The capacity constraints for the OVRP impose that the total demand of any s-route cannot exceed the capacity parameter $Q > 0$, and the total distance of the route cannot exceed a given value $L > 0$. That is $d(R_k) \leqslant Q$ and $l(R_k) \leqslant L$ for any subroute R_k of R. Therefore,

Definition 1. The OVRP is to find a set of open routes with origin at the depot, satisfying the capacity and distance constraints at the minimum total cost.

4. Problem formulation

The OVRP with distance constraints can be formulated using a set covering formulation similar to the one proposed by Balinski and Quandt (1964) for the VRP. Consider a binary matrix B of dimension $n \times m$ where n is the number of clients and m is the number of possible s-routes. The matrix elements $b_{ik} \in B$ take the values of 0 or 1. When a client i is included in the s-route R_k, $b_{ik} = 1$. Otherwise, $b_{ik} = 0$. Using the previous notation and the one introduced in Section 3, the OVRP with capacity and distance constraints can be formulated using the following variables:

$$r_k = \begin{Bmatrix} \begin{cases} 1, & \text{if } s - \text{route } r_i \text{ is part for the solution}, \quad j, k \in V^+ \\ 0, & \text{otherwise.} \end{cases} \end{Bmatrix}$$

Then a set covering formulation for the OVRP is as follows:

$$\text{Min} \sum_{k \in R} c_{R_k} \tag{1}$$

$$\text{s.t.} \sum_{k \in R} b_{ik} R_k = 1 \qquad j \in V^+ \tag{2}$$

$$d(R_k) \leqslant Q \qquad k \in R \tag{3}$$

$$l(R_k) \leqslant Q \qquad k \in R \tag{4}$$

$$R_k \in \{0, 1\} \qquad k \in R \tag{5}$$

The solution's total cost is given by expression (1) while constraint (2) guarantees that all customers are visited. Capacity and maximum distance are controlled by constraints (3) and (4) respectively.

5. Algorithm's description

The extended use of heuristic and metaheuristic approaches to solve the OVRP is explained by the NP-Hard nature of this combinatorial optimization problem. As was mentioned in Section 2, there are many heuristics and metaheuristics available in the literature. Some of them are based on the evolution of populations like genetic algorithms (Holland, 1962), particle swarm optimization (Eberhart & Kennedy, 1995), swallow swarm optimization (Neshat, Sepidnam, & Sargolzaei, 2013), gene expression programming (Ferreira, 2002), scatter search (Glover, 1998), ant colony optimization (Dorigo & Di Caro, 1999), and rainfall optimization (Kaboli, Selvaraj, & Rahim, 2017), among many others. Others use a construction and improvement scheme, like simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983), tabu search and path relinking (Glover & Laguna, 1998), variable neighborhood search (Mladenović & Hansen, 1997), GRASP (Feo & Resende, 1995) and the gravitational search algorithm (Rashedi, Nezamabadi-Pour, & Saryazdi, 2009). Additionally, there are many other approaches including the hybrid ones not mentioned in this brief review.

The heuristic selection process for solving a given problem is not trivial and affects the results. The context of the problem is relevant in this selection process since no single heuristic performs better than the others for all problems. The decision to use a BRKGA for solving the OVRP was made based on its excellent performance in combinatorial optimization problems (Gonçalves, Resende, & Mendes, 2011; Gonçalves & Resende, 2012; Ruiz et al., 2015; Resende, Toso, Gonçalves, & Silva, 2012).

A BRKGA evolves populations of random key vectors (individuals) that contain fractional values in the interval [0, 1]. A decoder is required to transform such vectors into solutions of the problem and compute their objective function. A BRKGA divides the population into two groups: the elite group containing the individuals with the best fitness function and the non-elite which contains the rest of the population. The crossover operation introduces bias by always selecting an elite individual to mate with either another elite or non-elite individual. Furthermore, bias is also encouraged since the best fitness parent chromosomes have a higher probability of being inherited by the offspring (See Fig. 4). Fig. 2 presents a flow diagram of the proposed heuristic. The first step is to initialize the populations of individuals with random values in the interval [0, 1]. After initialization, all key vectors are decoded and its fitness is computed. The next step is to classify the individuals into elite and non-elite according to its fitness. The crossover starts once the individuals are classified always using one elite parent to procreate the offspring. Finally, a given amount of mutants are introduced in the population to complete the next generation. The mutants are created by generating new random-key vectors. Once the next generation is complete, all individuals are transformed again into solutions (and its fitness computed) using the decoder. The process repeats until the stop criterion is reached.

For a more extensive and detailed description of the BRKGA, please refer to the work by Gonçalves and Resende (2011).

5.1. Predecessor assignment decoder for the OVRP

For solving the OVRP using the BRKGA, the algorithm proposed in this paper uses the predecessor decoder. This decoder was first proposed by Ruiz-y Ruiz (2013). Fig. 3 and Algorithm 1 present respectively the flow diagram and pseudo-code for the predecessor decoder. The first step in the decoding process is to transform the key-vectors into solutions. Due to the decoder's nature, the resulting solutions might be infeasible. Infeasible solutions enter a feasibility recovery phase to become feasible. Then, solutions enter the local search starting with neighborhood $N1$ and continuing with neighborhoods $N2$ and $N3$. A solution enters the local search again only if its fitness was improved at least in one explored neighborhood. The solution leaves the local search when no improvement is found and enters the improvement phase. In the improvement phase solutions are transformed using a modified version of Prim's algorithm with the purpose of finding a better solution. The resulting solutions are encoded and sent back as key-vectors to continue with the evolutionary process. Strategic oscillation is implemented in the decoding procedure, when a given number of generations without improvement is reached. The idea behind this procedure is to help the algorithm escape from local optima. Comprehensive explanations on the decoding procedure, feasibility recovery procedure, local search and improvement phase, and strategic oscillation are found in Sections 5.2, 5.3, 5.4 and 5.5 respectively.

Algorithm 1. Pseudo-code for predecessor decoder.

```
procedure Predecessor-assignment
  Input: $\mathcal{X}, V, E, Q, w, c$
  Output: Assignment array $a$,predecessor vector $p$
2 $a_j \leftarrow 0$, for $j = 1, \dots, n$;
3 $p_j \leftarrow f(l_j, \mathcal{X}_j)$ for $j = 1, \dots, n$;
4 Build $R$ using predecessor vector $p$ ;
5 for $i = 1, \dots, n$ do
6     if $(R_i)$ is infeasible then
7         Execute feasibility recovery procedure $F(R_i)$;
8     end
9     $R' \leftarrow R_i$;
10 end
11 Execute local search $LS(R')$;
12 Execute improvement phase $Imp(R')$;
13 $a \leftarrow R'$;
14 $p \leftarrow R'$;
15 return
```

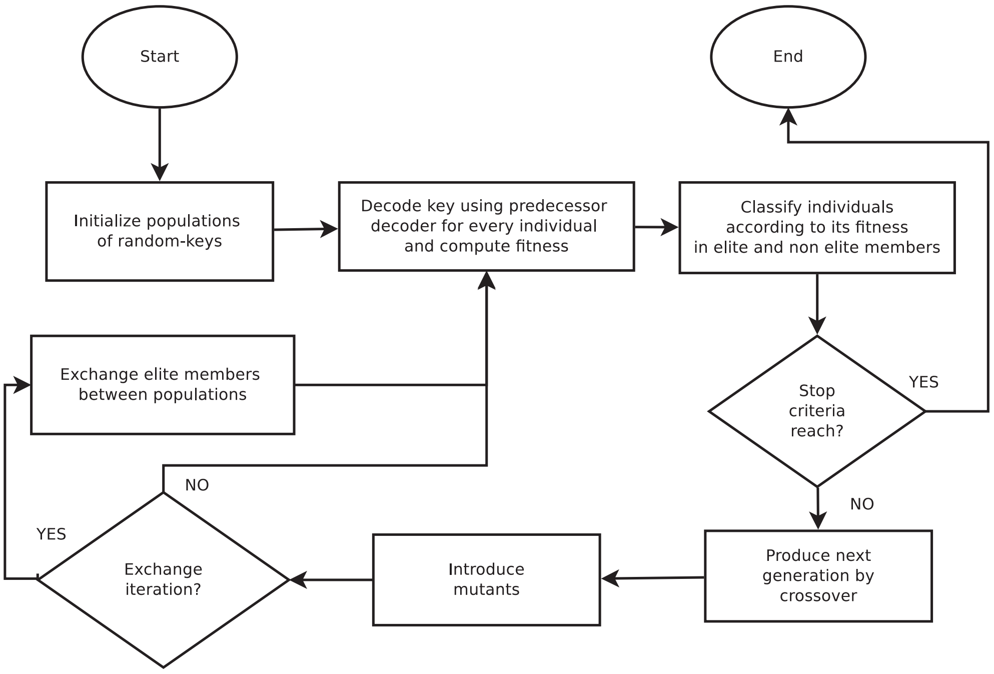
Start
End
Initialize populations of random-keys
Decode key using predecessor decoder for every individual and compute fitness
Classify individuals according to its fitness in elite and non elite members
Exchange elite members between populations
Stop criteria reach?
YES
NO
NO
YES
Exchange iteration?
Introduce mutants
Produce next generation by crossover

Fig. 2. Flow diagram for the BRKGA algorithm.

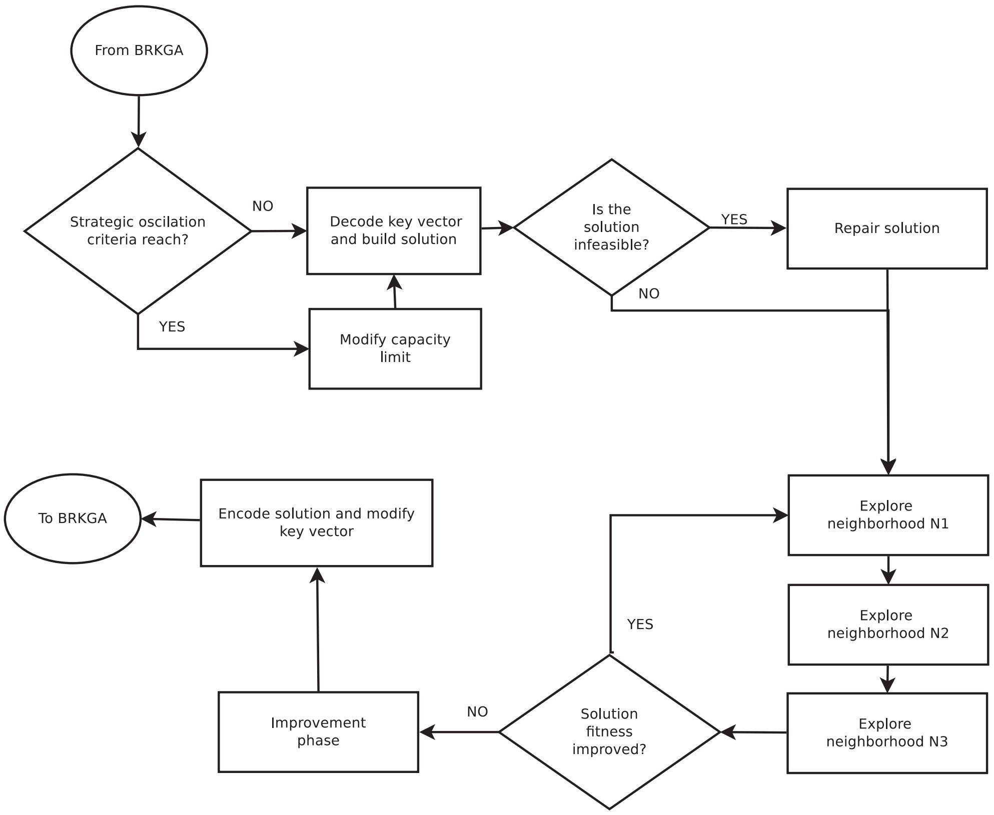
From BRKGA
Strategic oscilation criteria reach?
NO
Decode key vector and build solution
Is the solution infeasible?
YES
Repair solution
NO
YES
Modify capacity limit
To BRKGA
Encode solution and modify key vector
Explore neighborhood N1
Explore neighborhood N2
YES
Improvement phase
NO
Solution fitness improved?
Explore neighborhood N3

Fig. 3. Flow diagram for the predecessor decoder.

5.2. Decoding procedure

The input required by the predecessor assignment decoder is the graph $G = (V, A)$, the random key vector $\mathcal{X}$, the vector of associated demands d, and a predecessor list $l_j \subset V$ for each vertex $j \in V^+$. The random-key vector $\mathcal{X}$ contains $n - 1$ fractional values, which are associated to the $n - 1$ clients of set V^+. The predecessor list l_j includes vertices $i \in V,\ i \neq j$ that fulfill the condition $c(i, j) < c(0, j)$. Note that when $c(i, j) > c(0, j)$, the arc (i, j) will not be part of the optimal solution since connecting j directly to the depot has a lower cost. For every client, $j \in V^+$, the predecessor list l_j and the key vector fractional value $\mathcal{X}_j$ are used to define which client will be visited before the client j. Fig. 4 presents an example of how the codification procedure works. In this example, vertex 2 has a key value of 0.47, which is used along

	Predecessor list for vertices l_j									
Interval	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
[0-0.25)	2	3	1	10	2	4	0	0	0	4
[0.25-0.5)	3	5	2	8	9	10	8	10	7	8
[0.5-0.75)	5	9	5	7	7	8	9	7	5	0
[0.75-1]	9	0	9	0	0	0	5	4	8	7

Vertex	1	2	3	4	5	6	7	8	9	10
Key	0.13	0.47	0.10	0.23	0.36	0.08	0.01	0.17	0.09	0.28
Predecessor	2	5	1	10	9	4	0	0	0	8

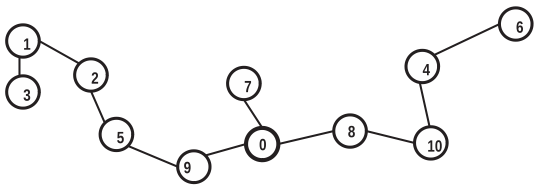

Fig. 4. Predecessor decoder example.

with the predecessor list to define its predecessor. Since 0.47 falls in the second interval, the selected predecessor for vertex 2 is vertex 5.

5.3. Feasibility recovery phase for the predecessor assignment decoder

The predecessor decoder transmits genetic information better when compared to other decoders (Ruiz-y Ruiz, 2013). Unfortunately, it also has some drawbacks like the fact that it can lead to infeasible routes that violate connectivity, capacity or distance constraints. Therefore, a procedure for recovering the solution's feasibility after decoding is required. A route can be infeasible in 3 ways: (i) the route is not connected to the depot and forms a cycle; (ii) the route violates capacity constraints; (iii) instead of a route a tree is obtained (see Fig. 5).

When the constructed s-route s_i forms a cycle, it is repaired by replacing one arc in the cycle with an arc from another s-route s_j that has available capacity. In the case where s_i violates the capacity constraint, a subset $s_k \subset s_i$ is removed and connected to a s-route s_j with available capacity. Note that for both cases, if there is no s-route s_j with available capacity, then s_i is connected to the depot. Finally, for an s-route s_i containing trees, arcs are removed and replaced until a route is obtained. Fig. 6 shows examples of infeasible solutions after recovering feasibility. For all three cases, the substitution of arcs is made considering cost criteria; in other words, the procedure selects the cheapest available arc that can recover the route's feasibility.

5.4. Local search and improvement phase

The BRKGA improves its performance using a local search procedure after decoding.For the local search, three neighborhoods were considered: ($N1$, $N2$, and $N3$). The exploration of such neighborhoods is made sequentially until no improvement is possible. Therefore, a locally optimal solution for all three neighborhoods is obtained. Every neighborhood is explored until no further improvement is found, and then a new neighborhood is explored. The local search finishes when no improvement is accomplished after exploring all three neighborhoods. A brief description for each neighborhood is presented below.

$N1$: involves swapping two vertices in different s-routes, i.e. $i \in V(s\text{-}R_k)$ and $j \in V(s\text{-}R_m)$, with $k \neq m$. Fig. 7 illustrates a move in $N1$.

$N2$: includes reassignments in which vertex $i \in V(s\text{-}R_k)$ is reassigned to another s-route, say $s\text{-}R_m$. Fig. 8 illustrates a move in neighborhood $N2$.

$N3$: is a generalization of $N2$, where a subroute R_{ki} of an s-route $s\text{-}R_k$ is reassigned to a different s-route $s\text{-}R_m$. Route merging is allowed in this neighborhood since any s-route is also a subroute. Fig. 9 illustrates a move in neighborhood $N3$.

After the decoding and local search are performed, an improvement phase is applied to every s-routes in the solution. Since the obtained s-routes are capacity feasible, the set of vertices $S(R_k)$ is used to build

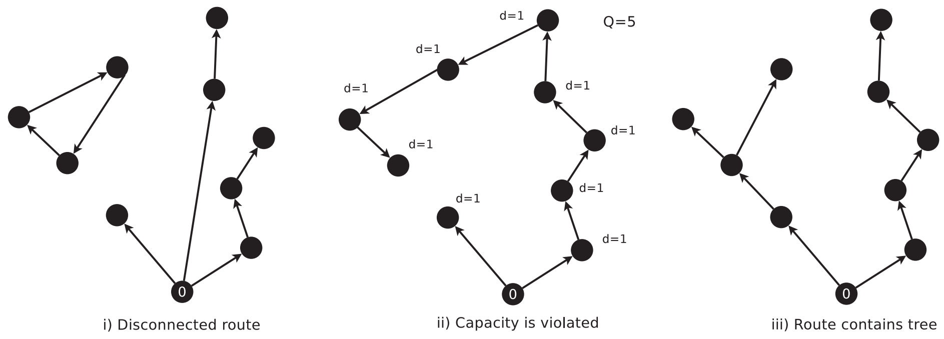

Fig. 5. Infeasible solution examples after decoding.

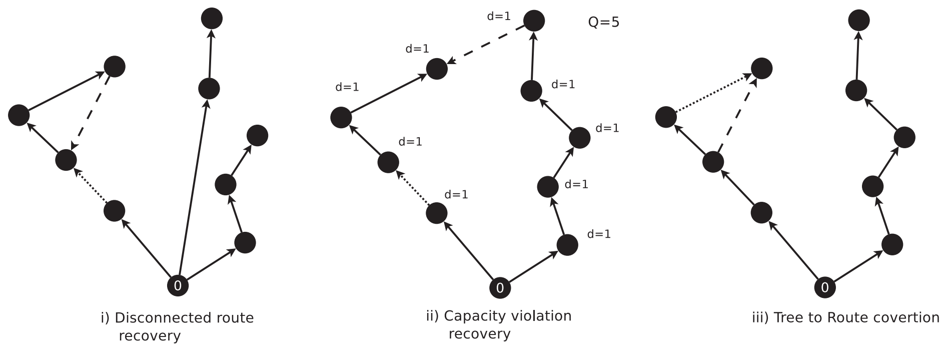
Q=5
d=1
0
i) Disconnected route recovery
ii) Capacity violation recovery
iii) Tree to Route covertion

Fig. 6. Feasibility recovery examples.

new *s*-routes. A modified version of Prim's algorithm (Prim, 1957) for the minimum spanning tree (MST) is employed. Considering that the OVRP can be formulated as a minimum spanning tree with additional constraints, Prim's procedure can also be modified to obtain valid solutions for the OVRP. The modified Prim's algorithm includes capacity, distance, and vertex degree constraints to generate valid solutions for the OVRP. It is important to remember that Prim's algorithm uses two sets of vertices, *connected* and *non-connected*. Notice that there are only two candidates from the *connected* set with respect to the *non-connected* set: the depot and the last vertex that passed from the *non-connected* set to the *connected* set. In this improvement phase, the distance constraints are also considered when connecting a vertex in the *non-connected* set to the current *s*-route, thus maintaining the solution's feasibility.

Since the procedure does not necessarily obtain the optimal solution for the subset of vertices $S(R_k)$, if the previous cost of R_k is less than the cost of R'_k, R_k is not modified. Algorithm 2 shows the pseudo-code for this phase. Set *NotConnected* contains the vertices that have not been connected to the route, and it is initialized in line 3, while set *Connected* is initialized in line 2 with the depot. Set *Solution* is initialized empty at line 4. Edges to be included in the solution are selected in line 6 and *Solution*, *NotConnected* and *Connected* sets are updated in lines 7, 8, and 9–12, respectively. The algorithm finishes when there are no more vertices to connect (line 5).

Algorithm 2. Pseudo-code for modified Prim's algorithm for the OVRP.

```
procedure Improvement-phase
    Input: $R_k, V, E, c$
    Output: Array of edges $Solution$
 2  Initialize set $Connected = \{0\}$;
 3  Initialize set $NotConnected = S(R_k)$;
 4  Initialize set $Solution = \emptyset$;
 5  while $NotConnected \neq \emptyset$ do
 6      Select $minarg\{e = (i, j) : i \in Connected, \quad j \in NotConnected\}$;
 7      $Solution \leftarrow e$;
 8      $NotConnected = NotConnected \setminus \{j\}$;
 9      $Connected = Connected \cup \{j\}$
10      if $i \neq 0$ then
11          $Connected = Connected \setminus \{i\}$
12      end
13  end
14  return
```

5.5. Strategic Oscillation

Strategic Oscillation (SO) is a procedure designed to help algorithms escape from local minima that was first proposed by Glover and Laguna (1997). This procedure helps the local search move between feasible and unfeasible solutions to avoid local optima. In this procedure, the violation of one or more constraints is allowed. Each time a constraint is violated by the solution, the objective function is penalized by using a penalty term β. The penalization for the solution is obtained by multiplying the amount of the violation in the restriction by the penalty term. For the problem, the capacity constraints were relaxed and a maximum amount of capacity violation was set to *MaxQ*. The SO for the final algorithm is the *alternate* proposed by Ruiz et al. (2015). In this approach, the penalty term starts with a value $\beta > 1$which is reduced gradually until β reaches zero. At this point, a value α very close to zero is used to set β to either $+\alpha$ or $-\alpha$. Using a negative value for the penalty term β encourages the algorithm to move into the infeasible area and avoid being trapped at local minima.

6. Instances and implementation details

Three sets of benchmark problems were used to test the algorithm's performance. The first set of instances was introduced by Christofides (1979). In this set we found 14 instances with 50 to 199 vertices, and capacities from 140 to 200. Instances $C1-C5$ and $C11-C12$ from this group do not consider maximum route distance constraints, while $C6-C10$ and $C13-C14$ do. The second set contains large size instances ranging from 200 to 440 vertices, and it was proposed by Li et al. (2007). Capacities range from 550 to 1000. All the instances in this set do not consider maximum distance constraints. The third set was extracted from the work by Golden, Wasil, Kelly, and Chao (1998) and contained 8 large instances. The smallest instance in this set contains 200 vertices and the largest 480, while capacities range from 550 to 1000. In this set, all instances include maximum distance constraints. The reason to include this third set was to test the algorithm with larger instances considering maximum length constraints. In order to compare the algorithm results for this third set, upper and lower bounds were computed using toher algorithms. The cutting plane algorithm proposed by Ruiz-y Ruiz and Ruiz-Barbosa (2018) was employed to obtain the lower bounds (LB) and the upper bounds (UB) were computed using the GRASP algorithm proposed in Ruiz-y Ruiz and Soto-Mendoza (2017). The implementation of the cutting plane algorithm was performed using XPress software.[1]

Table 1 shows the characteristics of all the benchmark instances,-where n specifies the number of vertices (including the depot), Q specifies the capacity of the vehicles, *MaxLength* specifies the maximum distance allowed for the routes, and *Distribution* specifies the customers

[1] http://www.fico.com/en/products/fico-Xpress-solver

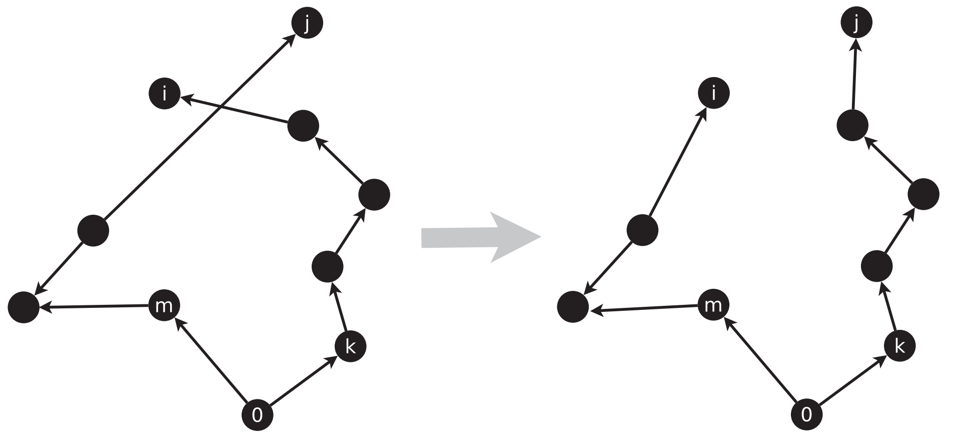

Fig. 7. Neighborhood $N1$, exchange of vertices.

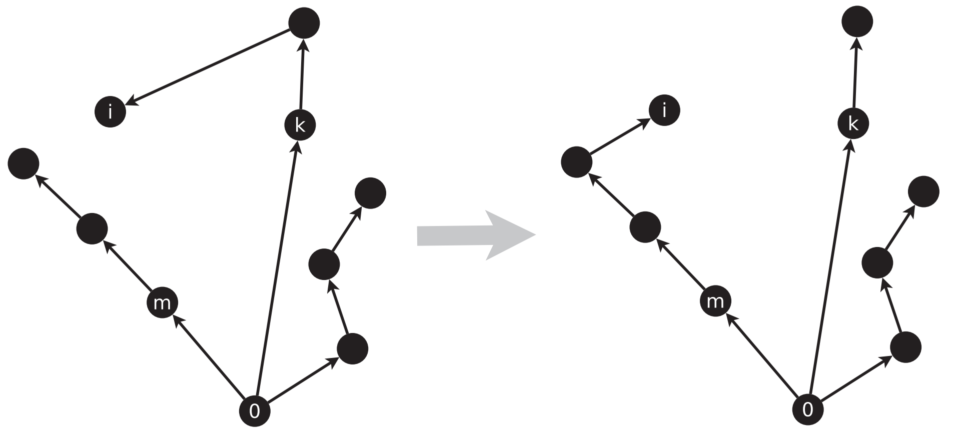

Fig. 8. Neighborhood $N2$, vertex reassignment.

distribution over the plane. Fig. 10 shows the customers distribution for instance $C1$ (randomly distributed) and Fig. 11 shows the customers distribution for instance $O1$ (axially distributed).

6.1. Preliminary experiments

A BRKGA requires four parameters to be specified: the proportion of elite individuals in the population p_e, the proportion of mutants in the population p_m, the elite parents probability to inherit their chromosomes ρ_e, and the number of iterations to exchange the best individual between populations *elitetoEx*. In order to define their values, a 3^k Taguchi design of experiments was employed ($k = 4$). For each factor (parameter) three levels were defined. Table 2 shows the different levels for each factor.

Using instances $C1$, $O1$, $K1$, the experiments were performed and the response variable chosen was the percentage deviation to the best-known solution. Table 3 presents the results for these preliminary experiments. The level with the lowest mean value for each parameter was selected. It is important to remember that the objective is to minimize the cost. Therefore, a lower gap is better. Thus, the selected values were $p_e = 0.25$, $p_m = 0.15$, $\rho_e = 0.65$, and $elitetoEx = 1$.

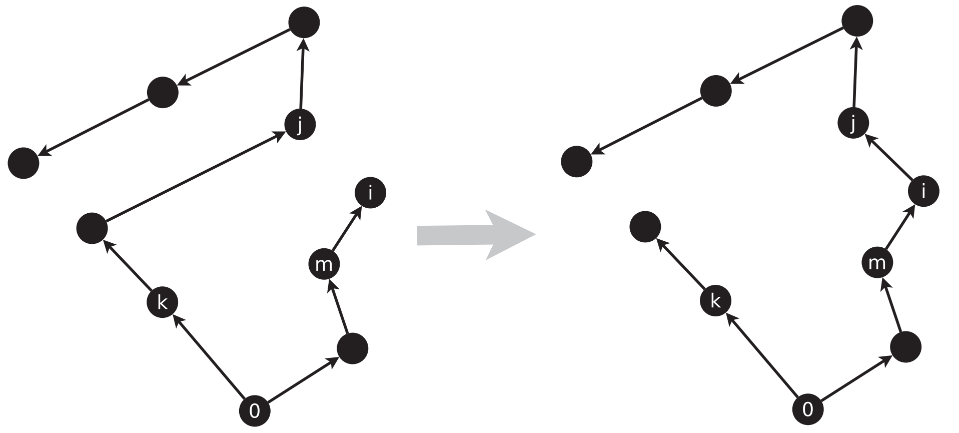

Fig. 9. Neighborhood $N3$, subroute reassignment.

Table 1
Test instances.

Instance	n	Q	MaxLength	Distribution
C1	50	160	∞	Random
C2	75	140	∞	Random
C3	100	200	∞	Random
C4	150	200	∞	Random
C5	199	200	∞	Random
C6	50	160	200	Random
C7	75	140	160	Random
C8	100	200	230	Random
C9	150	200	200	Random
C10	199	200	200	Random
C11	120	200	∞	Random
C12	100	200	∞	Random
C13	120	200	720	Random
C14	100	200	1040	Random
O1	200	900	∞	Axial
O2	240	550	∞	Axial
O3	280	900	∞	Axial
O4	320	700	∞	Axial
O5	360	900	∞	Axial
O6	400	900	∞	Axial
O7	440	900	∞	Axial
O8	400	1000	∞	Axial
K01	240	550	650	Axial
KO2	320	700	900	Axial
KO3	400	900	1200	Axial
KO4	480	1000	1600	Axial
KO5	200	900	1800	Axial
KO6	280	900	1500	Axial
KO7	360	900	1300	Axial
KO8	440	900	1200	Axial

6.2. Implementation details

The BRKGA heuristic for the OVRP was implemented in C++ based on the API developed by Toso and Resende (2014). The used compiler was g++ with the compilation options “-c” and “-O3”. A single thread was used to allow a fair comparison with the results from other authors. A DELL Optiplex running on Linux with an Intel Core2 Duo, 4 GB of RAM, and a CPU clock speed of 3.1 GHz, was used to perform the experiments. The algorithm’s parameters were: $p_e = 0.25n$, $p_m = 0.15$, $\rho_e = 0.65$, $elitetoEx = 1$. Two criteria were used to stop the algorithm: the first was to reach 25 iterations without improving the current best solution; the second was to reach a maximum CPU time of 7200 s.

7. Experimental results

The three previously mentioned sets of instances were used to test the proposed algorithm. The algorithm was run five times for each of the benchmark instances. Table 4 presents statistical information about the obtained results for each instance by the BRKGA. The column labeled *Instance* indicates the instance’s name, *Avg* presents the average value for the objective function after five runs and *Best* gives the objective function for the best solution found after five runs. The column labeled *%gap* presents the percentage deviation from the best solution found by the BRKGA compared to the best-known solution (BKS). It is important to note that a negative value in *%gap* means that the BKS for a given instance was improved. The column labeled *stdv* presents the standard deviation for the obtained solutions and *vc* shows the coefficient of variation. Additionally, *%mdif* shows the difference between the best and worst solution found during the experiments as a percentage of the average fitness value $100\left(\frac{Max - Min}{Avg}\right)$. Finally, *CPU* indicates the average CPU time for each replication and *Gen* presents the average iterations required to find the best solution for each run. Tables 6–8 compare the best solution found by the algorithm with the best solutions found by other algorithms in previous research.

The results in Table 4 shows the algorithm’s good performance. As mentioned before, the negative values in the column labeled *%gap* show that new best solutions for six of the fourteen instances in group *C* were found by the algorithm. The algorithm also found new upper bounds for five of the seven instances in group *C* that include maximum distance constraints. The reason such results were achieved is that the algorithm was specifically designed to consider the route length constraints along with the vehicle capacity ones. For the large size instances in group *O*, the algorithm only improved the best solutions for two of the eight instances. For the *O* set, the algorithm’s performance is not as good as for the *C* set. The most significant percentage deviation for all benchmark problems (1.14), is obtained in the second instance of this group. Observe that the algorithm performs better in the instances of this group with larger capacity values. For the benchmark problems in group *K* the comparison is made with the only available upper bounds. For all these instances the algorithm was able to improve the best-known solutions and therefore all values under column *%gap* are negative.

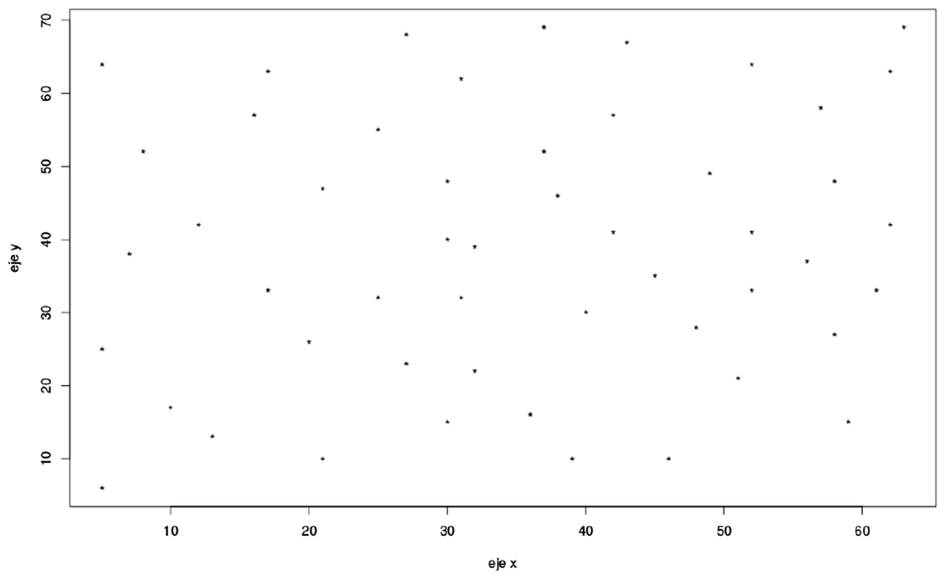

Fig. 10. Random customer distribution instance *C*1.

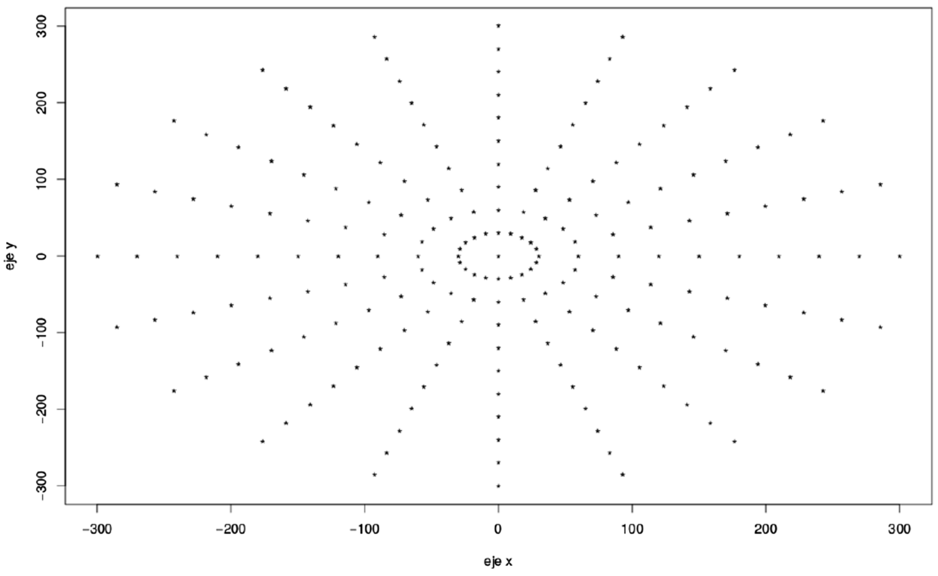

Fig. 11. Axial customer distribution instance $O1$.

Table 2
Levels for the parameters.

Parameter/level	L1	L2	L3
p_e	0.10	0.25	0.40
p_m	0.05	0.10	0.15
ρ_e	0.60	0.65	0.70
elitetoEx	1	10	15

Table 3
Mean response table.

Parameter/level	L1	L2	L3
p_e	1.008	1.000	1.012
p_m	1.005	1.011	1.004
ρ_e	1.010	1.005	1.006
elitetoEx	1.003	1.007	1.010

The results also show that there is an important correlation between the CPU time and the size of the instance. For example, for benchmark problems with fewer than 100 vertices the average CPU time was shorter than 50 s. For problems containing between 100 and 200, vertices the CPU time increased significantly but stayed below 3600 s. For large instances ($n > 200$), the CPU times crossed the 3600-s barrier. In fact, for instances with $n \geqslant 280$, the algorithm stopped after using the maximum CPU time allowed. For smaller benchmark problems, the algorithm stopped after reaching the maximum number of iterations without any improvement. Table 5 presents the computational effort required in each stage of the algorithm. The results show that 82.12% of CPU time was used in the local search phase exploring neighborhoods $N1$, $N2$ and $N3$. The improvement phase used 9.08%, while the feasibility recovery procedure required 5.08%. Finally, the percentage CPU time used in the de-codification/codification procedure was 3.72%. The extensive exploration in the already mentioned neighborhoods explains why the local search uses a lot of CPU time. The CPU time used by the feasibility recovery procedure is due to the numerous infeasible solutions obtained after de-codification. On average, in each iteration, around 63% of the generated solutions are infeasible. Of all the solutions that enter this phase, 96.31% worsen its fitness function, 2.98% improve it, and 0.71%remain unchanged.

The results also show that a small number of iterations lead to the best solution. Since an iteration represents the evolution of a generation, it is fair to say that compared to conventional genetic algorithms, this BRKGA converges faster in terms of generations due to the intensive local search to improve solutions after decoding. Actually, for only five of the 30 test instances, the algorithm required more than 50 iterations on average to find the best solution in each run ($C5$, $C10$, $C11$, $O5$, and $K06$). In general, the BRKGA performs better in instances with maximum distance constraints and any distribution of clients (random or axial). The precision of the algorithm is shown under *stdv*, *vc*, and *%mdif* since they present dispersion measures for the fitness value obtained among the different runs for the benchmark problems. Notice that the standard deviation values increase as the fitness value increases; therefore, it is easier to compare the dispersion among the different benchmark instances using *vc* and *%mdif* columns. As mentioned before, *vc* presents the coefficient of variation, thus removing the noise of the fitness value. It is clear that the dispersion values are higher for instances in group C than in the other two groups. Observe that the highest value (1.96) is found in instance $C11$ and is very different from all the other instances. The results for this instance revealed that the algorithm found an excellent solution on one run but worse solutions on the others. The comparison between the average and best solution values for this instance confirms the previous affirmation. Further analysis uncovered that the improvement phase hardly improved the solutions; in other words, the modified Prim's algorithm did not work properly over this instance. Continuation on the analysis showed that the instance presented a cluster-like distribution of the clients with one cluster close to the depot. This type of distribution explains why the improvement phase offered poor results. Modified Prim's algorithm starts by connecting the closest client to the depot; therefore, the algorithm always tries to connect at least one client from the cluster close to the depot and the rest from other cluster which leads to solutions with worse fitness.

Regarding the relative difference between the best and worst solution (*%mdif*), $C11$ is the benchmark problem with the highest value (5.23%). For 26 of the test instances this value is below 1%, meaning that there is a variation of less than 1% between the best and the worst solutions found. Observe that the exceptions are instances in group C ($C5$, $C10$, $C11$, $C13$) where clients are randomly distributed. In general, based on these results, it is fair to say that the algorithm is precise except in the instances with clients distributed in clusters.

Tables 6–8 compare BRKGA's results with other important algorithms for the OVRP. Numbers in bold are used to show that an algorithm was able to reach the best-known solution. The new best solutions found by the algorithm are presented in the table using bold and

Table 4
Results obtained by the BRKGA for all the test instances.

Instance	n	Q	Avg	Best	%gap	stdv	%stdv	%mdif	CPU	Gen
C1	50	160	412.96	**412.96**	0.00	0.00	0.00	0.00	21.19	21.6
C2	75	140	564.06	**564.06**	0.00	0.00	0.00	0.00	29.73	22.6
C3	100	200	644.36	**639.26**	0.80	0.00	0.00	0.00	158.19	34.4
C4	150	200	735.84	**733.13**	0.00	2.25	0.31	0.59	377.02	39.6
C5	199	200	876.23	871.21	0.25	4.76	0.54	1.42	970.40	62.2
C6	50	160	412.96	**412.96**	0.00	0.00	0.00	0.00	20.79	21.4
C7	75	140	567.59	**567.59**	−0.16	0.00	0.00	0.00	36.31	27.8
C8	100	200	649.38	646.75	0.33	1.67	0.26	0.58	133.34	28.2
C9	150	200	745.38	**743.51**	−1.70	1.76	0.24	0.49	432.09	42.8
C10	199	200	879.68	**873.89**	−0.20	6.35	0.72	1.62	1243.64	72.4
C11	120	200	693.39	**676.40**	−0.32	13.57	1.96	5.23	579.17	110.8
C12	100	200	536.73	536.20	0.37	0.49	0.09	0.17	64.06	23.0
C13	120	200	867.12	**862.74**	−3.77	4.10	0.47	1.14	413.48	44.6
C14	100	200	554.67	**554.67**	−6.29	0.00	0.00	0.00	94.39	24.6
O1	200	900	6020.01	**6003.08**	−0.26	2.37	0.04	0.09	2081.37	21.6
O2	240	550	4626.12	4609.25	1.14	7.99	0.17	0.40	4028.96	22.6
O3	280	900	7735.34	7735.16	0.05	0.25	0.00	0.01	7200.00	34.4
O4	320	700	7297.31	7280.98	0.38	19.58	0.27	0.68	7200.00	39.6
O5	360	900	9186.78	**9178.13**	−0.17	10.15	0.11	0.23	7200.00	62.2
O6	400	900	9861.98	9847.00	0.54	16.40	0.17	0.42	7200.00	21.4
O7	440	900	10508.44	10444.70	0.94	34.32	0.33	0.85	7200.00	27.8
O8	400	1000	12467.04	12449.60	0.28	23.50	0.19	0.46	7200.00	28.2
K01	240	550	4637.86	**4629.21**	−3.36	7.79	0.17	0.41	5917.65	14.8
KO2	320	700	7303.10	**7285.98**	−13.34	12.52	0.17	0.46	7200.00	11.2
KO3	400	900	9879.08	**9837.78**	−12.84	25.13	0.25	0.68	7200.00	9.2
KO4	480	1000	12475.34	**12431.50**	−13.22	35.37	0.28	0.75	7200.00	9.8
KO5	200	900	6006.34	**6004.65**	−5.33	2.23	0.04	0.07	2850.78	7.8
KO6	280	900	7734.46	**7731.21**	−9.66	1.86	0.02	0.06	7200.00	6.0
KO7	360	900	9204.44	**9188.11**	−13.52	12.81	0.14	0.36	7200.00	7.8
KO8	440	900	10527.46	**10488.50**	−13.53	22.74	0.22	0.51	7200.00	9.0

Table 5
CPU time distribution.

Phase	% CPU
Decodification/codification	3.78
Feasibility recovery	5.02
Local search	82.12
Improvement	9.08

underlined formats. Several heuristics and metaheuristics have been developed to solve the OVRP. The most important in terms of solution quality were selected and compared with the BRKGA. Among these algorithms are the bumblebee mating optimization algorithm (BBMO) by citetmarinakis2014bumble; the wide solution neighborhoods metaheuristic (BLSA) by Zachariadis and Kiranoudis (2010); the honey bee mating optimization algorithm (HBMO) by Marinakis and Marinaki (2011); the tabu search algorithms, (TSF) by Fu et al. (2005) and (TSB) by Brandão (2004); the record-to-record travel algorithm (ORTR) by Li et al. (2007); the variable neighborhood search algorithm (MVNS) by Fleszar et al. (2009); hybrid evolution strategy (HES) by Repoussis et al. (2010); the adaptive large neighborhood search (ALNS) by Pisinger and Ropke (2007); the ant colony optimization algorithm (ACO) by Li et al. (2009); the ant colony system (ACS) by Li and Tian (2006);and the hybrid particle swarm optimization algorithm (HPSO) by Marinakis and Marinaki (2012).

Table 6 presents the comparison for instances in group C. As mentioned in Section 6, instances $C1 - -C5$ and $C11 - -C12$, refer to instances with no maximum distance constraints. For this subset of instances, the BLSA found six of the seven best-known solutions, outperforming the other algorithms. Although the BRKGA reaches only five of the seven best-known solutions, it found similar results including a new best-known solution (instance $C11$). The instances $C6 - -C10$ and $C13 - -C14$ include maximum distance constraints; in this case, the BRKGA outperforms the other algorithms by reaching or improving six of the seven best-known solutions. The BRKGA is outperformed by almost all the other algorithms in instance $C8$. The BRKGA was expected to perform better in this subset of instances since it was designed to consider maximum distance constraints in the decoding and local search procedures. Overall, the BRKGA performs better for the instances in group C compared to the other algorithms from previous research.

The results in Table 7 compare the BRKGA's performance with other algorithms from the literature for instances in set O. BRKGA's performance is good, although not the best. It is capable of finding new best-known solutions for instances 01 and 05. The best performing algorithm for this set is the BLSA by Zachariadis and Kiranoudis (2010). Compared with the other algorithms, the BRKGA struggles with instances $O2$ and $O7$. The latter was due to the generation of more routes than required during the feasibility recovery procedure, which for axially distributed instances leads to solutions with higher costs.

Finally, Table 8 compares the results for instances in group K with the upper bounds obtained using the GRASP algorithm proposed by Ruiz-y Ruiz and Soto-Mendoza (2017) and the lower bounds from the cutting plane algorithm by Ruiz-y Ruiz and Ruiz-Barbosa (2018). The values in column *LB* are the lower bounds obtained by the cutting plane algorithm. The column labeled *%gaplb* presents the average percentage deviation between the best solution found by the BRKGA and the lower bounds. The results show that the BRKGA outperforms the GRASP and obtains solutions closer to the lower bounds.

8. Concluding remarks

This paper presented a BRKGA for solving the OVRP with capacity and maximum distance constraints, although some test instances do not include the distance constraints. The proposed BRKGA employs a codification procedure which helps to reduce the problems combinatorics

Table 6
Comparison between BRKGA results and previous research for instances in group C.

Instance	n	Q	BKS	BRKGA	%gap	BBMO	BLSA	HBMO	TSF	ACS
C1	50	160	412.96	**412.96**	0.00	**412.96**	**412.96**	**412.96**	416.06	416.06
C2	75	140	564.06	**564.06**	0.00	**564.06**	**564.06**	**564.06**	567.14	571.7
C3	100	200	639.26	**639.26**	0.80	**639.26**	**639.26**	640.08	643.05	649.02
C4	150	200	733.13	**733.13**	0.00	735.18	**733.13**	738.49	738.94	748.4
C5	199	200	869.00	871.21	0.25	872.15	**869.00**	878.25	878.95	1017.28
C6	50	160	412.96	**412.96**	0.00	**412.96**	**412.96**	**412.96**	**412.96**	–
C7	75	140	568.49	**567.59**	−0.16	568.95	575.25	568.49	568.49	–
C8	100	200	644.63	646.75	0.33	**644.63**	**644.63**	647.26	647.94	–
C9	150	200	756.38	**743.51**	−1.70	757.24	761.41	761.28	764.15	–
C10	199	200	875.67	**873.89**	−0.20	879.38	884.28	903.10	903.10	–
C11	120	200	678.54	**676.40**	−0.32	678.54	678.54	680.15	724.46	685.32
C12	100	200	534.24	536.20	0.37	535.28	**534.24**	536.37	534.71	536.33
C13	120	200	894.19	**862.74**	−3.77	897.10	898.18	922.28	903.82	–
C14	100	200	581.81	**554.67**	−6.29	592.16	593.95	600.66	593.08	–
Instance	n	Q	BKS	ORTR	MVNS	HES	TSB	ALNS	ACO	HPSO
C1	50	160	412.96	416.06	416.06	416.06	416.06	416.06	416.06	416.06
C2	75	140	564.06	567.14	567.14	567.14	574.05	567.14	567.14	567.14
C3	100	200	639.26	639.74	639.74	639.74	641.60	641.76	639.74	639.74
C4	150	200	733.13	**733.13**	**733.13**	**733.13**	740.80	**733.13**	**733.13**	735.29
C5	199	200	869.00	924.96	905.96	894.11	953.40	896.08	977.95	895.79
C6	50	160	412.96	**412.96**	412.96	**412.96**	**412.96**	**412.96**	**412.96**	**412.96**
C7	75	140	568.49	568.49	596.47	584.15	634.54	583.19	568.49	583.19
C8	100	200	644.63	**644.63**	**644.63**	**644.63**	**644.63**	645.16	**644.63**	644.79
C9	150	200	756.38	**756.38**	760.06	764.56	785.20	757.84	758.97	759.81
C10	199	200	875.67	876.02	**875.67**	888.46	884.63	**875.67**	886.08	878.49
C11	120	200	678.54	682.54	682.12	682.12	683.40	682.12	683.11	682.12
C12	100	200	534.24	**534.24**	**534.24**	**534.24**	535.10	**534.24**	**534.24**	535.49
C13	120	200	894.19	896.50	904.04	910.26	943.66	909.80	904.01	904.04
C14	100	200	581.81	591.87	591.87	591.87	597.30	591.87	591.87	592.58

Table 7
Comparison between BRKGA results and previous research for instances in group O.

Instance	n	Q	BKS	BRKGA	%gap	BBMO	BLSA
O1	200	900	6018.52	**6003.08**	−0.26	6021.11	6018.52
O2	240	550	4557.38	4609.25	1.14	4557.38	**4557.38**
O3	280	900	7731.00	7735.16	0.05	7735.14	**7731.00**
O4	320	700	7253.20	7280.98	0.38	7267.18	**7253.20**
O5	360	900	9193.15	**9178.13**	−0.16	9198.25	9193.15
O6	400	900	9793.72	9847.00	0.54	9798.19	**9793.72**
O7	440	900	10347.70	10444.70	0.94	10351.18	**10347.70**
O8	400	1000	12415.36	12449.60	0.28	12418.57	**12415.36**
Instance	n	Q	BKS	HBMO	ORTR	HES	HPSO
O1	200	900	6018.52	6023.48	6018.52	6018.52	6023.25
O2	240	550	4557.38	4561.18	4584.55	4583.70	4557.89
O3	280	900	7731.00	7745.16	7732.85	7733.77	7734.28
O4	320	700	7253.20	7287.49	7291.89	7271.24	7268.23
O5	360	900	9193.15	9201.25	9197.61	9254.15	9201.28
O6	400	900	9793.72	9809.48	9803.80	9821.09	9797.28
O7	440	900	10347.70	10401.24	10374.97	10363.40	10352.29
O8	400	1000	12415.36	12429.57	12429.56	12428.20	12419.25

Table 8
Comparison between BRKGA results and previous research for instances in group K.

Instance	n	Q	GRASP	BRKGA	%gap	LB	%gaplb
K01	240	550	4790.31	**4629.21**	−3.480	4422.69	4.46
KO2	320	700	8407.07	**7285.98**	−15.387	6829.92	6.26
KO3	400	900	11286.40	**9837.78**	−14.725	9248.74	5.99
KO4	480	1000	14325.87	**12431.50**	−15.238	11685.40	6.00
KO5	200	900	6342.66	**6004.65**	−5.629	5510.59	8.23
KO6	280	900	8557.82	**7731.21**	−10.692	7208.15	6.77
KO7	360	900	10624.48	**9188.11**	−15.633	8616.25	6.22
KO8	440	900	12129.07	**10488.50**	−15.642	9779.58	6.76

by using a predecessor list. The algorithm includes a feasibility recovery phase, an improvement phase, and a local search procedure combined with a strategic oscillation. The integration of all these elements was carefully made to obtain good results. Since a BRKGA requires the definition of specific parameters, a design of experiments was used to determine their best values. As already mentioned, the OVRP is promising in the context of route design considering that it is a difficult combinatorial optimization problem of type NP-hard. Despite the problem's complexity, the proposed algorithm was able to find new upper bounds for 16 from a total of 30 benchmark instances. For the first set (C), the algorithm reaches or improves the best-known solution for 11 of the 14 instances with a modest computational effort. For the second group (O), which do not include maximum distance constraints, the algorithm is only able to improve two of the eight best known solutions. Finally, for the third group (K), the algorithm improves all the instances best-known solutions. The algorithm obtained better results with randomly distributed clients and struggled in cases with clients distributed in clusters where the improvement phase was almost unable to find better solutions. The algorithm is designed to perform extensive local search which helps to reduce the number of generations (iterations) required to reach convergence. For this reason, most of the CPU time is spent during this process (82.12%) increasing the required computational effort as well as the instance's size. The latter is due to the scaling of the possible movements within each local search neighborhood. The experiment results show the BRKGA's precision and effectiveness and explain its good performance when compared to the results of previous algorithms taken from the literature. Despite the excellent results in terms of new best-known solutions found, future improvement could be investigated in the CPU time-reduction under a larger instances configuration.

Appendix A. Nomenclature

Avg Average value for the objective function after five runs.

Best The objective function for the best solution found after five

runs.
CPU Average CPU time for each replication.
Distribution Customers' distribution over the plane.
Gen Average iterations in which the best solution was found for each run.
MaxLength Maximum distance allowed for the routes.
n Number of vertices (including the depot).
Q Vehicle's capacity.
%gap Percentage deviation from the best solution found by the BRKGA compared to the BKS.
%gaplb
%stdv Standard deviation as a percentage of the fitness value $100\frac{stdv}{Avg}$.
stdv Standard deviation for the solutions obtained for each instance.
%mdif Difference between the best and worst solution found during the experiments as a percentage of the average fitness value $100\frac{Max-Min}{Avg}$.
ACO Ant colony optimization algorithm.
ACS Ant colony system.
ALNS Adaptive large neighbourhood search.
BBMO Bumble bee mating optimization algorithm.
BKS Best Known Solution.
BLSA Wide solution neighbourhoods metaheuristic.
BRKGA Biased random key genetic algorithm.
GRASP Greedy randomized adaptive search procedure.
HBMO Honey bee mating optimization algorithm.
HES Hybrid evolution strategy.
HPSO Hybrid particle swarm optimization algorithm.
LB Lower bounds.
MST Minimum spanning tree.
MVNS Variable neighborhood search algorithm.
ORTR Record-to-record travel algorithm.
OVRP Open vehicle routing problem.
SO Strategic oscillation.
TSB Tabu search algorithm by citetBRANDAO2004552.
TSF Tabu search algorithm by Fu et al. (2005).
UB Upper bounds.
VRP Vehicle routing problem.

References

Atefi, R., Salari, M., Coelho, L. C., & Renaud, J. (2018). The open vehicle routing problem with decoupling points. *European Journal of Operational Research, 265*(1), 316–327.

Balinski, M. L., & Quandt, R. E. (1964). On an integer program for a delivery problem. *Operations Research, 12*(2), 300–304.

Bauer, J., & Lysgaard, J. (2015). The offshore wind farm array cable layout problem: a planar open vehicle routing problem. *Journal of the Operational Research Society, 66*(3), 360–368.

Brandão, J. (2004). A tabu search algorithm for the open vehicle routing problem. *European Journal of Operational Research, 157*(3), 552–564. < http://www.sciencedirect.com/science/article/pii/S0377221703002388 > .

Brandão, J. (2018). Iterated local search algorithm with ejection chains for the open vehicle routing problem with time windows. *Computers & Industrial Engineering, 120*, 146–159.

Brito, J., Martínez, F. J., Moreno, J., & Verdegay, J. L. (2015). An aco hybrid metaheuristic for close–open vehicle routing problems with time windows and fuzzy constraints. *Applied Soft Computing, 32*, 154–163.

Cao, E., Lai, M., & Yang, H. (2014). Open vehicle routing problem with demand uncertainty and its robust strategies. *Expert Systems with Applications, 41*(7), 3569–3575.

Christofides, N. (1979). Combinatorial optimization. In Nicos Christofides (Vol. Ed.), *A Wiley-Interscience publication, based on a series of lectures, given at the summer school in combinatorial optimization, held in Sogesta, Italy, May 30th–June 11th, 1977: Vol. 1*. Chichester: Wiley.

Cuda, R., Guastaroba, G., & Speranza, M. G. (2015). A survey on two-echelon routing problems. *Computers & Operations Research, 55*, 185–199.

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science, 6*(1), 80–91.

Dorigo, M., & Di Caro, G. (1999). Ant colony optimization: A new meta-heuristic. *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406). Vol. 2. Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)* (pp. 1470–1477). IEEE.

Dutta, J., Barma, P. S., Kar, S., & De, T. (2019). A modified kruskal's algorithm to improve genetic search for open vehicle routing problem. *International Journal of Business Analytics (IJBAN), 6*(1), 55–76.

Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory. *Proceedings of the sixth international symposium on Micro Machine and Human Science, 1995. MHS'95* (pp. 39–43). IEEE.

Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization, 6*(2), 109–133.

Ferreira, C. (2002). Gene expression programming in problem solving. *Soft computing and industry* (pp. 635–653). Springer.

Fleszar, K., Osman, I. H., & Hindi, K. S. (2009). A variable neighbourhood search algorithm for the open vehicle routing problem. *European Journal of Operational Research, 195*(3), 803–809.

Fu, Z., Eglese, R., & Li, L. Y. (2005). A new tabu search heuristic for the open vehicle routing problem. *Journal of the Operational Research Society, 56*(3), 267–274.

Gharaei, A., & Pasandideh, S. H. R. (2017). Four-echelon integrated supply chain model with stochastic constraints under shortage condition. *Industrial Engineering & Management Systems, 16*(3), 316–329.

Glover, F. (1998). A template for scatter search and path relinking. *Lecture Notes in Computer Science, 1363*, 13–54.

Glover, F., & Laguna, M. (1997). *Tabu search.* Kluwer.

Glover, F., & Laguna, M. (1998). Tabu search. *Handbook of combinatorial optimization* (pp. 2093–2229). Springer.

Golden, B. L., Wasil, E. A., Kelly, James P., & Chao, I.-M. (1998). *The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets, and computational results.* US, Boston, MA: Springer33–56 doi: https://doi.org/10.1007/978-1-4615-5755-5_2.

Gonçalves, J., & Resende, M. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics, 17*, 487–525.

Gonçalves, J. F., & Resende, M. G. (2012). A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research, 39*(2), 179–190.

Gonçalves, J. F., Resende, M. G., & Mendes, J. J. (2011). A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics, 17*(5), 467–486.

Holland, J. H. (1962). Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM), 9*(3), 297–314.

Hosseinabadi, A. A. R., Kardgar, M., Shojafar, M., Shamshirband, S., & Abraham, A. (2016). Gravitational search algorithm to solve open vehicle routing problem. *Innovations in bio-inspired computing and applications* (pp. 93–103). Springer.

Hosseinabadi, A. A. R., Vahidi, J., Balas, V. E., & Mirkamali, S. S. (2018). Ovrp_gels: Solving open vehicle routing problem using the gravitational emulation local search algorithm. *Neural Computing and Applications, 29*(10), 955–968.

Jordan, W. C., & Burns, L. D. (1984). Truck backhauling on two terminal networks. *Transportation Research Part B: Methodological, 18*(6), 487–503.

Kaboli, S. H. A., Selvaraj, J., & Rahim, N. (2017). Rain-fall optimization algorithm: A population based algorithm for solving constrained optimization problems. *Journal of Computational Science, 19*, 31–42.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

Kritzinger, S., Tricoire, F., Doerner, K. F., Hartl, R. F., & Stützle, T. (2017). A unified framework for routing problems with a fixed fleet size. *International Journal of Metaheuristics, 6*(3), 160–209.

Lalla-Ruiz, E., Expósito-Izquierdo, C., Taheripour, S., & Voß, S. (2016). An improved formulation for the multi-depot open vehicle routing problem. *OR Spectrum, 38*(1), 175–187.

Letchford, A. N., Lysgaard, J., & Eglese, R. W. (2007). A branch-and-cut algorithm for the capacitated open vehicle routing problem. *Journal of the Operational Research Society, 58*(12), 1642–1651.

Li, F., Golden, B., & Wasil, E. (2007). The open vehicle routing problem: Algorithms, large-scale test problems, and computational results. *Computers & Operations Research, 34*(10), 2918–2930.

Li, X., & Tian, P. (2006). An ant colony system for the open vehicle routing problem. *International workshop on ant colony optimization and swarm intelligence* (pp. 356–363). Springer.

Li, X.-Y., Tian, P., & Leung, S. C. H. (2009). An ant colony optimization metaheuristic hybridized with tabu search for open vehicle routing problems. *Journal of the Operational Research Society, 60*(7), 1012–1025. https://doi.org/10.1057/palgrave.jors.2602644.

Liu, R., Jiang, Z., & Geng, N. (2014). A hybrid genetic algorithm for the multi-depot open vehicle routing problem. *OR Spectrum, 36*(2), 401–421.

Maleki, F., & Yousefikhoshbakht, M. (2019). A hybrid algorithm for the open vehicle routing problem. *Iran University of Science & Technology, 9*(2), 355–371.

Marinakis, Y., & Marinaki, M. (2011). A honey bees mating optimization algorithm for the open vehicle routing problem. *2011 Genetic and evolutionary computation conference*. Taylor and Francis.

Marinakis, Y., & Marinaki, M. (2012). A hybrid particle swarm optimization algorithm for the open vehicle routing problem. *International conference on swarm intelligence* (pp. 180–187). Springer.

Marinakis, Y., & Marinaki, M. (2014). A bumble bees mating optimization algorithm for the open vehicle routing problem. *Swarm and Evolutionary Computation, 15*, 80–94.

Min, H. (1989). The multiple vehicle routing problem with simultaneous delivery and pick-up points. *Transportation Research Part A: General, 23*(5), 377–386.

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

Neshat, M., Sepidnam, G., & Sargolzaei, M. (2013). Swallow swarm optimization algorithm: A new method to optimization. *Neural Computing and Applications, 23*(2), 429–454.

Niu, Y., Yang, Z., Chen, P., & Xiao, J. (2018). Optimizing the green open vehicle routing problem with time windows by minimizing comprehensive routing cost. *Journal of Cleaner Production, 171*, 962–971.

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research, 34*(8), 2403–2435.

Prim, R. (1957). Shortest connection matrix network and some generalizations. *Bell System Technical Journal, 36*, 1389–1401.

Rashedi, E., Nezamabadi-Pour, H., & Saryazdi, S. (2009). Gsa: a gravitational search algorithm. *Information Sciences, 179*(13), 2232–2248.

Repoussis, P. P., Tarantilis, C. D., Bräysy, O., & Ioannou, G. (2010). A hybrid evolution strategy for the open vehicle routing problem. *Computers & Operations Research, 37*(3), 443–455.

Resende, M. G., Toso, R. F., Gonçalves, J. F., & Silva, R. M. (2012). A biased random-key genetic algorithm for the steiner triple covering problem. *Optimization Letters, 6*(4), 605–619.

Ruiz, E., Albareda-Sambola, M., Fernández, E., & Resende, M. G. (2015). A biased random-key genetic algorithm for the capacitated minimum spanning tree problem. *Computers & Operations Research, 57*, 95–108.

Ruiz-y Ruiz, H. E. (2013). *The capacitated minimum spanning tree problem. Ph.D. thesis.* Universitat Politècnica de Catalunya.

Ruiz-y Ruiz, E., & Ruiz-Barbosa, A. (2018). *A cutting plane algorithm for the open vehicle routing problem with capacity and distance constraints. Tech. rep.* Instituto Tecnológico de Saltillo.

Ruiz-y Ruiz, E., & Soto-Mendoza, V. (2017). *A grasp algorithm for the open vehicle routing problem. Tech. rep.* Instituto Tecnológico de Saltillo.

Salari, M., Toth, P., & Tramontani, A. (2010). An ilp improvement procedure for the open vehicle routing problem. *Computers & Operations Research, 37*(12), 2106–2120.

Sánchez-Oro, J., López-Sánchez, A. D., & Colmenar, J. M. (2017). A general variable neighborhood search for solving the multi-objective open vehicle routing problem. *Journal of Heuristics,* 1–30.

Sariklis, D. (1997). *Open vehicle routing problem: Description, formulations and heuristic methods. Ph.D. thesis.* London School of Economics and Political Science (University of London).

Sariklis, D., & Powell, S. (2000). A heuristic method for the open vehicle routing problem. *Journal of the Operational Research Society, 51*(5), 564–573.

Schopka, K., & Kopfer, H. (2016). An adaptive large neighborhood search for the reverse open vehicle routing problem with time windows. *Logistics management* (pp. 243–257). Springer.

Schrage, L. (1981). Formulation and structure of more complex/realistic routing and scheduling problems. *Networks, 11*(2), 229–232.

Şevkli, A. Z., & Güler, B. (2017). A multi-phase oscillated variable neighbourhood search algorithm for a real-world open vehicle routing problem. *Applied Soft Computing, 58*, 128–144.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research, 35*(2), 254–265.

Soto, M., Sevaux, M., Rossi, A., & Reinholz, A. (2017). Multiple neighborhood search, tabu search and ejection chains for the multi-depot open vehicle routing problem. *Computers & Industrial Engineering, 107*, 211–222.

Syslo, M., Deo, N., & Kowalik, J. (1983). *Discrete optimization algorithms with Pascal programs.* Prentice Hall.

Tarantilis, C. D., Ioannou, G., Kiranoudis, C. T., & Prastacos, G. P. (2005). Solving the open vehicle routing problem via a single parameter metaheuristic algorithm. *Journal of the Operational Research Society, 56*(5), 588–596.

Toso, R. F., & Resende, M. G. C. (2015). A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software, 30*(1), 81–93.

Vincent, F. Y., Jewpanya, P., & Redi, A. P. (2016). Open vehicle routing problem with cross-docking. *Computers & Industrial Engineering, 94*, 6–17.

Vincent, F. Y., & Lin, S.-Y. (2015). A simulated annealing heuristic for the open location-routing problem. *Computers & Operations Research, 62*, 184–196.

Yousefikhoshbakht, M., & Mahmoodi Darani, N. (2019). A combined metaheuristic algorithm for the vehicle routing problem and its open version. *Journal of AI and Data Mining, 7*(1), 169–179.

Yu, S., Ding, C., & Zhu, K. (2011). A hybrid ga–ts algorithm for open vehicle routing optimization of coal mines material. *Expert Systems with Applications, 38*(8), 10568–10573.

Zachariadis, E. E., & Kiranoudis, C. T. (2010). An open vehicle routing problem metaheuristic for examining wide solution neighborhoods. *Computers & Operations Research, 37*(4), 712–723.

Zuñiga, B. C., & Mendoza, A. M. (2018). Propuesta de un modelo de ruteo de vehículos abierto en una institución prestadora de servicios de salud. *Entramado, 14*(2), 288–298.