Memetic Comp. (2012) 4:89–108
DOI 10.1007/s12293-011-0070-y

REGULAR RESEARCH PAPER

Memetic algorithm for the Traveling Car Renter Problem: an experimental investigation

Marco César Goldbarg · Paulo Henrique Asconavieta · Elizabeth Ferreira Gouvêa Goldbarg

Received: 30 September 2010 / Accepted: 27 August 2011 / Published online: 16 September 2011

Abstract The Traveling Car Renter Problem (CaRS) is a generalization of the Traveling Salesman Problem where the tour can be decomposed into contiguous paths that are travelled by different rented cars. When a car is rented in a city and delivered in another, the renter must pay a fee to return the car to its home city. Given a graph G and cost matrices associated to cars available for rent, the problem consists in determining the minimum cost Hamiltonian cycle in G, considering also the cost paid to deliver a car in a city different from the one it was rented. The latter cost is added to the cost of the edges in the cycle. This paper describes the general problem and some related variants. Two metaheuristic approaches are proposed to deal with CaRS: GRASP hybridized with Variable Neighborhood Descent and Memetic Algorithm. A set of benchmark instances is proposed for the new problem which is utilized on the computational experiments. The algorithms are tested on a set of 40 Euclidean and non-Euclidean instances.

Keywords Traveling Car Renter Problem · Memetic algorithm · GRASP · Variable neighborhood search

1 Introduction

The Traveling Salesman Problem (TSP) is a classic Combinatorial Optimization problem which consists in finding the minimum length Hamiltonian cycle in an edge-weighted graph $G = (N, M)$, $N = \{1, \ldots, n\}$ is the set of nodes, $M = \{1, \ldots, m\}$ is the set of edges, where cost c_{ij} is associated with the edge that connects vertices i and j. The TSP is NP-hard [11] and one of the most intensively investigated problems in Combinatorial Optimization. The TSP has several important practical applications and a number of variants [12], such as the peripatetic salesman [15], the M-tour TSP [29], the Colorful TSP [31] and the Robust TSP [21], the fuzzy transport traveling salesman problem [10], among others. This paper introduces a new problem, named Traveling Car Renter Problem (CaRS), which generalizes the TSP. As stated previously, the aim of the TSP is to find a Hamiltonian cycle in a graph G with minimal cost among all Hamiltonian cycles in G. The primary application of TSP is on a set of cities represented by the node set of G where the roads between these cities are represented by the set of edges of G and the costs to travel between each pair of cities i and j are represented in elements of square matrix C with order n. The costs in matrix C can be associated to expenses involving the use of a given vehicle that is used to do the journey between each pair of cities. However, in the case where a user rents cars, it may be more advantageous to use different cars in different parts during the travel among the n cities. A different cost can be associated to each car to do the journey between each pair of cities. Thus, in CaRS, the number of cost matrices associated to the graph corresponds to the number of different cars that can be used during the journey among the n cities. In addition to expenses associated to each car, other costs may be considered, such as the fee that has to be paid to deliver a car in a city different from the one it was rented. These fees are also expressed on a square matrix, one for each car, in which an element in line i and column j represents the fee that has to be paid to deliver the corresponding car in city j once it was rented in city i. CaRS

M. C. Goldbarg · P. H. Asconavieta · E. F. G. Goldbarg (✉)
Universidade Federal do Rio Grande do Norte, Natal, Brazil
e-mail: beth@dimap.ufrn.br

M. C. Goldbarg
e-mail: gold@dimap.ufrn.br

P. H. Asconavieta
Instituto Federal de Educação, Ciência e Tecnologia Sul-rio-grandense, Pelotas, Brazil
e-mail: asconavietaph@hotmail.com

models important applications in tourism and transportation areas, being a complex TSP variant that challenges the state-of-art. In this paper the new problem and some of its versions are presented, its complexity is discussed and some related problems are briefly overviewed.

Metaheuristic techniques are often successfully applied to complex problems. In this paper, two metaheuristics are applied to CaRS: Memetic Algorithms [16,17,22,23] and GRASP [9,27,28] hybridized with Variable Neighborhood Search [20]. Two algorithms of each class are developed for the new problem. A first computational experiment compares each pair of algorithms developed in accordance with the same metaheuristic technique. Then, the best algorithms, one from each class, are compared on another experiment.

Since CaRS is a new problem, there are no benchmark instances available in the literature to test the performance of the proposed methods. Therefore, in order to carry out computational experiments, a set of instances is introduced for the new problem, named CaRSLIB. This set contains Euclidean and non-Euclidean symmetric instances with number of cities ranging from 14 to 300 and number of cars between 2 and 5. A set of 40 instances is used in the computational experiments reported in this paper.

CaRS problem and several of its variants are introduced in Sect. 2. Section 3 presents the metaheuristic methods proposed for the investigated problem. Section 4 presents the results of computational experiments comparing the performance of the proposed approaches. Statistical tests are applied to support conclusions on the behavior of the proposed algorithms. According to those results, the Memetic Algorithm is pointed as the best approach for CaRS on the set of tested instances. Final conclusions and future directions for the research of algorithms for CaRS and its variations are addressed in Sect. 5.

2 The Traveling Car Renter Problem

The car rental industry is a multi-billion dollar sector of several countries economy [33]. In 2007, for example, the US segment of the car rental industry presented averages about $18.5 billion in revenue a year with approximately 1.9 million rental vehicles [19]. Today over 90 significant economic size car rental companies exist in the world market [5]. The importance of the car rental business can be measured by the enterprise turnover and by the size of companies that provide the service. In 2006, the Enterprise Holdings Inc. which owns today the National Car Rental, Alamo Rent A Car and WeCar earned about 9.04 billion dollars [7]. These numbers represent only part of the market that also has other major car rental networks such as Dollar and Hertz. The world market in 2012 is estimated at 52.6 billion dollars [4]. Besides being itself a major business, expenses on car rentals may represent a significant portion of the activities involving tourism and business. Currently the rental options are becoming increasingly diversified with the expansion of the companies, justifying the search for rent schemes that minimize the total cost of this transportation.

Among the various logistic problems of this branch of activity, the literature describes specific studies of combinatorial optimization in fleet assignment [18], the strategic and tactical fleet planning [24], demand forecast [8], car fleet management with maintenance constraints [13] and pool segmentation in logistics management process for large-scale car rental business [33]. Logistic problems in the car rental industry are reviewed by [32]. Although the car rental industry viewpoint has been extensively focused on those papers, the customer's point of view has not yet been the subject of published research. Usually, under the viewpoint of car rental users, the goal is to minimize the costs to move from a starting point to a destination. On the other hand, when someone rents a car, it is assumed that requirements of comfort and safety are met. The car rental user must add to the costs paid to the rental car company, at least two other costs: fuel and tolls payment.

Let $G = (N, M, W)$ be a graph where $N = \{1, \ldots, n\}$ represents the set of vertices, $M = \{1, \ldots, m\}$ is the set of edges and $W = \{1, \ldots, w\}$ is the set of distances between the vertices associated with each edge of M. The problem described in this paper has the following features:

1. Several types of cars are available for rent, each of them with its own specific operational costs which include rent cost, fuel consumption and tolls payment. The latter fee may depend on the car type and on the specific roads which are travelled. The value paid to the rental company can also be associated with a cost per kilometer. Thus, without loss of generality, these costs can be considered as a function of each car on a value associated to the edges (i, j) of graph G. The operational cost of car k to traverse edge (i, j) is denoted by c_{ij}^k in this paper.
2. A car rented in a given company can only be returned in a city where there is an agency of the same company. It is therefore not allowed to rent a car from a given company to travel a certain segment of the route, if that car cannot be returned in the last city, that is, there is not an agency of the same company in the last city of the travelled segment.
3. Whenever it is possible to rent a car in city i and return it in city j, $i \neq j$, there is an extra for returning the car to city i, its home city. Variable d_{ij}^k represents the fee to return car k to city j when k was rented in city i, $i \neq j$.
4. The tour begins and ends in the city where the first car is rented. This city is called *starting city* or CaRS base.

5. The return cost is null in case the tour is completed with a single car which is delivered in the same place it was rented. This case corresponds to the classic TSP considering the operational costs associated only with the rented car.
6. Cars with same characteristics rented in a single company can be hired under different costs depending on the city they were rented and on contract negotiation conditions. Therefore, without loss of generality, the rent designation can be efficiently controlled by decisions related to cars, not considering companies. The set $K = \{1, \ldots, ncars\}$, $|K| = ncars$, is the set of different cars that can be in the solution.
7. The cost to return a rented car to its home city may be strictly associated with the path between the city where the car is delivered and the city where the car was rented. This cost can also be a result of independent calculation.

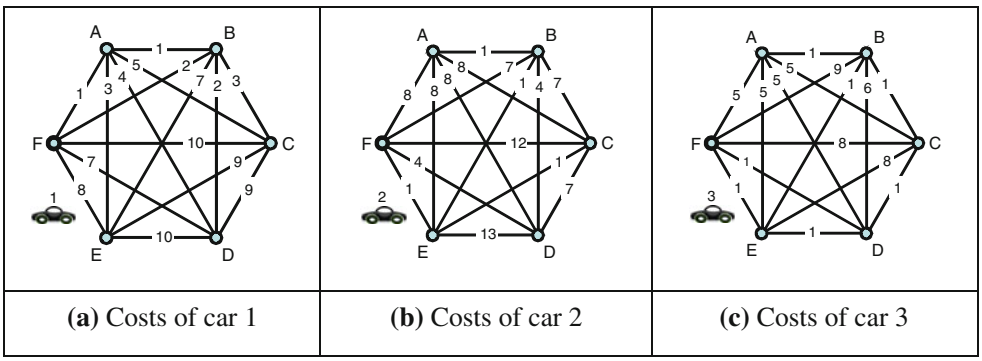
The objective of the proposed problem is to find the Hamiltonian cycle in a graph G that, starting on a given initial vertex, minimizes the sum of total operating costs regarding rented cars. The total operating costs are composed of a parcel that unifies rent expenses and other expenses in a value associated to each edge of G and a parcel associated to the cost of returning cars from the cities they are delivered to the cities they were rented. The return fees are calculated for each car k and the corresponding pair of cities where k was rented and delivered in the cycle. A solution to CaRS, where t cars are used, can be thought as a sequence S of t paths, $p_1, \ldots, p_t$. These paths are vertex-disjoint except for the first and last vertices. The first vertex of path p_r is the last vertex of path p_{r-1}, $1 < r \leq t$, that immediately precedes p_r in S, the first vertex of p_1 being the last vertex of p_t. Analogously, the last vertex of path p_r is the first vertex of path p_{r+1}, $1 \leq r < t$, that immediately succeeds p_r in S, the last vertex of p_t being the first vertex of p_1. Paths p_r and p_{r+1} in S correspond to different cars.

Figure 1 illustrates, in a complete graph with six vertices, a typical instance of CaRS. Three cars are available for rent in the example shown in Fig. 1, named 1, 2 and 3. Figures 1a, b and c show the costs associated to each edge of the complete graph corresponding to cars 1, 2 and 3, respectively. Unlike the TSP, the costs of solutions of CaRS problems are dependent on the city chosen as the starting point, that is, the car renter base. It occurs due to the fact that the fee paid to return a car in a city different from the one it was rented depends on those two cities. The tour's starting point in the example illustrated in Fig. 1 is vertex F.

The fees that have to be paid to return cars 1, 2 and 3 to cities F, B and C, respectively, when each car is rented in any other city are illustrated in Fig. 2. Those costs are shown as underlined integers next to the corresponding vertices. Figure 2a shows the values associated with the return fees associated to car 1 when it is rented in the city corresponding to vertex F. Figure 2b shows the values associated to car 2 when it is rented in city B and Fig. 2c shows the corresponding values to car 3 rented in city C. In the general case return costs are defined for each car and each pair of cities.

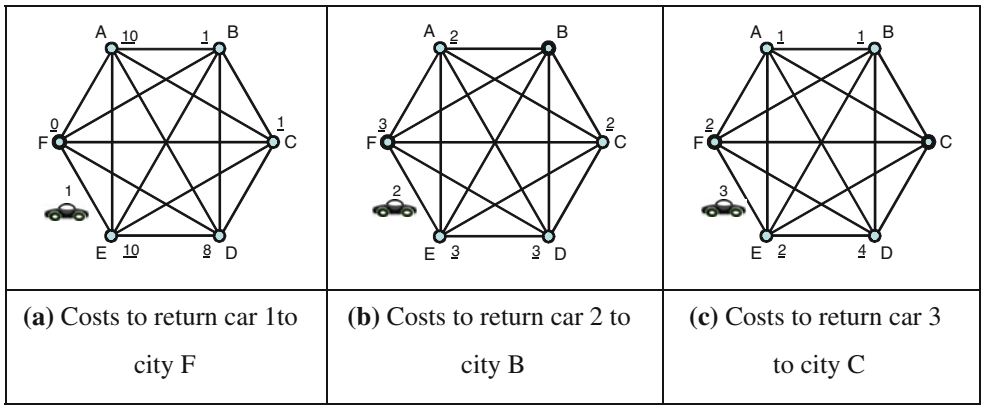
A solution for the problem exemplified in Figs. 1 and 2 is exhibited in Fig. 3. This solution considers a case where all available cars are rented and no car is rented more than once. The cost of the cycle, according to the solution shown in Fig. 3, corresponds to the cost of path F-A-B traversed with car 1, plus the cost of path B-E-C traversed with car 2, plus the cost of path C-D-F traversed with car 3. The cost corresponding to the values assigned to the edges of the complete graph is 6 unities. To this value it is necessary to add the cost of returning each car to its starting point. Car 1, delivered in city B, has to be returned to city F. Figure 2a shows that this return fee is 1. Car 2 has to be returned to B from C and car 3 has to be returned to C from F. Each of the latter fees is 2 unities. Thus, the cost of the solution presented in Fig. 3 is 11 unities.

CaRS has several variants which are established in accordance with conditions of real problems. The problem can be classified according to cars availability, return alternatives, existence of symmetry in the associated cost matrix, existence of links between cities (graph sparseness), etc. The remaining of this section presents some variants of the



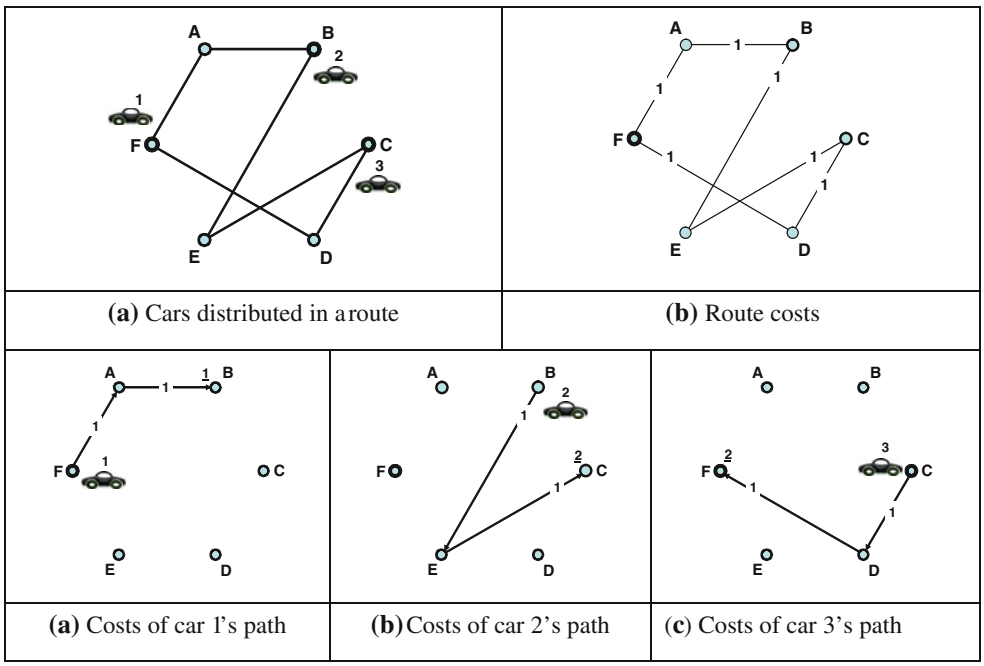
(a) Costs of car 1 **(b)** Costs of car 2 **(c)** Costs of car 3

Fig. 1 Costs associated to each car



(a) Costs to return car 1to city F | **(b)** Costs to return car 2 to city B | **(c)** Costs to return car 3 to city C

Fig. 2 Costs to return a car to the city it was rented from a different city



(a) Cars distributed in a route | **(b)** Route costs

(a) Costs of car 1's path | **(b)** Costs of car 2's path | **(c)** Costs of car 3's path

Fig. 3 Route cost of problem illustrated in Fig. 1

proposed problem in Sect. 2.1, an application on flexible manufacturing systems in Sect. 2.2 and a discussion on CaRS difficulty in Sect. 2.3.

2.1 Variants of the Traveling Car Renter Problem

The new proposed problem admits several specific situations being classified according to:

1. Availability of rental cars

Since there is no guarantee that agencies of all rental companies are present in all cities, it may not be necessarily assumed that any car can be rented in any city. The case in which all cars can be rented in all cities is called *total*. In any other case the problem is called *partial*. In this study, when no observation is made otherwise, the problem is considered total.

2. Alternatives to return cars to the cities they were rented

Since there is no guarantee that all rental companies operate services to receive cars in all cities, it may not be necessarily assumed that all cars can be returned in any city. The case in which all cars can be delivered in all cities is called *unrestricted*. In any other situation the problem is called *restricted*. In this paper, when no observation is made otherwise, the problem is considered unrestricted.

3. Contract integrity

When the problem does not allow the same type of car is rented more than once during the tour, the problem is called *without repetition*, in this case $k \geq ncars$. A special case of the *without repetition* problem is when all cars have to be rented, that is, $k = ncars$. In this case the problem is called *exact*. When car repetition is allowed the problem is called

with repetition. In this paper, when no observation is made otherwise, *without repetition* problems are considered.

4. Calculation of the costs to return a car

The costs to return cars to their starting points may be independent of network topology or network restrictions. The problem is called *free* when the return costs do not depend on the network topology and restrictions. The problem is said *bonded* when the cost to return a car to its starting city, named also car's base, takes into account the path traversed by the car to be go to its base. In this paper, when no observation is made otherwise, the considered problem is *free*.

5. Symmetry of distances between cities

When $c_{ij}^k = c_{ji}^k$ for $1 \leq i, j \leq n$, $1 \leq k \leq ncars$, the problem is *symmetric*, otherwise the problem is *asymmetric*.

6. Existence of links in the graph that models the problem

When the graph that models the problem is complete, the problem is called *complete*, otherwise the problem is called *incomplete*.

2.2 An application in flexible manufacturing system optimization

A flexible manufacturing system (FMS) has independent manufacturing nodes distributed in a production plant. The manufacture executes several distinct industrial tasks so that, in general, there are no production lines previously organized in the plant, and the transport between manufacturing nodes cannot always be carried out mainly by fixed equipment such as conveyor belts. An FMS transportation system may consist of different types of mobile carriers, mostly with electric propulsion, having different operating costs per unit of displacement. As a general guideline, the manufacture must plan their transportation system with each new production requirement. Also as a general guideline, a work-piece in FMS typically requires to be processed on a subset of workstations of the plant, not all. It is not rare the situation where each workstation can perform one or more operations on the work-piece with constant cost and depending only on the work-piece and on the operations under its responsibility, not depending on other previous or future operations on the work-piece. For a situation like this and considering that the FMS has to process a batch of similar work-pieces and considering also that there are k different carriers available to meet the demand of displacement of parts between the workstations, the problem of minimizing the total cost of transporting the batch between different workstations of the FMS can be modeled as a path or a circuit of the traveling car renter in which the carriers play the role of the cars and the circuit connects the workstations that will be necessary for processing the work-pieces. For the case where the piece should be delivered in the same place where it is initially received in the work layout, the cycle model is characterized. Otherwise the path model is characterized.

2.3 The difficulty of solving CaRS

The problem basically consists in determining a Hamiltonian cycle in a graph G composing paths built with the vertices of G. Let $T = \{1, \ldots, t\}$ denote the set of indices of up to t subgraphs H_r of $G, r \in T$. Calling $V(H_r)$ the vertices of H_r, the subgraphs H_r of CaRS have the following properties.

$$\bigcup_{r=1}^{t} V(H_r) = N \tag{1}$$

$$|V(H_s) \cap V(H_r)| \leq 1 \qquad \forall r, s \tag{2}$$

Constraint (1) determines that the union of all paths visits all vertices of G. Constraints (2) implies that two different subgraphs never have more than one vertex in common, a condition to prevent formation of subcycles. Note that the constraints (1) and (2) are not sufficient to guarantee the cycle of the CaRS. It is also necessary that the t subgraphs considered three to three, four to four, and so on, until $t - 1$ to $t - 1$, do not have more than one vertex in common.

Once this problem deals with Hamiltonian paths, each path done by a car in one subgraph H_r visits all vertices of H_r. The path of subgraph H_r has to be assigned to a car different from the cars assigned to neighbor paths during the construction of an Hamiltonian cycle in G. The costs of the edges of each subgraph correspond to the operation costs of the car traversing H_r. Furthermore, when $t \geq 2$ the total cost considers the return cost of each car rented in city i and returned in city $j, i \neq j$. Hamiltonian cycle and Hamiltonian path problems are well known NP-complete problems [11]. Due to what was previously exposed, the difficulty of solving CaRS is at least the same as the TSP. Nevertheless, although some solutions of the TSP are also solutions of CaRS, the latter has a number of feasible solutions greater than the former and incorporates all the requirements of the TSP, like other several classes of vehicle routing problems which are known to be more difficult than the TSP [25].

The Traveling Salesman Problem is a particular case of CaRS in the situation where there is only one vehicle available for rent. Note that the solution space of CaRS is exponentially greater than the solution space of the Traveling Salesman Problem. Considering $G = (N, M)$ a complete graph and that CaRS is total, unrestricted and without repetition, any permutation of the vertices of G is a feasible solution for the rental car problem considering only one of the k possible cars. Once there are k cars available for rent,

there are $k.n!$ different feasible configurations that meet the condition of use of only one different car in CaRS. From $k \geq 2$ each Hamiltonian cycle of the traveling car renter can be partitioned in up to k-subpaths in sequence and to each possibility of composition of such partitions a distinct cost for the route can be assigned. The possibility of this single value is guaranteed due to the composition of the costs related to the return fee of the rented cars with the costs of the trajectories of independent costs of each car. The number of possible partitions associated to each Hamiltonian cycle in G is equal to the number of different ways the set of n cities of the cycle can be divided in s groups of cities that are visited by s different cars, $k \geq s \geq 2$. The number that counts this process of division of a set in disjoint sets that go from 2 to k is the Stirling number of the second type. In this way, the number of configurations of the space of solutions of CaRS is at least $O(2^n)$ greater than the space of solutions of the Traveling Salesman Problem, since that for $k = 2$ the associated Stirling number is $O(2^n)$ [1]. For a general case of CaRS the dimension of the space of solutions is still greater once there is not a theoretical limit for the number of different cars to be considered in the problem.

3 Metaheuristic Algorithms

This section presents four heuristics for the investigated problem. The first two are Greedy Randomized Adaptive Search Procedures (GRASP) [9] with Variable Neighborhood Descent (VND) [20] in the local search phase. The other heuristics are Memetic Algorithms [22].

Solutions in the GRASP algorithms, as well as chromosomes in the memetic algorithms, are represented in 2-dimensional arrays with n elements as illustrated in Fig. 4a, b. Figure 4a shows a solution for the instance presented in Figure 3a. Figure 4b presents a solution for an instance with $n = 11$ and five cars. The second row corresponds to the sequence of cities visited in the tour. The elements in the first row correspond to cars. Let car c be assigned to the cities in the second row corresponding to indices i_1 to i_m. It means that car c is rented in city i_1 and delivered in city i_{m+1}. The last city visited by a car is not assigned to that car in the chromosome, since the car is returned on that city and another car is rented there to continue the tour. The starting city (city 0) is not represented as the final destination. In Fig. 4b four of the five available cars are used. The tour begins at city 0 with car 2 which passes through cities 6, 4, 3, 10 and is delivered in city 7 where car 1 is rented. Car 1 proceeds to city 9 passing through city 1. Car 5 is rented in city 9, passes through cities 2 and 5 and is delivered in city 8 where the last car, 4, is rented. Car 4 is delivered in the starting city.

Algorithm 1 : GVND1

```
1   main(nameInstance, #max_iter, α)
2     instanceRead(nameInstance)
3     for i ←1 to nCar
4       exact_Sol[i] ← Concorde(i)
5     f(Sol*) ←∞
6     for i ←1 to #max_iter
7       construct_phase(exact_Sol ,Sol,α)
8       Sol ← LocalSearchVND (Sol)
9       if (f(Sol) < f(Sol*))
10        Sol* ← Sol; f(Sol*) ←f(Sol)
11    return(Sol*)
```

3.1 GVND1

In this section the first GRASP hybridized with VND, named GVND1, is presented. The algorithm has a pre-processing phase where *nCar* optimal TSP solutions are obtained with the Concorde TSP Solver [2,3], one for each available car, where *nCar* is the number of cars available in the considered instance. The pseudo-code of GVND1 is presented in Algorithm 1 where the input parameters are: the name of the instance to be processed, *nameInstance*, the number of GRASP iterations, *#max_iter*, and the size of the restrict candidate list, α.

Solution *Sol* is built in procedure *construct_phase*() whose pseudo-code is presented in Algorithm 2. Then, *Sol* is submitted to the VND procedure, *LocalSearchVND*(), described in Algorithm 3.

To construct solution *Sol*, the algorithm adds to an initially empty solution a set of paths that are built iteratively and assigned to different cars. In Algorithm 2 a path between the cities stored in variables *iCity* and *dCity* is built and assigned to the car stored in variable *iCar*. Variable *Av_cities* stores the set of cities yet not assigned to *Sol*. The car is chosen randomly in procedure *random_available-_car*(). Initially variable *iCity* is set to the home city. This variable stores the starting point of the path being built at a given iteration of the main loop. Analogously, variable *dCity* stores the final city of the path. The destination city is chosen randomly from a restricted candidate list, *RCL*, built in step 4. That list contains α cities with the smallest return fees related to the city stored in *iCity* for the considered car. The path between the cities stored in *iCity* and *dCity* is built in procedure *build_path*(). The first path is built between the home city and *dCity*. The path is taken from the exact TSP solution corresponding to the car stored in *iCar*, *exact_Sol*[*iCar*]. That is, the path between cities 0 and *dCity* of the Hamiltonian cycle built on the complete graph corresponding to *iCar* is first added to solution *Sol*. In the next iterations of the main loop, cities that already belong to *Sol* are removed from the path obtained from the optimal TSP solution corresponding to *iCar* in procedure *build_path*(). Suppose that city b, between cities a and c in the path built in the *i-th* iteration of the main loop, is already in

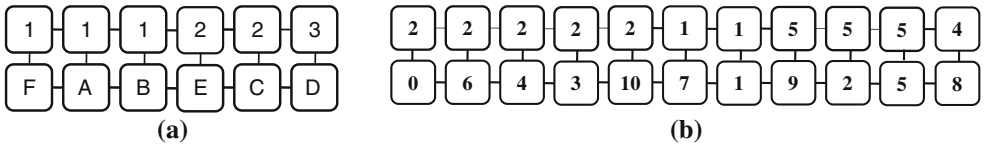


Fig. 4 **a** Solution for instance of Fig. 3a; **b** solution of a problem with 11 cities and 5 available cars

Algorithm 2 : construct_phase(*exact_Sol* ,*Sol*,α)

```
1   iCity ←0; #n_available_car ← nCar; Av_cities ← N\{0};
2   while ((#n_available_car > 1) and (Sol is not complete))
3       iCar ←random_available_car( )
4       RCL ←build_RCL(α,iCity,iCar, Av_cities)
5       dCity ←select_city(RCL); Av_cities ← Av_cities \{dCity};
6       path[iCar] ←build_path(exact_Sol[iCar],iCity, dCity)
7       set_path_car(Sol,path[iCar])
8       iCity ←dCity; n_available_car ← n_available_car - 1
9   end_while
10  if ((#n_available_car = 1) and (Sol is not complete))
11      iCar ←last_available_car( )
12      path[iCar] ← nearest_neighbor(iCity)
13  end_if
14  set_path_car(Sol,path[iCar])
15  return (Sol)
```

Algorithm 3: LocalSearchVND (*Sol*)

```
1   i ← 1; Sol* ← Sol
2   repeat
3    if (i = 1)
4      Sol ← InvertSol(Sol)
5      Sol ← Insert&Saving(Sol)
6    end_if
7    if (i = 2)
8      Sol ← Shift(Sol)
9    end_if
10   if (f(Sol) < f(Sol*))
11     Sol* ← Sol; f(Sol*) ←f(Sol); i ← 1
12   end_if
13   else i ← i + 1
14   end_if_else
15  until (i ≥ 3)
16  return (Sol*)
```

a path built in iteration j, $j < i$. Then the procedure removes city b from the i-th path and includes a link between cities a and c. The path corresponding to *iCar* is stored in variable *path*[*iCar*]. The main loop halts if all cities were included is *Sol*, that is a solution has been built, or if only one car lasts. In the latter case, a path has to be built including all cities yet not assigned to any path in *Sol*. This task is accomplished with a modification of the Nearest Neighbor Heuristic, a known constructive heuristic proposed for the TSP [14], that builds a Hamiltonian path with the remaining cities. The destination city of the last iteration of the main loop is the initial city of the last path, stored in *iCity*. Suppose that #*cities* are still not assigned to *Sol*. Then heuristic Nearest Neighbor is applied to those #*cities*, starting at *iCity* and considering edge weights corresponding to the last available car. Finally, the last path is added to *Sol* in step 14 of Algorithm 2. The home city is implicitly included in thes last path of *Sol* as the last city, completing the cycle represented in *Sol*.

In the local search phase a VND metaheuristic was used to explore the search space of three neighborhood structures named *InvertSol*, *Insert&Saving* and *Shift*, as shown in Algorithm 3. First solution *Sol* is submitted to *InvertSol* procedure, if it is improved the best current solution, *Sol** is updated. *InvertSol* is a simple low time consuming heuristic that reverses the sequences of cities of an input solution. With the reversal, though the same cars traverse the same sets of cities, the cost of rent and fees for returning the car to where they were rented change. *InvertSol* returns the reversed solution only if the latter is better than the former, otherwise the input solution is returned by the procedure. After, *Sol* is submitted to the *Insert&Saving* local search in which insertions of cars that are not yet in *Sol* are investigated. Consider that k cars, $k < nCar$, are assigned to the paths in *Sol*, then $av\_cars = nCar - k$ cars are still available. *Insert&Saving* randomly chooses an available car and searches the best position to insert it in *Sol*. Let $P = (i_1, i_2, \ldots, i_z)$ be the path traveled with car c, where c is rented in city i_1 and returned in city i_{z+1}. The insertion of a new car c' is checked in cities i_r, $1 \leq r \leq z$. The insertion of c' in i_r divides P into two paths, $P' = (i_1, \ldots, i_r)$ and $P'' = (i_{r+1}, \ldots, i_z)$, assigned to cars c' and c, respectively. The procedure verifies the cost of solutions resultant from the insertion of the new car in each point of *Sol*. All *av_cars* are tested. If any of these insertions produces a solution with a cost lower than the cost of the input solution, the new solution is set as the current solution in the local search. The procedure continues until all non-assigned cars have been considered for insertion. When an improvement is not achieved with the *Insert&Saving* method, the algorithm proceeds to the *Shift* local search. In the latter method a neighboring solution *Sol*' of *Sol* is generated by exchanging two consecutive cities within the path assigned to one car. Clearly, the exchanging of two cities within the path of a given car changes the cost of the path traveled by the considered car. Nevertheless, depending on the exchanged cities, the operation can also impact the costs of paths immediately before and after the considered path. Let $S = (i_1, i_2, \ldots, i_{x-1}, i_x, \ldots, i_{y-1}, i_y, \ldots, i_z)$ be a sequence of cities in *Sol* that defines three paths traveled by cars c_1, c_2 and c_3 that are rented in cities i_1, i_x, i_y

and returned in cities i_x, i_y, i_{z+1}, respectively. If city i_r is exchanged with city i_{r+1}, $x < r < y - 1$, the change in the cost of the path traveled by car c_2 depends only on the weights assigned to the edges of the corresponding path. Rent cost and returning rates remain the same. Nevertheless, if $r = x$, the cities where car c_1 is returned and car c_2 is rented also change. Similarly, if $r = y - 1$, the cities where car c_2 is returned and car c_3 is rented also change.

3.2 GVND2

GVND2 is a version of GVND1 with modifications in the constructive phase and in the *Insert&Saving* method. Algorithm 4 presents the pseudo-code of the constructive phase of GVND2, where *qcar* cars are used to build solution *Sol*. Variable *qCar* stores an integer randomly selected in the interval [1,*nCar*] (step 2 of Algorithm 4) with uniform probability distribution. In step 3 a list of pairs (*iCar*,*dCity*), named *Av_pairs*, is built with all associations between *nCars* and *n* cities, where *iCar* and *dCity* stand for a car and a destination city, respectively. Given any city, *iCity*, a cost can be assigned to each pair of *Av_pairs*, being the return fee paid when *iCar* is rented in city *iCity* and delivered in *dCity*. The RCL, with size α, is built with such pairs which are evaluated accordingly to the city considered as the initial one at each iteration of the main loop, *iCity*. The RCL is built in step 5 and in step 6 one pair is chosen based on a roulette wheel method where the lower the return fee the higher the associated probability. The remaining of Algorithm 4 is much the same as Algorithm 2, except for step 10 where the list *Av_pairs* is updated by removing from it all pairs in which the first element corresponds to *iCar* or the second element corresponds to any city assigned to *path*[*iCar*].

Procedure LocalSearchVND () is the same in both GVND algorithms, but in GVND2 *Insert&Saving* has two phases. The first phase is the same described for GVND1. In the second phase, the procedure seeks to change the points where the cars are exchanged. Suppose that car c_1 is rented and delivered in cities i_i and i_y, respectively, and car c_2 is rented in i_y and delivered in i_z. The city where car c is returned is considered its *exchanging point*. The procedure tests all cities between i_{y+1} and i_z as exchanging points for c_1. It is done for all cars, except the last one.

3.3 Memetic Algorithm MA1

Algorithm 5 shows the pseudo-code of the Memetic Algorithm (MA) developed for CaRS. The input parameters are: number of generations (*nOffspring*), population size (*sizePop*), recombination rate (*txCros*) (the number of individuals that reproduce in each generation), mutation rate (*txMuta*) and the population renewal rate (*txRenw*).

Algorithm 4 : construct_phase(*exact_Sol*,*Sol*,α)

```
1   iCity ← 0;
2   qcar ← IRandom(1, nCar); #n_available_car ← qCar;
3   Av_cities ← N\{0}; Av_pairs ← {1,...,nCar}× Av_cities;
4   while ((#n_available_car > 1) and (Sol is not complete))
5       RCL ← build_RCL(α, iCity,Av_pairs)
6       (dCity,iCar) ← select_pair(RCL);
7       path[iCar] ← build_path(exact_Sol[iCar],iCity, dCity)
8       set_path_car(Sol,path[iCar])
9       update(Av_pairs, iCar, path[iCar])
10      iCity ← dCity; n_available_car ← n_available_car – 1
11  end_while
12  if ((#n_available_car = 1) and (Sol is not complete))
13      iCar ← random_available_car( )
14      path[iCar] ← nearest_neighbor(iCity)
15  end_if
16  set_path_car(Sol,path[iCar])
17  return (Sol)
```

Algorithm 5 : Memetic Algorithm for CaRS

```
1   main(nameInstance,sizePop,nOffspring,txCros,txMuta,txRenw)
2    instanceRead(nameInstance)
3    Pop[] ← generateInitPop(sizePop)
4    for i ← 1 to sizePop
5        Pop[i] ←LocalSearchVND(Pop[i])
6    end_for
7    for i ← 1 to nOffspring
8        for j ← 1 to sizePop*txCros do
9            dad,mom ← parentsSelection()
10           son1, son2 ← Crossover(dad,mom)
11           son1, son2 ← carsMutation(son1,son2,txMuta)
12           LocalSearchVND(son1,son2)
13           if ( son1,son2 < Pop[dad],Pop[Mom] )
14               Pop[dad] ← son1, Pop[mom] ← son2
15           end_if
16       end_for
17       generateNewIndividuals(sizePop*txRenw)
18   end_for
19  return(Pop[0])
```

The fitness of each chromosome is given by the inverse of the objective function, which means that the lower the value of the objective function the fittest the chromosome is. The initial population is generated with a version of the Nearest Neighbor heuristic in procedure *generateInitPop*() which receives the size of the population as input parameter. Let *nCar* be the number of available cars of a given instance. Initially, the algorithm selects randomly a car c and a destination city j, $j \neq 0$. Then a path between cities 0 and j is built. Beginning at city 0, the procedure seeks the nearest city in accordance to the edge weights corresponding to car c. The procedure adds cities iteratively to c's path until reaching city j. At this point, if there are cities yet not visited, city j is set as the new origin. A new car and a new destination city are randomly selected. The procedure continues until all cities are added to the tour or until there is only one available car. In the latter case, the path has to include all remaining cities and the same version of the Nearest Neighbor heuristic utilized at the end of the constructive phase of

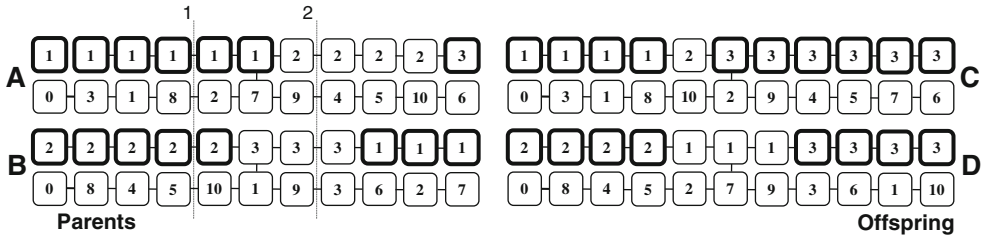


Fig. 5 Recombination operator

GVND algorithms is used. Finally, city 0 is added to the path of the last car. In steps 4 to 6, each individual of the initial population is submitted to the same VND procedure used in GVND1.

Parents for recombination are selected with the roulette wheel method in procedure *parentsSelection*(). A multi-point recombination operation adapted for CaRS is used to generate two children. The number of points in the crossover is randomly generated in the interval [1,4]. The recombination operator is illustrated in Fig. 5 on an instance with $n = 11$ and 3 cars. Two parents, A and B, generate offspring C and D. A 2-points operator is used in the example. The two crossing over points are shown with the parents. Chromosome C inherits its first and third parts from A and the second part from B. Similarly, chromosome D inherits the first and last parts from B and the second part from A. It may be necessary to restore feasibility of solutions represented in offspring. Restoration can be necessary due to infeasibility regarding routes or cars assignment. For example, immediately after inheriting its parts from parents, the route of chromosome C is [0 3 1 8 10 1 9 4 5 10 6] which is not feasible since cities 1 and 10 appear twice and cities 2 and 7 are missing. In the restoration procedure the route of chromosome C is replaced by [0 3 1 8 10 * 9 4 5 * 6] with asterisks replacing the second time cities 1 and 10 appear. Each asterisk is replaced by a missing city chosen at random, in this case by cities 2 and 7. Infeasibility is also observed for chromosome C regarding cars assignment. The car assignment of chromosome C, [1 1 1 1 2 3 3 2 2 2 2 3], is not feasible, for the problem considered in this paper requires each car being rented once. The restoration procedure replaces car repetitions by asterisks. Then each asterisk is replaced by the car which appears in the first preceding position. Thus, the car assignment of chromosome C is replaced by [1 1 1 1 2 3 3 * * * * *] and each asterisk is replaced by car 3. Chromosome D is processed similarly.

Solutions resulting from recombination are submitted to mutation in procedure *carsMutation*(). The mutation operator verifies which vehicles are not in the solution represented in the chromosome. Consider that *list_A* stores the cars not assigned to chromosome A. Each vehicle in *list_A* is inserted in A by the mutation operator. The mutation rate defines the

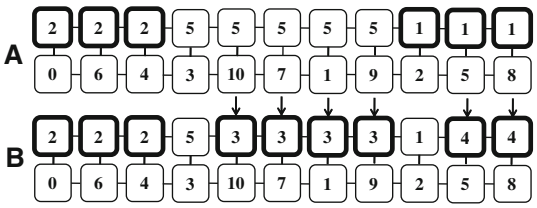


Fig. 6 Mutation operator

number of positions (cities) that are assigned to the entering cars. Figure 6 illustrates mutation on chromosome A which represents a solution for an instance with $n = 11$ and five cars. Cars 3 and 4 are not used in the solution shown in chromosome A. Two cities are randomly chosen to be the starting points of cars 3 and 4. In this example, cities 10 and 5 are chosen as starting points for cars 3 and 4, respectively. Considering value 3 for the mutation rate, vehicle 5 is replaced by vehicle 3 in cities 10, 7 and 1. Car 4 replaces car 1 in cities 5 and 8, since the number of cities assigned to car 4 is smaller than the one fixed as the mutation rate. It occurs since there are no cities after 5 and 8 in the chromosome. The sequence of cars in the resulting chromosome is [2 2 2 5 3 3 3 5 1 4 4]. Once car 5 appears in two different paths, the solution needs to be restored. The same restoration function used to repair cars assignment described previously is used. The repairing function removes the second occurrence of car 5, resulting in [2 2 2 5 3 3 3 * 1 4 4] and replaces the asterisk by car 3.

After recombination and mutation, the offspring is submitted to local search. The first memetic algorithm version, MA1, uses the same local search procedure implemented in GVND1. The resulting solutions are compared with their parents and the best two, among the four, individuals survive.

Part of the current population is replaced by new solutions in procedure *generateNewIndividuals*(). The new chromosomes are generated with the constructive method used to create the initial population. Variable *txRenw* defines the percentage of the population that is replaced by the new individuals. Therefore, *sizePop*txRenw* new individuals are created to replace the *sizePop*txRenw* worst individuals of the

current population. This renewal process promotes diversification and prevents premature convergence.

3.4 Memetic Algorithm MA2

The second memetic algorithm is pretty similar to the one presented in the previous section. The differences are in generation of the initial population and in the local search procedure utilized to improve chromosomes. As in MA1, the initial population is generated with the version of the nearest neighbor heuristics described in Sect. 3.3, however the number of cars available to each initial individual is randomly selected in the interval [1,*nCar*]. Furthermore, *nCar* optimal TSP solutions obtained with the Concorde TSP solver [3], one for each car, are included in the initial population. After recombination and mutation the offspring is submitted to local search. The same VND procedure used in GVND2 is implemented in MA2.

4 Computational experiments

Since the problem introduced in this paper is new, a library of instances, named CaRSLib, was created with the purpose of testing the proposed algorithms. These instances have the following features: *total* (all cars can be rented in all cities), *unrestricted* (all cars can be delivered in all city), *without repetition* (each car can be rented at most once), *free* (return fees are not correlated with the instance topology), *symmetric* (costs to go from city i to city j and vice-versa are equal) and *complete* (complete graph—there is an edge between each pair of cities). The set consists of Euclidean and non-Euclidean instances. For each set, three groups of instances were created. These groups concern primary weights associated to the edges. Primary edge weights of graphs of the first group of instances were taken from real maps. Primary edge weights of graphs that model the second group of instances were generated randomly according to a uniform probability distribution in the range [10,500]. The third group of instances is based on TSPLIB instances [26]. If an instance is defined to have *ncars* cars, then the weight associated to each edge corresponding to each car $t, 1 \leq t \leq ncars$, was randomly chosen, with uniform probability, in the range $[1.1w, 2.0w]$, where w stands for the primary edge weight. Return fees of each instance were randomly generated according to a uniform probability distribution in the range $[0.05\mu, 0.10\mu]$ where μ stands for the mean of the weights of the edges of the graph that corresponds to the given instance. The dataset, file formats and the description of each group of instances are available at http://www.dimap.ufrn.br/lae/en/projects/CaRS.php. The experiments were performed on 20 Euclidean and 20 non-Euclidean instances. Each group is formed with ten real map instances, five random instances and five TSPLIB type instances. The number of cities varies between 14 and 300 and the number of vehicles varies between 2 and 5.

This section is divided in three subsections. Sections 4.1 and 4.2 present comparisons between the two versions of each metaheuristic approach, GVND and Memetic Algorithms, respectively. The best algorithms from each type are compared in Sect. 4.3. Thirty independent executions of each algorithm were performed for each instance on a PC Intel Xeon QuadCore W3520 2.8 GHz, 8G RAM running Scientific Linux 5.5 64 bits.

To assess the performance of the proposed algorithms the following methodology was used. First, each algorithm was executed according to given stopping criteria. The stopping conditions for GVND versions were maximum number of re-starts set to 300 or 90 re-starts without improving the best current solution. The stopping criteria for MA versions were $nOffspring = 20$ or $maxGen = 0.30 \times nOffspring$ generations without improvement of the best current solution. The maximum processing time each algorithm spent on one of thirty executions on each instance was annotated. Given that algorithms A_1 and A_2 are being compared, two groups of experiments were performed taking into account the maximum processing times found in the first experiment. In the first group of experiments of the second stage the maximum processing times annotated for algorithm A_1 were given as stopping condition for algorithm A_2 for each corresponding instance. In the second group of experiments, the maximum processing times annotated for algorithm A_2 were given to algorithm A_1. Since the results do not conform to a normal distribution, the non-parametric test named U test [6] was applied to identify significant differences among the results produced by the two compared algorithms. Significance level 0.05 was adopted to analyze the results from the U test. Once computational effort is fixed for each pair of compared algorithms in each group of experiments, a statistical test for proportions comparison was applied to compare overall results of each group. The proportions comparison test proposed by [30] was used to compare success rates between the tested methods. Success is defined in terms of a target established in advance. Here, the success of method A_1, when compared to method A_2, is stated when A_1 achieves a result better than the one achieved by A_2 on a given problem instance. The confidence levels provided by the one-tailed proportions comparison test were calculated with the tool available at http://qualopt.eivd.ch/stats/?page=stats.

Preliminary tests to tune the parameters of the proposed algorithms were executed on a set of 20 CaRSLib instances with number of cities ranging from 14 to 300 and 2 to 5 vehicles. Parameter α of the GVND algorithms was set to 0.25. The values chosen for the parameters of the Memetic Algorithms are: $sizePop = 30, txCros = 0.60, txMuta = 0.40$ and $txRenw = 0.15$.

4.1 Comparison between the GVND versions

This section presents the comparison between the performances of the GVND algorithmic versions presented in Sects. 3.1 and 3.2.

Tables 1 and 2 show the results obtained with GVND1 and GVND2 on Euclidean and non-Euclidean instances, respectively, regarding maximum processing times obtained by GVND1. Similarly, Tables 3 and 4 show results for maximum processing times fixed by GVND2. Columns *Name*, *City* and *Car* show, respectively, the name, the number of cities and the number of available cars of each instance. Column *#Best* shows the best solution found during experiments performed with the four proposed algorithms. Columns *Avg*, *Best* and *Freq*, respectively, show the average, the value of the best solution and the number of times the best solution reported in column *#Best* was found by each algorithm. Column $T(s)$ shows the maximum processing time and column *p-value* shows the values obtained with the U test. Values presented in the latter column are rounded to two decimal places. Given significance level 0.05, values greater than 0.95 indicate the best performance of GVND2 while values less than 0.05 points out the latter as the best algorithm.

Table 1 shows that algorithms GVND1 and GVND2 perform similarly on Euclidean instances when processing times are fixed by GVND1. This fact can be observed in column *p value* where each algorithm performs significantly better than the other on 10 instances. A slightly better performance of algorithm GVND1 than GVND2 is observed in column *Freq* concerning the number of best solutions obtained by each algorithm.

Table 2 shows that conclusions are rather different from the previous ones on the set of non-Euclidean instances. Column $p-$ value shows that GVND1 is outperformed by algorithm GVND2 on 18 of the 20 tested instances. The results presented in column *Freq* reinforce the conclusion that GVND2 exhibits the best performance, since this algorithm finds out the best solution of 11 instances while the other one does not obtain the best solution of any instance. The test for proportions comparison points out GVND2 as the best algorithm with 99.996 confidence level.

Results on both sets of instances when maximum processing times are fixed by GVND2 are shown in Tables 3 and 4. Table 3 shows that, although slightly superior results are exhibited by GVND2 on Euclidean instances, the results obtained by both GVND versions are still quite similar on this set of instances. Column *Freq* confirms this conclusion, once GVND1 and GVND2 find, in average, 22.67 and 21.33% of the overall best solutions, respectively.

Column *p value* of Table 4 shows that GVND2 exhibits its superior performance to GVND1 on 19 of 20 instances. According to the proportions comparison test, GVND2 is pointed out as the best approach with confidence level 99.999998. Columns *Freq* show that no best solution is found by GVND1 and 22.67% overall best solutions are found by GVND2.

In summary, the results presented in Tables 1, 2, 3 and 4 lead to the conclusion that on the set of Euclidean instances, the GVND versions behave similarly. Nevertheless, on the other set, GVND2 is clearly devised as the best GVND approach. Therefore, GVND2 is used in the computational experiments reported in Sect. 4.3 when it is compared with the best Memetic Algorithm version.

4.2 Comparison between the Memetic Algorithm versions

This section presents the comparison between the performances of the memetic algorithms presented in Sects. 3.3 and 3.4. Tables 5 and 6 show the results on Euclidean and non-Euclidean instances, respectively, when processing times are fixed by MA1.

Table 5 shows that MA2 finds better *Best* and *Avg* results than MA1 on 15 and 17 Euclidean instances, respectively. MA1 exhibits better results than MA2 on 2 instances regarding column *Avg* and on 1 instance regarding column *Best*. Column *p value* shows favorable results for MA2 on 17 instances while the best performance of MA1 is indicated on 2 instances only. This result confirms the ones observed in column *Avg*. The test for proportions comparison applied on these results show that MA2 behaves significantly better than MA1 with confidence level 99.987. The overall mean of best solutions found by MA1 and MA2, given by the arithmetic mean of columns *Freq*, is 12.67 and 20%, respectively.

Similar conclusions can be drawn regarding non-Euclidean instances when the results shown in Table 6 are analyzed. MA2 exhibits significantly better results than MA1 on 19 instances as indicated in column *p value*. Better average results are obtained by MA2 on 18 non-Euclidean instances, when the processing time is fixed by MA1. Confidence level 99.999 is provided by the proportions comparison test applied to these results, indicating the superior performance of MA2.

Tables 7 and 8 show the results on Euclidean and non-Euclidean instances, respectively, when processing times are fixed by MA2. Although, the performance of MA1 is slightly improved on Euclidean instances when results presented in Tables 5 and 7 are compared, conclusions on the best behavior presented by MA2 do not change. Column *p-value* of Table 7 shows that the U test indicates that MA2 behaves significantly better than MA1 on 14 Euclidean instances, while the latter outperforms the former on 3 instances. Better results for MA2 against MA1 are presented in columns *Avg* and *Best* of Table 7 on 15 and 13 instances. Applying the proportions comparison test to results concerning the number of best average values achieved by both algorithms, MA2 is indicates as the best approach with confidence level 98.88.

Table 1 GVND results on Euclidean instances, processing times fixed by GVND1

INSTANCE				GVND2			GVND1			T(s)	p value
Name	City	Car	#Best	Avg	Best	Freq	Avg	Best	Freq		
BrasilRJ14e	14	2	297	306	306	0	*297*	*297*	30	1	1
BrasilRN16e	16	2	375	401	401	0	*375*	*375*	30	1	1
BrasilPR25e	25	3	510	540	532	0	*510*	*510*	29	2	1
BrasilAM26e	26	3	495	505	505	0	*495*	*495*	27	3	1
BrasilMG30e	30	4	576	*587*	*583*	0	603	595	0	5	*0*
BrasilSP32e	32	4	615	*621*	*621*	0	633	626	0	8	*0*
BrasilRS32e	32	4	529	558	556	0	*537*	*529*	11	8	1
BrasilCO40e	40	5	703	*745*	*703*	1	807	805	0	18	*0*
BrasilNO45e	45	5	848	*874*	*848*	3	1,008	1,008	0	23	*0*
BrasilNE50e	50	5	756	*833*	*778*	0	963	940	0	43	*0*
Betim100e	100	3	1,402	*1,511*	*1,416*	0	1,723	1,708	0	78	*0*
Vitoria100e	100	5	1,471	*1,487*	*1,471*	24	1,802	1,642	0	155	*0*
PortoVelho200e	200	3	2,947	3,313	3,294	0	*3,142*	*3,041*	0	466	1
Cuiaba200e	200	3	3,174	3,801	3,801	0	*3,379*	*3,212*	0	686	1
Belem300e	300	4	3,806	5,034	5,034	0	*4,635*	*4,563*	0	1,804	1
berlin52eA	52	3	8948	9,140	9,139	0	*9,020*	*8,991*	0	20	1
eil76eB	76	4	1,911	*1,998*	*1,911*	2	2,228	2,158	0	87	*0*
rat99eB	99	5	3,206	3,453	3,431	0	*3,439*	*3,351*	0	194	0.96
rd100eB	100	4	9,940	*9,968*	*9,942*	0	10,107	9,951	0	103	*0*
st70eB	70	4	1,893	*1,981*	*1,937*	0	2,201	2,085	0	77	*0*

Best results are emphasized in italics

Table 2 GVND results on non-Euclidean instances, processing times fixed by GVND1

INSTANCE				GVND2			GVND1			T(s)	p value
Name	City	Car	#Best	Avg	Best	Freq	Avg	Best	Freq		
BrasilRJ14n	14	2	170	175	175	0	*171*	*171*	0	1	1
BrasilRN16n	16	2	194	*194*	*194*	30	203	203	0	1	*0*
BrasilPR25n	25	3	246	*264*	*256*	0	311	305	0	5	*0*
BrasilAM26n	26	3	225	*236*	*233*	0	242	239	0	5	*0*
BrasilMG30n	30	4	288	*321*	*288*	1	375	352	0	11	*0*
BrasilSP32n	32	4	272	*292*	*272*	1	336	298	0	12	*0*
BrasilRS32n	32	4	312	*326*	*312*	1	372	344	0	15	*0*
BrasilCO40n	40	5	671	*718*	*671*	1	826	755	0	39	*0*
BrasilNO45n	45	5	689	*734*	*689*	8	889	770	0	55	*0*
BrasilNE50n	50	5	773	*861*	*773*	1	1,044	874	0	81	*0*
Londrina100n	100	3	1,329	*1,363*	*1,339*	0	1,783	1,629	0	192	*0*
Osasco100n	100	4	1,272	*1,407*	*1,332*	0	2,000	1,910	0	191	*0*
Aracaju200n	200	3	2,590	*2,987*	*2,813*	0	3,686	3,223	0	903	*0*
Teresina200n	200	5	2,780	*2,893*	*2,780*	6	3,793	3,261	0	1,407	*0*
Curitiba300n	300	5	5,474	6,616	6,616	0	*6,125*	*5,680*	0	3,388	1
berlin52nA	52	3	1,476	*1,521*	*1,476*	3	1,777	1,661	0	41	*0*
ch130n	130	5	2,671	*2,881*	*2,671*	10	4,706	3,855	0	478	*0*
d198n	198	4	5,866	*6,106*	*5,866*	8	7,138	6,529	0	1,330	*0*
kroB150n	150	3	4,349	*4,465*	4,445	0	5,368	*4,414*	0	464	*0*
rd100nB	100	4	1,872	*2,115*	*1,893*	0	2,953	2,623	0	205	*0*

Best results are emphasized in italics

Table 3 GVND results on Euclidean instances, processing times fixed by GVND2

INSTANCE				GVND2			GVND1			T(s)	p value
Name	City	Car	#Best	Avg	Best	Freq	Avg	Best	Freq		
BrasilRJ14e	14	2	297	297	297	30	297	297	30	1	–
BrasilRN16e	16	2	375	375	375	30	375	375	30	1	–
BrasilPR25e	25	3	510	510	510	30	510	510	30	1	–
BrasilAM26e	26	3	495	*495*	*495*	30	504	501	0	2	1
BrasilMG30e	30	4	576	600	*576*	1	*597*	589	0	2	*0*
BrasilSP32e	32	4	615	636	628	0	*620*	*615*	2	2	*0*
BrasilRS32e	32	4	529	*536*	*529*	11	558	556	0	4	1
BrasilCO40e	40	5	703	814	806	0	*761*	*733*	0	7	*0*
BrasilNO45e	45	5	848	1,006	954	0	*892*	*886*	0	10	*0*
BrasilNE50e	50	5	756	965	963	0	*829*	*756*	1	15	*0*
Betim100e	100	3	1,402	*1,496*	*1,402*	1	1,761	1,560	0	127	1
Vitoria100e	100	5	1,471	1,916	1,757	0	1,652	1,622	0	70	*0*
PortoVelho200e	200	3	2,947	*3,086*	*2,947*	1	3,272	3,270	0	453	1
Cuiaba200e	200	3	3,174	3,447	*3,174*	1	*3,388*	3,388	0	349	*0*
Belem300e	300	4	3,806	*4,345*	*3,806*	1	5,354	5,354	0	891	1
berlin52eA	52	3	8,948	9,149	9,133	0	*8,948*	*8,948*	30	10	*0*
eil76eB	76	4	1,911	2,227	2,115	0	*2,006*	*1,914*	0	82	*0*
rat99eB	99	5	3,206	3,427	3,294	0	3,443	*3,206*	1	185	1
rd100eB	100	4	9,940	10,128	9,957	0	*9,972*	*9,940*	1	68	*0*
st70eB	70	4	1,893	2,173	2,046	0	*1,994*	*1,893*	3	58	*0*

Best results are emphasized in italics

Table 4 GVND results on non-Euclidean instances, processing times fixed by GVND2

INSTANCE				GVND2			GVND1			T(s)	p value
Name	City	Car	#Best	Avg	Best	Freq	Avg	Best	Freq		
BrasilRJ14n	14	2	170	217	217	0	*170*	*170*	30	1	*0*
BrasilRN16n	16	2	194	260	260	0	*194*	*194*	30	1	*0*
BrasilPR25n	25	3	246	325	320	0	*251*	*246*	7	4	*0*
BrasilAM26n	26	3	225	236	229	0	*231*	*225*	10	4	*0*
BrasilMG30n	30	4	288	372	315	0	*332*	*312*	0	7	*0*
BrasilSP32n	32	4	272	321	307	0	*292*	*274*	0	7	*0*
BrasilRS32n	32	4	312	409	367	0	*346*	*321*	0	7	*0*
BrasilCO40n	40	5	671	845	796	0	*688*	*673*	0	17	*0*
BrasilNO45n	45	5	689	922	771	0	*755*	*700*	0	28	*0*
BrasilNE50n	50	5	773	1,071	995	0	*865*	*778*	0	41	*0*
Londrina100n	100	3	1,329	1,785	1,639	0	*1,356*	*1,329*	1	228	*0*
Osasco100n	100	4	1,272	2,060	1,826	0	*1,505*	*1,272*	1	240	*0*
Aracaju200n	200	3	2,590	3,501	3,242	0	*2,788*	*2,590*	12	1,189	*0*
Teresina200n	200	5	2,780	*3,722*	*3,368*	0	4,021	3,710	0	975	1
Curitiba300n	300	5	5,474	6,082	5,849	0	*5,474*	*5,474*	30	2,721	*0*
berlin52nA	52	3	1,476	1,813	1,765	0	*1,603*	*1,506*	0	36	*0*
ch130n	130	5	2,671	4,434	3,981	0	*3,423*	*3,013*	0	522	*0*
d198n	198	4	5,866	7,289	*6,469*	0	*6,931*	6,715	0	1,610	*0*
kroB150n	150	3	4,349	5,300	4,835	0	*4,458*	*4,349*	14	675	*0*
rd100nB	100	4	1,872	2,844	2,629	0	*2,001*	*1,872*	1	289	*0*

Best results are emphasized in italics

Table 5 Comparison of Memetic Algorithms on Euclidean instances with processing times fixed by MA1

INSTANCE				MA2			MA1			T(s)	p value
Name	City	Car	#Best	Avg	Best	Freq	Avg	Best	Freq		
BrasilRJ14e	14	2	294	294	294	30	294	294	25	1	*0*
BrasilRN16e	16	2	375	*375*	375	30	376	375	27	1	*0.04*
BrasilPR25e	25	3	510	*510*	510	30	515	510	17	2	*0*
BrasilAM26e	26	3	467	490	467	1	*481*	467	3	4	1
BrasilMG30e	30	4	541	*577*	*543*	0	596	563	0	6	*0*
BrasilSP32e	32	4	601	*621*	*601*	1	624	615	0	8	0.1
BrasilRS32e	32	4	506	546	532	0	*523*	*512*	0	8	1
BrasilCO40e	40	5	709	*740*	*709*	1	824	801	0	17	*0*
BrasilNO45e	45	5	829	*885*	*829*	1	993	897	0	25	*0*
BrasilNE50e	50	5	814	*877*	*841*	0	963	953	0	31	*0*
Betim100e	100	3	1,394	*1,395*	*1,394*	1	1,642	1,401	0	128	*0*
Vitoria100e	100	5	1,381	*1,414*	*1,381*	8	1,922	1,814	0	98	*0*
PortoVelho200e	200	3	2,382	*2,555*	*2,382*	1	3,134	2,871	0	771	*0*
Cuiaba200e	200	3	2,376	*2,517*	*2,376*	1	3,415	3,052	0	705	*0*
Belem300e	300	4	3,112	*3,482*	*3,112*	1	4,434	4,282	0	2,032	*0*
berlin52eA	52	3	8,948	*8,983*	*8,948*	13	9,094	8,948	4	27	*0*
eil76eB	76	4	1,864	*1,983*	*1,865*	0	2,069	1,986	0	61	*0*
rat99eB	99	5	3,239	*3,304*	*3,242*	0	3,525	3,339	0	128	*0*
rd100eB	100	4	9,933	*9,948*	*9,933*	1	10,385	9,994	0	161	*0*
st70eB	70	4	1,891	*2,030*	*1,907*	0	2,158	2,037	0	54	*0*

Best results are emphasized in italics

Table 6 Comparison of Memetic Algorithms on non-Euclidean instances with processing times fixed by MA1

INSTANCE				MA2			MA1			T(s)	p value
Name	City	Car	#Best	Avg	Best	Freq	Avg	Best	Freq		
BrasilRJ14n	14	2	167	167	167	9	167	167	2	1	*0.01*
BrasilRN16n	16	2	188	*191*	*188*	2	195	190	0	1	*0*
BrasilPR25n	25	3	229	*239*	*229*	1	256	235	0	4	*0*
BrasilAM26n	26	3	204	215	207	0	*212*	*204*	1	5	1
BrasilMG30n	30	4	279	*302*	282	0	328	*279*	1	8	*0*
BrasilSP32n	32	4	265	*282*	*265*	1	296	287	0	13	*0*
BrasilRS32n	32	4	280	*314*	*286*	0	340	304	0	9	*0*
BrasilCO40n	40	5	625	*665*	*625*	1	743	668	0	20	*0*
BrasilNO45n	45	5	613	*680*	*624*	0	764	667	0	32	*0*
BrasilNE50n	50	5	682	*757*	*682*	1	861	707	0	46	*0*
Londrina100n	100	3	1,278	*1,369*	*1,278*	1	1,592	1,471	0	146	*0*
Osasco100n	100	4	1,117	*1,337*	*1,117*	1	1,442	1,150	0	125	*0*
Aracaju200n	200	3	2,281	*2,518*	*2,281*	1	2,744	2,467	0	928	*0*
Teresina200n	200	5	1,971	*2,385*	*2,103*	0	2,551	2,233	0	841	*0*
Curitiba300n	300	5	3,334	*3,845*	*3,344*	0	4,076	3,676	0	2,398	*0*
berlin52nA	52	3	1,385	*1,460*	*1,392*	0	1,642	1,480	0	38	*0*
ch130n	130	5	2,424	*2,751*	*2,490*	0	3,020	2,493	0	237	*0*
d198n	198	4	4,297	*4,890*	*4,431*	0	5,449	4,887	0	828	*0*
kroB150n	150	3	3,402	*3,876*	*3,402*	1	4,259	3,845	0	420	*0*
rd100nB	100	4	1,681	*1,937*	*1,775*	0	2,271	1,890	0	140	*0*

Best results are emphasized in italics

Table 7 Comparison of Memetic Algorithms on Euclidean instances with processing times fixed by MA2

INSTANCE				MA1			MA2			T(s)	p value
Name	City	Car	#Best	Avg	Best	Freq	Avg	Best	Freq		
BrasilRJ14e	14	2	294	294	294	29	294	294	30	1	0.17
BrasilRN16e	16	2	375	375	375	30	375	375	30	1	–
BrasilPR25e	25	3	510	*510*	510	30	522	510	1	2	1
BrasilAM26e	26	3	467	*486*	*467*	4	494	493	0	3	1
BrasilMG30e	30	4	541	597	573	0	*578*	*541*	1	8	0
BrasilSP32e	32	4	601	625	617	0	*621*	*613*	0	7	0
BrasilRS32e	32	4	506	*524*	*506*	1	537	524	0	7	1
BrasilCO40e	40	5	709	811	806	0	*786*	*726*	0	23	*0*
BrasilNO45e	45	5	829	973	910	0	*932*	*848*	0	30	*0*
BrasilNE50e	50	5	814	959	883	0	*869*	*814*	1	35	*0*
Betim100e	100	3	1,394	1,604	1,396	0	*1,425*	1,396	0	247	*0*
Vitoria100e	100	5	1,381	1,729	1,586	0	*1,495*	*1,381*	4	292	*0*
PortoVelho200e	200	3	2,382	2,997	2,764	0	*2,413*	*2,413*	0	1,862	*0*
Cuiaba200e	200	3	2,376	3,166	2,968	0	*2,502*	*2,492*	0	1,682	*0*
Belem300e	300	4	3,112	4,458	3,937	0	*3,672*	*3,298*	0	5,184	*0*
berlin52eA	52	3	8,948	*9,085*	8,948	1	9,090,	8,948	5	43	0.83
eil76eB	76	4	1,864	2,073	1,987	0	*1,946*	*1,864*	1	184	*0*
rat99eB	99	5	3,239	3,495	3,376	0	*3,304*	*3,239*	1	248	*0*
rd100eB	100	4	9,933	10,346	10,047	0	*9,951*	*9,951*	0	255	*0*
st70eB	70	4	1,891,	2,128	1,993	0	*1,970*	*1,891*	1	148	*0*

Best results are emphasized in italics

Table 8 Comparison of Memetic Algorithms on non-Euclidean instances with processing times fixed by MA2

INSTANCE				MA1			MA2			T(s)	p value
Name	City	Car	#Best	Avg	Best	Freq	Avg	Best	Freq		
BrasilRJ14n	14	2	167	168	167	1	*167*	167	6	0	0.17
BrasilRN16n	16	2	188	195	192	0	*191*	*188*	4	1	*0*
BrasilPR25n	25	3	229	260	241	0	*241*	*229*	1	5	*0*
BrasilAM26n	26	3	204	*210*	*206*	0	213	207	0	6	1
BrasilMG30n	30	4	279	333	304	0	*299*	*286*	0	9	*0*
BrasilSP32n	32	4	265	294	280	0	*284*	*275*	0	12	*0*
BrasilRS32n	32	4	280	338	302	0	*311*	*280*	1	13	*0*
BrasilCO40n	40	5	625	747	663	0	*660*	*627*	0	24	*0*
BrasilNO45n	45	5	613	769	723	0	*667*	*613*	1	39	*0*
BrasilNE50n	50	5	682	867	771	0	*736*	*687*	0	48	*0*
Londrina100n	100	3	1,278	1,522,	1,450	0	*1,338*	*1,278*	1	492	*0*
Osasco100n	100	4	1,117	1,419	1,268	0	*1,259*	*1,126*	0	385	*0*
Aracaju200n	200	3	2,281	2,735	2,544	0	*2,446*	*2,331*	0	2,491	*0*
Teresina200n	200	5	1,971	2,526	2,272	0	*2,241*	*1,971*	1	2,514	*0*
Curitiba300n	300	5	3,334	3,983	3,499	0	*3,726*	*3,334*	1	6,599	*0*
berlin52nA	52	3	1,385	1,590	1,489	0	*1,447*	*1,385*	1	86	*0*
ch130n	130	5	2,424	2,998	2,502	0	*2,630*	*2,424*	1	743	*0*
d198n	198	4	4,297	5,309	4,809	0	*4,665*	*4,297*	1	2,756	*0*
kroB150n	150	3	3,402	4,140	3,859	0	*3,675*	*3,491*	0	1,527	*0*
rd100nB	100	4	1,681	2,228	1,927	0	*1,832*	*1,681*	1	464	*0*

Best results are emphasized in italics

Results shown in Table 8 confirm the best performance of MA2 regardless the maximum processing time adopted in the experiments. Column *p value* shows that MA2 and MA1 exhibit significant better performance among each other on 18 and 1 non-Euclidean instances, respectively. Best results on columns *Avg* and *Best* are also obtained by MA2 on 19 and 18 instances. Confidence level 99.999 is pointed out by the proportions comparison test on the number of successes concerning average values.

Once experimental results support MA2 as the best algorithmic approach among the two tested memetic algorithms, it was chosen to be compared with the best GVND version as reported in Sect. 4.3.

4.3 Comparison between GVND2 and MA2

In this section the best algorithms among the ones examined in Sects. 4.1 and 4.2, GVND2 and MA2, are compared. First, the potential of these algorithms in obtaining optimal solutions for a set of small instances is verified. The optimal solutions are obtained with an exact backtracking algorithm. This method implicitly enumerates all possible configurations utilizing permutations of the available cars. The algorithm was implemented in C++ and executed on an Intel Core Duo 1.67 GHz, 2GB RAM running Linux. The algorithm solved eighteen Euclidean and non-Euclidean instances with n between 10 and 16 and 2 cars. Column *Backtrack* of Table 5 shows the value of optimal solutions in column *#Best* and the processing time spent by the exact algorithm in column $T(s)$. Columns *GVND2* and *MA2* show the best solution (column *Best*), the percent deviation from the best solution (column*GAP*) and the average processing time spent by GVND2 and MA2 to reach the optimal solution in thirty independent executions for each instance.

The results presented in Table 9 show that MA2 reaches the optimal solutions of all tested instances while GVND2 misses 5 best results among the 18 instances tested. It can be observed that the success rate of GVND2 is 78% with maximal deviation from the best solution being 3%. The processing times of both heuristic algorithms are significantly smaller than the execution time of the exact algorithm.

Tables 10, 11, 12 and 13 present results on Euclidean and non-Euclidean instances with maximum processing times fixed by GVND2 and MA2. Considering significance level 0.05, p values less than 0.05 indicate that MA2 outperforms GVND2. The best performance of GVND2 over MA2 is indicated with p values greater than 0.95.

Tables 10 and 11 present results for Euclidean instances with maximum processing times fixed by GVND2 and MA2, respectively. Columns *p value* of both tables show that significant differences between the values produced by the tested algorithms do exist for 16 instances when the processing

Table 9 Results of Backtrack, GVND2 and MA2 on small instances

INSTANCE			BACKTRACK		GVND2			MA2		
Name	City	Car	T(s)	#Best	T(s)	Best	GAP	T(s)	Best	GAP
Mauritania10e	10	2	1	*540*	1	*540*	*0.00*	1	*540*	*0.00*
Mauritania10n	10	2	1	*571*	1	*571*	*0.00*	1	*571*	*0.00*
Colombia11e	11	2	19	*620*	1	*620*	*0.00*	1	*620*	*0.00*
Colombia11n	11	2	14	*639*	1	640	*0.00*	1	*639*	*0.00*
Angola12e	12	2	266	*719*	1	*719*	*0.00*	1	*719*	*0.00*
Angola12n	12	2	144	*656*	1	*656*	*0.00*	1	*656*	*0.00*
Peru13e	13	2	1,953	*672*	1	*672*	*0.00*	1	*672*	*0.00*
Peru13n	13	2	1,847	*693*	1	*693*	*0.00*	1	*693*	*0.00*
Libia14e	14	2	31,273	*730*	1	*730*	*0.00*	1	*730*	*0.00*
Libia14n	14	2	28,331	*760*	1	764	0.01	1	*760*	*0.00*
BrazilRJe	14	2	44,104	*294*	1	297	0.01	1	*294*	*0.00*
BrazilRJn	14	2	35,263	*167*	1	170	0.02	1	*167*	*0.00*
Congo15e	15	2	455,788	*756*	1	*756*	*0.00*	1	*756*	*0.00*
Congo15n	15	2	412,212	*886*	1	*886*	*0.00*	1	*886*	*0.00*
Argentina16e	16	2	7603,200	*955*	1	*955*	*0.00*	1	*955*	*0.00*
Argentina16n	16	2	7612,310	*894*	1	*894*	*0.00*	1	*894*	*0.00*
BrasilRN16e	16	2	7609,203	*375*	1	*375*	*0.00*	1	*375*	*0.00*
BrasilRN16n	16	2	7613,217	*188*	1	194	0.03	1	*188*	*0.00*

Best results are emphasized in italics

Table 10 Comparison of GVND2 and MA2 on Euclidean instances with maximum processing time fixed by GVND2

INSTANCE				GVND2			MA2			T(s)	p value
Name	City	Car	#Best	Avg	Best	Freq	Avg	Best	Freq		
BrasilRJ14e	14	2	294	297	297	0	*294*	*294*	30	1	0
BrasilRN16e	16	2	375	*375*	*375*	30	*375*	*375*	30	1	–
BrasilPR25e	25	3	510	*510*	*510*	30	521	520	0	1	*1*
BrasilAM26e	26	3	493	504	501	0	*494*	*494*	0	2	*0*
BrasilMG30e	30	4	541	597	589	0	*596*	*577*	0	2	0.93
BrasilSP32e	32	4	609	*620*	615	0	621	*609*	1	2	0.71
BrasilRS32e	32	4	515	558	556	0	*536*	*515*	1	4	*0*
BrasilCO40e	40	5	700	*761*	733	0	799	*700*	1	7	*1*
BrasilNO45e	45	5	848	*892*	886	0	949	*854*	01	10	*1*
BrasilNE50e	50	5	756	*829*	*756*	1	879	845	0	15	*1*
Betim100e	100	3	1,396	1,761	1,560	0	*1,427*	*1,401*	0	127	*0*
Vitoria100e	100	5	1,381	1,652	1,622	0	*1,580*	*1,398*	0	70	*0*
PortoVelho200e	200	3	2,413	3,272	3,270	0	*2,428*	*2,413*	29	453	*0*
Cuiaba200e	200	3	2,492	3,388	3,388	0	*2,503*	*2,503*	0	349	*0*
Belem300e	300	4	3,298	5,354	5,354	0	*3,701*	*3,701*	0	891	*0*
berlin52eA	52	3	8,948	*8,948*	*8,948*	30	9,112	*8,948*	2	10	*1*
eil76eB	76	4	1,864	2,006	1,914	0	*1,954*	*1,865*	0	82	*0*
rat99eB	99	5	3,206	3,443	*3,206*	1	*3,302*	3,207	0	185	*0*
rd100eB	100	4	9,937	9,972	9,940	0	*9,950*	*9,937*	1	68	*0*
st70eB	70	4	1,877	*1,994*	*1,893*	0	1,997	1,907	0	58	0.42

Best results are emphasized in italics

Table 11 Comparison of GVND2 and MA2 on Euclidean instances with maximum processing time fixed by MA2

INSTANCE				MA2			GVND2			T(s)	p value
Name	City	Car	#Best	Avg	Best	Freq	Avg	Best	Freq		
BrasilRJ14e	14	2	294	*294*	*294*	30	297	297	30	1	*0*
BrasilRN16e	16	2	375	*375*	*375*	30	*375*	*375*	30	1	–
BrasilPR25e	25	3	510	522	*510*	1	*510*	*510*	30	2	*1*
BrasilAM26e	26	3	493	*494*	*493*	3	503	501	0	3	*0*
BrasilMG30e	30	4	541	*578*	*541*	1	594	589	0	8	*0*
BrasilSP32e	32	4	609	621	*613*	0	*619*	615	0	7	*0.99*
BrasilRS32e	32	4	515	*537*	*524*	0	557	556	0	7	*0*
BrasilCO40e	40	5	700	786	*726*	0	*747*	733	0	23	*1*
BrasilNO45e	45	5	848	932	*848*	2	*885*	*848*	0	30	*1*
BrasilNE50e	50	5	756	869	814	0	*811*	*766*	0	35	*1*
Betim100e	100	3	1,396	*1,425*	*1,396*	1	1,758	1,560	0	247	*0*
Vitoria100e	100	5	1,381	*1,495*	*1,381*	4	1,610	1,494	0	292	*0*
PortoVelho200e	200	3	2,413	*2,413*	*2,413*	30	3,270	3,270	0	1,862	*0*
Cuiaba200e	200	4	2,492	*2,502*	*2,492*	1	3,388	3,388	0	1,682	*0*
Belem300e	300	4	3,298	*3,672*	*3,298*	1	5,354	5,354	0	5,184	*0*
berlin52eA	52	3	8,948	9,090	*8,948*	5	*8,948*	*8,948*	30	43	*1*
eil76eB	76	4	1,864	*1,946*	*1,864*	1	1,980	1,934,	0	184	*0*
rat99eB	99	5	3,206	*3,304*	*3,239*	0	3,436,	3,252	0	248	*0*
rd100eB	100	4	9,937	9,951,	9,951	0	*9,949*	*9,940*	0	255	*1*
st70eB	70	4	1,877	1,970	1,891	0	*1,936*	*1,877*	6	148	*0.99*

Best results are emphasized in italics

Table 12 Comparison of GVND2 and MA2 on non-Euclidean instances with maximum processing time fixed by GVND2

INSTANCE				GVND2			MA2			T(s)	p value
Name	City	Car	#Best	Avg	Best	Freq	Avg	Best	Freq		
BrasilRJ14n	14	2	167	170	170	0	*167*	*167*	8	1	*0*
BrasilRN16n	16	2	188	194	194	0	*191*	*188*	7	1	*0*
BrasilPR25n	25	3	229	251	246	0	*240*	*231*	0	4	*0*
BrasilAM26n	26	3	207	231	225	0	*217*	*207*	1	4	*0*
BrasilMG30n	30	4	286	332	312	0	*300*	*288*	0	7	*0*
BrasilSP32n	32	4	274	292	*274*	3	*287*	276	0	7	*0.02*
BrasilRS32n	32	4	280	346	321	0	*316*	*291*	0	7	*0*
BrasilCO40n	40	5	618	688	673	0	*662*	*618*	1	17	*0*
BrasilNO45n	45	5	613	755	700	0	*675*	*617*	0	28	*0*
BrasilNE50n	50	5	670	865	778	0	*736*	*670*	1	41	*0*
Londrina100n	100	3	1,269	*1,356*	1,329	0	1,358	*1,269*	1	228	*0.97*
Osasco100n	100	4	1,126	1,505	1,272	0	*1,255*	*1,154*	0	240	*0*
Aracaju200n	200	3	2,331	2,788	2,590	0	*2,495*	*2,346*	0	1,189	*0*
Teresina200n	200	5	1,971	4,021	3,710	0	*2,248*	*2,019*	0	975	*0*
Curitiba300n	300	5	3,334	5,474	5,474	0	*3,711*	*3,397*	0	2,721	*0*
berlin52nA	52	3	1,385	1,603	1,506	0	*1,488*	*1,420*	0	36	*0*
ch130n	130	5	2,269	3,423	3,013	0	*2,567*	*2,269*	1	522	*0*
d198n	198	4	4,271	6,931	6,715	0	*4,808*	*4,271*	1	1,610	*0*
kroB150n	150	3	3,445	4,458	4,349	0	*3,791*	*3,445*	1	675	*0*
rd100nB	100	4	1,681	2,001	1,872	0	*1,876*	*1,707*	0	289	*0*

Best results are emphasized in italics

Table 13 Comparison of GVND2 and MA2 on non-Euclidean instances with maximum processing time fixed by MA2

INSTANCE				MA2			GVND2			T(s)	p value
Name	City	Car	#Best	Avg	Best	Freq	Avg	Best	Freq		
BrasilRJ14n	14	2	167	*167*	*167*	6	170	170	0	1	*0*
BrasilRN16n	16	2	188	*191*	*188*	4	194	194	0	1	*0*
BrasilPR25n	25	3	229	*241*	*229*	1	249	246	0	5	*0*
BrasilAM26n	26	3	207	*213*	*207*	3	231	225	0	6	*0*
BrasilMG30n	30	4	286	*299*	*286*	1	330	306	0	9	*0*
BrasilSP32n	32	4	274	*284*	275	0	287	*274*	7	12	0.07
BrasilRS32n	32	4	280	*311*	*280*	1	339	321	0	13	*0*
BrasilCO40n	40	5	618	*660*	*627*	0	684	673	0	24	*0*
BrasilNO45n	45	5	613	*667*	*613*	1	751	713	0	39	*0*
BrasilNE50n	50	5	670	*736*	*687*	0	857	779	0	48	*0*
Londrina100n	100	3	1,269	1,338	*1,278*	0	*1,337*	1,322	0	492	0.44
Osasco100n	100	4	1,126	*1,259*	*1,126*	1	1,468	1,325	0	385	*0*
Aracaju200n	200	3	2,331	*2,446*	*2,331*	1	2,644	2,590	0	2,491	*0*
Teresina200n	200	5	1,971	*2,241*	*1,971*	1	3,922	3,710	0	2,514	*0*
Curitiba300n	300	5	3,334	*3,726*	*3,334*	1	5,474	5,474	0	6,599	*0*
berlin52nA	52	3	1,385	*1,447*	*1,385*	1	1,571	1,506	0	86	*0*
ch130n	130	5	2,269	*2,630*	*2,424*	0	3,286	2,689	0	743	*0*
d198n	198	4	4,271	*4,665*	*4,297*	0	6,964	6,715	0	2,756	*0*
kroB150n	150	3	3,445	*3,675*	*3,491*	0	4,386	4,349	0	1,527	*0*
rd100nB	100	4	1,681	*1,832*	*1,681*	1	1,981	1,829	0	464	*0*

Best results are emphasized in italics

time is established by GVND2 and 19 instances when the processing time is established by MA2. Column *p value* of Table 10 also shows that MA2 outperforms GVND2 on 11 instances, whilst the latter outperforms the former on 5 instances. With the processing time of MA2 fixed for both algorithms, Table 11 shows that MA2 presents the best results on 11 instances, whilst GVND2 presents the best results on 8 instances. The test to compare success rates shows that the confidence level to accept that MA2 performs better than GVND2 is 0.983972 when the processing time is fixed by GVND2 and 0.892580 when the processing times are fixed by MA2. Therefore, adopting significance level 0.05, these results support the conclusion that MA2 outperforms GVND2 when the time is fixed by the GVND2. The same conclusion cannot be drawn when the time is determined by the MA2 with the adopted significance level.

The best average results are presented by MA2 and GVND2 on 13 and 8 instances of Table 10, respectively. A similar result is observed in Table 11 where the former algorithm presents 12 best average results and the latter presents 9 best average results. These numbers of success concerning the average solution does not point any of the two tested algorithms as the best one at significance level 0.05. Nevertheless, the test for proportions comparison does point a significant difference if the success rate concerns the best solution found by each algorithm. Table 10 shows that these rates are 16/20 for MA2 and 6/20 for GVND2. Rates of 17/20 and 7/20 are verified in Table 11 for MA2 and GVND2, respectively. The results observed in Tables 10 and 11 lead to confidence levels 0.999596 and 0.999664 indicating that the success rates of MA2 are higher than the ones presented by GVND2. Therefore, the former algorithm can be considered to outperform the latter regarding the best solutions found on the set of Euclidean instances with significance level less than 0.01.

The results presented in columns *p value* of Tables 12 and 13 show that significant differences exist on 20 and 18 non-Euclidean instances, respectively, when significance level 0.05 is considered. Table 12 shows that MA2 outperforms GVND2 on 19 instances, whilst the latter behaves better than the former on 1 instance. Submitting these results to the one-tailed proportions comparison test, one can conclude, with confidence level 1, that MA2 presents higher success rate than GVND2. A similar conclusion is drawn for data provided in Table 13. Column *p value* shows the best performance of MA2 on 18 instances, whilst GVND2 does not present any significant best result. Similar conclusions can be drawn observing the results presented in columns *Best* and *Avg* of Tables 12 and 13. Therefore, regardless the computational effort fixed on each group of experiments, MA2 significantly outperforms GVND2 on tested non-Euclidean instances.

5 Conclusions

This paper presented the Traveling Car Renter Problem (CaRS), a new generalization of the classic Traveling Salesman Problem. An experimental investigation was carried out to compare two metaheuristic approaches proposed for this new problem: GRASP (Greedy Randomized Search Procedure) hybridized with VND (Variable Neighborhood Descent) and Memetic algorithms. The algorithms were applied to 40 Euclidean and non-Euclidean instances of the CaRSLib benchmark which is proposed for this problem. An exact procedure established the optimal solutions of 4 among the 40 instances, whilst the proposed heuristics established the first upper limits for the remaining 36 instances. Statistical tests are applied to the results generated by the proposed algorithms in order to support conclusions on their behaviors concerning quality of solution.

To establish a fair basis of comparison, the effect of the computational effort demanded by each algorithm is neutralized by comparing them with fixed processing times. These execution times are established in accordance to the requirements of each algorithm for its best performance. Therefore, the proposed algorithms are tested twice, first with the processing times fixed by the best performance of one algorithm and then with the processing times fixed by the best performance of the other. Thus, a superior qualitative behavior can be considered conclusive when it holds for both processing time conditions.

Results of statistical tests for proportions comparison proposed by [30] show that the proposed Memetic algorithm outperforms the hybrid GRASP/VND method on Euclidean and non-Euclidean instances. These tests showed that significant higher success rates are found for MA2 on Euclidean instances concerning the set of results provided on 30 independent executions of each algorithm and the best result achieved on each of the 20 instances. For the set of non-Euclidean instances, confidence level 1 is pointed out by the proportions comparison test for the same previous indicators and also for the rate of average solutions provided by each algorithm.

This paper presented also other six variants for the introduced problem, opening up the topic for future research.

Acknowledgments Paulo Asconavieta would like to thank the financial support provided by CAPES-Brazil. Elizabeth F. G. Goldbarg and Marco C. Goldbarg also acknowledge the CNPq for their partial financial support.

References

1. Amdeberhan T, Manna D, Moll VH (2008) The 2-adic valuation of Stirling numbers. Exp Math 17(1):69–82

2. Applegate DL, Bixby R, Chvátal V, Cook W (1995) Finding Cuts in the TSP, DIMACS technical Report 95-05, Rutgers University. http://www.tsp.gatech.edu/methods/papers/index.html
3. Applegate DL, Bixby R, Chvátal V, Cook W (2006) The traveling salesman problem: a computational study. Princeton University Press, Princeton
4. Car Rental (2008) Web Page. http://thrifty4.com/article.cfm/id/284920
5. Car (2008) Car Rental Business, Global Strategic Business Report Global Industry Analysts, Inc., 278. http://www.researchandmarkets.com/reports/338373/
6. Conover WJ (2001) Practical nonparametric statistics. 3rd edn. Wiley, New York
7. Conrad C, Perlut A (2006) Enterprise Rent-A-Car Hits New Billion-Dollar Revenue Mark for 3rd Consecutive Year, Enterprise rent-a-car. http//www.enterpriseholdings.com/NewsReleases/Enterprise_FYO6_Sept06.pdf
8. Edelstein M, Melnyk M (1977) The pool control system. Interfaces 8(1):21–36
9. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Global Optim 6:109–133
10. Földesi P, Botzheim J (2010) Modeling of loss aversion in solving fuzzy road transport traveling salesman problem using eugenic bacterial memetic algorithm. Memetic Comput. 2:259–271
11. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman and Company, New York
12. Gutin G, Punnen AP (2002) The traveling salesman problem and its variations, Series: *Combinatorial Optimization*. Kluwer Academic Publishers, Dordrecht
13. Hertz A, Schindl D, Zufferey N (2009) A solution method for a car fleet management problem with maintenance constraints. J Heurist 5:425–450
14. Johnson DS, McGeoh LA (2002) Experimental analysis of heuristics for the STSP. In: Guttin G, Punnen A (eds) The Traveling Salesman Problem and its Variations. Kluwer Academic Publishers, Dordrecht, pp 369–443
15. Krarup J (1975) The peripatetic salesman and some related unsolved problems. In: Combinatorial Programming Methods and Applications. Reidel, Dordrecht, pp 173–178
16. Krasnogor N, Smith JE (2005) A tutorial for competent memetic algorithms: model, taxonomy, and design issues. IEEE Trans Evol Comput 9(5):474–488
17. Le MN, Ong Y-S, Jin Y, Sendhoff B (2009) Lamarckian memetic algorithms: local optimum and connectivity structure analysis. Memetic Comput 1:175–190
18. Lia Z, Taob F (2010) On determining optimal fleet size and vehicle transfer policy for a car rental company. Comput Oper Res 37:341–350
19. Mars-Proietti L (2007) Encyclopedia of global industries, 4th edn. Grey House Publishing, New York
20. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24:1097–1100
21. Montemanni R, Barta J, Mastrolilli M, Gambardella LM (2007) The robust traveling salesman problem with interval data. Transp Sci 41(3):366–381
22. Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithm. caltech concurrent computation program. California Institute of Technology, USA
23. Moscato P, Cotta C (2010) A modern introduction to memetic algorithms. In: Gendreau M, Potvin J-Y (eds) Handbook of metaheuristics, international series in operations research & management science, vol 146. Springer, Berlin, pp 141–183
24. Pachon JE, Iakovou B, Ip C, Aboudi R (2003) A synthesis of tactical fleet planning models for the car rental industry. IIE Trans 35(9):907–916
25. Ralphs TK, Kopman L, Pulleyblank WR, Trotter LE (2003) On the capacitated vehicle routing problem. Math Program Ser B 94:343–359
26. Reinelt G (1995) TSPLIB95. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95
27. Resende MGC, Ribeiro CC (2003) Greedy randomized adaptive search procedures. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics, international series in operations research & management science, vol 57. Kluwer Academic Publishers, Dordrecht, pp 141–183
28. Resende MGC, Ribeiro CC (2010) Greedy randomized adaptive search procedures: advances and applications. In: Gendreau M, Potvin J-Y (eds) Handbook of metaheuristics, international series in operations research & management science, vol 146. Springer, Berlin, pp 283–320
29. Russel R (1977) An effective heuristic for the m-tour traveling salesman problem with some side conditions. Oper Res 25:517–524
30. Taillard E, Waelti P, Zuber J (2008) Few statistical tests for proportions comparison. Eur J Oper Res 185:1336–1350
31. Xiong Y, Golden B, Wasil E (2007) The Colorful Traveling Salesman Problem. In: Extending the horizons: advances in computing, optimization, and decision technologies, Operations Research/Computer Science Interfaces Series 37, Springer, USA, pp 115–123
32. Yang Y, Jin W, Hao X (2008) Car rental logistics problem: a review of literature. IEEE International Conference on Service Operations and Logistics, and Informatics. IEEE/SOLI 2008, vol 2, pp 2815–2819
33. Yang Y, Jin W, Hao X (2009) Dynamic pool segmentation model and algorithm in the car rental industry. J Comput 4(12):202–1208