RAIRO-Oper. Res. 55 (2021) S1725–S1746
https://doi.org/10.1051/ro/2020053

RAIRO Operations Research
www.rairo-ro.org

AN ILS-BASED HEURISTIC APPLIED TO THE CAR RENTER SALESMAN PROBLEM

Sávio S. Dias[1,*], Luidi G. Simonetti[1] and Luiz Satoru Ochi[2]

Abstract. The present paper tackles the Car Renter Salesman Problem (CaRS), which is a Traveling Salesman Problem variant. In CaRS, the goal is to travel through a set of cities using rented vehicles at minimum cost. The main aim of the current problem is to establish an optimal route using rented vehicles of different types to each trip. Since CaRS is $\mathcal{NP}$-Hard, we herein present a heuristic approach to tackle it. The approach is based on a Multi-Start Iterated Local Search metaheuristic, where the local search step is based on the Random Variable Neighborhood Descent methodology. An Integer Linear Programming Formulation based on a Quadratic Formulation from literature is also proposed in the current study. Computational results for the proposed heuristic method in euclidean instances outperform current state-of-the-art results. The proposed formulation also has stronger bounds and relaxation when compared to others from literature.

Mathematics Subject Classification. 90C11, 90C59, 90B06.

Received August 26, 2019. Accepted May 17, 2020.

1. Introduction

A broadly studied problem in the Combinatorial Optimization field is the Traveling Salesman Problem (TSP). This $\mathcal{NP}$-Hard problem focus on finding a Hamiltonian cycle at minimum cost in an edge-weighted graph [7]. The TSP suits several real-world situations, and it triggers significant interest in its variants and in the problem itself. Some recent TSP variants introduce the possibility of several salesmen to cover all vertices, known as Multiple TSP [1]. Others introduce optional vertices, known as hotels, in which the salesman may start or finish a trip in order to satisfy a trip length constraint, such as the TSP with Hotel Selection [27]. For a more general survey on TSP and its variants, see Ilavarasi and Joseph [14].

A frequently studied area in recent Combinatorial Optimization literature is the car rental business, due to its significant growth in the last few years [8, 23]. Most of this literature concentrates on the companies' side, aiming: profit maximization [25]; vehicle fleet planning [8, 13, 15, 20]; or vehicle demand forecast [5]. However, the consumer side has been less prioritized. Thus, a recently assessed TSP variant regarding this matter is the Car Renter Salesman Problem (CaRS) [10].

Keywords. Traveling salesman, heuristics, integer programming, transportation, manufacture processes.

[1] PESC/Coppe – Federal University of Rio de Janeiro, Rio de Janeiro, Brazil.
[2] Computing Institute – Fluminense Federal University, Niterói, Brazil.
*Corresponding author: sdias@cos.ufrj.br

The CaRS concerns tourists who are willing to visit some places (or cities) at minimum cost. Such cost depends on the type of rented vehicle and some extra fees. CaRS was initially proposed as a classical TSP generalization, having applications on tourism, transport, and manufacturing fields, modeling critical situations in the vehicle renting department for tourism, and also on flexible manufacturing [9].

Initially described as a tourism-driven problem, CaRS has its application drawn directly from the Car Rental Industry. Albeit, this is an oversimplified scope for this problem, as CaRS suits in public transportation and manufacture processes. In public transportation, the vehicles symbolize different options of transportation, *e.g.*, subways, buses, trains, airlines. While in manufacturing processes, different machines can perform equivalent tasks but with different performances and at different costs, depending on some criteria. CaRS also suits in Multilayer Circuits design, as specific robots with different setup costs better assemble some components. Hence, in this paper, we consider the nomenclature and definitions of the problem as firstly proposed in the literature, *i.e.*, in a tourism-driven outlook.

In CaRS, the renter (or tourist) must travel through a set of places or cities using a vehicle. There is a set of vehicle types available, and the tourist can choose any car for any trip. We herein understand "trip" as a fraction of the Hamiltonian Cycle traveled using the same vehicle type. Thus, all trips together lead to a solution. Notice that, by definition, intersections between different trips are not allowed. Accordingly, many variants may emerge by taking some constraints into account: vehicle type availability, places to return a vehicle type, number of uses for a vehicle type, and a version wherein the vehicle return fee is a time function.

Another way of seeing CaRS is comparing it with the Clustered TSP [2], where the fundamental difference between the two problems is that in the Clustered TSP, the total number of used vehicles is given, represented by the clusters. Thus, vertices in a cluster match vertices for a vehicle to cover. One could also compare CaRS with the Multiple TSP, where the return fee could account for the weight of returning to the depot vertex. Also note that, in the Multiple TSP, every vehicle has a set starting point, and the number of vehicles used is a problem constant. The Colorful TSP [28] also shares similarities with CaRS, contrasting that edges have only one possible vehicle or transport company (color), the goal is to minimize the number of colors, and there is no continuity guarantee in the colors of incident edges.

Since CaRS is a generalization of the TSP, where one must choose the set of optimal vehicles to cover edges in the Hamiltonian cycle, its optimization version is also $\mathcal{NP}$-Hard [10]. Thus, the task of solving CaRS is non-trivial, since there is no known algorithm able to find optimal solutions in polynomial time for such class of problems. Therefore, as exact approaches require a high computational effort to solve this problem, making them prohibitive in most cases, the use of heuristics stands as an alternative. In that scenario, metaheuristics attempt to solve the problem, often leading to high-quality solutions in a shorter computational time.

There are some heuristics available in the literature to deal with CaRS. A GRASP (Greedy Randomized Adaptive Search Procedure) using VND (Variable Neighborhood Descent) as a local search method was presented along with a Memetic Algorithm by Goldbarg *et al.* [10] to deal with euclidean and non-euclidean instances. The Transgenetic Algorithm [11] is a metaheuristic inspired by the mutualistic intracellular endosymbiotic evolution. It is the state-of-the-art of the euclidean instances concerning this problem. More recently, da Silva and Ochi [3] proposed an Evolutionary Algorithm (EA) and a Mixed Integer Linear formulation. However, this formulation worked only as a local search mechanism within the EA algorithm, leading to a hybrid heuristic, which is the current state-of-the-art of the non-euclidean instances.

Goldbarg *et al.* [11] also proposed a Quadratic Formulation, which optimally solved instances up to 16 vertices and two vehicles. The authors have reported the formulation as linearized, but with no linearization method present in the paper. Also, the formulation lacks crucial constraints for truly modeling the problem. Hence, this paper will further explore this formulation as a way to fix it. More recently, Goldbarg *et al.* [12] presented an exact study with three new formulations for CaRS. In this study, the first formulation proposed intends to correct the one presented by Goldbarg *et al.* [11]. The second one wrongly formulates CaRS, enforcing any feasible solution to use at least two vehicles, while the last assumes that there are only non-negative renting coefficients. This paper presents results for those formulations by testing them in a set of 50 instances. da Silva Menezes *et al.* [4] presented a CaRS variant with prize collection along with a Memetic Algorithm.

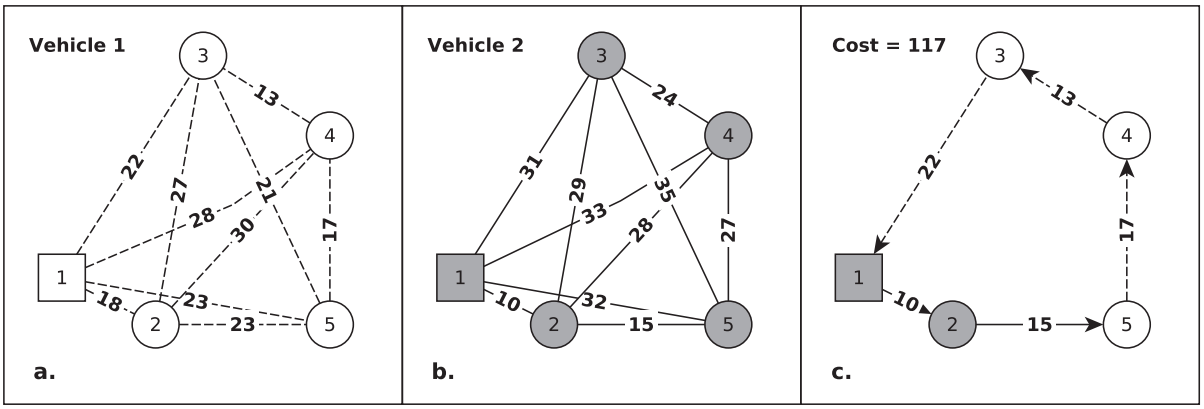

FIGURE 1. Operational costs and feasible solution for an instance with $n = 5$ and $|C| = 2$. (a) Vehicle 1. (b) Vehicle 2. (c) Cost = 117.

The following section of this paper presents a formal definition of the problem at hand (Sect. 2). An Integer Linear Formulation study is described in Section 3, followed in Section 4, by the heuristic methods developed. Section 5 presents computational results for both approaches, along with comparisons to the current state-of-the-art. Finally, the conclusion and some insights about future research are available in Section 6.

2. Problem description

Given a graph $G = (V, E)$, where V ($|V| = n$) is a set of vertices representing cities and E is the set of edges between any pair of vertices in V. A set of different vehicle types C is available in each city. For any $c \in C$ and any $i, j \in V$:

- D_{ij}^c: operational cost for traveling from i to j using vehicle type c, or weight of edge (i, j) using c;
- F_{ij}^c: return fee to be paid if a car type c is rented in vertex i and returned in vertex j.

The goal is to find a Hamiltonian cycle at minimum cost by taking into account the operational costs and return fees. The tourist should start and finish the cycle in the initial vertex. Thus, it is mandatory renting the first vehicle in v_1 and returning the last one in v_1. Outside those restricted cases, the tourist may rent and return vehicles at any vertex, any moment, in its route.

The problem considered in this paper complied with the following features and constraints:

- A vehicle can be rented and returned in any vertex;
- A vehicle type can be rented once at most, *i.e.*, if a vehicle of type c returns, that vehicle should not be rented again. Note that, in this problem, there is only one vehicle per type. So, in this text, both "vehicle" or "vehicle type" descriptions are alike;
- Return fees do not rely on graph topology or any other constraints;
- Operational costs are symmetric, *i.e.*, $D_{ij}^c = D_{ji}^c$. However, there is no guarantee of whether $F_{ij}^c = F_{ji}^c$.

Figure 1 shows an example of an instance as an edge-weighted graph with five vertices and two vehicles. Where, Figures 1a and 1b show the traveling operational costs between each pair of vertices for vehicles 1 and 2, respectively. Also, Figure 1c shows a feasible solution with its respective cost. The returns fees associated with every vehicle and their possible renting/returning place are in Table 1.

In the solution presented in Figure 1c, a vehicle of type 2 covers the first trip starting at vertex 1 (renting), traveling through vertex 2, and finishing in vertex 5 (returning). This trip contributes 35 to the total cost of the solution. The second trip, using vehicle 1, starts where the last trip ended (vertex 5), travels across vertices

TABLE 1. Return fees for instance with $n = 5$ e $|C| = 2$.

	Vehicle 1					Vehicle 2				
Vertices	v_1	v_2	v_3	v_4	v_5	v_1	v_2	v_3	v_4	v_5
v_1	26	26	28	28	30	11	14	10	12	10
v_2	24	24	26	26	28	26	20	28	30	30
v_3	30	30	30	32	34	14	20	24	18	18
v_4	30	30	32	30	34	20	26	22	15	24
v_5	30	36	38	38	30	20	26	22	24	30

4 and 3, and finishes at vertex 1. This trip sums 52 of operational costs plus 30 of return fee, totaling 117 of total solution cost.

It is essential to notice that a feasible solution from classical TSP using any available vehicles is also a feasible solution for CaRS. That happens because one may choose a single vehicle to cover the whole route, and thus making a single trip. For instance, the same route presented in Figure 1c. would cost $93 + 26 = 119$ if traveled using vehicle 1, and $107 + 11 = 118$ if traveled using vehicle 2. Also, if one wishes to produce a solution renting a vehicle type multiple times, thus virtually making multiple vehicles of a type, it suffices to copy its coefficients as a new vehicle in instance data.

3. Integer Linear Programming Formulation

As stated before, the Integer Quadratic Programming Formulation applied to CaRS was proposed by Goldbarg *et al.* [11]. This formulation, referred herein as (P1), has a non-linear objective function and incorrectly models the problem. Those problems were partially assessed in Goldbarg *et al.* [12], while this section explores these details and propose some solutions.

In the following formulation, variable x^c_{ki} takes value 1 if i is visited in k-th order by vehicle c, and 0 otherwise. y^c_i takes value 1 if c is rented in vertex i, and 0 otherwise. Finally, z^c_j takes value 1 if c is returned in vertex j, 0 otherwise.

The original formulation (P1) is non-linear, which makes the application of exact algorithms for integer linear programming impractical. The authors did not report the linearization method implemented. Hence, we propose a linearization in order to avoid this problem. Consider two new sets of variables for this formulation: w^c_{ij}, takes value 1 if vehicle c is rented in i and returned in j, *i.e.*, $y^c_i z^c_j = 1$, 0 otherwise; and h^c_{ij}, takes value 1 if the edge (i, j) is used by vehicle c, *i.e.*, $\sum_{k=1}^{n-1} x^c_{ki} \sum_{c' \in C} x^{c'}_{(k+1)j} = 1$, 0 otherwise. In the new formulation, named as (P2), a vehicle does not visit its returning vertex, as the next vehicle in the route visit this vertex since it is its renting vertex.

$$\text{(P2)}\ \min \sum_{c \in C} \left(\sum_{(i,j) \in E} D^c_{ij} h^c_{ij} + \sum_{i \in V \setminus \{1\}} \sum_{j \in V \setminus \{i\}} F^c_{ij} w^c_{ij} + \sum_{j \in V} F^c_{1j} w^c_{1j} \right) \tag{3.1}$$

$$\text{Subject to:} \sum_{c \in C} \sum_{k=2}^{n} x^c_{ki} = 1, \qquad \forall i \in V \setminus \{1\} \tag{3.2}$$

$$\sum_{c \in C} \sum_{i \in V \setminus \{1\}} x^c_{ki} = 1, \qquad \forall k = 2, \ldots, n \tag{3.3}$$

$$\sum_{c \in C} x^c_{11} = 1 \tag{3.4}$$

$$\sum_{c \in C} y^c_1 = 1 \tag{3.5}$$

$$\sum_{c\in C} z_1^c = 1 \tag{3.6}$$

$$\sum_{c\in C} y_i^c - \sum_{c\in C} z_i^c = 0, \qquad \forall i \in V\setminus\{1\} \tag{3.7}$$

$$\sum_{j\in V\setminus\{i,1\}} h_{ij}^c = \sum_{k=2}^{n-1} x_{ki}^c, \qquad \forall c \in C, \forall i \in V\setminus\{1\} \tag{3.8}$$

$$\sum_{j\in V\setminus\{1\}} h_{1j}^c = x_{11}^c, \qquad \forall c \in C \tag{3.9}$$

$$h_{i1}^c = x_{ni}^c, \qquad \forall c \in C, \forall i \in V\setminus\{1\} \tag{3.10}$$

$$\sum_{j\in V\setminus\{i\}} h_{ji}^c = \sum_{k=2}^{n} x_{ki}^c + z_i^c - y_i^c, \qquad \forall c \in C, \forall i \in V\setminus\{1\} \tag{3.11}$$

$$\sum_{c\in C} h_{1i}^c = \sum_{c\in C} x_{2i}^c, \qquad \forall i \in V\setminus\{1\} \tag{3.12}$$

$$x_{ki}^c + \sum_{\bar{c}\in C} x_{(k+1)j}^{\bar{c}} - 1 \le h_{ij}^c, \qquad \forall c \in C, \forall i \in V, \forall j \in V\setminus\{1\}, \; i \ne j, \forall k = 1,\dots,n-1 \tag{3.13}$$

$$\sum_{c\in C} (h_{ij}^c + h_{ji}^c) \le 1, \qquad \forall i,j \in V, i \ne j \tag{3.14}$$

$$\sum_{j\in V} w_{ij}^c = y_i^c, \qquad \forall c \in C, \forall i \in V \tag{3.15}$$

$$\sum_{i\in V} w_{ij}^c = z_j^c, \qquad \forall c \in C, \forall j \in V \tag{3.16}$$

$$\sum_{i\in V} y_i^c \le 1, \qquad \forall c \in C \tag{3.17}$$

$$y_1^c + z_1^c \le 2 - \sum_{j\in V\setminus\{1\}} y_j^{\bar{c}}, \qquad \forall c \in C, \forall \bar{c} \in C\setminus\{c\} \tag{3.18}$$

$$\sum_{k=2}^{n} x_{ki}^c \le \sum_{j\in V} y_j^c - z_i^c, \qquad \forall c \in C, \forall i \in V\setminus\{1\} \tag{3.19}$$

$$y_i^c \le \sum_{k=2}^{n} x_{ki}^c, \qquad \forall c \in C, \forall i \in V\setminus\{1\} \tag{3.20}$$

$$x_{11}^c = y_1^c, \qquad \forall c \in C \tag{3.21}$$

$$\sum_{i\in V\setminus\{1\}} x_{ni}^c = z_1^c, \qquad \forall c \in C \tag{3.22}$$

$$x_{ki}^c \in \{0,1\}, \qquad \forall c \in C, \forall k = 1,\dots,n, \; \forall i \in V \tag{3.23}$$

$$y_i^c \in \{0,1\}, \qquad \forall c \in C, \forall i \in V \tag{3.24}$$

$$z_j^c \in \{0,1\}, \qquad \forall c \in C, \forall j \in V \tag{3.25}$$

$$w_{ij}^c \in \{0,1\}, \qquad \forall c \in C, \forall i,j \in V \tag{3.26}$$

$$h_{ij}^c \in \{0,1\}, \qquad \forall c \in C, \forall i,j \in V, i \ne j. \tag{3.27}$$

Notice that when (P1) was firstly proposed, v_1 was the last visited vertex, *i.e.*, $\sum_{c\in C} x^c_{n1} = 1$. Here, v_1 is the first visited vertex, in order to make the formulation simpler. Also, note that $w_{ii} = 0, \forall i \neq 1$.

The objective function (3.1) in the formulation above aims to minimize the sum of operational costs and return fees of all used vehicles. Now, the proposed sets of variables prevent variable multiplication from the original objective function in (P1). Constraints (3.2) assure that each vertex is visited in a different order and by one vehicle only. Constraints (3.3) follow a generic visitation order for every vertex, except for the initial one set in constraint (3.4). Constraints (3.5) and (3.6) make sure of the rental and returning of only one vehicle in origin. Constraints (3.7) guarantee that, if one returns a vehicle in any vertex other than the origin, this vertex must be renting point of a new vehicle. Constraints (3.8) guarantee an edge leaving any vertex should it be visited by vehicle c, with constraints (3.9) and (3.10) being special cases for the initial vertex. As constraints (3.11) and (3.12) ensure an edge arriving at any visited vertex, should it not be rented there by vehicle c. In constraints (3.13), vehicle c can cover an edge only if j is visited immediately after i, with c also visiting i. Constraints (3.14) ensure that an edge can at most be covered by a vehicle once. Then, constraints (3.15) and (3.16) presents the links among variables y^c_i, z^c_j, and w^c_{ij}. Constraints (3.17) assure that one vehicle, at most, can be rented in every vertex. Constraints (3.18) show that, if there is no other vehicle rented, a vehicle can be rented and returned in v_1. Constraints (3.19) state that a vehicle does not visit the returning vertex itself, though the next vehicle in the route does. Constraints (3.20) and (3.21) assure that if a vehicle visits a vertex, it is renting place eligible. At last, constraints (3.22) state that, if a vehicle returns in v_1, that vehicle alone visit the last vertex in the cycle. Constraints (3.23) to (3.27) are variables integrality constraints.

Constraints (3.17) to (3.22) were presented herein and added to (P2) as a way to cover the missing cases from the original formulation (P1) since, without them, the formulation would not generate a feasible search space and a proper optimal solution. Some examples of constraint absent from (P1) are link among variables x^c_{ki}, y^c_i, and z^c_j; prevention of a vehicle from being rented and returned in the same vertex other than in v_1; multiple rents of a vehicle type, and; rent of multiple vehicles in the same vertex.

4. Proposed algorithm

We propose a heuristic method since there is no knowledge of polynomial-time algorithms for optimally solve the problem at hand. Also, exact methods demand a significantly higher computational cost as instances grow in size, as can be seen in Section 5. In order to tackle CaRS, Algorithm 1 describes a heuristic based on Iterated Local Search (ILS) [17] using a Multi-Start approach (MILS). The MILS algorithms have been successfully used in the literature to tackle several TSP variants [24,26].

Algorithm 1: MILS(input, $ms_{\max}$, $ils_{\max}$, α, seed).

```
1  begin
2      s* ← ∅; f(s*) ← ∞
3      for i ← 1, ..., ms_max do
4          s' ← build_sol(input, α, seed)
5          for j ← 1, ..., ils_max do
6              s' ← outer_RVND(s', seed)
7              if f(s') < f(s*) then
8                  s* ← s'
9              end
10             s' ← perturbation(s', seed)
11         end
12     end
13     Return s*
14 end
```

Initially, the algorithm generates a solution using a greedy randomized criterion for each one of the $ms_{\max}$ iterations based on input data from the instance (line 4). The built solution is then refined through the $ils_{\max}$ applications of a local search method (line 6) and perturbed (line 10) to escape from a local optimum. The acceptance criterion is exclusively elitist in lines 7–9, *i.e.*, a solution generated according to the local search method will only be accepted if it is better than the current optimum.

The following subsections explore in full detail the solution representation, construction, local search, and perturbation methods.

4.1. Solution representation

We define two data structures in order to represent a solution properly:

- `route`: a n-dimensional array with the vertices visitation order, such that `route[i]` $\in V$, $\forall i = 0, \ldots, n-1$;
- `v_pos`: an array of dimension $3 \times |C|$, with the following information about every trip: vertex position inside `route` to rent a vehicle; vertex position inside `route` to return a vehicle; and the vehicle type. This array also keeps vehicles' usage order. The remaining positions are nullified when the number of trips is smaller than the number of vehicles available.

Thus, the solution presented in Figure 1 is represented as `route` $= [1, 2, 5, 4, 3]$ and `v_pos` $= [(0, 2, 2), (2, 5, 1)]$. Notice that the returning point in the last trip is an invalid position in the `route` array. It happens because the last vehicle returns to the first vertex in order to close the cycle.

4.2. Constructive procedure

Algorithm 2 shows the initial solution construction step used in the present paper. This algorithm presents a greedy randomized behavior, like the one proposed by Feo and Resende [6]. The idea of such a method is to build a diversified and well-distributed route in vertices using all vehicle types available. Thus, the local search will have a vast search space for the achievement of better solutions.

Firstly, the algorithm starts the Candidate List (CL) using all vertices except for the v_1 in the graph, and it starts the set of empty trips using all vehicles available (lines 2 and 3). The algorithm assigns an equal number of vertices to every vehicle (lines 4 and 5), except when $\frac{n}{|C|}$ is not integer; in this case, the exceeding vertices go to the last vehicle. Then, a trip is built for every vehicle, renting in v_1 first.

The algorithm chooses a vehicle randomly from the set of empty trips, with the trip set to start from its renting vertex (lines 8–10). A Restrict Candidate List (RCL) is created in the next step (lines 11–14) using all elements in CL satisfying equation (4.1), in order to randomly choose a vertex to return the current vehicle, and rent a new one. Notice that, if there is only one vehicle left, its returning place shall be v_1 due to problem definition. The r_1 is the returning vertex of vehicle c in the current trip s^c, r_2 is the renting vertex of vehicle c in the current trip, and α is a parameter outlining the minimum requirement to a vertex r become a possible returning place for c in equation (4.1).

$$\text{RCL} = \left\{ r \in \text{CL} \mid F^c_{r_1 r} \leq \min_{j \in \text{CL}} (F^c_{r_1 j}) + \alpha \left[\max_{j \in CL} (F^c_{r_1 j}) - \min_{j \in CL} (F^c_{r_1 j}) \right] \right\}. \tag{4.1}$$

Chosen the renting and returning vertices of the vehicle, the trip is then individually set. For that, the algorithm assesses every vertex in the CL through its insertion in every position of the trip. Such assessment is conducted through equation (4.2), based on Penna *et al.* [22], to avoid late insertion of vertices located away from the renting and returning places of the current vehicle.

$$g(c, k, r_1, r_2) = \begin{cases} D^c_{kj} - \gamma(D^c_{r_2 k} + D^c_{k r_1}) & \text{if insertion in the beginning} \\ D^c_{ik} - \gamma(D^c_{r_2 k} + D^c_{k r_1}) & \text{if insertion in the end} \\ (D^c_{ik} + D^c_{kj} - D^c_{ij}) - \gamma(D^c_{r_2 k} + D^c_{k r_1}) & \text{if insertion in the middle} \end{cases}. \tag{4.2}$$

Algorithm 2: build_sol(input, α, seed).

```
1  begin
2      Initialize CL
3      Let S = {s^1, ..., s^|C|} be a set of empty trips
4      for i ← 1, ..., |C| − 1 do  w_{s^i} ← ⌊n/|C|⌋
5      w_{s^|C|} ← ⌈n/|C|⌉
6      r_1 ← v_1 ; S̄ ← ∅
7      while S ≠ ∅ do
8          Select s^c ∈ S at random
9          S ← S − {s^c}
10         s^c ← s^c ∪ {r_1}
11         if S = ∅ then  RCL ← {v_1}
12         else  Create RCL
13         r_2 ← r_1
14         r_1 ← random element from RCL
15         CL ← CL −{r_1}
16         while CL ≠ ∅ and |s^c| < w_{s^c} do
17             k' ← argmin_{k∈CL} {g(c, k, r_1, r_2)}
18             s^c ← s^c ∪ {k'}
19             CL ← CL −{k'}
20         end
21         S̄ ← S̄ ∪ s^c
22     end
23     Return S̄
24 end
```

Equation (4.2) evaluates possible positions to insert vertex k, where i and j refer to the previous and following vertices referring to the position of vertex k, respectively. Parameter γ weights the distance influence of vertex k on the renting and returning vertices if inserted in the current trip, and takes a value at random from set $\{0.00, 0.05, 0.10, \ldots, 1.65, 1.70\}$. Previous experiments in Subramanian and Battarra [26] defined this set.

A vertex $k' \in CL$ representing the lowest function value $g(c, k, r_1, r_2)$ will be added to its corresponding position in trip s^c and removed from CL. Thus, recur on this step until s^c reached its maximum number of vertices (lines 16–20), then adding it to the end of route $\bar{S}$ (line 21). Finally, with the trips of all vehicles, the algorithm returns $\bar{S}$ as a solution.

4.3. Local search

For the local search methodology, we propose a random Variable Neighborhood Descent. It is a classical VND variation proposed by Mladenović and Hansen [18]. In this methodology, the order of application of the neighborhoods is random rather than manual, as in classical VND. In the literature, Penna *et al.* [22] used a similar approach.

The neighborhood structures are divided into two groups in the present paper, depending on their application goal, namely outer-trip and inner-trip. The outer-trip neighborhoods make moves in the reference solution s by taking into account trips performed by different vehicles, whereas the inner-trip neighborhoods only account for moves on the same trip. The following subsections will further detail the implemented neighborhoods.

Algorithm 3 shows the methodology that sets the application order of the outer-trip neighborhoods randomly (line 2). With `NUM_OUT_NEIGH` being the amount of outer-trip neighborhoods, in every iteration the randomly chosen i-th neighborhood is applied to the reference solution (line 5). If the reference solution improves using the current neighborhood, then an inner-trip RVND will be applied over the recently found best solution leading

to a neighborhood count reset (lines 6–9). When the algorithm finds no best solution through the current neighborhood, the next randomly set outer-trip neighborhood is applied (line 10). The execution finishes and leads to the best solution found so far when all outer-trip neighborhoods are applied, finding no improvement.

Algorithm 3: outer_RVND(s, seed).

```
1  begin
2      OuterNeighborhoodList ← random_start(seed)
3      i ← 0
4      while i < NUM_OUT_NEIGH do
5          s′ ← apply_neigh(OuterNeighborhoodList[i], s)
6          if f(s′) < f(s) then
7              s ← s′
8              s ← inner_RVND(s, seed)
9              i ← 0
10         else i ← i + 1
11     end
12     Return s
13 end
```

If Algorithm 3 improves the reference solution in outer-trip fashion, it calls Algorithm 4 in order to attempt to improve the reference solution for each trip individually. Algorithm 4 has a similar working to that of Algorithm 3, but using only inner-trip neighborhoods and its amount (`NUM_IN_NEIGH`). Also, returning the best neighbor.

Algorithm 4: inner_RVND(s, seed).

```
1  begin
2      InnerNeighborhoodList ← random_start(seed)
3      i ← 0
4      while i < NUM_IN_NEIGH do
5          s′ ← apply_neigh(InnerNeighborhoodList[i], s)
6          if f(s′) < f(s) then
7              s ← s′
8              i ← 0
9          else i ← i + 1
10     end
11     Return s
12 end
```

4.3.1. Outer-trip neighborhoods

We present in this paper nine outer-trip neighborhoods. Of those, seven are a λ-interchange adaptation. The λ-interchange schemes are classical Vehicle Routing Problem (VRP) [19] neighborhood structures. In this paper, we propose two new neighborhoods as an attempt to change the size of the trips and vehicle usage.

The neighborhood structures use exhaustive search, taking all possible combinations (or neighbors) into account, in order to obtain the best solution, *i.e.*, the best improvement strategy. Neighborhoods based on the λ-interchanges do not consider the renting or returning vertices for modifications. Let s^x and s^y be trips such that $s^x \neq s^y$ since the following neighborhoods apply only to outer-trips. These neighborhoods consider all pairs of trips. Thus, we define the neighborhood structures as follow:

– `Swap(1,1)`: swaps one vertex $v_i \in s^x$ with another vertex $v_j \in s^y$. Thus, applying the move, $v_i \in s^y$ and $v_j \in s^x$.

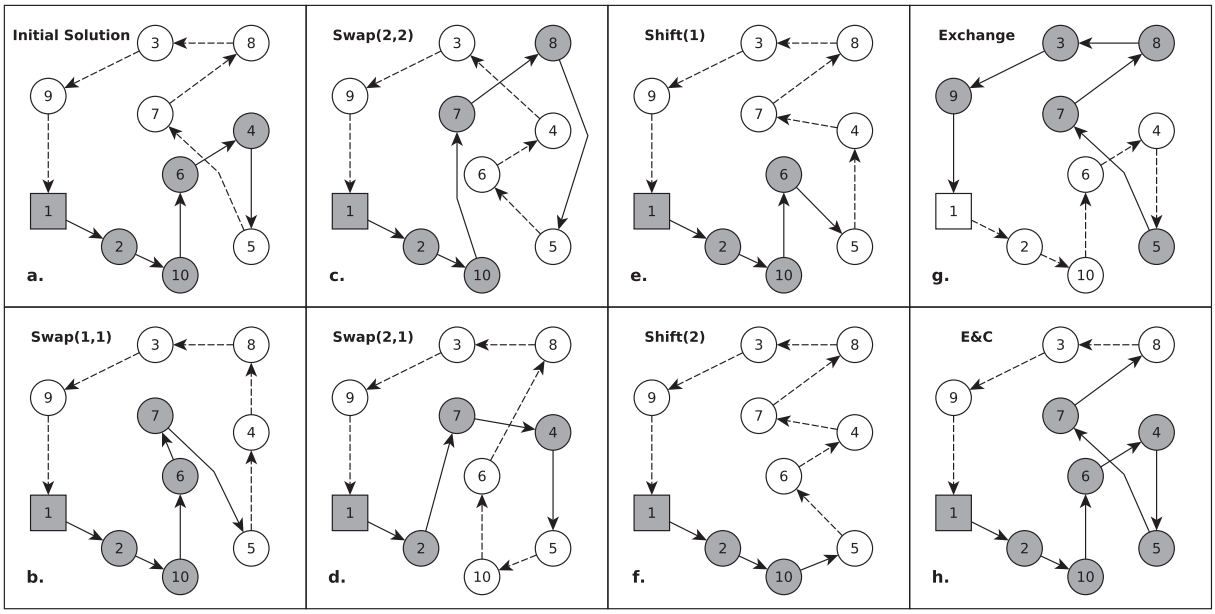

FIGURE 2. Outer-trip neighborhoods. (a) Initial solution. (b) `Swap(1,1)`. (c) `Swap(2,2)`. (d) `Swap(2,1)`. (e) `Shift(1)`. (f) `Shift(2)`. (g) `Exchange`. (h) `E&C`.

- `Swap(2,2)`: swaps two adjacent vertices $v_i, v_{i+1} \in s^x$ with two other adjacent vertices $v_j, v_{j+1} \in s^y$. Thus, applying the moves, $v_i, v_{i+1} \in s^y$ and $v_j, v_{j+1} \in s^x$.
- `Swap(3,3)`: swaps three adjacent vertices $v_i, v_{i+1}, v_{i+2} \in s^x$ with three other adjacent vertices $v_j, v_{j+1}, v_{j+2} \in s^y$. Thus, applying the moves, $v_i, v_{i+1}, v_{i+2} \in s^y$ and $v_j, v_{j+1}, v_{j+2} \in s^x$.
- `Swap(2,1)`: swaps two adjacent vertices $v_i, v_{i+1} \in s^x$ with one vertex $v_j \in s^y$. Thus, applying the moves, $v_i, v_{i+1} \in s^y$ and $v_j \in s^x$.
- `Shift(1)`: shifts one vertex $v_i \in s^x$ to trip s^y. All positions in the trip are considered to insert $v_i \in s^y$, except for the renting and/or returning places.
- `Shift(2)`: shifts two adjacent vertices $v_i, v_{i+1} \in s^x$ to trip s^y. All positions in the trip are considered to insert $v_i, v_{i+1} \in s^y$, except for the renting and/or returning places.
- `Shift(3)`: shifts three adjacent vertices $v_i, v_{i+1}, v_{i+2} \in s^x$ to trip s^y. All positions in the trip are considered to insert $v_i, v_{i+1}, v_{i+2} \in s^y$, except for the renting and/or returning places.
- `Vehicle Exchange`: swaps trips between two vehicles. Let c_i be the vehicle performing trip s^x and c_j performing trip s^y. Thus, by applying the move, vehicle c_i will perform trip s^y, and vehicle c_j will perform trip s^x. Notice that, when this neighborhood applies some moves, one must reevaluate the whole solution, because of the vehicles renting and returning vertices change, as well as edge weights in each trip.
- `Extend & Contract (E&C)`: modifies the renting and returning places of adjacent trips. Evaluation should occur to the moves in order to address every vertex changed in the trip. The idea is to extend the trip of one vehicle and contract the trip of another.

Figure 2 holds examples of some outer-trip neighborhoods with Figure 2a presenting an initial solution for reference in every neighborhood move applied. Also, Figure 3 displays the changes made in the data structures used to represent the solution after every neighborhood move. In Figure 2b, trips 1 and 2 swaps vertices 4 and 7 through `Swap(1,1)` while in Figure 2c, `Swap(2,2)` swaps vertices 6 and 4 by vertices 7 and 8 of their trips. Figure 2d performs a `Swap(2,1)` example where vertices 10 and 6 are swapped by vertex 7. `Shift(1)` and `Shift(2)` examples are in Figures 2e and 2f, where vertex 4 is shifted to a position after vertex 5 in trip 2,

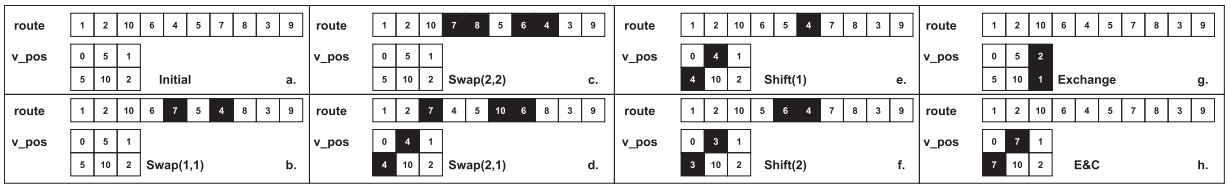

FIGURE 3. Solution representation changes for outer-trip neighborhoods. (a) Initial. (b) `Swap(1,1)`. (c) `Swap(2,2)`. (d) `Swap(2,1)`. (e) `Shift(1)`. (f) `Shift(2)`. (g) `Exchange`. (h) `E&C`.

in first example, and vertices 6 and 4 is shifted to a position after vertex 5 in trip 2. Figure 2g shows an example of the `Vehicle Exchange` neighborhood and Figure 2h presents an `Extend & Contract (E&C)` instance. In this case, the first trip extends from vertex 5 (vehicle returning place) to vertex 8. Thus, the vehicle performing trip 1 will be rented at vertex 1 and returned at vertex 8, whereas the vehicle of trip 2 will be rented at vertex 8 and returned at vertex 1.

It is crucial emphasizing that exchanging vertices between trips guides the search for a better overall route, in edge cost, as this favors vehicles with better costs for some portion of the graph. In that direction, neighborhoods that exchange vehicle types or their renting/returning places better contribute to the setup costs. Also, due to the exchanging nature of these neighborhoods, they are prone to perform better when the triangle inequality holds, *i.e.*, for euclidean instances. While one may find that difficult to achieve in non-euclidean instances, as seen in experimental results.

The size of neighborhoods based on λ-interchanges is $\mathcal{O}(n^2)$. Neighborhood `Vehicle Exchange` has size $\mathcal{O}(|C|^2)$ and neighborhood `E&C`, $\mathcal{O}(|C|)$. Regarding the neighborhoods herein proposed, which handle trip changes directly, a computational effort of $\mathcal{O}(n)$ stands due to the need for reevaluation of a non-constant portion of the solution. On the other hand, in the other neighborhoods, this reevaluation process may be conducted in $\mathcal{O}(1)$.

4.3.2. Inner-trip neighborhoods

We propose eight inner-trip neighborhoods. They are all based on the classical λ-interchanges proposed for the VRP. The best improvement strategy is also employed to search for better solutions. Excepting for `Reverse`, all other neighborhoods do not consider the renting and returning vertices. Thus:

- `Swap(1,1)`, `Swap(2,2)`, `Swap(3,3)`, `Shift(1)`, `Shift(2)`, `Shift(3)`: similar to those described in the previous section, however, the moves are performed in the vertices of the same trips, only, *i.e.*, $s^x = s^y$.
- `2-Opt`: deletes two non-adjacent edges and adds another two edges to the selected trip in order to make a new route. That may change orientation on a portion of the trip since some vertices will be visited early but at no solution cost whatsoever.
- `Reverse`: reverses the trip of a vehicle c, with the trip now starting at its previous returning place and finishing at its previous renting place. This neighborhood is valid only if $F^c_{ij} > F^c_{ji}$, because reversing the orientation in vehicle trips itself does not affect the solution cost, since they are symmetric. Thus, no other modification, other than its start and finish places, is made on the trip. Notice that this neighborhood is only applied if $|C| > 2$ because changing the renting place from the first trip and the returning place from the last trip would result in an infeasible solution.

Figure 4 presents examples of an initial solution (Fig. 4a), as well as moves made by neighborhoods `2-Opt` (Fig. 4b) and `Reverse` (Fig. 4c). Moreover, Figure 5 illustrates the changes made in the data structures used to represent the solution after every move. In Figure 4b, trip 1 has edges $(6, 4)$ and $(5, 7)$ removed with new edges $(6, 5)$ and $(4, 7)$ added to it. While in Figure 4c, trip 2 has a reverse move performed, where vertex 8 is the new

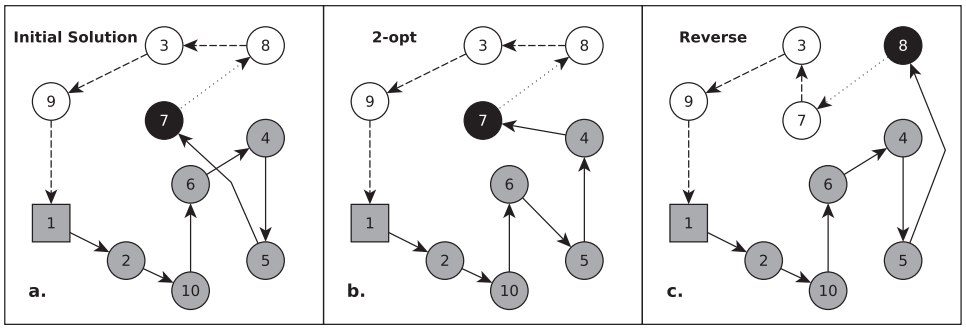

FIGURE 4. Inner-trip neighborhoods. (a) Initial solution. (b) `2-opt`. (c) `Reverse`.

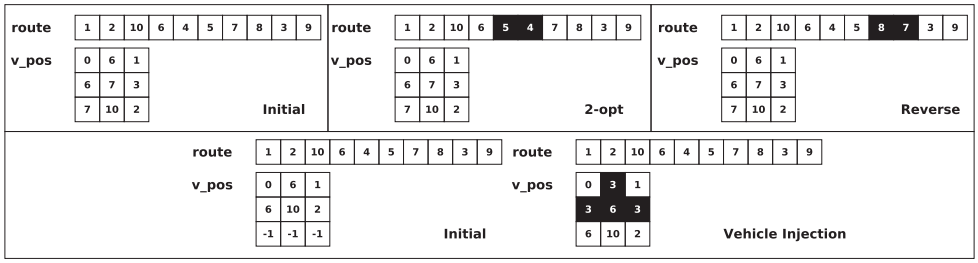

FIGURE 5. Solution representation changes for inner-trip neighborhoods and perturbation.

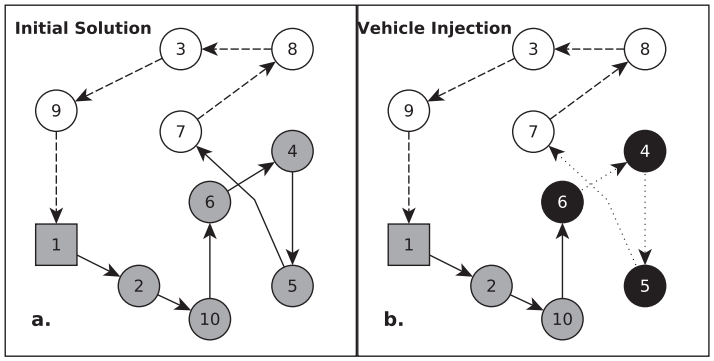

FIGURE 6. Vehicle injection perturbation. (a) Initial solution. (b) `Vehicle Injection`.

renting place of the vehicle. Size of neighborhoods `Swap`, `Shift`, and `2-Opt` is $\mathcal{O}(n^2)$, with a constant time for solution reevaluation. As for neighborhood `Reverse`, it has a size of $\mathcal{O}(C)$, with $\mathcal{O}(n)$ for solution reevaluation.

4.4. Perturbation

We adapt three outer-trip neighborhood structures as perturbation mechanisms: `Shift(1)`, `Swap(1,1)`, and `Vehicle Exchange`. The vertices and trips (or vehicles) in these structures are randomly selected, in opposition to the improvement guiding local search. Thus, the move is unpredictably applied and, in most cases, getting worse solutions, but allowing escape from local optima.

We also propose a new perturbation mechanism called `Vehicle Injection`. This perturbation starts by selecting two vehicles. Let $s^x = \{v_k, v_{(k+1)}, v_{(k+2)}, \ldots, v_{(k+p)}\} \in S$ be the trip covered by the vehicle x in a feasible solution S, where v_i and v_p is the renting and returning place, respectively; also let y be a vehicle available in the instance that is not performing any trip in solution S, thus $s^y = \varnothing$. The idea is to build trip s^y, inserting vehicle y into the solution, using a portion of vertices from s^x. For that, select at random a starting point to s^y, either v_k or $v_{(k+p)}$ from s^x, suppose v_k. Then, randomly assign the number of vertices in s^x to be covered by s^y, suppose m. Thus, after the perturbation, $s^x, s^y \in S$, where $s^x = \{v_{(k+m)}, v_{(k+m+1)}, \ldots, v_{(k+p)}\}$ and $s^y = \{v_k, v_{(k+1)}, \ldots, v_{(k+m)}\}$. Figure 6 shows the injection of a new vehicle on the first trip. The injected trip starts in the returning point (vertex 7), covering three vertices from the first trip. Among the four perturbation herein presented, a single one is randomly chosen and applied in every iteration. See Figure 5 for changes reference in solution representation.

5. Computational results

The heuristic tests happened in an architecture Intel i7-870 2.93 GHz with 8 GB RAM, while the exact approaches occurred in the same architecture with 16 GB RAM. The operating system used was Ubuntu 14.04 64-bit. All approaches are coded in C++ language and compiled with g++ 4.8.2. All tests performed in a single thread. The integer linear programming model ran using the IBM ILOG CPLEX 12.6.1 framework as a solver platform. For tests, CPLEX standard preprocessing, heuristics, and cuts parameters were disabled.

A preliminary test using the *irace* package [16] with 20 arbitrarily-chosen instances helps define the MILS parameters. These training instances, taken from the CaRSlib (`http://www.dimap.ufrn.br/lae/en/projects/CaRS.php`), vary in size and are separated from those used for testing and comparisons. It was possible running 500 experiments in the *irace* configuration. The Restricted Candidate List outlining parameter α was set to take values in interval $\alpha \in [0.01, 0.99]$, the Multi-Start iterations limit parameter $ms_{\max} \in \{5, 6, \ldots, 29, 30\}$ and the ILS iterations limit parameter $ils_{\max} \in \{20, 21, \ldots, 69, 70\}$. The returned values are $\alpha = 0.47$, $ms_{\max} = 28$ and $ils_{\max} = 61$.

Test with the methods proposed also occurred in another set of instances, proposed by Goldbarg *et al.* [11] and used in tests by da Silva and Ochi [3] and Goldbarg *et al.* [12]. Those instances were provided through e-mail by authors. Explanation of instances configuration and generation methods is in the first paper. The instances for this problem follow a euclidean and a non-euclidean pattern in order to define distances between vertices. Consider the euclidean set to assess the heuristic method proposed, in comparison to current state-of-the-art, since the EA algorithm proposed by da Silva and Ochi [3] did not lead to good results when tested in this set.

In order to check the efficiency of the neighborhoods presented in Section 4.3, Table 2 shows the results of a study. The tests carried in this study used a subset of 20 instances from the ones in the literature. This study presents two sets of experiments. One uses each neighborhood structure individually as the local search for the heuristic, starting from the same initial solution. The other set of experiments runs the full RVND, as described in the previous section, and enumerates pieces of information about every neighborhood when used.

Associated to each neighborhood in Table 2, the columns in Single mean: Sol, the average solution obtained on those 20 instances executing the neighborhood as single local search method; T (s), the average time needed to reach the solution using the neighborhood as single local search; Avg (%), the average of proportional improvement applying the neighborhood to a given reference solution during execution as single local search. For the columns RVND: #Usage, the number of times in which the RVND uses the referred neighborhood for improving a reference solution inside RVND; Avg (%), the average of marginal improvement when applying the neighborhood to a reference solution inside RVND as a whole, *i.e.*, given the number of times it was called inside RVND, how much (on average) does the neighborhood improved the reference solution.

Table 2 shows that all neighborhoods have a similar average computational time, when executed standalone, except for the neighborhoods `Exchange` and `E&C`. This behavior is explained due to these neighborhoods' nature, since they modify a more meaningful portion of the route, thus requiring a more considerable effort in recalculating the solution cost. The extra effort rewards with an average improvement higher than other neighborhoods.

TABLE 2. Neighborhoods' efficiency study.

Neighborhood	Single			RVND	
	Sol	T (s)	Avg (%)	#Usage	Avg (%)
Outer-trip					
`Swap(1,1)`	6453.55	1.52	9.69	2665	3.18
`Swap(2,2)`	6417.15	2.80	10.29	862	5.32
`Swap(2,1)`	6392.60	1.45	10.46	2283	4.11
`Swap(3,3)`	6474.95	1.56	11.34	516	7.75
`Shift(1)`	5942.80	1.34	8.45	8052	1.73
`Shift(2)`	6256.30	2.14	8.73	3230	2.87
`Shift(3)`	6151.00	1.56	9.32	1994	4.18
`Exchange`	6760.75	3.01	12.43	139	36.88
`E&C`	6661.95	7.18	11.37	725	6.59
Inner-trip					
`Swap(1,1)`	6763.90	1.93	2.73	11 889	0.68
`Swap(2,2)`	6750.05	1.11	2.47	4552	0.95
`Swap(3,3)`	6889.00	1.53	2.15	3572	1.19
`2-opt`	6666.80	1.03	2.29	21 859	0.89
`Reverse`	6959.30	2.44	0.98	277	1.08
`Shift(1)`	6639.55	1.36	3.08	12 321	0.72
`Shift(2)`	6695.65	1.24	2.71	11 764	0.91
`Shift(3)`	6756.30	1.38	2.64	8038	1.07

It is worth noticing the general inefficiency of Inner-trip neighborhoods, as a single, in achieving a good average solution when compared to the Outer-trip's. However, these neighborhoods are more used than outer-trips due to their definition, limiting application to each trip separately.

Furthermore, neighborhood `Exchange` has the best average improvement, both as single and in full RVND setting. Although, this neighborhood is the least used because of the rarity of cases in which it applies. Also, note it has the worse solution value when used as a single local search due to its nature that does not allow much neighborhood depth.

Table 3 shows the results obtained with the formulation proposed in the current paper for solving instances with a small number of vertices. The goal is to prove optimality of some instances and show that results found through MILS are of high quality and of shorter computational time than exact methods.

Instance information is in the column Instance of the tables in the current section. The name and number of vertices are listed, respectively, in columns Name and n, while the number of vehicles in $|C|$. Columns Reported (P1) present the results as reported by Goldbarg *et al.* [11], who used formulation (P1) (Tab. 3). Even with some demonstrated missing constraints, the reported results are feasible, according to previous e-mail contact with the authors regarding this subject. Those results reportedly used the GNU GLPK solver version 4.47 in a computer presenting architecture Intel i5 with 8 GB RAM, running an Ubuntu Linux 12.04 64-bit operating system. Columns Formulation (P2) show results obtained through formulation (P1) after corrections and linearization herein proposed. Columns MTZ [3] present results obtained running the formulation proposed by da Silva and Ochi [3] as an exact approach in this paper since that formulation served only as a local search procedure before. Finally, the columns MILS show heuristic results for the appropriate instances in a single execution. Those columns show the best UB from each method, *i.e.*, the best feasible solution found, and the amount of time (in seconds) needed, T (s). The best upper bound or proved optimal of each instance is boldface highlighted. The instances where the upper bound value contains an α symbol mean an execution terminated by out of memory.

TABLE 3. Results for small instances.

Instance			Reported (P1)		Formulation (P2)		MTZ [3]		MILS	
Name	n	$\|C\|$	UB	T (s)	UB	T (s)	UB	T (s)	UB	T (s)
Mauritania10e	10	2	**540**	0.20	**540**	0.03	**540**	0.15	**540**	0.01
Mauritania10n	10	2	**571**	0.90	**571**	0.03	**571**	3.43	**571**	0.02
Colombia11e	11	2	**620**	2.30	**620**	2.17	**620**	0.22	**620**	0.02
Colombia11n	11	2	**639**	0.20	**639**	0.06	**639**	42.68	**639**	0.03
Angola12e	12	2	**719**	2.50	**719**	1.70	**719**	0.58	**719**	0.02
Angola12n	12	2	**656**	17.40	**656**	0.16	**656**	1074.56	**656**	0.02
Peru13e	13	2	**672**	1.00	**672**	0.12	**672**	0.34	**672**	0.03
Peru13n	13	2	**693**	17.40	**693**	0.10	**693**	305.31	**693**	0.03
Libia14e	14	2	**730**	13.50	**730**	0.10	**730**	2.58	**730**	0.03
Libia14n	14	2	**760**	247.90	**760**	0.12	**760**	27 856.00	**760**	0.06
Congo15e	15	2	**756**	2.50	**756**	2.72	**756**	3.23	**756**	0.06
Congo15n	15	2	**886**	0.80	**886**	0.29	**886**	25 867.79	**886**	0.04
Argentina16e	16	2	**955**	45.90	**955**	67.53	**955**	16.13	**955**	0.08
Argentina16n	16	2	**894**	176.50	**894**	2.08	897$^{\alpha}$	52 969.02	**894**	0.06
BrasilRJ14e	14	2	**294**	49.20	**294**	5.66	**294**	19.59	**294**	0.02
BrasilRJ14n	14	2	**167**	32.00	**167**	0.53	**167**	6716.38	**167**	0.03
BrasilRN16e	16	2	**375**	31.80	**375**	1.02	**375**	39.45	**375**	0.03
BrasilRN16n	16	2	**188**	1.30	**188**	0.73	**188**	83.08	**188**	0.09
BrasilPR25e	25	3	509	70 000.00	**508**	24.45	**508**	5019.14	**508**	0.34
BrasilPR25n	25	3	227	70 000.00	**226**	23.36	227$^{\alpha}$	55 188.31	**226**	1.41
BrasilAM26e	26	3	469	70 000.00	**467**	2935.92	**467**	54 406.41	**467**	0.08
BrasilAM26n	26	3	**202**	70 000.00	**202**	2182.27	201$^{\alpha}$	47 677.81	**202**	1.04
BrasilMG30e	30	4	547	70 000.00	**529**	245.74	542	70 000.00	**529**	0.88
BrasilMG30n	30	4	276	70 000.00	275	70 000.00	272$^{\alpha}$	62 798.61	**271**	2.71
BrasilSP32e	32	4	656	70 000.00	610	70 000.00	841$^{\alpha}$	3352.53	**588**	1.64
BrasilSP32n	32	4	261	70 000.00	**254**	5254.49	266	70 000.00	**254**	2.81

Formulation (P2) was able to find all previously proved optimal solutions and to prove optimal of other instances under significantly smaller computational effort. The MILS was able to reach all previously and newly proved optimal solutions, as well as to propose new upper bounds to instances not yet optimally solved. Nevertheless, the exact methods start to struggle when the number of vehicle types rises, *e.g.*, instance `BrasilSP32e`. As can be seen in Table 3, even with the difference of architecture of (P1), our linearized formulation (P2) has shown to be competitive and more robust than the others. When ran through an exact method, MTZ formulation seems to be unstable and have memory issues, although able to find optimal solution for some instances.

Following the recent results in the literature, Table 4 presents a comparison between (P2) and the two best formulations presented by Goldbarg *et al.* [12]. Those formulations were also implemented in the same environment as (P2) with CPLEX tuning parameters disabled, in order to check formulations' strength. As employed in Goldbarg *et al.* [12], this experiment set a time limit of 10 000 s. In this table, the column GAP accounts for the relative difference between upper bound and lower bound found when the execution ended, either by timeout, out of memory, or optimality criteria. Column LR accounts for the CPLEX linear relaxation in the root node of the branch-and-bound tree. Note that this information may not be the pure linear relaxation of the model itself since even with preprocessing and other CPLEX parameters turned off, the black box solver may use unknown private information to boost its processes.

As seen in Table 4, (P2) still has good results for those new instances tested, especially when one compares its linear relaxation with the UB. However, the formulations proposed by Goldbarg *et al.* [12] have better overall

TABLE 4. Comparison with the results from Goldbarg *et al.* [12].

Instance		Formulation (P2)				DFJ [12]				GG [12]			
Name	$\|C\|$	UB	GAP	T (s)	LR	UB	GAP	T (s)	LR	UB	GAP	T (s)	LR
Arabia14e	5	**851**	0.00	2.78	820.47	**851**	0.00	23.86	695.81	**851**	0.00	10.11	695.85
Argelia15e	3	**840**	0.00	44.41	752.00	**840**	0.00	12.34	670.57	**840**	0.00	59.83	671.21
Argentina16e	2	**955**	0.00	56.22	895.00	**955**	0.00	4.27	832.33	**955**	0.00	76.26	772.67
Australia16e	4	**1051**	0.00	1276.32	917.00	**1051**	0.00	394.43	839.73	**1051**	0.00	652.36	847.87
BrasilAM26e	3	485	5.37	10 000.00	430.17	**467**	0.00	753.83	393.32	483	4.48	10 000.00	399.36
BrasilMG30e	3	**529**	0.00	3093.94	498.75	**529**	0.00	1864.27	425.41	556	15.42	10 000.00	435.59
BrasilRJ14e	2	**294**	0.00	3.46	284.50	**294**	0.00	5.72	220.15	**294**	0.00	15.01	224.31
Canada17e	4	**1251**	0.00	190.66	1162.00	**1251**	0.00	172.11	1032.50	**1251**	0.00	405.81	1036.81
Cazaquistao15e	5	**904**	0.00	18.16	847.17	**904**	0.00	38.30	734.50	**904**	0.00	343.19	733.71
China17e	3	**1003**	0.62	10 000.00	879.13	**1003**	0.00	325.87	785.25	**1003**	0.00	4277.38	789.06
EUA17e	2	**912**	0.00	9.06	838.00	**912**	0.00	0.58	786.63	**912**	0.00	20.35	802.31
India16e	3	**1035**	0.00	12.63	1003.00	**1035**	0.00	2.75	869.47	**1035**	0.00	17.59	867.87
Indonesia14e	3	**799**	0.00	1.48	782.17	**799**	0.00	0.76	711.31	**799**	0.00	3.51	714.31
Libia14e	2	**730**	0.00	1.47	702.00	**730**	0.00	0.63	651.23	**730**	0.00	4.29	652.46
Mexico14e	4	**789**	0.00	12.34	733.00	**789**	0.00	9.47	652.23	**789**	0.00	35.52	647.69
Russia17e	5	**1061**	0.00	185.59	936.97	**1061**	0.00	1756.53	849.88	**1061**	0.00	1349.48	860.56
Sudao15e	4	**823**	0.00	3.87	792.50	**823**	0.00	34.26	666.86	**823**	0.00	41.34	672.50
att48eA	3	–	–	10 000.00	32 385.00	–	–	2983.21$^{\alpha}$	27 398.31	**38 779**	25.22	10 000.00	28 366.43
att48eB	4	–	–	10 000.00	32 334.87	–	–	5710.24$^{\alpha}$	27 338.51	–	–	10 000.00	28 305.36
berlin52eA	3	–	–	10 000.00	8052.50	–	–	3845.97$^{\alpha}$	7213.47	**13 528**	43.21	10 000.00	7383.22
berlin52eB	4	–	–	10 000.00	7863.00	–	–	3453.48$^{\alpha}$	6956.19	**13 868**	47.02	8539.11	7143.06
eil51eA	3	–	–	10 000.00	1169.50	**1508**	20.65	10 000.00	1043.54	1755	37.24	10 000.00	1078.08
eil51eB	5	–	–	10 000.00	1096.00	–	–	4383.61$^{\alpha}$	933.40	–	–	10 000.00	950.50
pr76eA	3	–	–	10 000.00	99 978.50	–	–	8080.90$^{\alpha}$	78 560.15	–	–	10 000.00	84 283.25
pr76eB	4	–	–	10 000.00	100 146.83	–	–	3711.58$^{\alpha}$	78 785.12	–	–	10 000.00	84 509.89
Arabia14n	5	**1026**	0.00	5.00	918.00	**1026**	0.00	110.31	633.42	**1026**	0.00	2141.77	635.08
Argelia15n	3	**863**	0.00	2.82	782.50	**863**	0.00	2728.67	651.71	865	12.02	10 000.00	650.57
Argentina16n	2	**894**	0.00	3.80	892.00	**894**	0.00	0.35	775.67	**894**	0.00	4924.42	714.27
Australia16n	4	**1061**	0.00	28.31	976.00	**1061**	0.00	1865.64	819.17	1070	13.52	10 000.00	819.33
BrasilAM26n	3	**202**	0.00	4952.43	190.75	**202**	0.00	9.64	178.32	203	8.61	10 000.00	178.88
BrasilMG30n	3	309	12.62	10 000.00	256.33	**279**	8.54	10 000.00	226.85	282	16.92	10 000.00	227.31
BrasilRJ14n	2	**167**	0.00	0.64	166.50	**167**	0.00	0.21	140.65	**167**	0.00	71.10	140.15
Canada17n	4	**1136**	0.00	49.02	1069.50	**1136**	0.00	1239.41	776.75	**1136**	0.00	9229.08	781.00
Cazaquistao15n	5	**1043**	0.00	15.52	936.65	**1043**	0.00	8244.63	723.57	**1043**	2.22	10 000.00	730.14
China17n	3	**918**	0.00	9.58	843.50	934	2.85	10 000.00	781.50	926	12.62	10 000.00	778.69
EUA17n	2	**822**	0.00	6.04	822.00	**822**	0.00	0.89	731.38	**822**	0.00	9496.63	728.63
India16n	3	**985**	0.00	21.15	928.50	**985**	0.00	135.56	724.17	**985**	0.00	3952.49	724.80
Indonesia14n	3	**796**	0.00	2.44	721.00	**796**	0.00	7.92	600.46	**796**	0.00	510.71	600.77
Libia14n	2	**760**	0.00	0.12	760.00	**760**	0.00	0.22	650.31	**760**	0.00	1210.62	649.69
Mexico14n	4	**902**	0.00	4.72	781.00	**902**	0.00	287.96	616.77	**902**	0.00	6223.15	616.15
Russia17n	5	**1094**	0.00	1832.99	944.50	**1094**	15.15	10 000.00	746.38	**1094**	19.40	10 000.00	747.81
Sudao15n	4	**1020**	0.00	7.66	891.17	**1020**	0.00	1706.32	711.71	1022	11.86	10 000.00	709.00
att48nA	3	–	–	10 000.00	942.83	**1015**	5.66	10 000.00	871.38	1018	11.14	10 000.00	895.06
att48nB	4	–	–	10 000.00	722.67	**839**	13.89	10 000.00	602.06	1098	42.15	10 000.00	620.38
berlin52nA	3	–	–	10 000.00	1227.50	**1348**	6.30	10 000.00	1117.86	–	–	10 000.00	1136.20
berlin52nB	4	–	–	10 000.00	789.75	**908**	16.19	10 000.00	670.82	–	–	10 000.00	678.24
eil51nA	3	–	–	10 000.00	948.75	**1094**	8.88	10 000.00	844.90	1201	26.07	10 000.00	869.70
eil51nB	5	–	–	10 000.00	832.38	**955**	17.20	10 000.00	717.64	–	–	10 000.00	732.54
pr76nA	3	–	–	10 000.00	1143.00	**5094**	77.19	5141.55$^{\alpha}$	1096.24	–	–	10 000.00	1093.60
pr76nB	4	–	–	10 000.00	1306.42	**1458**	11.96	4052.28$^{\alpha}$	1197.32	–	–	10 000.00	1218.27

performances for instances with more number of vertices. Table 5 presents a broader study comparing (P2) and formulation DFJ from Goldbarg *et al.* [12]. In this study, the MILS heuristic UB is given to the solver for each formulation, comparing those formulations performance, and its gaps from LR. For Table 5, column: LB is the best lower bound obtained when the execution ended; Heuristic presents the MILS heuristic upper bound given to the formulation; LR (%) presents the gap from the linear relaxation in root node to the given UB; LR (s) presents the time, in second, spent to solve the LR; Nodes presents the number of nodes of the branching tree.

The results presented in Table 5 shows how the linear relaxation of Formulation (P2) gives a better edge to the branch and bound. An example of this is in non-euclidean instances, where the more considerable gap between the LR and Heuristic UB was 8.82%. Thus, at the end of the time limit, it has a smaller gap between the bounds than those from Formulation DFJ, being able to solve five instances in the root node optimally. The number of nodes generated in the branching tree is also significantly shorter in Formulation (P2), which in the case of larger instances, leads to an out-of-memory problem for Formulation DFJ. Finally, Formulation (P2) was also able to reach higher lower bounds in all instances, except one. Finally, Formulation (P2) showed competitive results against state-of-the-art, being also robust when dealing with larger instances in comparison with state-of-the-art.

For every instance, the heuristic executes 30 times using different seeds, with the seed value set as the execution number in the discrete interval $[1, 30]$. Other approaches from the literature also ran 30 executions. Avg is the average result in all executions. Best is the best solution for all executions. T (s) is the average computational time required to find the reported solution. GAP $= 100(S_1 - S_2)/S_2$, wherein: S_1 is the MILS result, and S_2 is the best result reported in the literature (either best or average solution) for the corresponding instance, which is the relative difference between two heuristics in the following tables. The heuristics presenting the best results in every instance are boldface highlighted.

Table 6 shows the average and best results of the euclidean instances found in 30 executions of the method proposed in the present paper (MILS). The results are compared to a Memetic Algorithm (MA) [10] and to a Transgenetic Algorithm (TA) [11], which is the state-of-the-art of euclidean instances. The gap here measures over average results for all approaches. Those results from the literature were also coded in C++ and found in a computer presenting architecture Intel Xeon QuadCore W3520 2.8 GHz with 8 GB RAM running a Scientific Linux 5.5 64-bit operating system.

Some observations can be drawn from Table 6 since the performance of architectures from both MA/TA and MILS are comparable [21]. Table 6 shows that MILS leads to high-quality bounds demanding a computational time much smaller than previous heuristics. Average GAP compared to the state-of-the-art is negative, where 24 out of 30 instances account for better average performance, a fact that required about four times shorter computational time. The results show that MILS reaches the best solutions in 25 out of 30 instances, thus providing new best solutions to 10 of them.

The MILS assesses non-euclidean instances as a complimentary study in order to find its performance alongside existing literature. Table 7 shows the best solution results of non-euclidean instances. The results are compared to TA and to a hybrid Evolutionary Algorithm (EA + ALSP) [3], which is the state-of-the-art of non-euclidean instances. These EA + ALSP results were found in a computer presenting architecture Intel i7 3630-QM 2.4 GHz with 8 GB RAM running a Windows 8.0 64-bit operating system. CPLEX 12.6.1 was also used as a MIP optimizer in this method.

The MILS shows competitive results, although the goal of the current study is to propose a heuristic able to tackle euclidean instances, in comparison to non-euclidean instances previously found in the literature [21]. Although the solution lost in quality against the non-euclidean EA + ALSP heuristic, it was still able to reach good solutions in shorter computational time, with an average GAP of 0.75%. MILS also outperformed TA in the non-euclidean instances and was able to prove optimality in two new instances.

At last, the results presented in this section explored the problem thoroughly. In the heuristic approach, the MILS has proven competitive with current state-of-the-art, outperforming previous literature for euclidean instances, while struggling in solution quality for non-euclidean instances. It happened despite no major neighborhood invention, leaving an opportunity for a broader neighborhood study that better suits the heuristic.

TABLE 5. Comparison using heuristic UB.

Instance			Formulation (P2)						DFJ [12]					
Name	$\|C\|$	Heuristic	LB	T (s)	LR	LR (%)	LR (s)	Nodes	LB	T (s)	LR	LR (%)	LR (s)	Nodes
Arabia14e	5	851	**851.00**	1.55	823.00	3.40	0.43	79	**851.00**	2.63	695.81	22.30	0.07	1861
Argelia15e	3	840	**840.00**	28.44	782.00	7.42	0.38	3625	**840.00**	8.06	670.57	25.27	0.05	9663
Argentina16e	2	955	**955.00**	26.35	895.00	6.70	0.10	4772	**955.00**	1.53	832.33	14.74	0.05	2920
Australia16e	4	1051	**1051.00**	1430.78	925.00	13.62	0.78	89 259	**1051.00**	236.28	839.73	25.16	0.06	191 164
BrasilAM26e	3	467	448.24	10 000.00	430.00	8.60	0.26	163 458	**467.00**	53.94	393.32	18.73	0.09	27 256
BrasilMG30e	3	529	**529.00**	50.38	496.00	6.65	8.56	12	**529.00**	8554.69	425.41	24.35	0.16	2 716 995
BrasilRJ14e	2	294	**294.00**	1.29	283.06	3.86	0.10	258	**294.00**	0.51	220.15	33.55	0.01	1004
Canada17e	4	1251	**1251.00**	713.64	1162.00	7.66	0.83	20 362	**1251.00**	166.66	1032.50	21.16	0.07	141 987
Cazaquistao15e	5	904	**904.00**	8.37	872.50	3.61	0.84	626	**904.00**	29.56	734.50	23.08	0.07	22 129
China17e	3	1003	**1003.00**	2906.55	881.00	13.85	0.51	128 185	**1003.00**	645.65	785.25	27.73	0.06	797 773
EUA17e	2	912	**912.00**	32.60	838.00	8.83	0.13	4776	**912.00**	0.23	786.62	15.94	0.05	327
India16e	3	1035	**1035.00**	1.17	1012.00	2.27	0.22	92	**1035.00**	1.49	869.47	19.04	0.06	1760
Indonesia14e	3	799	**799.00**	0.40	787.50	1.46	0.13	47	**799.00**	0.30	711.31	12.33	0.05	296
Mexico14e	4	789	**789.00**	0.89	768.75	2.63	0.21	18	**789.00**	8.55	652.23	20.97	0.06	9443
Libia14e	2	730	**730.00**	0.65	704.00	3.69	0.08	175	**730.00**	0.62	651.23	12.10	0.04	1296
Russia17e	5	1061	**1061.00**	540.52	963.75	10.09	0.80	19 288	**1061.00**	244.45	849.87	24.84	0.08	136 919
Sudao15e	4	823	**823.00**	0.39	807.50	1.92	0.38	0	**823.00**	31.33	666.86	23.41	0.06	31 556
att48eA	3	34 571	**32 397.00**	10 000.00	32 397.00	6.71	110.39	6493	29 438.39$^{\alpha}$	16 598.37	27 372.81	26.30	0.24	2 222 595
att48eB	4	34 582	**32 347.17**	10 000.00	32 347.17	6.91	209.47	1803	27 547.55$^{\alpha}$	2566.39	27 338.51	26.50	0.31	365 383
berlin52eA	3	8948	**8068.75**	10 000.00	8068.75	10.90	167.98	1302	7626.45	10 000.00	7213.47	24.05	0.25	2 157 610
berlin52eB	4	8751	**7871.90**	10 000.00	7871.90	11.17	251.76	728	7305.45$^{\alpha}$	2918.96	6956.19	25.80	0.37	301 201
eil51eA	3	1356	**1249.17**	10 000.00	1188.00	14.14	179.21	5438	1155.53	10 000.00	1043.54	29.94	0.26	2 183 582
eil51eB	5	1307	**1113.17**	10 000.00	1113.17	17.41	302.94	436	999.70	10 000.00	933.40	40.03	0.44	560 395
pr76eA	3	109 617	**99 990.50**	10 000.00	99 990.50	9.63	1174.58	44	79 284.82$^{\alpha}$	3407.87	78 560.15	39.53	0.57	220 855
pr76eB	4	110 098	**100 170.75**	10 000.00	100 170.75	9.91	2284.81	4	79 617.12	10 000.00	78 785.12	39.74	0.81	402 365
Arabia14n	5	1026	**1026.00**	0.64	1026.00	0.00	0.64	0	**1026.00**	993.38	633.42	61.98	0.06	1 130 480
Argelia15n	3	863	**863.00**	0.64	863.00	0.00	0.61	1	**863.00**	7713.62	651.71	32.42	0.05	14 380 709
Argentina16n	2	894	**894.00**	0.82	892.00	0.22	0.29	77	**894.00**	0.06	775.67	15.26	0.05	10
Australia16n	4	1061	**1061.00**	0.60	1061.00	0.00	0.51	5	1047.86	10 000.00	819.17	29.52	0.06	11 970 401
BrasilAM26n	3	202	**202.00**	88.86	202.00	0.00	9.62	1310	**202.00**	13.38	178.32	13.28	0.10	7852
BrasilMG30n	3	274	**270.00**	10 000.00	270.00	1.48	43.07	32 212	246.67	10 000.00	226.84	20.79	0.15	6 785 959
BrasilRJ14n	2	167	**167.00**	0.27	167.00	0.00	0.10	38	**167.00**	0.06	140.65	18.73	0.04	17
Canada17n	4	1148	**1136.00**	6.97	1130.00	0.53	1.02	393	1031.50	10 000.00	776.75	46.25	0.06	9 685 491
Cazaquistao15n	5	1043	**1043.00**	0.82	1043.00	0.00	0.82	0	**1043.00**	6651.36	723.57	44.15	0.07	6 854 498
China17n	3	918	**918.00**	0.53	864.50	6.19	0.53	0	893.12	10 000.00	781.50	17.47	0.06	11 761 975
EUA17n	2	822	**822.00**	0.46	822.00	0.00	0.30	6	**822.00**	0.08	731.37	12.39	0.05	47
India16n	3	985	**985.00**	0.88	983.00	0.20	0.36	67	**985.00**	408.27	724.17	36.02	0.05	694 074
Indonesia14n	3	796	**796.00**	0.58	787.50	1.08	0.14	58	**796.00**	195.95	600.46	32.57	0.05	396 107
Libia14n	2	760	**760.00**	0.09	698.37	8.82	0.09	0	**760.00**	0.08	650.31	16.87	0.04	35
Mexico14n	4	907	**902.00**	3.48	850.25	6.09	0.27	452	**902.00**	4978.71	616.77	46.25	0.06	7 311 706
Russia17n	5	1126	**1061.48**	10 000.00	1029.00	6.32	1.62	522 477	895.19	10 000.00	746.37	47.25	0.08	11 277 422
Sudao15n	4	1020	**1020.00**	4.40	954.90	6.82	0.33	329	902.11	10 000.00	711.71	43.32	0.06	12 239 984
att48nA	3	993	**976.00**	10 000.00	976.00	1.74	101.98	10 968	930.55	10 000.00	894.38	11.03	0.32	4 265 126
att48nB	4	811	**786.50**	10 000.00	786.50	3.12	654.26	1521	689.38$^{\alpha}$	4244.98	622.98	30.18	0.38	765 498
berlin52nA	3	1308	**1297.00**	10 000.00	1297.00	0.85	381.35	4366	1216.79	10 000.00	1135.23	15.22	0.32	3 128 172
berlin52nB	4	877	**855.50**	10 000.00	855.50	2.51	521.42	102	749.29	10 000.00	681.71	28.65	0.47	2 179 392
eil51nA	3	1075	**1044.25**	10 000.00	1043.87	2.98	162.11	1414	941.26	10 000.00	878.26	22.40	0.39	3 243 362
eil51nB	5	903	**876.08**	10 000.00	870.69	3.71	453.84	2697	749.86$^{\alpha}$	3105.30	732.04	23.35	0.66	337 925
pr76nA	3	1197	**1187.67**	10 000.00	1187.67	0.79	2039.23	6	1136.19	10 000.00	1096.24	9.19	1.05	1 006 711
pr76nB	4	1404.00	**1343.80**	10 000.00	1343.80	4.48	2237.43	50	1222.71$^{\alpha}$	4833.98	1218.27	15.25	1.33	140 890

TABLE 6. Results for euclidean instances.

Instance			MA			TA			MILS			
Name	n	$\|C\|$	Avg	Best	T (s)	Avg	Best	T (s)	Avg	Best	GAP	T (s)
BrasilRJ14e	14	2	**294**	**294**	1.00	**294**	**294**	1.00	**294.00**	**294**	**0.00%**	0.02
BrasilRN16e	16	2	**375**	**375**	1.00	**375**	**375**	1.00	**375.00**	**375**	**0.00%**	0.03
BrasilPR25e	25	3	523	510	2.00	**508**	**508**	12.00	**508.00**	**508**	**0.00%**	0.34
BrasilAM26e	26	3	477	468	3.00	**467**	**467**	13.00	**467.00**	**467**	**0.00%**	0.08
BrasilMG30e	30	4	549	530	8.00	532	**529**	26.00	**530.33**	**529**	**−0.31%**	0.88
BrasilSP32e	32	4	606	**588**	7.00	593	**588**	27.00	**590.57**	**588**	**−0.41%**	1.64
BrasilRS32e	32	4	496	494	7.00	493	**491**	24.00	**491.00**	**491**	**−0.41%**	0.86
BrasilCO40e	40	5	696	672	23.00	676	**668**	51.00	**673.87**	**668**	**−0.32%**	5.55
BrasilNO45e	45	5	857	**829**	30.00	840	**829**	70.00	**830.93**	**829**	**−1.08%**	3.22
BrasilNE50e	50	5	766	**756**	35.00	763	**756**	70.00	**758.23**	**756**	**−0.62%**	7.21
Livramento30e	30	3	**739**	**739**	2.00	**739**	**739**	16.00	**739.00**	**739**	**0.00%**	0.14
Pelotas50e	50	3	1288	1274	17.00	1265	1249	77.00	**1244.20**	**1244**	**−1.64%**	4.35
BoaVista80e	80	4	1725	1707	49.00	1666	1591	350.00	**1579.40**	**1560**	**−5.20%**	28.55
Betim100e	100	3	1401	**1394**	247.00	1408	**1394**	348.00	**1396.40**	**1394**	**−0.33%**	58.01
Vitoria100e	100	5	**1357**	**1354**	292.00	1362	**1354**	382.00	1376.23	**1354**	1.42%	62.63
JoaoPessoa140e	140	4	2557	2521	249.00	2461	2368	1297.00	**2276.80**	**2243**	**−7.48%**	174.54
Natal160e	160	5	2792	2753	298.00	2688	2641	1944.00	**2506.90**	**2486**	**−6.74%**	379.16
PortoVelho200e	200	3	2375	2327	1862.00	2376	2312	3165.00	**2294.50**	**2284**	**−3.39%**	553.00
Cuiaba200e	200	3	2398	2329	1682.00	2332	2275	3039.00	**2237.17**	**2215**	**−4.07%**	737.85
Belem300e	300	4	3095	3007	5184.00	3056	**2985**	9649.00	**3028.70**	2997	**−0.89%**	3339.81
att48eA	48	3	**34 572**	**34 571**	14.00	34 643	**34 571**	39.00	34 596.60	**34 571**	0.07%	1.62
berlin52eA	52	3	8950	**8948**	43.00	**8949**	**8948**	59.00	**8949.27**	**8948**	**0.00%**	3.46
ch130e	130	5	8931	8818	190.00	8828	8729	893.00	**8733.97**	**8631**	**−1.07%**	93.33
eil76eB	76	4	1826	1756	184.00	1779	1703	420.00	**1665.30**	**1649**	**−6.39%**	37.76
lin105e	105	5	17 053	**16 916**	100.00	**16 988**	**16 916**	384.00	17 038.27	16 958	0.30%	41.85
pr107e	107	5	46 867	46 827	104.00	**46 840**	**46 812**	394.00	46 925.70	46 836	0.18%	41.07
rat99eB	99	5	3188	3113	248.00	3133	3042	309.00	**2942.83**	**2904**	**−6.07%**	66.04
rd100eB	100	4	9954	**9909**	255.00	**9951**	**9909**	255.00	9961.23	9911	0.10%	33.44
st70eB	70	4	1898	1849	148.00	1858	1777	310.00	**1739.70**	**1708**	**−6.37%**	25.65
w100eB	100	4	**8310**	**8310**	82.00	**8310**	**8310**	331.00	8422.57	8364	1.35%	33.84
Average					378.90			798.53			**−1.65%**	191.20

TABLE 7. Best results for non-euclidean instances.

Instance			TA		EA + ALSP		MILS		
Name	n	$\|C\|$	Best	T (s)	Best	T (s)	Best	GAP	T (s)
BrasilRJ14n	14	2	**167**	2.00	**167**	0.60	**167**	**0.00%**	0.03
BrasilRN16n	16	2	**188**	4.00	**188**	0.60	**188**	**0.00%**	0.09
BrasilPR25n	25	3	**226**	16.00	**226**	10.60	**226**	**0.00%**	1.41
BrasilAM26n	26	3	**202**	16.00	**202**	10.60	**202**	**0.00%**	1.04
BrasilMG30n	30	4	**271**	32.00	**271**	21.30	**271**	**0.00%**	2.71
BrasilSP32n	32	4	**254**	39.00	**254**	26.00	**254**	**0.00%**	3.28
BrasilRS32n	32	4	**269**	40.00	**269**	26.60	**269**	**0.00%**	2.81
BrasilCO40n	40	5	576	84.00	576	56.00	**575**	**−0.17%**	8.96
BrasilNO45n	45	5	551	104.00	548	69.30	**546**	**−0.36%**	12.50
BrasilNE50n	50	5	618	147.00	**611**	98.00	618	1.15%	14.74
Canoas30n	30	4	**376**	37.00	**376**	24.60	**376**	**0.00%**	2.87
Santos50n	50	5	**392**	153.00	**382**	102.00	**382**	**0.00%**	14.48
Macapa80n	80	5	616	624.00	**599**	416.00	608	1.50%	58.27
Londrina100n	100	3	1186	1107.00	**1146**	738.00	1153	0.61%	121.83
Osasco100n	100	4	993	999.00	**964**	666.00	974	1.04%	133.46
Cuiaba140n	140	4	1339	2718.00	**1293**	1200.00	1313	1.55%	367.98
PortoVelho160n	160	3	1426	4595.00	**1382**	1200.00	1399	1.23%	561.09
Aracaju200n	200	3	1942	7349.00	**1839**	2400.00	1868	1.58%	1248.27
Teresina200n	200	5	1410	8884.00	**1343**	2400.00	1369	1.94%	1649.99
Cuiaba300n	300	5	2222	37 391.00	**2100**	3600.00	2129	1.38%	1611.77
att48nA	48	3	993	134.00	**988**	89.30	**988**	**0.00%**	11.92
berlin52nA	52	3	1326	181.00	**1303**	120.60	1305	0.15%	17.46
ch130n	130	5	1696	2831.00	**1632**	1200.00	1664	1.96%	390.05
d198n	198	4	3188	11 993.00	**3036**	2400.00	3069	1.09%	1617.16
kroB150n	150	3	2966	4472.00	**2845**	1200.00	2876	1.09%	494.01
pr107n	107	5	1698	1589.00	**1631**	1054.30	1653	1.35%	221.14
rat99nB	99	5	1399	1310.00	**1349**	873.30	1371	1.63%	141.00
rd100nB	100	4	1412	1251.00	**1357**	834.00	1360	0.22%	156.19
st70nB	70	4	910	415.00	**879**	276.60	904	2.84%	44.57
w100nB	100	4	1670	1166.00	**1615**	777.30	1630	0.93%	175.30
Average				2989.43		729.72		0.75%	302.88

As in the exact approach, the studies comparing the proposed integer linear formulation proved new optimal solutions with its strong linear relaxation. Albeit, that brings some time limit drawbacks in larger instances, as a heavier linear model demands more time to find a feasible solution. So, the tests giving the heuristic upper bound to the CPLEX solver aiming for the best lower bound, concluded that the formulation proposed is better suited.

6. Conclusions

The current paper presented a new approach in an attempt to solve the Car Renter Salesman Problem using a methodology based on the Iterated Local Search metaheuristic, including an intensive local search procedure along with the classical and novel neighborhood structures. This paper also presented a linearization for a quadratic integer formulation along with some sets of constraints that allowed the previous formulation to be optimally solved efficiently using integer linear programming solver, also proving optima in five previously unproven instances. As some instances were optimally solved, it was verified MILS's efficiency in returning good solutions at a significantly shorter computational time when compared to the exact approaches.

We also conducted a study comparing the linearized formulation herein proposed with some new formulations. In this study, the proposed formulation proved to be capable of reaching reasonable solutions, and also has a better linear relaxation when compared to the other ones from the literature. As a way to check that, and to try solving more significant instances for the problem, we set the MILS heuristic upper bound to both formulations. The results have shown that the proposed formulation was able to give better lower bounds than others from literature due to its better linear relaxation, also resulting in a significantly smaller branching tree to be explored.

Tests using euclidean instances proved that MILS was capable of accomplishing its goal as it presented very competitive and efficient results in this set of instances, and outperformed the previous state-of-the-art. In some cases, MILS was even able to set upper bounds better than previous literature demanding smaller computational effort. The MILS was tested in non-euclidean instances and compared to previous literature as a sideway experiment. Although it was not the goal of the present study, the results were positive. Notwithstanding, the heuristic was not able to outperform the state-of-the-art, but it was able to show good quality results in a shorter period-of-time. In some cases, it even presented new best solutions.

Overall, the algorithm showed good results for exploring CaRS. However, there is still room for improvement, mostly to the non-euclidean instances. Different constructive methods may provide different results in this set. It is also worth noticing the effortless adaptability of the proposed method to CaRS variants since it relies on the ILS methodology, a metaheuristic known for its simplicity.

The current state-of-the-art for this problem still has room for improvement in both exact and heuristic approaches. For heuristic approaches, the proposal of new neighborhood structures that cover and refine the presented MILS aiming to tackle non-euclidean instances or a new populational heuristic that works with penalized infeasible solutions. With that in mind, one could trade computational effort for a more extensive search that better explores the solution space. Also, for exact approaches, the proposal of new models based on classical TSP modeling, exploring the key features and differences between these problems. Moreover, one could try different ways to attack the problem on the exact spectrum, exploiting the extensive literature approaches on similar problems. One significant example is the Column Generation employed in several variants of TSP and VRP problems.

Acknowledgements. The authors kindly thank the support provided by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnolóico), FAPERJ (Fundação de Amparo Pesquisa do Estado do Rio de Janeiro), and also the computational resources supplied by Optimization Laboratory (LabOtim) from Coppe/UFRJ. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

References

[1] T. Bektas, The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega* **34** (2006) 209–219.

[2] J.A. Chisman, The clustered traveling salesman problem. *Comput. Oper. Res.* **2** (1975) 115–119.

[3] A.R.V. da Silva and L.S. Ochi, An effcient hybrid algorithm for the Traveling Car Renter Problem. *Expert Syst. App.* **64** (2016) 132–140.

[4] M. da Silva Menezes, M.C. Goldbarg and E.F. Goldbarg, A memetic algorithm for the prize-collecting Traveling Car Renter Problem. In: 2014 IEEE Congress on Evolutionary Computation (CEC). IEEE, New York, NY (2014) 3258–3265.

[5] M. Edelstein and M. Melnyk, The pool control system. *Interfaces* **8** (1977) 21–36.

[6] T.A. Feo and M.G.C. Resende, A probabilistic heuristic for a computational diffcult set covering problem. *Oper. Res. Lett.* **8** (1989) 67–71.

[7] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman Co., New York, NY (1979).

[8] D.K. George and C.H. Xia, Fleet-sizing and service availability for a vehicle rental system via closed queueing networks. *Eur. J. Oper. Res.* **211** (2011) 198–207.

[9] Global Industry Analysts, Inc., Car rental business – global strategic business report, Tech. Rep. 338373, research and markets. http://www.researchandmarkets.com/reports/338373/car_rental_business_global_strategic_business (2014).

[10] M.C. Goldbarg, P.H. Asconavieta and E.F.G. Goldbarg, Memetic algorithm for the traveling car renter problem: an experimental investigation. *Memet. Comput.* **4** (2012) 89–108.

[11] M.C. Goldbarg, E.F. Goldbarg, P.H. Asconavieta, M.d.S. Menezes, H.P. Luna, A transgenetic algorithm applied to the Traveling Car Renter Problem. *Expert Syst. App.* **40** (2013) 6298–6310.

[12] M.C. Goldbarg, E.F. Goldbarg, H.P. Luna, M.S. Menezes and L. Corrales, Integer programming models and linearizations for the Traveling Car Renter Problem. *Optim. Lett.* **12** (2018) 743–761.

[13] A. Hertz, D. Schindl and N. Zufferey, A solution method for a car fleet management problem with maintenance constraints. *J. Heuristics* **15** (2009) 425–450.

[14] K. Ilavarasi and K.S. Joseph, Variants of travelling salesman problem: a survey. In: International Conference on Information Communication and Embedded Systems (ICICES2014). IEEE, New York, NY (2014) 1–7.

[15] Z. Li and F. Tao, On determining optimal fleet size and vehicle transfer policy for a car rental company. *Comput. Oper. Res.* **37** (2010) 341–350.

[16] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle and M. Birattari, The irace package, iterated race for automatic algorithm configuration, *Oper. Res. Perspect.* **3** (2011). DOI: 10.1016/j.orp.2016.09.002

[17] H.R. Lourenco, O.C. Martin and T. Stützle, Iterated local search: framework and applications. In: Handbook of Metaheuristics. Vol 146 of *International Series in Operations Research & Management Science*. Springer, New York, NY (2010) 363–397.

[18] N. Mladenović and P. Hansen, Variable neighborhood search. *Comput. Oper. Res.* **24** (1997) 1097–1100.

[19] I.H. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Ann. Oper. Res.* **41** (1993) 421–451.

[20] J.E. Pachon, E. Iakovou, C. Ip and R. Aboudi, A synthesis of tactical fleet planning models for the car rental industry. *IIE Trans.* **35** (2003) 907–916.

[21] PassMark, Passmark Software Pty Ltd – CPU benchmarks. [Accessed October 24, 2016 by https://www.cpubenchmark.net] (2016).

[22] P.H.V. Penna, A. Subramanian and L.S. Ochi, An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *J. Heuristics* **19** (2013) 201–232.

[23] S. Seay and A. Narsing, Transitioning to a lean paradigm: a model for sustainability in the leasing and rental industries. *Acad. Strategic Manage. J.* **12** (2013) 113.

[24] M.M. Silva, A. Subramanian, T. Vidal and L.S. Ochi, A simple and effective metaheuristic for the minimum latency problem. *Eur. J. Oper. Res.* **221** (2012) 513–520.

[25] C. Steinhardt and J. Gönsch, Integrated revenue management approaches for capacity control with planned upgrades. *Eur. J. Oper. Res.* **223** (2012) 380–391.

[26] A. Subramanian and M. Battarra, An iterated local search algorithm for the Travelling Salesman Problem with Pickups and Deliveries. *J. Oper. Res. Soc.* **64** (2013) 402–409.

[27] P. Vansteenwegen, W. Souffriau and K. Sörensen, The travelling salesperson problem with hotel selection. *J. Oper. Res. Soc.* **63** (2012) 207–217.

[28] Y. Xiong, B. Golden and E. Wasil, The colorful traveling salesman problem. In: Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies. Springer, New York, NY (2007) 115–123.