Evolutionary algorithms for the Traveling Car Renter with Passengers

Gustavo de Araujo Sabry*, Marco Cesar Goldbarg†, Elizabeth Ferreira Gouvêa Goldbarg‡, Matheus da Silva Menezes§ and José Gomes Lopes Filho¶
*†‡¶*Departamento de Informática e Matemática Aplicada*
Universidade Federal do Rio Grande do Norte
Natal, Brazil
Email: guga_sabry@hotmail.com*, gold@dimap.ufrn.br†, beth@dimap.ufrn.br‡, zefilho@msn.com¶
§*Departamento de Ciências Naturais, Matemática e Estatística*
Universidade Federal Rural do Semi-Árido
Mossoró, Brazil
Email: matheus@ufersa.edu.br§

***Abstract*—The Traveling Car Renter with Passengers (*CaRSP*) is a generalization of the Traveling Salesman Problem (*TSP*) where the tour can be decomposed into contiguous paths that are travelled by different rented cars and allows the vehicles to be shared with other passengers to reduce expenses. The definition of the proposed problem involves the combination of two important concepts that are currently being widely used in the field of transportation: car rental and ride-sharing. This paper defines the *CaRSP* and presents four evolutionary algorithms to solve it: a genetic algorithm, a memetic algorithm and hybridizations of both using transgenetic vectors (transposons and recombinant plasmids). The metaheuristics are tested in a set of 30 Euclidean, non-Euclidean, symmetric and asymmetric instances. We use a framework to define the best parameter settings for each algorithm. Computational experiments are performed in two stages to ensure the algorithms are compared properly. The obtained results are submitted to statistical analysis. An algorithmic study is reported presenting the first heuristic results for *CaRSP*.**

***Index Terms*—Traveling Car Renter with Passengers, Evolutionary Computation, Memetic Algorithm, Genetic Algorithm**

I. Introduction

The Traveling Car Renter with Passengers (*CaRSP*) is a variant of the Traveling Salesman Problem (*TSP*). It allows the salesman to rent and drive different vehicles. Besides, the salesman can give rides to passengers and share trip costs. Nowadays, we observe an increment of services such as car rental and ride-sharing in transportation. According to [17], the global car rental market is expected to register a compound annual growth rate (*CAGR*) of approximately 7.5% between 2019 and 2024. In this same period, the ride-sharing market is expected to register a *CAGR* of 19.2% [18].

In 2017, the transportation sector was the largest source (29%) of greenhouse gas emissions in the USA and most of this air pollution was caused by passenger cars and light-duty trucks, leading to serious negative health effects [6]. Single-occupant trips, which in 2019 corresponded to approximately 76.4% of the american people that drives to work [2], combined with the high number of vehicles on the road increases the traffic congestion, gas emissions, fuel consumption and stress among people [26].

According to [25], these effects could be avoided by the efficient use of rented vehicles combined with the practice of ride-sharing. Some methodologies developed for the car rental industry may be adapted to meet future needs of shared-use vehicle systems [21].

In this paper, we investigate *CaRSP*, the ride-sharing version of the Traveling Car Renter (*CaRS*). It combines characteristics of both *CaRS* – classified as NP-hard [10]; and aspects of ride-sharing problems that are equally NP-hard [1].

CaRS is a generalization of the *TSP*, proposed by [9], that allows several cars with different costs to be available for the salesman's tour. An extra fee is charged when a car is delivered to a city different from the one it was rented. The objective is to find a route and a sequence of rented cars that minimize the travel costs and extra fees paid by the salesman. Heuristic approaches for *CaRS* include greedy randomized adaptive search, evolutionary algorithms and local search [4], [9], [10]. Mathematical formulations are proposed in [11].

In *CaRSP*, the salesman is allowed to share trip expenses with passengers. There is a list of potential passengers demanding rides from different cities to several destinations. Each one has a budget, i.e., a maximum fee that he/she agrees to pay. The objective is to find a route, a sequence of rented cars, and a set of passengers that minimize the salesman's travel costs. The problem investigated in this study is deterministic and static, i.e., the problem parameters are known with certainty and do not change.

An analysis about *CaRSP* and its subproblems is presented in [23] and mathematical formulations are proposed in [22].

Evolutionary algorithms are commonly used and have great performance solving combinatorial problems similar to *CaRSP*, as shown in [4], [9], [10]. The transgenetic vec-

This research was supported by *CNPq* (Conselho Nacional de Desenvolvimento Científico e Tecnológico), Brazil, under grants 302387/2016-1, 306702/2017-7, and 420999/2018-3 and by *UFERSA* (Universidade Federal Rural do Semi-Árido). The computational experiments were supported by *NPAD* (High Performance Computing Center) at *UFRN* (Universidade Federal do Rio Grande do Norte).

tors manipulate the chromosomes, modifying their codes and promoting the random variation that is necessary for the exploration and exploitation of the search space. They are capable of performing a local search on chromosomes allowing the overall search to become less stagnant over time and has proven to be efficient operators as shows [10], [12].

This study introduces *CaRSP* and proposes four evolutionary algorithms: a genetic algorithm, a memetic algorithm, and hybridizations of them with transgenetic vectors. The experiments used a set of 30 instances containing from 14 to 100 cities, from 2 to 5 cars, and from 51 to 290 passengers. We present the first heuristic results for *CaRSP*.

The problem is defined in Section II. Section III describes the proposed metaheuristics. The results of the computational experiments and the statistical analysis used to compare the performance of the proposed algorithms are presented in Section IV. Finally, some conclusions and future research directions are addressed in Section V.

II. Problem description

We consider graph $G = (N, M)$, where N is the set of nodes (cities) and M is the set of arcs (roads). The salesman drives rented cars along the tour. A set of cars, C, is available to be rented and delivered in any city.

Specific operational attributes are associated with each car, including fuel consumption, toll payment and other rental costs. Whenever there is a vehicle exchange, an extra fee must be paid, exclusively by the salesman, related to the company cost of returning a car from the city it was delivered back to the city it was rented. The vehicles have matrices of operational costs and return rates with different costs. Each car has its specific capacity, i.e., the number of passengers it can transport. Each car can only be rented once.

There is a set of potential passengers, L, i.e., people demanding rides. Each $l \in L$ demands a ride from an origin to a destination, and agrees to pay, at most, a pre-defined fee for the trip. Once on board, the passenger can only leave the vehicle at his/her destination.

The cost of traveling each arc is divided equally among the occupants of the car traversing it, including the driver. The tour begins and ends at city 1, the home-city of the salesman. The objective is to find the minimum cost spanning cycle regarding the viewpoint of the driver. *CaRSP* involves inter-connected decisions about the sequence of the cities, the order of used vehicles, the cities where the cars are rented/delivered and the decision of which passengers must be transported. These aspects must be considered to reach the best solution.

Figure 1 illustrates a solution for an instance with six cities where the salesman drives two different cars along the tour: cars 2 and 1. There is no need to use all available cars. Every city is visited once and the tour's initial vertex is city *1*. Initially, car *2* is rented and used to travel the following arcs: (*1*,*5*), (*5*,*6*), (*6*,*3*), and (*3*,*4*). In city *4*, there is a car exchange and car *2* is returned to the city it was rented, i.e., city *1*. Car *1* is rented and used to continue the tour through the following arcs: (*4*,*2*) and (*2*,*1*). The tour ends in the initial vertex and car *1* must be returned from city *1* to city *4*.

Along the tour, the salesman must reduces his costs by sharing the seats of the used vehicles. The total of passengers on board at each arc must respect the used vehicle's capacity. A passenger on board must be picked-up at his/her origin and dropped-off at his/her destination respecting his/her budget. Some passengers are not able to travel since their destinations are visited before their origins. Passenger *4* travels arcs (*3*,*4*), (*4*,*2*), and (*2*,*1*). Passenger *6* travels arcs (*5*,*6*), (*6*,*3*), and (*3*,*4*). Passenger *9* travels arcs (*5*,*6*) and (*6*,*3*). Passengers *11* and *13* travels arcs (*5*,*6*), (*6*,*3*), (*3*,*4*), and (*4*,*2*). Budget and car capacity constraints must be satisfied.

The cost to travel each arc is calculated by dividing its operational cost by the total of passengers on board traversing it, including the salesman. The cost of the tour is the sum of the cost of each traveled arc. The objective is to minimize the cost of the solution, which is given by the sum of the cost of the tour and the fees of all returned cars along the trip.

Since the *TSP* is NP-hard, as demonstrated by [8], and is a special case of the *CaRSP* with only one car and no passengers, the latter also belongs to NP-hard.

III. Genetic and Memetic Algorithms

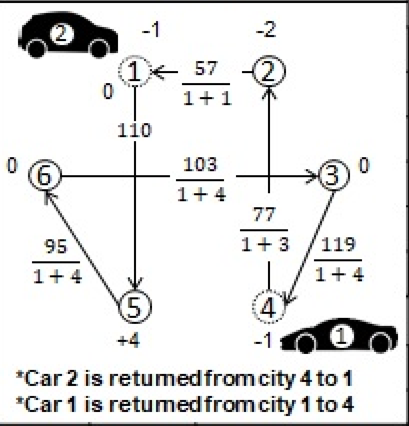
Genetic Algorithms (*GA*), proposed by [13], are inspired by Darwinism. It is a metaheuristic that has biological inspiration, emphasizing the simulation of evolutionary processes in artificial systems. It uses a population-based approach, considering the aspects of reproduction, recombination, mutation, and natural selection.

Memetic Algorithms (*MA*), proposed by [19], are inspired by Lamarckism and can be defined as hybrid evolutionary algorithms [24]. They incorporate local search strategies into the *GA* to combine the advantages of efficient heuristics with population-based evolution [5]. This concept states that the memetic approach considers a non-genetic improvement in the population with self-reproduction, inherited characteristics, random changes in the genotype, and survival regulated by a combination of abilities.

Transgenetic Algorithms (*TA*), proposed by [12], are inspired on the mutualistic intra-cellular endosymbiotic evolution. This type of biological evolution involves a host cell, endosymbionts fixed in the host's cytoplasm and vectors, or mechanisms, that allow endosymbionts and host to exchange genetic information. In the natural endosymbiotic evolution, endosymbionts are finally absorbed by their hosts with the formation of a new organism. In the computational metaphor this event is characterized by the stagnation of the fitness or the convergence of the population of chromosomes [10].

Transposons and recombinant plasmids are two operators from the *TAs* whose purpose is to reduce stagnations. The transposon is a transgenetic vector that identifies a region of an endosymbiont and manipulates the latter by rearranging the genes of the identified region [10]. The recombinant plasmid combines information from the host within an endosymbiont. It may partially or fully consist of a heuristic, such as a constructive one. Transposons and recombinant plasmids are

Solution



*Car 2 is returned from city 4 to 1
*Car 1 is returned from city 1 to 4

Car	Arc	Return Cost
2	(4,1)	34
1	(1,4)	9

Car	Arc	Arc Cost	#Passenger on board		Capacity	Cost
2	(1,5)	110	0	≤	4	110
2	(5,6)	95	4	≤	4	19
2	(6,3)	103	4	≤	4	20.6
2	(3,4)	119	4	≤	4	23.8
1	(4,2)	77	3	≤	5	19.25
1	B-A	57	1	≤	5	28.5

Pass. ID	Orig.	Dest.	Cost		Budget
4	3	1	71.55	≤	72.56
6	5	4	68.8	≤	74.72
9	5	3	45	≤	82.49
11	5	2	88.05	≤	138.1
13	5	2	88.05	≤	106.74

Total Cost = 110 + 19 + 20.6 + 23.8 + 19.25 + 28.5 + Return (CAR_2, 4, 1) + Return (CAR_1, 1, 4)
Total Cost = 110 + 19 + 20.6 + 23.8 + 19.25 + 28.5 + 34 + 9 = 264.15

Fig. 1. Example of solution for the *CaRSP*

operators used within the algorithms presented in this study.

The following sections describe the *GAs* and *MAs* proposed in this study. Section III-A defines the proposed algorithms.

Section III-B presents the structure of a chromosome. The method used to generate the initial population is reported in Section III-C. Sections III-D, III-E and III-F describes, respectively, crossover, mutation, and local search procedures. Section III-G defines the approaches used for the transposon and recombinant plasmid.

A. Proposed algorithms

We implemented four evolutionary algorithms: *GA1*, *GA2*, *MA1*, and *MA2*. Their general features are defined as follows.

- ***GA1*** – Traditional *GA*;
- ***GA2*** – *GA1* hybridized with transgenetic vectors;
- ***MA1*** – Traditional *GA* hybridized with local search;
- ***MA2*** – *MA1* hybridized with transgenetic vectors.

The pseudocode of the proposed metaheuristics is presented in Algorithm 1 and explained in the subsequent sections. The procedure *NaturalSelection* (step 21) remove the worst chromosomes in the population.

Algorithm 1: *CaRSP* – Proposed algorithms

```
1  Input: instanceName, numInd, repRate, mutRate, lsRate
2  Output: bestIndividual
3  begin
4      readInstance (instanceName);
5      pop ← generateInitialPopulation (numInd);
6      while ($numAvaliacoes < |N| \times |C| \times 500$) do
7          for ($i = 1, ..., repRate$) do
8              father, mother ← selectParents (pop)
9              son1, son2, son3, son4 ← Crossover (father, mother);
10             pop ← insertOffspring (pop, son1, son2, son3, son4);
11         for ($i = 1, ..., mutRate$) do
12             ind ← selectIndividual (pop);
13             pop[ind] ← Mutation (pop[ind]);
14         for ($i = 1, ..., lsRate$) do                      } Steps used in
15             ind ← selectIndividual (pop);                   } MA1 and MA2
16             pop[ind] ← localSearch (pop[ind]);              }
17         if ($Random(0, 1) = 0$) then                        }
18             pop ← Transposon (pop);                         } Steps used in
19         else                                                } GA2 and MA2
20             pop ← RecombinantPlasmid (pop);                 }
21         pop ← NaturalSelecion (pop, numInd);
22     bestIndividual ← returnBestIndividual (pop);
23 return (bestIndividual);
24 end
```

B. Chromosome

To represent a *CaRSP* solution, a chromosome must contain information about the sequence of cities visited by the salesman, the cars used, and the passengers whose demands were satisfied. Thus, two $|N|$-sized lists and a $|L|$-sized list represent each chromosome. Figure 2 illustrates the chromosome that represents the solution shown in Figure 1.

Cities	1	5	6	3	4	2							
Cars	2	2	2	2	1	1							
Passengers	0	0	0	1	0	1	0	0	1	0	1	0	1

Fig. 2. Chromosome

The chromosome presented in Figure 2 is the same solution explained in Section II. Associated with each potential passenger, there is an origin, a destination and a budget, which in turn are parameters of the problem.

Value 1 in position *i* of the *Passengers* list indicates that the demand of the the *i*-th passenger was satisfied, i.e., he/she has been boarded. The fitness is calculated by the sum of the costs of each traversed arc divided by the occupants of the vehicle and the fees of all returned cars along the trip.

C. Initial Population

Each individual of the initial population is generated randomly following these three steps:

- *Define sequence of cities* – The sequence of visited cities is generated randomly;
- *Define order of cars* – The order of used vehicles is generated randomly. The initial car, the number of vehicles in the solution and the index of the next car exchange are chosen at random. For all upcoming visits, there are two possibilities: keep the same car or exchange the vehicle for one that was not used yet;
- *Passenger boarding* – The passenger boarding scheme is defined by a heuristic presented in Algorithm 2.

The passenger boarding procedure, shown in Algorithm 2, analyzes passengers' demand at each arc of the tour (steps 3–9) and considers only passengers whose origin is before the destination (step 11). Steps 12–23 calculates each passenger's cost to travel from his/her origin to his/her destination. The cost of traveling each arc is calculated by dividing the cost of the arc (step 14) by the passengers' demand, including the salesman (steps 18). If this demand is greater than the vehicle's limit (step 15), this cost is divided by its capacity (step 16). If a passenger's travel cost meets his budget, then he/she is added to the list of potential passengers L^* (steps 20–21). Otherwise he/she is removed from the demand list (steps 22–23).

Heuristic boarding (step 24) consists of boarding each passenger $l \in L^*$. For each visit with more boarded passengers than the vehicle's capacity, passengers are removed until the vehicle's limit is respected. Passengers with fewer arcs traveled between his/her origin and destination are more likely to be removed of the solution. The cost for each on board passenger is calculated and those exceeding the financial limit are removed. For each person removed, it is necessary to recalculate the costs of the other passengers and repeat the procedure of removing passengers until the cost of each on board passenger respects his/her budget. This heuristic generates a feasible solution that respects all restrictions of the problem.

D. Crossover

The crossover combines information about cities and cars to generate the offspring. As shown in Algorithm 3, we use the strategy defined by the traditional single-point crossover operator.

The function *selectParents*, shown in step 3, selects two different individuals at random, *father* and *mother*, to generate four children (*c1*, *c2*, *c3* and *c4*). *c1* and *c3* inherit genetic material from the mother (step 4). *c2* and *c4* inherit genetic material from the father (step 5).

The point that defines which cities the children will inherit from the other parent is set at step 6. *c1* and *c3* inherit cities from the the second part of *father* (step 7). *c2* and *c4* inherit cities from the second part of *mother* (step 8). *c1* inherit the list of cars from *father* (step 9). *c2* inherit the list of cars from *mother* (step 10).

The function *repairChromosome*, shown in step 11, replaces each repeated city for one that was not visited that results in lower costs to traverse the incident arcs. The *repairPassengers* function, shown in step 12, removes all boarded passengers which now are unable to travel and attempts to board other potential passengers respecting all constraints. This procedure is similar to the *HeuristicBoarding* presented previously, in Section III-C.

Algorithm 2: *CaRSP* – Heuristic Passenger Boarding

```
1  Input: Chromosome c;
2  begin
3      for ($l = 1, ..., |L|$) do
4          origIndex[l] ← getOrigIndex (c, l)
5          destIndex[l] ← getDestIndex (c, l)
6          if ($destIndex[l] = 1$) then
7              destIndex[l] ← $|N|$
8          for ($i = origIndex[l], ..., destIndex[l]$) do
9              demands[i] ← demands[i] + 1
10     for ($l = 1, ..., |L|$) do
11         if ($origIndex[l] < destIndex[l]$) then
12             passengerCost ← 0;
13             for ($i = origIndex[l], ..., destIndex[l]$) do
14                 cost ← Cost (c.Cars[i], c.Cities[i], c.Cities[i+1])
15                 if ($demands[i] > carCapacity[c.Cars[i]]$)
                     then
16                     cost ← cost / (carCapacity[c.Cars[i]] + 1)
17                 else
18                     cost ← cost / (demands[i] + 1)
19                 passengerCost ← passengerCost + cost
20             if ($passengerCost \leq Budget[l]$) then
21                 $L^*$ ← AddPassenger (l)
22             else
23                 demands ← RemoveDemands (demands, l)
24     c ← HeuristicBoarding (c, $L^*$, demands)
25 return (c)
26 end
```

Algorithm 3: *CaRSP* – Crossover

```
1  Input: Population p
2  begin
3      Chromosome father, mother ← selectParents (p)
4      Chromosome c1, c3 ← mother
5      Chromosome c2, c4 ← father
6      point ← rand (2, $|N| - 1$)
7      c1, c3 ← InheritCities (father, point, $|N|$)
8      c2, c4 ← InheritCities (mother, point, $|N|$)
9      c1.Cars ← father.Cars
10     c2.Cars ← mother.Cars
11     c1, c2, c3, c4 ← repairChromosomes (c1, c2, c3, c4)
12     c1, c2, c3, c4 ← repairPassengers (c1, c2, c3, c4)
13 return (c1, c2, c3, c4)
14 end
```

E. Mutation

We propose four mutation procedures to the input chromosome: *removeCar*, *addCar*, *swapUsedcars*, and *exchangeCars*.

Procedure *removeCar* chooses a vehicle to be removed. This procedure is repetead for each used car. Cities that were visited using this vehicle are now visited using the previous or the next car, whichever leads to the best possible solution.

Procedure *addCar* chooses an unused vehicle to be inserted before every rental/delivery of the used cars. This procedure is repetead for each unused vehicle. Assuming that the unused car *c"* was added to the solution before the rental of car *c'*, the group of cities that was previously visited only by car *c'* are now visited by *c"* and *c'*, respectively. The sequence of cities do not change and the choice of which cities will be visited by each vehicle is given by the combination that leads to the best possible solution.

Procedure *swapUsedCars* swaps each pair of used cars. The sequence of cities do not change, only the group of visited cities by both swapped vehicles. The sequence of used cars is defined by the combination that leads to the best solution.

Procedure *exchangeCars* exchanges each used car for each unused one. The sequence of cities do not change, only the vehicle that is used to visit them. The sequence of used cars is defined by the combination that leads to the best solution.

After each of these procedures, passengers are boarded using the heuristic shown in Algorithm 2. Each chromosome *c* has a variable *c.nStagnation* used for analyzing whether a chromosome is stagnated in a local minimum. If after these procedures there are no improvements in the fitness of chromosome *c*, then this variable is incremented by one. Otherwise, it is set to zero. This variable is used by recombinant plasmid presented in Section III-G2.

F. Local Search

We propose six local search procedures to the input chromosome: *passengersCars*, *passengersCities*, *lkhPassengersAndCosts*, *2Opt*, *swapNotRequiredCities*, and *changePassengers*.

Given the sequence of cities of a chromossome, procedure *passengersCars* analyzes passengers' demands along this tour, i.e., passengers whose pickup city is visited before the drop-off city. For each arc traveled, if the demand is greater than the capacity of the used vehicle, it's verified if there are benefits to exchange the used car for an unused one with more seats or to rearrange the used cars to satisfy these demands. The sequence of used cars is defined by the combination that leads to the best possible solution.

Procedure *passengersCities* is a 2-swap local search that is applied on the sequence of cities. It aims to increase the number of passengers able to board, i.e., passengers whose pickup city is visited before the drop-off city and whose travel's cost from their origin to their destination with all car seats occupied meets their budgets. The sequence of cars do not change. The sequence of visited cities is defined by the combination that leads to the best possible solution.

Considering that Lin-Kernighan heuristic (*LKH*) is one of the best heuristics for solving *TSP* problems [15], the procedure *lkhPassengersAndCosts* splits a chromosome into *TSP* subproblems to be solved by Helsgaun implementation of *LKH* [14]. Given a chromosome, each *TSP* subproblem is composed by the group of cities visited by each used car. To solve each *TSP* subproblem, *LKH* considers that the cost of the arc connecting two cities *A* and *B* is divided by the number of passengers whose origin is *A* and destination *B*, including the driver. The order of used cars is maintained, only the sequence of cities visited by each one is changed.

Procedure *2Opt* is an adapted version of the traditional local search algorithm proposed by [3]. Given a chromosome, it rearranges each pair of traversed arcs, but keeps the order of used vehicles. The sequence of traversed arcs is defined by the combination that leads to the best possible solution.

Given a chromosome, procedure *swapNotRequiredCities* creates a list with all cities where there are no passengers being picked-up or dropped-off. Then, a 2-swap local search is applied only on the cities of this list, without changing the order of used vehicles. The output is given by the combination of cities that leads to lowest cost possible.

Procedure *changePassengers* replaces boarded passengers for unboarded ones. It creates a list with boarded passengers that could be replaced for unboarded ones. This generally happens in parts of the tour where there is a high demand of passengers, which can be identified according to their origins and destinations. This local search only performs replacement of passengers that lead to viable solutions, i.e., that respect all the restrictions of the problem. The passenger boarding scheme is defined by the solution that leads to the best solution.

After each of these procedures, except for *changePassengers*, passengers are boarded using the heuristic shown in Algorithm 2. Each chromosome *c* has a variable *c.nStagnation* used for analyzing whether a chromosome is stagnated in a local minimum. For both mutation and local search procedures we use Lamarckian evolution, i.e., if the fitness of the original solution is worse than the corrected one, it is replaced in the population. If after these procedures there are no improvements in the fitness of chromosome *c*, then this variable is incremented by one. Otherwise, it is set to zero. This variable is used by recombinant plasmid presented in Section III-G2.

G. Transgenetic Vectors

We propose two transgenetic vectors, which are used in metaheuristics *GA2* and *MA2*: transposons and recombinant plasmids, presented, respectively, in Sections III-G1 and III-G2. At each iteration, one transgenetic vector, selected randomly, is applied to all chromosomes. Both methods are equally likely to be selected, with a 50% chance.

1) Transposon: Transgenetic vector whose information is a rule for rearranging some genes of the manipulated chromosomes [10]. It identifies some genes and modifies them. For example, it can apply a local search just in a predefined part of the chromosome.

Given a chromosome, a vehicle exchange occurs when a car *c'* is delivered in city *i* and a car *c"* is rented in the same city to continue the tour. This procedure consists in anticipating/postponing vehicles' exchange in part of the solution. The part of the solution in which this technique is applied is defined by two randomly selected points. This results in solutions with the same sequence of cities and vehicles, the only changes are at the vehicles' exchange points. The combination that leads to the best solution is returned.

2) Recombinant Plasmid: Combines information obtained from two or more sources. It can also be an adhoc heuristic to construct part of a solution. This procedure consists in inherit information about an individual's sequence of visited cities and used cars and apply an exact passenger boarding technique, presented below. The recombinant plasmid is applied to each chromosome, *c*, that have not improved for a period greater than two iterations, i.e., chromosomes with variable $c.nStagnation > 2$. The goal is to remove each of those individuals from its stagnation point.

The proposed method is based on [22], which presents a linearization whose strategy is to replace the divisions by successive subtractions. Considering a fraction *N/D*, where *N* is the numerator and *D* the denominator, the transformation is restricted to cases where *D* is an integer, $D \geq 1$. According to [22], a fraction *N/D* can be rewritten as in equation (1).

$$\frac{N}{D} = N - \sum_{i=1}^{D-1} \frac{N}{i\,(i+1)} \tag{1}$$

Considering we have the sequence of cities and vehicles of a chromosome, it's possible to obtain the cost of each visit. This cost is used as the fraction's numerator. The number of passengers on board at each visit, i.e., the number of seats occupied, represents the denominator and will be defined by the proposed method. The procedure presented in equation (1) is repeated for each visit of the salesman. The exact formulation must define the passenger boarding sequence respecting all restrictions of the problem.

There is a preprocessing stage that eliminates passengers whose destinations are before their origins or that the budget is not enough to travel considering the car is full. It creates a list, L^*, containing all passengers able to board, i.e., persons whose pickup and drop-off points are feasible regarding the tour. It increases the computational efficiency of the procedure.

Expressions (2)-(8) show the mathematical formulation for the problem of assigning passengers to a solution represented by variable *sol*, containing the sequence of cities and vehicles. It requires that the initial city is also inserted as the final city. The parameters and variables are defined as follows.

Parameters

c_i: cost to use car *sol.Cars*[i] to travel from *sol.Cities*[i] to *sol.Cities*[i+1];

$k^{sol.Cars[i]}$: capacity of car *sol.Cars*[i];

L^*: list created in the preprocessing stage explained previously containing all passengers able to board;

bud(l): maximum fee *l* agrees to pay, $l \in L^*$;

iorg(l): index, of *sol.Cities*, for the origin of passenger *l*, $l \in L^*$;

idst(l): index, of *sol.Cities*, for the destination of passenger *l*, $l \in L^*$.

Variables

e_l: binary variable indicating whether passenger l was picked-up ($e_l = 1$) or not ($e_l = 0$);

g_{ih}: binary variable indicating whether seat *h* is occupied on the *i*-th visited city ($g_{ih} = 1$) or not ($g_{ih} = 0$). The seats are occupied sequentially, starting from the first;

q_{li}: real variable that indicates the cost, for passenger *l*, of visiting the *i*-th city.

$$\min \sum_{i \in N} \left(c_i - \sum_{h=1}^{k^{sol.Cars[i]}} \frac{g_{ih} c_i}{h(h+1)} \right) \tag{2}$$

subject to

$$\sum_{l \in P_i} e_l = \sum_{h=1}^{k^{sol.Cars[i]}} g_{ih}, \qquad \begin{smallmatrix} \forall i \in N \\ P_i = \{l \in L^* \mid iorg(l) \leq i < idst(l)\} \end{smallmatrix} \tag{3}$$

$$\sum_{l \in P_i} e_l \leq k^{sol.Cars[i]}, \qquad \begin{smallmatrix} \forall i \in N \\ P_i = \{l \in L^* \mid iorg(l) \leq i < idst(l)\} \end{smallmatrix} \tag{4}$$

$$c_i e_l - \sum_{h=1}^{k^{sol.Cars[i]}} \frac{g_{ih} c_i}{h(h+1)} \leq q_{li}, \qquad \begin{smallmatrix} \forall i \in N \\ \forall l \in L^* \mid \\ iorg(l) \leq i < idst(l) \end{smallmatrix} \tag{5}$$

$$\sum_{i \in N} q_{li} \leq bud(l), \qquad \forall l \in L^* \mid iorg(l) \leq i < idst(l) \tag{6}$$

$$g_{ih}, e_l \in \{0, 1\}, \qquad \begin{smallmatrix} \forall i \in N \\ l \in L^* \\ h = 1, \ldots, k^{sol.Cars[i]} \end{smallmatrix} \tag{7}$$

$$q_{li} \in \mathbb{R}_+ \tag{8}$$

This formulation assumes that the sequence of visited cities and used cars are previously known in *sol*. The objective function (2) consists of minimizing the sum of the costs of each visit *i* divided equally between the occupants of the vehicle on each traversed arc. It uses the strategy proposed in [22] and presented in equation (1) to replace the division operations. Constraint (3) couples variables e_l and g_{ih}, defining when passenger *l* has boarded, i.e., when *l* travels from his/her origin to his/her destination occupying a seat. Constraint (4) ensures that the capacity of each used vehicle is never exceeded. Equation (5) and (6) guarantees that the travel cost of each passenger not exceeds his/her budget. Constraints (7) and (8) ensures the integrity and nonnegativity of decision variables.

IV. Computational experiments

We executed the experiments on a computer with an Intel Core i7-4790 3.6 GHz x 8, 16 GB RAM DDR4 with Ubuntu 16.04 LTS operating system. The *CaRSP* instances are described in Section IV-A. The strategy used to define the best parameter settings for each metaheuristic is described in Section IV-B. To compare the performance of the algorithms, we divided the experiments into two stages. In the first stage, described in Section IV-C, we executed the algorithms with the parameters defined previously, in Section IV-B, using the number of function evaluations (*NFE*) as the stopping criterion. In the second stage, described in Section IV-D, the experiments were performed again using the highest average time obtained among all algorithms, in the first stage, as the stopping criterion. A statistical analysis to compare the obtained results is presented in Section IV-E. The same initial population is used by all proposed metaheuristics, so the experiments are performed under the same circumstances.

In the local search named *lkhPassengersAndCosts*, we used *LKH-3.0.3* version, released in July 2018 and available at http://akira.ruc.dk/~keld/research/LKH/.

A. Instances

The instances created for the *CaRSP* are described in [22] and available at http://www.dimap.ufrn.br/lae/downloads/Instances_CaRSP.rar. Instances were named as <id><n><distance type><cost type>, where *id* is a sequence with 2 or 3 characters, n is the number of cities, distance type "e" (Euclidean) or "n" (non-Euclidean), and cost type "s" (symmetric) or "a" (asymmetric). For example, the instance named Afe4ns means that the *id* is "Afe", the graph has 4 cities, it is non-Euclidean, and symmetric.

B. Parameter settings

The parameters, described below, were defined by using the irace framework (version 3.3), available at http://iridia.ulb.ac.be/irace/, which implements a number of automatic configuration procedures. It offers iterated racing procedures, which have been used successfully to automatically configure various state-of-the-art algorithms [16]. Each metaheuristics' best parameters found by irace are shown in Table I.

- **Number of individuals (*numInd*)**: population size;
- **Reproduction rate (*repRate*)**: corresponds to the portion of the population that will reproduce at each iteration;
- **Mutation rate (*mutRate*)**: represents the portion of the population that will mutate at each iteration;
- **Local search rate (*lsRate*)**: determines the portion of the population in which local search will be applied at each iteration.

TABLE I
BEST PARAMETERS FOR EACH METAHEURISTIC DEFINED BY IRACE

	numInd	*repRate*	*mutRate*	*lsRate*
GA1	193	0.43	0.11	-
GA2	191	0.44	0.14	-
MA1	131	0.23	0.24	0.4
MA2	194	0.25	0.13	0.27

C. First stage

The experiments were performed using the best parameters found previously, shown in Table I. The number of function evaluations, which is given empirically by $|N| \times |C| \times 500$, was used as the stopping criterion. Each metaheuristic solved all instances 30 times. The column *"First stage of the experiments"*, of Table II, shows the average solutions and execution times obtained by the algorithms. Results show that *MA1* obtained the highest average execution time in almost all instances.

D. Second stage

The execution time was used as the stopping criterion in this stage. The time limit, for each instance, was defined by the metaheuristic that obtained the highest average time in the previous stage. Each metaheuristic solved all instances 30 times. The column *"Second stage of the experiments"*, of Table II, shows the average solutions and execution times obtained by the algorithms. Results show that *MA2* and *MA1* obtained, respectively, better solutions in 24 and 6 instances.

E. Statistical analysis

For the statistical analysis of the obtained results in the second stage, the instances were divided into Euclidean, non-Euclidean, symmetric and asymmetric. First, we applied the Friedman test [7], a non-parametric statistical test used to detect differences in treatments across multiple test attempts. We obtained significant differences, with p-values less than 0.05, to each instance class. Then, we used Nemenyi test [20], a post-hoc test intended to find the groups of data that differ after a statistical test of multiple comparisons, such as the Friedman test. This test makes comparisons between each pairwise of the proposed metaheuristics in all instance groups. The results are shown in Table III.

The p-values obtained in the comparisons between each *GA* and between each *MA* are inconclusive, i.e., above 0.05. Comparisons between both *MAs* and *GAs* presents conclusive results for all cases, i.e., p-values less than 0.05, except between *GA2* and *MA1* in Euclidean and asymmetric instances.

V. CONCLUSIONS

This paper described an algorithmic study for the *CaRSP* based on evolutionary algorithms. We implemented a genetic algorithm, a memetic algorithm, and hybridizations of both using transgenetic vectors. We presented the first heuristic results for the proposed problem.

We reported the results of an experiment with 30 instances, divided into Euclidean, non-Euclidean, symmetric and asymmetric. The computational experiments were performed in two stages to ensure the algorithms are compared properly and the obtained results were submitted to statistical analysis.

The statistical analysis showed that memetic algorithms outperformed genetic algorithms in almost all cases, except in comparisons between *GA2* and *MA1* in Euclidean and asymmetric instances.

Although the statistical tests presented inconclusive results in the comparison between the *MAs*, we observed that *MA2* reached better solutions in 24 of the 30 instances, as shown in column *"Second stage of the experiments"*, of Table II, where the algorithms were submitted to the same execution time.

Future works will investigate hybridizations of exact methods and other metaheuristics. In addition, we intend to develop versions of the proposed problem that could be applied to dynamic changing situations, becoming even closer to reality.

REFERENCES

[1] N. Agatz, A. Erera, M. Savelsbergh and X. Wang, "Optimization for dynamic ride-sharing: a review," European Journal of Operational Research, vol. 223, pp. 295–303, 2012.

[2] City Lab, "The great divide in how americans commute to work," https://www.citylab.com/transportation/2019/01/commuting-to-work-data-car-public-transit-bike/580507, 2019, accessed in 15 July 2019.

[3] G. A. Croes, "A method for solving traveling salesman problems," Operations Research, vol. 6(6), pp. 791–812, 1958.

[4] D. Felipe, E. F. G. Goldbarg and M. C. Goldbarg, "Scientific algorithms for the car renter salesman problem," 2014 IEEE Congress on Evolutionary Computation, pp. 873–879, 2014.

[5] J. Digalakis and K. Margaritis, "Performance comparison of memetic algorithms," Journal of Applied Mathematics and Computation, Elsevier Science, vol. 158, pp. 237–252, 2004.

TABLE II
RESULTS OBTAINED IN COMPUTATIONAL EXPERIMENTS

			First stage of the experiments								Second stage of the experiments							
			GA1		*GA2*		*MA1*		*MA2*		*GA1*		*GA2*		*MA1*		*MA2*	
Instance	$\|C\|$	$\|L\|$	Sol	T(s)	Sol	T(s)	Sol	T(s)	Sol	T(s)	Sol	T(s)	Sol	T(s)	Sol	T(s)	Sol	T(s)
Bra14e	2	51	405.33	2.31	399.46	2.40	190.33	**5.65**	193.78	2.95	405.33	5.65	386.29	5.65	189.86	5.65	**179.45**	5.65
Bra14na	2	51	315.67	2.28	311.90	2.38	196.45	**7.78**	230.43	3.08	315.67	7.77	261.49	7.77	198.60	7.77	**176.11**	7.77
Bra14ns	2	51	247.17	2.31	246.45	2.39	134.10	**7.44**	151.26	3.02	247.17	7.44	227.40	7.44	132.81	7.44	**130.62**	7.44
Arg15e	3	46	1920.67	3.45	1915.42	3.52	955.63	**12.42**	1002.87	4.69	1920.67	12.42	1499.78	12.42	924.39	12.42	**889.42**	12.42
Arg15na	3	46	2164.67	3.42	2082.33	3.58	1326.90	**13.49**	1387.65	4.80	2164.67	13.49	1403.37	13.49	1322.53	13.49	**1203.49**	13.49
Arg15ns	3	46	2091.67	3.58	2070.90	3.78	1000.45	**14.24**	1088.38	4.77	2091.67	14.24	1217.03	14.24	1025.52	14.24	**979.09**	14.24
Bra16e	2	47	571.70	2.59	571.70	2.54	361.44	**9.35**	361.50	3.24	571.70	9.35	550.45	9.35	362.24	9.35	**355.16**	9.35
Bra16na	2	47	362.87	2.45	362.87	2.56	190.24	**8.69**	215.54	3.26	362.87	8.69	338.22	8.69	184.44	8.69	**183.68**	8.69
Bra16ns	2	47	398.83	2.56	392.66	2.71	190.01	**9.03**	233.69	3.29	398.83	9.03	337.94	9.03	210.78	9.03	**202.88**	9.03
Bra25e	3	65	1160.00	8.72	1153.03	9.30	620.03	**20.37**	640.62	10.06	1160.00	20.37	1078.98	20.37	625.53	20.37	**606.12**	20.37
Bra25na	3	65	627.00	7.67	623.60	7.97	398.78	**21.53**	434.75	10.64	627.00	21.53	561.96	21.53	398.90	21.53	**392.82**	21.53
Bra25ns	3	65	668.50	7.87	656.18	10.64	385.92	**21.59**	411.08	10.64	668.50	21.59	571.51	21.59	**394.05**	21.59	395.12	21.59
Bra30e	4	94	1147.17	18.83	1066.61	20.90	580.18	**41.52**	587.12	22.23	1147.17	41.52	864.66	41.52	584.38	41.52	**559.37**	41.52
Bra30na	4	94	715.50	19.65	694.55	19.03	422.07	**46.02**	434.01	23.98	715.50	46.02	592.94	46.02	**424.79**	46.02	442.84	46.02
Bra30ns	4	94	635.00	19.93	631.33	26.11	421.50	44.44	429.22	**48.14**	635.00	48.14	578.34	48.14	**418.89**	48.14	425.82	48.14
Bra40e	5	114	1698.50	38.86	1503.41	49.78	812.02	**90.23**	824.87	59.94	1698.50	90.23	1230.07	90.23	816.00	90.23	**807.98**	90.23
Bra40na	5	114	1250.50	37.56	1219.27	38.42	828.23	**89.27**	839.28	56.09	1250.50	89.27	1073.59	89.27	827.74	89.27	**825.90**	89.27
Bra40ns	5	114	1398.00	41.95	1345.95	42.78	825.28	**86.08**	839.79	59.96	1398.00	86.08	1095.17	86.08	**828.71**	86.08	838.93	86.08
Bra45e	5	133	2555.50	54.72	2136.56	49.03	1054.01	**106.56**	1068.38	75.53	2555.50	106.56	1696.40	106.56	1053.23	106.56	**1048.50**	106.56
Bra45na	5	133	1746.00	55.67	1690.22	56.07	1060.51	**118.71**	1077.17	74.37	1746.00	118.71	1494.71	118.71	**1036.98**	118.71	1055.12	118.71
Bra45ns	5	133	1771.00	49.40	1704.72	55.41	1073.02	**111.34**	1085.17	67.82	1771.00	111.34	1487.94	111.34	1067.41	111.34	**1066.65**	111.34
Bra50e	5	118	2119.00	46.89	1864.59	52.92	1018.49	**112.01**	1026.82	80.20	2119.00	112.01	1515.11	112.01	1019.59	112.01	**1011.06**	112.01
Bra50na	5	118	1720.67	59.79	1711.44	61.55	1140.19	**134.89**	1132.93	92.95	1720.67	134.89	1508.26	134.89	1129.01	134.89	**1123.98**	134.89
Bra50ns	5	118	1744.00	64.18	1675.91	59.79	1085.79	**120.05**	1100.67	74.73	1744.00	120.05	1524.19	120.05	**1086.15**	120.05	1126.57	120.05
st70eB	4	204	5561.33	91.70	5477.19	100.04	2676.13	**185.65**	2731.72	115.04	5561.33	185.65	4978.70	185.65	2643.77	185.65	**2633.14**	185.65
st70naB	4	204	3812.00	97.00	3702.90	95.40	1918.00	**188.55**	2014.94	116.62	3812.00	188.55	3276.53	188.55	1880.42	188.55	**1865.11**	188.55
st70nsB	4	204	3859.00	87.65	3757.28	92.11	1853.10	**166.85**	1963.94	107.75	3859.00	166.85	3245.30	166.85	1900.80	166.85	**1766.32**	166.85
rd100eB	4	290	54760.00	188.91	54145.88	197.45	13086.70	**389.56**	13340.15	247.71	54760.00	389.56	50691.51	389.56	13293.45	389.56	**13050.57**	389.56
rd100naB	4	290	5626.00	193.28	5587.51	197.07	2998.88	**370.49**	3045.14	242.07	5626.00	370.49	5220.77	370.49	3047.20	370.49	**2897.26**	370.49
rd100nsB	4	290	5110.50	187.44	5077.23	197.99	3033.67	**371.69**	3085.31	237.42	5110.50	371.69	4911.23	371.69	3062.58	371.69	**2892.14**	371.69

TABLE III
P-VALUES OBTAINED IN NEMENYI POST-HOC TEST

		GA1	*GA2*	*MA1*	*MA2*
Euclidean Instances	*GA1*	-	0.30713	**0.00299**	**0.001**
	GA2	0.30713	-	0.30713	**0.00299**
	MA1	**0.00299**	0.30713	-	0.30713
	MA2	**0.001**	**0.00299**	0.30713	-
		GA1	*GA2*	*MA1*	*MA2*
non-Euclidean Instances	*GA1*	-	0.068212	**0.001**	**0.001**
	GA2	0.068212	-	**0.007913**	**0.001**
	MA1	**0.001**	**0.007913**	-	0.735188
	MA2	**0.001**	**0.001**	0.735188	-
		GA1	*GA2*	*MA1*	*MA2*
Symmetric Instances	*GA1*	-	0.068212	**0.001**	**0.001**
	GA2	0.068212	-	**0.017331**	**0.001**
	MA1	**0.001**	**0.017331**	-	0.457254
	MA2	**0.001**	**0.001**	0.457254	-
		GA1	*GA2*	*MA1*	*MA2*
Asymmetric Instances	*GA1*	-	0.307130	**0.001**	**0.001**
	GA2	0.307130	-	0.160247	**0.00986**
	MA1	**0.001**	0.160247	-	0.701825
	MA2	**0.001**	**0.00986**	0.701825	-

[6] Environmental Protection Agency, "Inventory of U.S. greenhouse gas emissions and sinks," https://www.epa.gov/ghgemissions/inventory-us-greenhouse-gas-emissions-and-sinks, 2017, Accessed in 15 July 2019.

[7] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," Journal of the American Statistical Association, vol. 32(200), pp. 675–701, 1937.

[8] M. R. Garey and D. S. Johnson, "Computers and intractability: a guide to the theory of np-completeness," vol. 1, Freeman, San Francisco, 1979.

[9] M. C. Goldbarg, P. H. Asconavieta and E. F. G. Goldbarg, "Memetic algorithm for the traveling car renter problem: an experimental investigation," Memetic Computing, vol. 4(2), pp. 89–108, 2012.

[10] M. C. Goldbarg, E. F. G. Goldbarg, P. H. Asconavieta, M. S. Menezes and H. P. L. Luna, "A transgenetic algorithm applied to the traveling car renter problem," Expert Systems with Applications, vol. 40(16), pp. 6298–6310, 2013.

[11] M. C. Goldbarg, E. F. G. Goldbarg, H. P. L. Luna, M. S. Menezes and L. Corrales, "Integer programming models and linearizations for the traveling car renter problem," Optimization Letters, vol. 12(4), pp. 743–761, 2017.

[12] M. C. Goldbarg, L. B. Bagi and E. F. G. Goldbarg, "Transgenetic algorithm for the traveling purchaser problem," European Journal of Operational Research, vol. 119(1), pp. 36–45, 2009.

[13] D. E. Goldberg, "Genetic algorithms in search, optimization, and machine learning," vol. 1, Addison-Wesley, Reading, MA, 1989.

[14] K. Helsgaun, "An effective implementation of the Lin-Kernighan traveling salesman heuristic," European Journal of Operational Research, vol. 126(1), pp. 106–130, 2000.

[15] D. Karapetyan and G. Gutin, "Lin-Kernighan heuristic adaptations for the generalized traveling salesman problem," European Journal of Operational Research, vol. 208(3), pp. 221–232, 2011.

[16] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres and M. B. T. Stützle, "The irace package: iterated racing for automatic algorithm configuration," Operations Research Perspectives, vol. 3, pp. 43–58, 2016.

[17] Mordor Intelligence, "Car rental market - growth, trends, and forecast (2019 - 2024)," https://www.mordorintelligence.com/industry-reports/car-rental-market, 2018, accessed in 15 July 2019.

[18] Mordor Intelligence, "Ridesharing market - growth, trends, and forecast (2019 - 2024)," https://www.mordorintelligence.com/industry-reports/ridesharing-market, 2018, accessed in 15 July 2019.

[19] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithm," Technical Report C3P 826, Caltech Con-Current Computation Program 158–79, 1989.

[20] P. B. Nemenyi, "Distribution-free multiple comparisons," PhD thesis, Princeton University, 1963.

[21] B. B. Oliveira, M. A. Caravilla and J. F. Oliveira, "Fleet and revenue management in car rental companies: a literature review and an integrated conceptual framework," Omega, vol. 71, pp. 11–26, 2016.

[22] G. A. Sabry, M. C. Goldbarg, E. F. G. Goldbarg, M. S. Menezes and Z. S. A. Calheiros, "Models and linearizations for the traveling car renter with passengers," unpublished.

[23] G. A. Sabry, B. C. H. Silva, M. C. Goldbarg, E. F. G. Goldbarg, M. S. Menezes and Z. S. A. Calheiros, "An analysis of the Traveling Car Renter with Passengers and its subproblems," Brazilian Journal of Development, vol. 5(11), pp. 24449–24470, 2019.

[24] E. Shahamatnia, R. Ayanzadeh, R. Ribeiro, and S. Setayeshi, "Adaptive imitation scheme for memetic algorithms," 2nd Doctoral Conferece on Computing, Electrical and Industrial Systems, pp. 109–116, 2011.

[25] B. Yu, Y. Ma, M. Xue, B. Tang, B. Wang, J. Yan and Y-M. Wei, "Environmental benefits from ridesharing: a case of Beijing," Applied Energy, vol. 191, pp. 141–152, 2017.

[26] Y. Zhang and Y. Zhang, "Exploring the relationship between ridesharing and public transit use in the United States," 2018 Journal of Environmental Research and Public Health, vol. 15(8), pp. 1–23, 2018.