2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

A Memetic Algorithm for the Prize-collecting Traveling Car Renter Problem

Matheus da Silva Menezes, Marco César Goldbarg, and Elizabeth F. G. Goldbarg

***Abstract*—This paper introduces a new variant of the Traveling Car Renter Problem, named Prize-collecting Traveling Car Renter Problem. In this problem, a set of vertices, each associated with a bonus, and a set of vehicles are given. The bonus represents a degree of satisfaction to visit the vertex. The objective is to determine a cycle that visits some vertices collecting, at least, a pre-defined bonus, i.e. reaching a pre-specified satisfaction, and minimizing the cost of the tour that can be traveled with different vehicles. A mathematical formulation is presented and implemented in a solver to produce results for sixty-four instances. A memetic algorithm is proposed and its performance is evaluated in comparison to the results obtained with the solver.**

I. Introduction

THIS paper investigates a variant of the Traveling Car Renter Problem (CaRS), named Prize-collecting Traveling Car Renter Problem (pCaRS). The former was introduced in [1] and is a generalization of the Traveling Salesman Problem (TSP). In the TSP, the objective is to find a Hamiltonian cycle in a weighted simple graph, G, given as input, that has the minimum cost among all Hamiltonian cycles of G. The cost is given by the sum of the weights of the edges of the Hamiltonian cycle. The TSP is a classical Combinatorial Optimization problem and a revision of models and algorithms to this problem is given in [2]. The cost of the edges of the TSP can be thought of as the cost of using a car to travel a road between two cities represented by two vertices of G. CaRS generalizes the TSP allowing that several cars, with different costs, are available to be used during the tour. This situation occurs, for instance, when a tourist wants to visit a set of cities traveling with rented cars paying as little as possible for the cars. There are several options of rental cars of different companies in each city. The multiplicity of options opens a range of alternatives. The problem consists in visiting a set of cities, starting and ending at the same point, minimizing the cost due to car rentals. To make the decision on which car to rent at each part of the route, the customer should consider, besides renting, costs due to fuel consumption and payment of tolls. If a car is rented in a city and delivered in a different one, then the user needs to consider also some extra fee to take the car back to its origin. Since the Traveling Salesman Problem is NP-hard [2] and a special case of CaRS when only one car is used in the tour, CaRS is also NP-hard. Metaheuristic algorithms were presented for cars in [3] [4].

It is usual the case where the tourist cannot visit all the existing attractions during a trip. In those cases it is interesting to maximize satisfaction visiting the most attractive points. The Mobile Tourist Guide, presented in [5], was designed to tourists who cannot visit all places they are interested in a large city. Some tourist trip design problems are introduced in [6] and an increasing volume of research has been dedicated to those problems [7]. A revision of routing problems in the car rental industry is presented in [8]. In [9] systems where rent and delivery occur in distinct places are investigated. Those papers, however, do not address the problem when more than one vehicle can be used by the tourists.

The pCaRS problem, investigated in this paper, is a generalization of the Prize-collecting Traveling Salesman Problem (PCTSP). The latter was introduced in [10] and is NP-hard. In the PCTSP a reward and a penalty are assigned to each city and one must choose a subset of cities to be visited so that the cost of the tour and the penalties associated with each non visited city are minimized and the total reward is at least a given parameter ω. pCaRS is a variant of CaRS that, as in PCTSP, a bonus is associated to each city. The bonus defines a satisfaction level in visiting the associated city. A minimal pre-defined cumulative satisfaction must be met. In pCaRS, it is also considered that the tour begins and ends at the same point. The objective is to select a subset of cities (points) to be visited so that the total traveling cost regarding rental cars is minimized and the total satisfaction is at least a given parameter ω. In this paper, the problem is formulated and results of the mathematical formulation implemented in the GLPK solver [11] are presented. A memetic algorithm is proposed to pCaRS and applied to sixty-four instances. The results obtained with the heuristic algorithm are compared to the ones obtained with the solver.

This paper is organized in other four sections, besides this one. The problem and the proposed mathematical formulation are presented in Section II. The memetic algorithm is presented in Section III. Computational experiments are reported in Section IV. Finally, some conclusions are presented in Section V.

II. The Prize-Collecting Car Renter Salesman

Let $G=(V,A)$ be a complete graph where V is a set with n nodes (cities) and A is a set of arcs (roads between cities). A

M. S. Menezes is with the Universidade Federal Rural do Semi-Árido - UFERSA Angicos, Brasil (matheus@ufersa.edu.br) e-mail: matheus@ufersa.edu.br / matheussmenezes@gmail.com).

M. C. Goldbarg., is with Dpto. de Informática e Matemática Aplicada – DIMAP of the Universidade Federal do Rio Grande do Norte – UFRN Natal, Brasil (marcocgold@gmail.com).

E. F.G Goldbarg is with Dpto. de Informática e Matemática Aplicada – DIMAP of the Universidade Federal do Rio Grande do Norte – UFRN Natal, Brasil (beth@dimap.ufrn.br).

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

bonus SV_i, $i = 1,\ldots,n$, is assigned to each city $i \in V$. In this problem, C denotes the set of cars. All cars are available for rent and delivery in all cities. Specific operational costs are associated with each car including fuel consumption, toll payment and rent cost. Since toll fees usually depend on the car type and on the length of the traveled route, it is possible to assume, without loss of generality, that the operational cost of each car to traverse edge $(i,j) \in A$ is a function of that car. The operational cost associated to car $c \in C$ to traverse edge $(i,j) \in A$ is denoted by d_{ij}^c. Vertex 1 is chosen as the starting and ending vertex. If car $c \in C$ is rented in city i and delivered in city j, $i \neq j$, a fee to take c back to city i, γ_{ij}^c, is paid. The mathematical formulation considers the following binary variables: f_{ij}^c with value 1 when car c traverses edge (i,j) from i to j and 0 otherwise; w_{ij}^c with value 1 when car c is rented in city j and delivered in i; a_i^c with value 1 when car c is rented in city i; e_i^c with value 1 when car k is delivered in city i. The formulation also considers parameter ω, the minimum total satisfaction to be reached in the tour, which is given by $\omega = 0.8\sum_{i \in V} SV_i$, and the integer variable u_i that gives the position of vertex i in the tour.

$$\min \sum_{c \in C} \sum_{i,j \in V} d_{ij}^c f_{ij}^c + \sum_{c \in C} \sum_{i,j \in V} \gamma_{ij}^c w_{ij}^c \quad (1)$$

$$\sum_{c \in C} \sum_{i \in V} f_{i1}^c = \sum_{c \in C} \sum_{j \in V} f_{1j}^c = 1 \quad (2)$$

$$\sum_{c \in C} \sum_{i \in V} f_{ih}^c = \sum_{c \in C} \sum_{j \in V} f_{hj}^c \leq 1 \qquad \forall h \in V \quad (3)$$

$$a_i^c = \left(\sum_{j \in V} f_{ij}^c \right) \left(\sum_{\substack{c' \in C \\ c' \neq c}} \sum_{h \in V} f_{hi}^{c'} \right) \qquad \forall\, c \in C,\ i \in V,\ i > 1 \quad (4)$$

$$e_i^c = \left(\sum_{j \in V} f_{ji}^c \right) \left(\sum_{\substack{c' \in C \\ c' \neq c}} \sum_{h \in V} f_{ih}^{c'} \right) \qquad \forall\, c \in C,\ i \in V,\ i > 1 \quad (5)$$

$$w_{ij}^c = a_j^c e_i^c \qquad \forall\, c \in C,\ i,j \in V \quad (6)$$

$$\sum_{c \in C} a_1^c = 1 \quad (7)$$

$$\sum_{i \in V} a_i^c \leq 1 \qquad \forall\, c \in C \quad (8)$$

$$\sum_{i \in V} a_i^c = \sum_{j \in V} e_j^c \qquad \forall\, c \in C \quad (9)$$

$$\sum_{i \in V} \left[\left(\sum_{c \in C} \sum_{j \in V} f_{ij}^c \right) SV_i \right] \geq \omega \quad (10)$$

$$2 \leq u_i \leq n \qquad \forall\, i = 2,\ldots,n \quad (11)$$

$$u_i - u_j + 1 \leq (n-1)\left(1 - \sum_{c \in C} f_{ij}^c\right) \qquad \forall\, i,j = 2,\ldots,n \quad (12)$$

$$f_{ij}^c,\ w_{ij}^c,\ a_i^c,\ e_i^c \in \{0,1\} \quad (13)$$

$$v_t \in \aleph \quad (14)$$

The objective function is expressed in (1). It includes the cost of traversing edges with different cars and fees related to cars return. Constraint (2) states that the tour begins and ends at city 1. Constraints (3) state that each vertex is visited, at most, once and if a car arrives at vertex i then a car must leave that vertex. Constraints (4) associate if a car is located at the node i and constraints (5) associate if a car is delivered in node i. Constraints (6) are related to a car being rented in a city and delivered in another. Constraint (7) assures that a car is rented in node 1. Constraints (8) ensure the each car is used only once, and constraints (9) guarantee that every located car is delivered. Constraint (10) assures that a minimum satisfaction level is met. Constraints (11) and (12) forbid subtours and were adapted from the Miller-Tucker-Zemlin (MTZ) TSP formulation presented in [12] and also utilized in [13]. Constraints (13) state that those variables are binary and (14) that variables u_i are positive integers.

Constraints 4-6 are quadratic and their variables are binary. Those constraints are linearized. The linearization presented in expressions 16-19 is known as usual linearization [14], was proposed in [15] and reformulated in [16]. A non linear constraint, as in (15), is replaced by the set of equations 16-19.

$$z = x \times y \quad (15)$$

$$z \leq x \quad (16)$$

$$z \leq y \quad (17)$$

$$z \geq x + y - 1 \quad (18)$$

$$z, x, y \in [0,1] \quad (19)$$

This linearization was applied to solve the problem in the GLPK solver as presented in [17].

A numerical example of the pCaRS is given in the Appendix.

III. Memetic Algorithm

Memetic algorithms were proposed in [18] and have been used with success in a wide range of applications [19]. The memetic algorithm proposed to pCaRS is presented in figure 1. It receives as input parameters the name of the instance, *inst*, the size of the population, *sizePop*, the crossover rate, *#Cross*, the percentage of the population considered as elite solutions, *#Elite*. In step 2 the data of the instance are read. The initial population is generated in step 3.

The chromosome is represented in a 2-dimensional array, as illustrated in figure 2, where the tour is represented in one dimension and the cars are represented in the other. In figure 2 the gray array represents the tour and the white one represents the cars. In this figure ten from twelve cities are visited. Note that cities 10 and 11 are not visited. This chromosome represents a solution where car 2 is hired in city 1 and delivered in city 4, car 3 is hired in city 4 and delivered in city 5 and car 1 is rented in city 5 and delivered in city 1.

```
1.  main(inst, sizePop,  #Cross, #Elite, limitIter)
2.  instanceRead(inst)
3.  Pop[ ] ← generateInitPop(sizePop); numIter ← 0
4.  multiOperatorsLocalSearch(Pop)
5.  elitePop[ ] ←generateElitePop(Pop, #Elite)
6.  while(numIter ++ < limitIter ) do
7.      off [ ] ← Crossover(Pop, elitePop[ ], #Cross )
8.      multiOperatorsLocalSearch (off)
9.      Pop[ ] ←binaryTournament(Pop, off)
10.     elitePop[ ] ←generateElitePop(Pop, #Elite)
11. end_while
12. end
```

Fig. 1. Main procedure of the memetic algorithm proposed to pCaRS

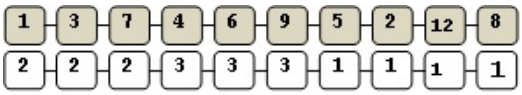


Fig. 2. Chromosome representation

The initial population is generated in procedure *generateInitPop*( ) with the population size, *sizePop*, as input parameter. A greedy heuristics adapted from CaRS is used to generate the chromosomes of the initial population. A car, c_1, is randomly chosen for the first city. Then city i, where car c_1 is delivered, is chosen at random. A path is created between cities 1 and i with the Nearest Neighbor heuristics considering the traveling costs associated with c_1. City i is the initial city of the next part of the tour. Then, a new car c_2 and a new city j are randomly chosen. Clearly, city j cannot be any city visited previously. A path is built between cities i and j with not visited cities. The procedure continues picking another city at random and building a path until there are no cars to add to the solution or the satisfaction level is reached. The cycle is closed going to the starting city with the last rented car.

The major chalenge of heuristic methods that use local search is to define an efficient strategy to cover the search space, mainly exploiting promising regions [13]. There are three areas where local seach can act in pCaRS: the best route, the best points to change cars and the satisfaction level. The local search phase of the proposed algorithm aims at dealing with these three areas in order to improve candidate solutions. Local search is perfomed in procedure *multiOperatorsSearch*( ) and is composed with the methods describe as follows.

- *removeSaving*: This method focuses on the cost of the tour. It consists in removing the cities with the lowest satisfaction scores from the candidate solution, while the satisfaction requirement is still met.
- *InvertSol*: This method inverts the visiting order of the cities in the candidate solution. The same cars are associated with the same cities, but the hiring and delivering points are exchanged. For instance, consider the tour (1, 2, 3, 4, 5) on five cities with car 1 being hired in city 1 and returned in city 3 and car 2 hired in city 3 and delivered in city 1. After application of the *InvertSol* method, the tour becomes (1,5,4,3,2) with car 2 being hired in city 1 and delivered in city 3 and car 1 being hired in city 3 and returned in city 1. It also focuses on the cost of the tour. The best of the original and inverted solutions remains in the population.
- *replaceSavingCar*: This procedure focuses on vehicles that are not yet in the solution examining the insertion of a new car, if possible. The new car replaces another in the solution. All cars not in the input solution are considered. One car not in the solution is inserted at each position iteratively. For example, suppose an instance where cars 1, 2, 3 and 4 can be rented. Consider a solution with five cities represented by the tour (1,2,3,4,5) with cars 2, 3 and 4 being hired(delivered), respectively, in cities 1,3,5(3,5,1). The vector assigning cars to cities is represented as (2,2,3,3,4). Car 1 is not used in this solution. Then, procedure *replaceSavingCar* examines possibilities of inserting car 1. First, car 1 is inserted in position 1 producing car sequence (1,2,3,3,4), then car 1 is inserted in positions 2, 3, 4, and 5 producing, respectively, (1,1,3,3,4), (1,1,1,3,4), (1,1,1,1,4), (1,1,1,1,1). In the next step, car 1 replaces the second car in the original car sequence, producing (2,1,3,3,4). The substitution goes on from that point, producing (2,1,1,3,4), (2,1,1,1,4) and (2,1,1,1,1). Next, the third position in the original sequence is set to 1 and the procedure continues until all possibilities are examined.
- *replaceSavingCit*: This procedure focuses on the satisfaction level, examining cities that are not yet in the solution. It examines substituting cities in the solution by cities out of it. All cities out of the input solution are considered for insertion. For example, consider the tour (1,3,4,5,6). City 2 is not in this tour. Then, the procedure examines the following tours: (1,2,4,5,6), (1,3,2,5,6), (1,3,4,2,6) and (1,3,4,5,2). The vector of cars associated with the input tour is not changed.
- *insertSavingCit*: This procedure also focuses on cities that are not yet in the input solution. It inserts a new city in the tour and does not remove any city. The car assigned to the new city is the same one assigned to the city immediately before the new one. The insertion of a new city is tested between every pair of cities of the input solution. All cities out of the input solution are considered for insertion. For example, consider again the tour (1,3,4,5,6). Then, the procedure examines the following tours: (1,2,3,4,5,6), (1,3,2,4,5,6), (1,3,4,2,5,6), (1,3,4,5,2,6) and (1,3,4,5,6,2).
- 2-opt: is a simple local search algorithm proposed in [20] to the TSP. A 2-opt move consists of eliminating two edges and reconnecting the two resulting paths in a different way to obtain a new tour. There is only one way to reconnect the paths that yields a different tour. Among all pairs of edges whose 2-opt exchange

decreases the tour length, the pair that gives the shortest tour is chosen. This procedure is then iterated until no such pair of edges is found. The resulting tour is called 2-optimal. In this case, the associated car sequence remains the same.

These local searches are applied in sequence in procedure *multiOperatorsSearch*( ) as depicted in figure 3. If the input solution represented in an individual is improved during local search, then the new individual replaces the old one.

```
1. multiOperatorsSearch(vector)
2.     removeSaving(vector)
3.     InvertSol (vector)
4.     insertSavingCit(vector)
5.     replaceSavingCit(vector)
6.     replaceSavingCar(vector)
7.     2Opt(vector)
8. end
```

Fig. 3. multiOperatorsLocalSearch procedure

An elite population is considered for recombination. The elite population, *elitepop*, is formed with *#Elite* percent best individuals of the current population in terms of their fitness values. Fitness is calculated with expression (1). Since, pCaRS is a minimization problem, the lowest the fitness the best the individual is.

The recombination procedure, *Crossover*( ), has three parameters: *Pop*, *elitepop* and *#Cross*. One parent comes from the current population, *Pop*, and the other comes from *elitepop*. Both are chosen at random with uniform probability. The third parameter, *#Cross*, stands for the recombination rate. The one-point crossover is utilized. Since the number of genes in chromosomes can vary, a random point is chosen in the range of indices of the smallest chromosome. The recombination of two parents, *A* and *B*, generates two offsprings, *C* and *D*, as illustrated in figure 4. The crossover point is illustrated with a dashed line in chromosomes *A* and *B*.

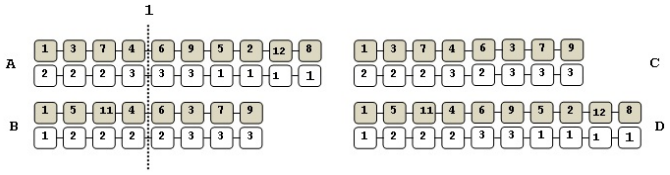


Fig. 4. Recombination operator

It may be necessary to restore feasibility of solutions generated during recombination. Infeasibility may occur regarding routes, cars assignment or total satisfaction. For example, in figure 4, chromosome C represents an infeasible solution, once cities 3 and 7 are visited twice and assigned vehicles. Infeasibility regarding the cities in the route occurs due to cities 3 and 7 appear twice each. Infeasibility regarding cars assignment occurs due to cars 3 and 2 are rented twice. The restoration procedure used in this work is the same presented in [1]. To restore feasibility regarding the cities in the route, the second time each city appears in chromosome C, it is replaced by an asterisk. For instance the sequence (1,3,7,4,6,3,7,9) is replaced by (1,3,7,4,6,*,*,9). Each asterisk is replaced by a city different from those in the chromosome, chosen at random, if such cities exist. The car assignment of chromosome C, (2,2,2,3,2,3,3,3), is also not feasible, for the problem considered in this paper requires each car is rented once. The restoration procedure replaces car repetitions by asterisks. Then each asterisk is replaced by the car which appears in the first preceding position. Thus, the car assignment of chromosome C is replaced by (2,2,2,3,*,*,*,*) and each asterisk is replaced by car 3.

If after restoring cities and cars, the chromosome does not meet the minimum satisfaction requirement, a procedure to restore satisfaction is executed. First, cities with low degree of satisfaction are replaced by others with better satisfaction levels. If the minimum satisfaction level is still not reached, then cities are added randomly at the end of the solution up to reaching the required minimum satisfaction. The car assigned to each new city is the one assigned to the city immediately before it.

IV. Computational Experiments

Since pCaRS is a new problem, an instance library, named pCaRSLIB, was created. They were adapted from the CaRSLIB instances [1] and are available at http://www.dimap.ufrn.br/lae/en/projects/CaRS.php. Those instances have the following characteristics: all cars can be rented in all cities; all cars can be delivered in all cities; each car can be rented only once; the fee paid to take a car back to its home city is not associated with the instance topology; symmetry, i.e., the costs to go from i to j and from j to i are equal; the underlying graph is complete. Instances are divided into two classes: Euclidean and non-Euclidean. Three groups of instances were created for each class. The difference between the instances of each group is on how the edges weights were generated. First, a primary set of edge weights is established for each group. Those weights are associated with the first car. In the first group of instances, the primary edge weights were taken from real maps. The weights of the second group were generated uniformly in the range [10,500]. The third group of instances is based on the TSPLIB [21]. The weight of each edge corresponding to car c, $1 < c \leq |C|$, was randomly chosen, with uniform probability, in the range $[1.1w_e,2.0w_e]$ where w_e stands for the primary weight of edge e. The satisfaction level assigned to each city was uniformly generated in the range [0,100].

The mathematical formulation presented in Section II was implemented in the GLPK software [11], version 4.45.2. Memory was limited to 14 Gb RAM. Processing time was limited to 80000 s. GLPK was finished if the problem was solved or if the limit due to memory or processing time was reached.

The parameters of the memetic algorithm were tuned in preliminary tests and were: *sizePop* = 150 and *#Cross* = 0.4, *limitIter* = 100 and *#Elite*=0.3.

Figures 5-7 illustrate the parameters *sizePop* and *#Cross* in three different instances used in the preliminary experiments. These figures show the typical behavior of the algorithm in a

computational experiment on 64 instances with 10 to 48 cities and 2 to 5 cars. The legend on the right hand side refers to tested values for *#Cross*. The figures show that the quality of the best solution did not improve significantly to values of *sizePop* greater than 150. Stability of the best solution was also reached for *#Cross* = 0.4.

The algorithms were executed on a PC Intel Core i5, 16G of RAM, running Ubuntu 12.04 64bits. Thirty independent executions of the Memetic Algorithm were performed for each instance.

Tables I and II show the results of the computational experiments. Columns *Name*, *nCity* and *nCar* are related to the characteristics of each instance being, respectively, the identification, the number of cities and the number of available cars. Columns *Min* and *T*(*s*), related to the GLPK solver, show the value of the best feasible integer solution and the processing time in seconds. Columns *Min*, *Av*, *F* and *T*(*s*), related to the Memetic Algorithm, show, respectively, the value of the minimum solution, the average of the minimum solutions on the thirty independent executions, the number of times the best solution was found and the average processing time in seconds. Column *Gap*, for each algorithm, presents the percent deviation of the value presented in column *Min* from a lower bound for the exact solution computed by the GLPK. Let the upper bound on the relative error due to rounding in floating point arithmetic in the manchine be $\in$, the gap is given in (19), where *best_mip* and *best_bound* stand for, best integer solution and the best relaxed solution, respectively.

$$GAP = \frac{|best\_mip - best\_bound|}{|best\_mip + \in|} \qquad (19)$$

Table I shows that GLPK solves all Euclidean instances, except for China17e, Russia17e, BrasilAM26e, BrasilMG30e and att48eA for which the solver stopped due to memory limitation. The percent deviation produced by the GLPK for those instances were 6.40, 10.47, 20.50 28.60 and 38.12, respectively. All optima found by GLPK were also found by the Memetic Algorithm, except for instance Brasil16e where the solution produced by the latter presented percent deviation 2.83 from the former. GLPK spent less time to solve eleven instances than the average processing time of the Memetic Algorithm. This fact occurred since the six local search phases of the latter algorithm, implemented in *multiOperatorSearch*, require significant processing times. Remark that on instances larger than those eleven, the computational effort spent on the local search phases benefits the search made by the Memetic Algorithm. The tables show that the performance of the Memetic Algorithm improves as instances grow larger. This tendency is also observed with the analysis of the data presented in Table II. On the other twenty-two Euclidean instances the Memetic Algorithm spent, in average, less processing time than the GLPK. In average, the GLPK spent 12089.39 seconds to produce solutions to the Euclidean instances while the Memetic Algorithm took 88.00 seconds. The average frequency of the best solution found by the Memetic Algorithm is 21.34 for the Euclidean instances.

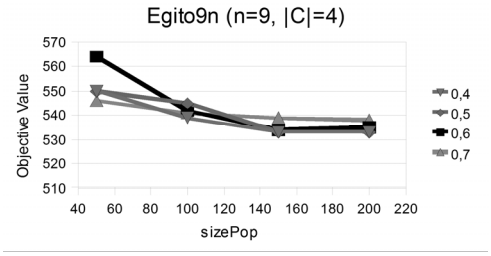


Fig. 5. Population size versus value of the best solution on Egito9n

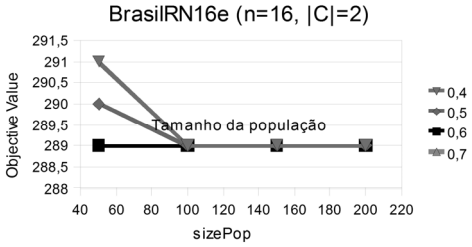


Fig. 6. Population size versus value of the best solution on BrasilRN16e

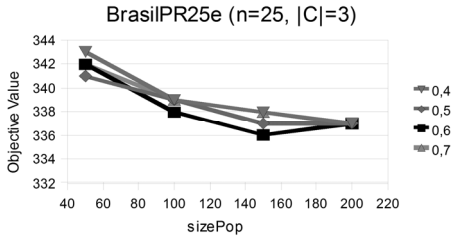


Fig. 7. Population size versus value of the best solution on BrasilPR25e

Table II shows that the GLPK did not solve nineteen non-Euclidean instances for which the algorithm stopped due to time or memory limitation. The algorithm stopped due to memory limitation on instances: Argentina16n, Argelia15n, India15n, India16n, Chade12n, Ira13n, Mexico14n, Canada17n, Arabia14n, Cazaquistao15n, Brasil16n and Russia17n. The algorithm stopped due to time limitation on instances: EUA17n, Mongolia13n, Sudao15n, Australia16n, BrasilAM26n, BrasilMG30n and att48nA. The percent deviations varied from 2 to 37.60. On sixteen among those nineteen instances the Memetic Algorithm found better solutions than those produced by the GLPK. The former algorithm also spent, in average, less processing time to obtain those solutions. The Memetic Algorithm spent, in average, from 14 to 1232 seconds to process the non-Euclidean instances. The results obtained with the GLPK took from 1 to 80000 seconds. The average frequency of the best solutions found by the Memetic Algorithm is 16.88 for the non-Euclidean instances.

V. CONCLUSION

This paper presented the Prize-collecting Car Renter Salesman Problem (pCaRS), a new variant of CaRS [2]. A

mathematical model was presented and submitted to the GLPK solver with time and memory limitations. A Memetic Algorithm that uses six local search procedures was proposed. An experimental investigation was carried out to investigate the potential of the proposed Memetic Algorithm. The solutions and processing times produced by the latter were compared with the results of the implementation of the mathematical model in the GLPK solver. A set with thirty two Euclidean and thirty two non-Euclidean instances was used in the experiments. Fourty exact solutions were found with the solver and the proposed heuristic algorithm established the first upper limits for other twenty four instances. The results of the computational experiments showed that for Euclidean instances the proposed Memetic Algorithm found almost all optima obtained with the GLPK with some advantage for the former. The Memetic Algorithm found also better solutions than the ones produced by the GLPK when it terminated due to time or memory limitations. The best performance due to quality of solution and processing time of Memetic Algorithm is observed on the non-Euclidean instances. The experiment also showed that as instances grow larger there is a tendency of the Memetic Algorithm improves its behavior in comparison to the GLPK solver.

As the problem proposed here is new, several innovations can be implemented for future research, such as: (a) to develop other metaheuristics to pCaRS, (b) to investigate other local search procedures and (c) to develop algorithms for other variations of the problem.

APPENDIX

An instance of pCaRS is illustrated in figure 8 on a complete graph with five vertices (cities) and three types of cars. A graph and a square matrix of order five are associated with each car. The graph associated with a car, shows the cost to use that car on each edge (the value on the edge). The satisfaction associated with each city (the number in brackets at each vertex) is shown in the three graphs. Those values are the same in the three graphs, once they do not depend on the car. Each car can be rented and delivered in any city. Thus, the element (i,j) of the matrix associated with each car shows the fee to deliver that car in city i when it was rented in the city j. Figure 9 shows the example of a solution to pCaRS. City A is the starting (and ending) point of the tour. Figure 9(a) shows that car 2 is rented in A, used to go from A to B and delivered in B. The cost associated with car 2 is the cost to travel road AB, 1, and the cost to be delivered in B, 1. The accumulated satisfaction up to this moment is given by 81+68 = 149. Figure 9(b) shows that car 3 is rented in city B, used to go travel roads BD and DE and delivered in E. The cost to go from B to E through D is 2 + 2 = 4. The cost to deliver car 3 in E is 3 The satisfaction to visit cities E and D is 73+27 = 100. Figure 9(c) shows that car 1 is rented in city E, goes from E to A, completing the tour, and is delivered in A. The cost to travel from E to A is 2 and the delivery fee is 3 Figure 9(d) shows the solution where the cities A, B, D and E form a tour. The costs associated to traveling the roads are 1 + 2 + 2 + 2 = 7. The costs associated with delivery fees are 1 + 3 + 2 = 6. The satisfaction to visit the cities in the tour is 81 + 68 +73 + 27 = 249. The final solution presented in figure 9(d) does not include city C. The final cost is 13 and the satisfaction reached is 249.

REFERENCES

[1] M. C. Goldbarg, A P.H. S. Asconavieta, E. F. G. Goldbarg "Memetic algorithm for the traveling car renter problem: an experimental investigation". *Memetic Computing*, v. 4, 2012, pp. 89-108.

[2] G. Gutin, and A.P. Punnen. "Traveling salesman problem and its variations", *Kluwer Academic Publishers*, Dordrecht, 2002

[3] P H. S. Asconavieta, M. C. Goldbarg and E. F. G. Goldbarg, "Evolutionary algorithm for the car renter salesman," *Proceedings of the IEEE CEC Congress on Evolutionary Computation* , vol. 1, pp. 593-600, 2011.

[4] M. C. Goldbarg, E. F. G. Goldbarg, P. H. Asconavieta, M. Menezes, H. P. L. Luna, "A transgenetic algorithm applied to the traveling car renter problem," *Expert Systems with Applications*, vol. 40, no 16, pp. 6298-6310, 2013.

[5] W. Souffriau, , P. Vansteenwegen, J. Vertommen, G. Vanden Berghe, and D. Van Oudheusden, "A personalised tourist trip design algorithm for mobile tourist guides". *Applied Artificial Intelligence* Vol. 22 , 2008, pp. 964–985.

[6] P. Vansteenwegen, D. Van Oudheusden. "The mobile tourist guide: An or opportunity". *OR Insights* vol. 20, 2007, pp. 21–27.

[7] P. Vansteenwegen, W. Souffriau, , G. Vander Berghe, D. Van Oudheusden, "The city trip planner: an expert system for tourists", *Expert Systems with Applications* 38, 2011, pp. 6540–6546

[8] Y. Yang; W. Jin, X. Hao. "Car rental logistics problem: A review of literature". In: *Service Operations and Logistics, and Informatics,* 2008. IEEE/SOLI 2008. IEEE International Conference on. [S.l.: s.n.], 2008. v. 2, p. 2815 - 2819.

[9] D. K. George,.C. H. Xia, "Fleet-sizing and service availability for a vehicle rental system via closed queueing networks". *European Journal of Operational Research*, v. 211, n. 1, p. 198 - 207, 2011.

[10] E. Balas, "The prize collecting traveling salesman problem". *Networks*, Vol. 19, 1989, pp. 621-636.

[11] GLPK (GNU Linear Programming Kit) package. Version 4.45.2, Avaliable: http://www.gnu.org/software/glpk/

[12] C. Miller, A. Tucker, R. Zemlin, "Integer programming formulations and travelling salesman problems". *Journal of the ACM* vol. 7, 1960, pp. 326–329.

[13] P. Vansteenwegen, W. Souffriau. and D. Van Oudheusden. "The orienteering problem: a survey". *European Journal of Operational Research*, 2010.

[14] L. Liberti, "Compact linearization for binary quadratic problems". *4OR*: Springer-Verlag , Volume 5, Issue 3, September 2007, pp. 231-245.

[15] R. Fortet. "Applications de l'algebre de boole en recherche operationelle". *Revue Francaise de Recherche Operationelle*, 4ed , 1960, pp. 17-26.

[16] F. Glover, E. Woolsey. "Converting the 0 - 1 polynomial programming problem to a 0 - 1 linear program". *Operations Research*, v. 22, n. 1, p. 180 - 182, 1974.

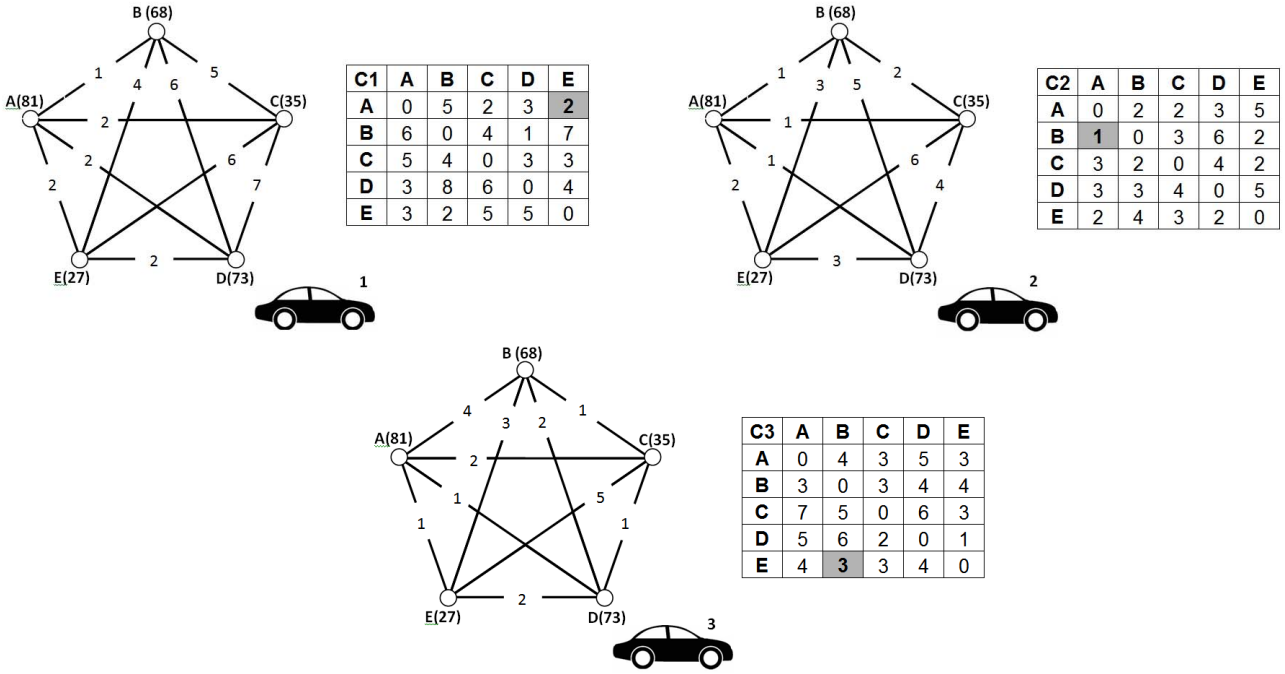
[17] AIMMS (2012, september, 12) "Integer Programming Tricks", Avaliable: http://www.aimms.com..

[18] P. Moscato. "On evolution, search, optimization, genetic algorithms and martial arts: Towards Memetic Algorithm". Caltech Concurrent Computation Program. California Institute of Technology, USA. 1989

[19] Y. S. Ong, M. H. Lim and X. S. Chen, "Research frontier: memetic computation - past, present & future", *IEEE Computational Intelligence Magazine*, Vol. 5, No. 2, 2010, pp. 24 -36

[20] G. A. Croes. "A method for solving traveling salesman problems" *Operations Res*. Vol. 6 , 1958 , pp., 791-812..

[21] G. Reinelt, "TSPLIB – A traveling salesman problem library", *ORSA Journal on Computing* vol. 3, 1991, pp. 376-384.



C1	A	B	C	D	E
A	0	5	2	3	2
B	6	0	4	1	7
C	5	4	0	3	3
D	3	8	6	0	4
E	3	2	5	5	0

C2	A	B	C	D	E
A	0	2	2	3	5
B	1	0	3	6	2
C	3	2	0	4	2
D	3	3	4	0	5
E	2	4	3	2	0

C3	A	B	C	D	E
A	0	4	3	5	3
B	3	0	3	4	4
C	7	5	0	6	3
D	5	6	2	0	1
E	4	3	3	4	0

Fig. 8. Operational costs and return fees for cars 1, 2 and 3.

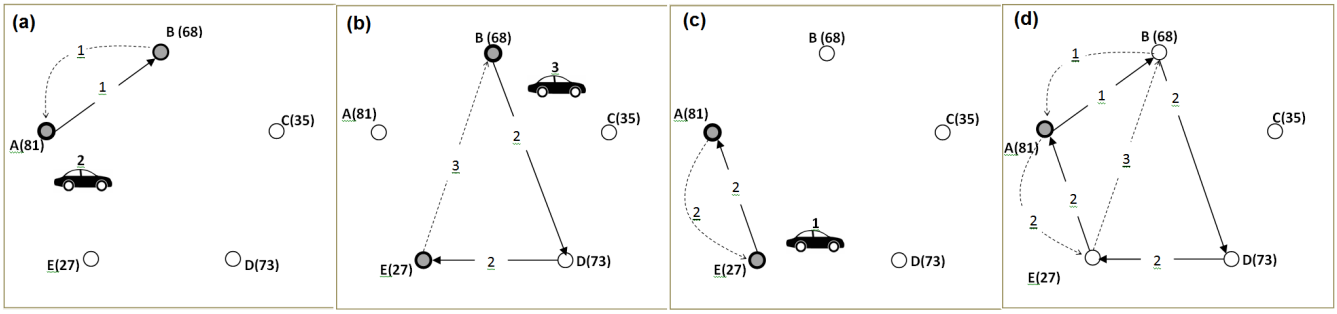


Fig. 9. Routes with edges associated with each car, the return fees and the final Hamiltonian cycle

TABLE I. RESULTS ON EUCLIDEAN INSTANCES

Instances	Exact (GLPK)	Memetic

Name	*nCity*	*nCar*	*Min*	*T(s)*	*GAP(%)*	*Min*	*Av*	*F*	*T(s)*	*GAP(%)*
Mauritania10e	10	2	**422**	5	0.00	**422**	422	30	17	0.00
Colombia11e	*11*	*2*	**326**	1	0.00	**326**	326	30	14	0.00
Angola12e	12	2	**490**	8	0.00	**490**	490	30	17	0.00
Peru13e	13	2	**556**	46	0.00	**556**	556	30	33	0.00
Libia14e	14	2	**521**	45	0.00	**521**	521	30	36	0.00
BrasilRJ14e	14	2	**230**	560	0.00	**230**	230	29	42	0.00
Congo15e	15	2	**513**	28	0.00	**513**	513	30	40	0.00
Argentina16e	16	2	**719**	2276	0.00	**719**	719	30	47	0.00
EUA17e	17	2	**602**	71,5	0.00	**602**	602	29	62	0.00
Bolivia10e	10	3	**384**	3	0.00	**384**	384	30	23	0.00
AfricaSul11e	11	3	**402**	11	0.00	**402**	402	30	24	0.00
Niger12e	12	3	**564**	53	0.00	**564**	567	24	39	0.00
Mongolia13e	13	3	**543**	607	0.00	**543**	545	1	31	0.00
Indonesia14e	14	3	**504**	22	0.00	**504**	504	28	43	0.00
Argelia15e	15	3	**487**	351	0.00	**487**	492	15	34	0.00
India16e	16	3	**705**	36	0.00	**705**	713	19	68	0.00
China17e	17	3	735	57332	6.40	**728**	731	13	86	5.50
Etiopia10e	10	4	**283**	2	0.00	**283**	283	30	17	0.00
Mali11e	11	4	**428**	10	0.00	**428**	428	30	28	0.00
Chade12e	12	4	**655**	861	0.00	**655**	657	25	48	0.00
Ira13e	13	4	**532**	49	0.00	**532**	533	24	64	0.00
Mexico14e	14	4	**492**	144	0.00	**492**	492	30	31	0.00
Sudao15e	15	4	**422**	46	0.00	**422**	422	29	28	0.00
Australia16e	16	4	**682**	453	0.00	**682**	710	3	98	0.00
Canada17e	17	4	**783**	1599	0.00	**783**	785	18	102	0.00
Arabia14e	14	5	**482**	50	0.00	**482**	482	30	37	0.00
Cazaquistao15e	15	5	**574**	1473	0.00	**574**	587	2	71	0.00
Brasil16e	16	5	**619**	718	0.00	637	637	28	64	2.83
Russia17e	17	5	760	80000	10.40	**750**	787	1	100	9.21
BrasilAM26e	26	3	371	80000	20.50	**338**	344	3	226	12.74
BrasilMG30e	30	3	419	80000	28.60	**368**	375	1	296	18.70
att48eA	48	3	25234	80000	38.12	**20444**	20954	1	950	23.62

TABLE II. RESULTS ON NON EUCLIDEAN INSTANCES

Instances			**Exact (GLPK)**			**Memetic**				
Name	*nCity*	*nCar*	*Min*	*T(s)*	*GAP(%)*	*Min*	*Av*	*F*	*T(s)*	*GAP(%)*
Mauritania10n	10	2	**306**	1	0.00	**306**	306	30	14	0.00
Colombia11n	11	2	**461**	224	0.00	**461**	461	30	19	0.00
Angola12n	12	2	**409**	50	0.00	**409**	409	30	21	0.00
Peru13n	13	2	**502**	3634	0.00	**502**	502	29	42	0.00
Libia14n	14	2	**504**	77307	0.00	**504**	504	30	34	0.00
BrasilRJ14n	14	2	**101**	58346	0.00	**101**	101	16	39	0.00
Congo15n	15	2	**573**	5747	0.00	**573**	573	27	35	0.00
Argentina16n	16	2	647	62007	22.70	**642**	644	6	56	22.10
EUA17n	17	2	**579**	80000	8.10	**579**	589	2	75	8.10
Bolivia10n	10	3	**448**	54	0.00	**448**	448	30	19	0.00
AfricaSul11n	11	3	**537**	7755	0.00	**537**	537	29	22	0.00
Niger12n	12	3	**607**	4069	0.00	**607**	630	1	34	0.00
Mongolia13n	13	3	**551**	80000	2.00	**551**	551	30	43	2.00
Indonesia14n	14	3	**522**	16188	0.00	**522**	522	30	38	0.00
Argelia15n	15	3	619	48859	17.30	**616**	620	8	65	16.90
India16n	16	3	734	73024	24.90	**723**	725	17	79	23.76
China17n	17	3	651	62162	20.10	**645**	654	3	72	19.36
Etiopia10n	10	4	**403**	153	0.00	**403**	403	30	22	0.00
Mali11n	11	4	**494**	262	0.00	**494**	494	30	32	0.00
Chade12n	12	4	654	69111	10.40	**649**	649	22	45	9.71
Ira13n	13	4	697	54936	20.90	**693**	693	10	41	20.44
Mexico14n	14	4	620	61358	18.70	**610**	612	24	38	17.37
Sudao15n	15	4	793	80000	29.50	**769**	776	2	67	27.30
Australia16n	16	4	551	80000	16.90	**525**	526	24	85	12.78
Canada17n	17	4	827	77542	23.80	**825**	872	2	112	23.62
Arabia14n	14	5	701	37832	27.80	**688**	693	6	76	26.44
Cazaquistao15n	15	5	843	67421	28.80	**830**	858	2	103	27.68
Brasil16n	16	5	769	32052	32.50	**742**	744	23	57	30.04
Russia17n	17	5	820	52468	37.60	**778**	782	8	118	34.23
BrasilAM26n	26	3	**107**	80000	17.80	**107**	108	4	197	17.80
BrasilMG30n	30	3	179	80000	30.20	**160**	161	4	352	21.91
att48nA	48	5	**443**	80000	17.50	**443**	598	1	1232	17.50