RAIRO Operations Research
Will be set by the publisher

REACTIVE GRASP FOR THE PRIZE-COLLECTING COVERING TOUR PROBLEM *

Glaubos Clímaco*[1], Isabel Rosseti[2], Rogério da Silva[3] and Marcos Guerine[4]

Abstract. This paper presents a greedy randomized adaptive search procedure (GRASP) for the prize-collecting covering tour problem, which is the problem of finding a route for traveling teams that provide services to communities geographically distant from large urban locations. We devised a novel hybrid heuristic by combining a reactive extension of the GRASP with Random Variable Neighborhood Search (VND) meta-heuristic for the purpose of solving the PCCTP. Computational experiments were conducted on a PCCTP benchmark from the literature, and the results demonstrate our approach provides a significant improvement in solving PCCTP and comparable with the state-of-the-art, mainly regarding the computational processing time.

Keywords: prize-collecting covering tour problem, hybrid heuristic, GRASP, VND

Mathematics Subject Classification. ???, ???

...

* *Corresponding author: francisco.glaubos@ufma.br*

[1] Universidade Federal do Maranhão, Av. dos Portugueses, 1966 - Vila Bacanga, Sao Luís, Brasil.

[2] Universidade Federal Fluminense, Instituto de Computação. Rua Passo da Pátria 156 São Domingos 24210-240 - Niterói, RJ - Brasil

[3] Institute of Education, Science and Technology of Piauí.R. Alvaro Mendes, 94 - Centro (Sul), Teresina - Brasil

[4] Federal Institute of Education, Science and Technology of Rio de Janeiro. Rua José Breves, 550, Centro, Pinheiral, RJ, Brazil

1. Introduction

The provision of social, medical or, legal assistance to communities geographically distributed and distant from large urban centers has been a concern of the various public, philanthropic or private entities. Deploying fixed units in these locations involves significant investments, and there is still a shortage of professionals with interest in being in those regions.

One of the most appropriate solutions is to attend to people who live in places far from the service centers is the use of traveling teams. As an example, the Brazilian Courts of Justice (TJ) have a project called Itinerant Justice, in which an adapted bus with a full forum structure is used to perform most of the legal services. This bus periodically arrives at cities far from the TJ units, thus covering a large number of citizens unable to have legal assistance.

In this context, the prize-collecting covering tour problem (PCCTP) arises. The objective of this problem is to find a minimum cost tour that passes through some mandatory cities (T) and, sometimes, others that are not mandatory (R). Each city has an associated prize, which represents the number of people served if the tour passes through it. Some cities may be inaccessible (W), and their inhabitants must travel to the closest city to receive assistance. Besides, the tour must guarantee a minimum attendance of people, so that the trip of a TJ bus is worthwhile.

The PCCTP is an NP-hard problem as it generalizes the NP-hard covering tour problem (CTP) [13]. Since it was first introduced by Gendreau *et al.* [13], several generalizations of the CTP have been proposed. Some of these variants refer to multi vehicles that must: collectively cover the cities of W [14]; minimize the sum of arrival times at visited locations, while the total duration of each tour does not exceed a preset time limit [11]; cover cities of W more than once [23]; and, handle a probabilistic coverage while maximizes the expected customer demand covered [17]. Other generalizations are three bi-objective versions that despite the traditional CTP objective: include the minimization of the cover [16]; maximize the number of covered cities W [10]; and, incorporate the minimization of expected uncovered stochastic demand [32].

All the aforementioned generalizations emerged from real-world needs that have not been considered by the classic CTP. In this work, we address the prize-collecting covering tour problem (PCCTP), in which every city $v \in V$ is associated with a prize p_v, and the tour must guarantee a preset minimum prize collection. This type of generalization, with prizes to be collected at the vertices, is common in the literature and has been widely addressed in other problems such as the traveling salesman problem [4], vehicle routing problem [31], and Steiner tree problem [5].

This problem is relatively new in the literature and there are two mathematical formulations describing it. The first one was proposed by Lyra [19] along with the problem and makes use of constraints based on flow to eliminate sub-routes in the solution. The second one was proposed by Silva [30], and it is based on multi-flow constraints to handle with sub-routes in the solution.

Since PCCTP is an NP-Hard problem, it is impracticable to solve large instances ($|N|$ greater than 100) using only exact methods [19]. Therefore, heuristic approaches are required to deal with such instances. Lyra [19] proposed a heuristic based on GRASP combined with a variation of the VNS (Variable neighborhood Search) metaheuristic [20]. Its heuristic approach also includes a path-relinking technique as an improving solutions component.

In the following, Silva [30] proposed new six heuristic methods, in which five of them are based on the ILS (Iterated Local Search) meta-heuristic [18], and one is an evolutionary heuristic. Silva [30] conducted experiments on test-problems proposed in his work and the ILS-RDM-CI was the one with the best overall performance.

The ILS-RDM-CI starts by generating an initial random solution through the cheapest Insertion (CI) heuristic. Then, at each iteration, a disturbance routine is performed followed by a Random Descent Method (RDM) procedure. The RDM applies a random movement over the best current solution (S) to generate a new neighbor solution $N(S)$. If this neighbor is better than the S, $N(S)$ becomes the new best current solution. Otherwise, another neighbor is generated. The RDM method stops when a maximum number of iterations without improvement is reached, and the whole ILS-RDM-IC heuristic stops when a maximum number of disturbances is reached.

In this paper, we propose a hybrid heuristic derived from the reactive GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristic [27] for solving the prize-collecting covering tour problem. The GRASP metaheuristic is performed, iteratively, in two main phases: constructive and local search. The first phase is based on a GENIUS heuristic [12] adapted for the PCCTP, while in the second phase, several moves are designed and combined in a VND (Random Variable neighborhood Descent) structure [20]. The resulting R-GRASP heuristic is fast, and the extensive computational results show the solutions to be equal to or better than those obtained by the best existing heuristic ILS-RDM-CI. Besides, we have implemented and tested the two mathematical formulations from the literature using a mixed-integer linear programming (MILP) solver, and compared the results with the heuristic.

The remainder of this paper is organized as follows. Section 2 defines formally the PCCTP and its existing mathematical formulations. Section 3 shows the proposed R-GRASP heuristic. Section 4 presents the computational results obtained by the proposed and the state-of-the-art heuristics; and in Section 5, some conclusions, and a few future works are drawn.

2. Problem definition

The definition of the PCCTP can be given as follows. Consider an undirected graph $G = (N, E)$, with the set of vertices $N = V \cup W$ and $V = R \cup T$. Let D be a coverage distance, c_e be a cost associated to each $e \in E$ and p_i be a prize associated to each vertex $i \in V$. Let T be a subset containing the vertices that

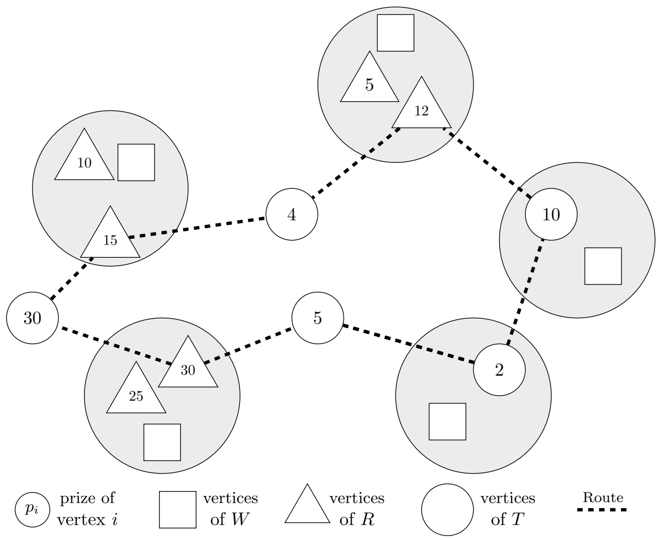

FIGURE 1. Example of a PCCTP solution.

must be visited and R a subset of vertices that are optional, and therefore may or may not be visited. Finally, let W be a subset containing the vertices that must be covered by another vertex from V, i.e., a vertex $i \in V$ covers a vertex $j \in W$ if $c_{ij} \leq D$. The goal of the PCCTP is to find a simple minimum cost cycle that visits all vertices in T, covers all vertices of W and collects at least a minimum prize (*PRIZE*). In Figure 1 is depicted a solution for an example of the PCCTP, in which the value of *PRIZE* was set to 100.

A real-world application for the PCCTP can be seen in the provision of a route for the TJ's bus case, mentioned in Section 1. The vertices of T would represent cities of a particular and important region and thus are defined as mandatory visiting points, while the vertices in R would represent cities in which the visits are optional. There is also a set of cities that shall be attended, but for some reason, it is impracticable and their population must be served by some neighboring cities whose distance is at most D unities away, avoiding large displacement of people. These last set of cities would be represented by the vertices in W. To be worth the release of a bus, it is necessary that a minimum number of people be attended. Therefore, the population of each city in T and R is represented by the prize of each vertex, and the total *PRIZE* to be collected corresponds to the minimum total number of citizens to be served.

2.1. Mathematical formulations

In the literature, there are two mathematical formulations for the PCCTP. The first formulation, proposed by Lyra [19], uses flow variables to avoid the formation of disconnected cycles. The other formulation proposed by Silva [30], uses multi-flow variables to avoid sub-cycles in the route.

In this section, let $G = (V, A)$ be a complete and directed graph in which V and A are, respectively, the sets of vertices and arcs, and u a root vertex belonging to T, chosen as the origin of the route. According to Lyra [19], the PCCTP can be formulated as a model of integer linear programming (ILP) using the following variables:

- z_{ij}: a non-negative integer variable that represents the amount of flow flowing in the arc (i, j);
- y_k: a binary variable for every $k \in V$. $y_k = 1$ if vertex k is in the route, and $y_k = 0$ otherwise;
- x_{ij}: a binary variable for every arc $(i, j) \in A$. It assumes the value equal to one if the arc (i, j) belongs to the route, and value equal to zero, otherwise.

In addition, c_{ij} represents the cost of using the arc $(i, j) \in A$, p_k indicates the premium associated with vertex $k \in V$, $PRIZE$ is the minimum prize to be collected for the route, and R_w is the set of all $k \in V$ vertices that cover $w \in W$. Thus, the ILP model can be defined as follows:

$$Min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

$$\text{s.a}: \sum_{k\in V} p_k y_k \geq PRIZE \tag{2}$$

$$\sum_{k\in R_w} y_k \geq 1, \qquad \forall w \in W \tag{3}$$

$$\sum_{(i,k)\in A} x_{ik} + \sum_{(k,j)\in A} x_{kj} = 2y_k, \qquad \forall k \in V \tag{4}$$

$$\sum_{j\in V} z_{kj} = \sum_{i\in V} z_{ik} + y_k, \qquad \forall k \in V \setminus \{u\} \tag{5}$$

$$\sum_{j\in V} z_{uj} = 1 \tag{6}$$

$$\sum_{j\in V} z_{ju} = \sum_{j\in V\setminus\{u\}} y_j, \tag{7}$$

$$x_{ij} \leq z_{ij}, \qquad \forall (i,j) \in A \tag{8}$$

$$x_{ij} \geq z_{ij}/(|V| + 1), \qquad \forall (i,j) \in A \tag{9}$$

$$y_k = 1, \qquad \forall k \in T \tag{10}$$

$$y_k \in \{0, 1\} \qquad \forall k \in R \tag{11}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i,j) \in A \tag{12}$$

$$z_{ij} \in \mathbb{Z}^+ \qquad \forall (i,j) \in A \tag{13}$$

In this formulation, the objective function (1) minimizes the cost of the route. Constraint (2) ensures that the minimum prize, $PRIZE$, is collected. The constraints (3) ensure that each vertex of W is covered by at least one vertex of the route. Constraints (4) are responsible for conserving the flow, while the constraints (5), (6) and (7) prevent disconnected cycles from the route vertex. Constraints (8), and (9) ensure that the routes generated by flow variables z_{ij} and binary variables x_{ij} coincide. Constraints (10) ensure that all vertices of T are in the route. Finally, constraints (11), (12) and (13) determine the domain of the variables y_k, x_{ij} and z_{ij}. The formulation of Silva [30] is composed of the following variables.

- x_{ij}: a binary variable that assumes value equal to one if the route contains the arc (i, j), and value equal to zero, otherwise;
- y_k: a binary variable that indicates if the vertex k is in the route or not, therefore, $y_t = 1$ for every $t \in T$;
- and z^k_{ij}: a non-negative integer variable representing the amount of flow from product k drained to the arc $(i, j) \in A$.

Then, the ILP formulation of Silva [30] can be presented as follows:

$$Min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{14}$$

$$s.a: \sum_{j\in V:j\neq i} x_{ij} = y_i, \quad \forall i \in V \tag{15}$$

$$\sum_{i\in V:i\neq j} x_{ji} = y_j, \quad \forall j \in V \tag{16}$$

$$z^k_{ij} \leq x_{ij}, \quad \forall (i,j) \in A, k \in V \tag{17}$$

$$\sum_{i\in V\setminus\{u\}} z^k_{ui} = y_k, \quad \forall k \in V \setminus \{u\} \tag{18}$$

$$\sum_{i\in V\setminus\{u\}} z^k_{iu} = 0, \quad \forall k \in V \setminus \{u\} \tag{19}$$

$$\sum_{i\in V\setminus\{k\}} z^k_{ik} = y_k, \quad \forall k \in V \setminus \{u\} \tag{20}$$

$$\sum_{j\in V\setminus\{k\}} z^k_{kj} = 0, \quad \forall k \in V \setminus \{u\} \tag{21}$$

$$\sum_{i\in V:i\neq j} z^k_{ij} - \sum_{i\in V:i\neq j} z^k_{ji} = 0, \quad \forall k, j \in V \setminus \{u\}, j \neq k \tag{22}$$

$$\sum_{i\in V} p_i y_i \geq PRIZE \tag{23}$$

$$\sum_{i\in R_w} y_i \geq 1 \quad \forall w \in W \tag{24}$$

$$y_i = 1 \quad \forall i \in T \tag{25}$$

$$y_i \in \{0,1\} \quad \forall i \in R \tag{26}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{27}$$

$$z_{ij}^{k} \in \mathbb{Z}^{+} \qquad \forall (i,j) \in A, k \in V \qquad (28)$$

In this formulation, the objective function (14) minimizes the total cost of the route. Constraints (15) and (16) guarantee that if a vertex is in the route, then that vertex has a degree equal to two. Constraints (17) limit the flow of products to the edge of the route. Constraints (18) ensure that only one product is shipped from the origin to each customer present on the route. Constraints (19) do not allow any product to return to its origin. Constraints (20) and (21) require that every vertex will receive its corresponding product, which in turn should not be sent to any other vertex. Constraints (22) guarantee the conservation of the flow of products that have not reached their destination vertices. Constraint (23) demands that the *PRIZE* will be collected. Constraints (24) guarantee that every vertex of W is covered by at least one vertex of the route. Constraints (25) require that all vertices of T be in the route. Finally, constraints (26), (27) and (28) represent the domain of the variables y_i, x_{ij}, and z_{ij}^{k}.

3. A GRASP algorithm

The GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristic is a multi-start process that can obtain sufficiently good solutions for computationally difficult problems. This method has been applied successfully in solving various optimization problems, in several areas such as scheduling [22], telecommunications [3], routing [15], partitioning, allocation, and assignment [27].

This metaheuristic is executed iteratively, and each iteration is composed of two phases: construction and local search. A feasible solution S is generated in the construction phase, and then its neighborhood $N(S)$ is explored by a local search, in order to find a better solution. This iterative process is repeated until a stop criterion is reached, that can be a limited number of iterations allowed, a number of iterations without solution improvement, among others. Once this stopping criterion is reached, the best solution found in all iterations S^* is returned.

3.1. Construction phase

For generating an initial solution for our proposed heuristic, we devised an adaptation of the GENIUS heuristic [12] originally proposed for the TSP and it is divided into two phases: a construction approach, the GENI step (Generalized Insertion), and a method of improvement, the Unstringing and Stringing (US) step.

The GENI is a generalized insertion-based method whose main characteristic is that the evaluation of the possible insertions of a vertex is not essentially limited to a position between consecutive vertices. In the GENI step, there are two different ways for inserting a vertex v. Suppose v will be inserted between two other vertices v_i and v_j of the route. Since the route has an orientation, consider v_k a vertex in the path from v_j to v_i, and v_l a vertex in the path from v_i to v_j. Also, consider that given a v_h vertex of the route, v_{h-1} is its predecessor and v_{h+1} is its successor. The two types of insertion are explained below, and illustrated in Figures 2 and 3:

- *Insertion type 1*: before inserting v, the edges (v_i, v_{i+1}), (v_j, v_{j+1}) and (v_k, v_{k+1}) are removed, and then replaced by the four new edges (v_i, v), (v, v_j), (v_{i+1}, v_k) and (v_{j+1}, v_{k+1}), such that $k \neq i$ and $k \neq j$. For the preservation of the route orientation, the direction of the paths (v_{i+1}, ..., v_j) and (v_{j+1}, ..., v_k) are inverted.

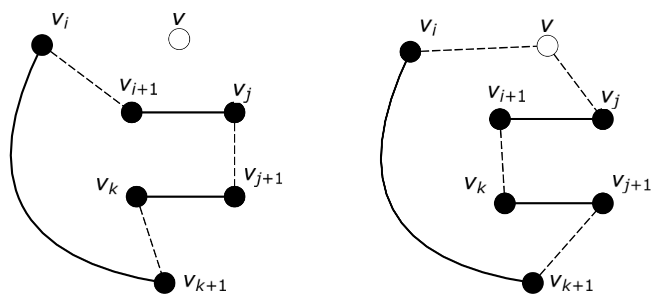

FIGURE 2. GENI - Insertion type 1 of vertex v between v_i and v_j.

- *Insertion type 2*: before the insertion, one verifies if $v_k \neq v_j$, $v_k \neq v_{j+1}$, $v_l \neq v_i$ and $v_l \neq v_{i+1}$. If it is true, so the removal of the following edges occurs: (v_i, v_{i+1}), (v_{l-1}, v_l), (v_j, v_{j+1}) and (v_{k-1}, v_k), followed by the insertion of the edges (v_i, v), (v, v_j), (v_l,v_{j+1}), (v_{k-1} , v_{l-1}) and (v_{i+1} , v_k). Thus, as in the previous insertion, the orientation of the paths (v_{i+1}, ..., v_{l-1}) and (v_l,..., v_j) are inverted to maintain the direction of the route.

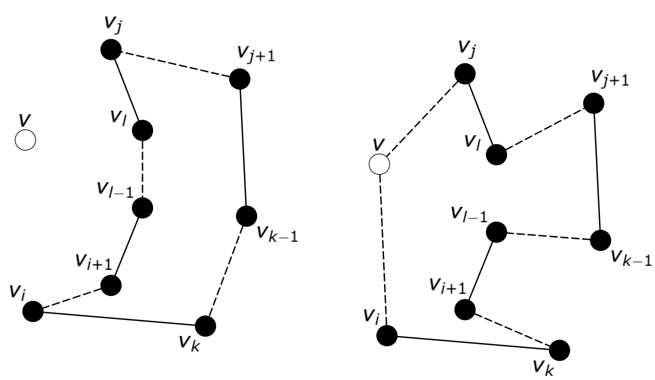

FIGURE 3. GENI - Insertion type 2 of the vertex v between v_i and v_j

In the Algorithm 1, the pseudo-code of the GENI step is illustrated, in which the method receives a partial route S and a randomly chosen vertex v to be inserted. In line 1, the neighborhood $N_p(v)$ is created for each vertex v. $N_p(v)$ consists of the p vertices of the route that are closest to v, according to the instance distance matrix. From lines 3 to 16, for every two neighbors of v and for each orientation of the route (clockwise and counter-clockwise), both types of insertion are tested. Finally, the best solution is returned on line 17.

Algorithm 1:GENI(S, v)

```
1:  N_p ← DefineNearbyVertices(V);
2:  f* ← ∞; S* ← S;
3:  for v_i, v_j ∈ N_p(v), v_i ≠ v_j do
4:    for orientation ∈ {clockwise,counter-clockwise} do
5:      S' ← insertion_type1(v_i, v_j, S, v, orientation);
6:      if f(S') < f* then
7:        S* ← S';
8:        f* ← f(S*);
9:      end if
10:     S'' ← insertion_type2(v_i, v_j, S, v, orientation);
11:     if f(S'') < f* then
12:       S* ← S'';
13:       f* ← f(S*);
14:     end if
15:   end for
16: end for
17: return S*
```

After the first step, the US step is applied to improve the route constructed. In this step, iteratively, each one of the vertices in S is removed and reinserted into S, using the GENI procedure. Similarly to the GENIUS construction step, the US has two types of removal for a given vertex. Let $N_p(v)$ be the p closest neighbors of a vertex v, the two types of removal are detailed below and illustrated in Figures 4 and 5:

- *US - Removal type 1*: let $v_j \in N_p(v_{i+1})$, and let $v_k \in N_p(v_{i-1})$ be a vertex in the path (v_{i+1},..., v_{j-1}) for a given orientation. When removing the vertex v_i, the edges (v_{i-1}, v_i), (v_i, v_{i+1}), (v_k, v_{k+1}) and (v_j, v_{j+1}) are also removed. Then, the route is reconnected using the edges (v_{i-1}, v_k), (v_{i+1}, v_j) and (v_{k+1}, v_{j+1}). Moreover, the direction of the paths (v_{i+1},..., v_k) and $(v_{k+1}, ..., v_j)$ must be inverted.
- *US - Removal type 2*: given an orientation of the route, consider $v_j \in N_p(v_i + 1)$ and v_k a vertex in the path $(v_{j+1}, ..., v_{i-2})$, in which $v_k \in N_p(v_{i-1})$, and $v_l \in N_p(v_k + 1)$ belongs to the path (v_j,...,v_{k-1}). When removing the vertex v_i from the route, one also removes the edges (v_{i-1}, v_i),-$(v_i, v_{i+1}), (v_{j-1}, v_j), (v_l, v_{l+1})$ and (v_k, v_{k+1}), and then, the connectivity

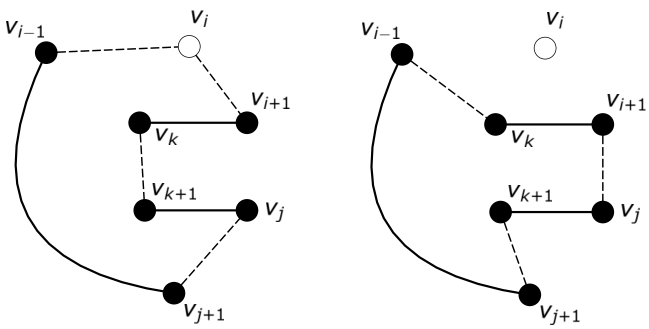

FIGURE 4. US - Removal type 1 of the vertex v_i.

of the route is re-established including the edges $(v_{i-1}, v_k), (v_{l+1}, v_{j-1})$,-$(v_{i+1}, v_j)$ and (v_l, v_{k+1}). Lastly, the direction of the paths $(v_{i+1}, ..., v_{j-1})$ and $(v_{l+1}, ..., v_k)$ are inverted to ensure the orientation of the route.

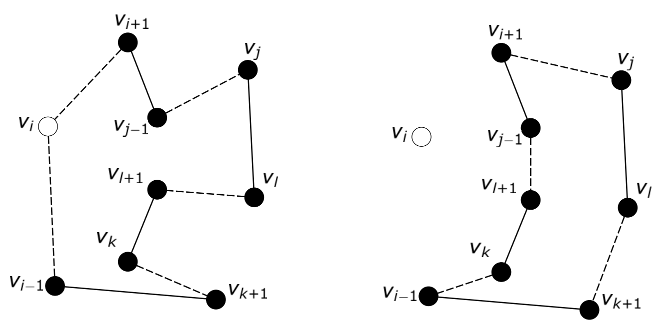

FIGURE 5. US - Removal type 2 of the vertex v_i.

The Algorithm 2 presents, in more detail, how the US step is performed. From lines 3 to 16, for each vertex in S and each direction of the route, the two types of removal are executed. For reinserting the vertex v in the route, every removal is followed by the two insertions types mentioned in the GENI step. Finally, in line 17, the route with the lowest cost is returned.

For the PCCTP, we have used the GENIUS procedure as follows. A partial route is generated by three vertices randomly chosen from T, and then, as long as the $PRIZE$ is not reached and there is any uncovered vertex $w \in W$, the following phases are performed: i) a candidate list (CL) is created containing all vertices outside the route; ii) a restricted candidate list (RCL) is formed by the best-quality vertices from CL; iii) a vertex is selected at random from the RCL and inserted in the route using the GENI procedure; and iv) one tries to improve the route by performing the US method. The pseudo-code of the GENIUS procedure is presented in Algorithm 3.

Algorithm 2: US(S)

```
S* ← S
f* ← f(S*)
for each v ∈ S do
    for orientation ∈ {clockwise, counter-clockwise} do
        S′ ← removal_type1(v, S, orientation)
        if (f(S′) < f*) then
            S* ← S′
            f* ← f(S*)
        end if
        S″ ← removal_type2(v, S, orientation)
        if (f(S″) < f*) then
            S* ← S″
            f* ← f(S*)
        end if
    end for
end for
return S*
```

A vertex is inserted into the RCL if its incremental cost is inferior to the threshold $c^{min} + (\alpha \times (c^{max} - c^{min}))$, where c^{min} and c^{max} are, respectively, the smallest and the largest incremental costs in CL. The greedy parameter $\alpha \in [0, 1]$ indicates how greedy the heuristic is, the higher the value of α the greedier.

For calculating the incremental cost, we used the following greedy function, which considers the insertion of a vertex k between two other vertices i and j.

$$g(k) = \min_{i,j \in V : i \neq j} (c_{ik} + c_{kj} - c_{ij}) \tag{29}$$

where c_{ij}, c_{ik} and c_{kj} are, respectively, the cost of the edges (i, j), (i, k) and (k, j).

3.2. Local search phase

Each solution built at the constructive phase is the starting point for a local search procedure in which we try to improve the solution. In our approach, the solution goes through a local search performed by the VND method [20]. Proposed by Mladenović and Hansen [20], the VND is a local search metaheuristic that uses different neighborhood structures. Given an ordered list of neighborhoods, the VND starts by exploring the first neighborhood of S, $N^k(S)$, and if a better solution is not found, the next neighborhood $N^{(k+1)}(S)$ is explored. Otherwise, it returns to the first neighborhood on the list. The algorithm stops when all neighborhood structures are explored, returning the best solution found.

Algorithm 3: GENIUS($seed$,α)

$i, j, k \leftarrow$ `SelectRandom`$(T, seed)$;
$S \leftarrow \cup \{i, j, k\}$;
$CL \leftarrow V$;
$CL \leftarrow CL \setminus \{i, j, k\}$
while $(\exists\ t \in T : t \notin S)$ **or** $(\exists$ not covered $w \in W)$ **or** ($PRIZE$ not collected) **do**
  $RCL \leftarrow$ `setRCL`(CL, α);
  $v \leftarrow$ `SelectRandom`$(RCL, seed)$;
  $S' \leftarrow GENI(S, v)$;
  $CL \leftarrow CL \backslash v$;
end while
$S'' \leftarrow US(S')$; *//route improvement*
return S'';

The VND is detailed in the pseudo-code of Algorithm 4. From lines 2 to 10, it tries the next neighborhood if the current solution is not improved. Otherwise, the local optimal solution for the current neighborhood is obtained, and k is set to 1 so that the loop restarts for the newly accepted solution. When none of the neighbors are able to improve S, i.e., $k > k_{max}$, the best current solution is returned.

For the VND, we used the following 15 neighborhood structures based on classical movements for the TSP as *swap* and *shift*, and heuristics as GENIUS and Cheapest Insertion. In particular, one of them, the *double_remove_simple_insert-_cheapest*, is proposed in this work.

Of the 15 neighborhood structures used, ten are intra-route:

(1) *shift*: changes the position of a vertex within the route;
(2) *swap*: swaps the position of two vertices in the route;
(3) *or-opt*: movement similar to *shift*, but the position of n vertices is changed;
(4) *2-opt*: removes two non-adjacent edges of the solution and inserts two new ones to keep the single cycle;
(5) *3-opt*: removes three non-adjacent edges and inserts three new ones, similar to *2-opt*;
(6) *remove_simple_re_insert_cheapest*: removes a vertex and reinserts it via the cheaper insertion;
(7) *remove_simple_re_insert_genius*: removes a vertex and reinsert it via GENIUS method;
(8) *remove_genius_re_insert_cheapest*: Removes a vertex using a GENIUS removal method and inserts using lower cost criteria between adjacent vertices;
(9) *remove_genius_re_insert_genius*: removes and reinsert a vertex of the route using GENIUS;
(10) *remove_cheapest_re_insert_genius*: removes using cheaper insertion criteria and re-enter via GENIUS;

and five of them are extra-route:

(1) *double_remove_simple_insert_cheapest*: tries to replace two vertices with a single one that does not belong to the solution;
(2) *remove_simple_insert_cheapest*: replaces a vertex of the route with an outside vertex, inserting via cheaper insertion;
(3) *remove_simple_insert_genius*: replaces a vertex of the route with an outside vertex, inserting via GENIUS;
(4) *remove_genius_insert_genius*: replaces a vertex of the route with an outside vertex, by removing and inserting via GENIUS; and
(5) *remove_genius_insert_cheapest*: removes a vertex from the solution via GENIUS and inserts a new one via cheapest insertion.

It is important to remark that only feasible movements are performed.

Algorithm 4: VND(S).

$k \leftarrow 1$
while $k \leq k\_max$ **do**
    $S' \leftarrow firstImprovingSolution(N^{(k)}(S))$;
    if *S' is better than S* **then**
        $S \leftarrow S'$;
        $k \leftarrow 1$;
    else
        $k \leftarrow k + 1$;
    end if
end while
return S

3.3. R-GRASP heuristic

In the construction phase of the R-GRASP, a preliminary computational experience showed that no value of α always produced the best results. Therefore, we decided to devise a reactive version of the GRASP proposed by Prais and Ribeiro [24], in which α is taken at random from a set of discrete values. Initially, all α values have the same probability of being chosen. In the iterative process, one keeps the value of the solutions obtained for each value of α. After a certain number of iterations, the probabilities are updated. Those corresponding to values of which have produced good solutions are increased and, conversely, those corresponding to values producing low-quality solutions are decreased. The Reactive GRASP (R-GRASP) is described in Algorithm 5, and the value of β is fixed at 10, as in [24].

From lines 1 to 5, some variables are initialized and, from lines 6 to 24, while the maximum number of iterations $maxIt$ is not reached, the following instructions are executed. In line 7 α^* is chosen at random from set $\mathcal{D}$ with probability of p_α. In lines 8 and 9, at each iteration, the solution construction and local search

Algorithm 5: R-GRASP($maxIt$, $seed$, $\beta = 10$)

1: $\mathcal{D} \leftarrow \{0.1, 0.2 \ldots, 0.9\}$ {`set of possible values for` α}
2: $n_{\alpha^*} \leftarrow 0$ {`number of iterations with` α^*, $\forall \alpha^* \in \mathcal{D}$}
3: $Sum_{\alpha^*} \leftarrow 0$ {`sum of values of solutions obtained with` α^*}
4: $p_{\alpha = \frac{1}{|\mathcal{D}|}} \quad \forall \alpha \in \mathcal{D}$
5: $S_{best} \leftarrow \infty$; $S_{worst} \leftarrow 0$; $it \leftarrow 0$
6: **while** ($it < maxIt$) **do**
7: Choose α^* from $\mathcal{D}$ with probability of p_α
8: $S \leftarrow GENIUS(seed, \alpha^*)$
9: $S' \leftarrow VND(S)$
10: **if** $S' < S_{best}$ **then**
11: $S_{best} \leftarrow S'$
12: **end if**
13: **if** $S' > S_{worst}$ **then**
14: $S_{worst} \leftarrow S'$
15: **end if**
16: $sum_{\alpha^*} \leftarrow sum_{\alpha^*} + S'$
17: $n_{\alpha^*} \leftarrow n_{\alpha^*} + 1$
18: **if** $mod(it, 15) == 0$ **then**
19: $mean_\alpha \leftarrow \frac{sum_{\alpha^*}}{n_{\alpha^*}}$
20: $eval_\alpha \leftarrow \left(\frac{S_{worst} - mean_\alpha}{S_{worst} - S_{best}}\right)^\beta \quad \forall \alpha \in D$
21: $p_\alpha \leftarrow \frac{eval_\alpha}{(\sum_{\alpha' \in D} eval_{\alpha'})} \quad \forall \alpha \in D$
22: **end if**
23: $it \leftarrow it + 1$;
24: **end while**
25: **return** S_{best};

procedures are performed. An initial solution S is built using the GENIUS method, adapted for the PCCTP, and then the local search is performed employing the VND method. At the end of the local search, the VND returns a solution value S' that is compared to the best current solution value (line 10). If S' is better than S_{best}, then the best solution value is updated (line 11). From lines 13 to 15, the worst solution value found so far is updated. From lines 18 to 22, the probabilities are updated every 15 GRASP iterations. Finally, in line 25, the best solution value obtained over all iterations is returned.

4. Computational results

In this section, it is presented the computational experiments performed with the R-GRASP, the ILS-RDM-CI, and the mathematical formulations. All strategies presented were implemented in C programming language and compiled with GCC version 4.7.0. All experiments were carried out on a 3.2GHz $IntelCore^{TM}$ i5 CPU under Linux Fedora 15 operational system. As regards the MIP solver, Gurobi optimizer [21] was used disabling its preprocessing heuristics, cuts, and the parallel mode set to none. We remark that the original source code of the

ILS-RDM-CI heuristic was kindly provided by its author Silva [30] and used in our computational experiments.

4.1. Instances

The instances used in this work were proposed by Silva [30], and consist of 144 test problems adapted from the traveling salesman problem (TSP) available at the TSPLIB repository [26]. To prevent any instance of the PCCTP from becoming an instance of the covering tour problem (CTP), the *PRIZE* is never satisfied with the collection of only the prizes associated with the mandatory vertices T.

These instances were named according to their original name in the TSPLIB, adding additional information about the subsets R, T, W and the percentage of prize to be collected, concerning the total available prize. For example, the instance *brazil58_R18_T20_W20_25* has the following characteristics: 58 vertices in total; 18 optional vertices R; 20 mandatory vertices T; 20 vertices to be covered W; and 25% of the total instance prize must be collected. These instances were made public in the Mendeley repository (see [6]).

4.2. Mathematical formulations

This section presents an unprecedented comparison between the two formulations present in the literature, Flow [19] and Multi-flow [30], since the Flow formulation has never been tested before in Silva's instances [30]. Then, the formulation with the best results will be compared with the performance of the proposed heuristic.

Table 1 presents the results obtained by executing these formulations. The first column refers to the name of the instances. From the second column, the solution, the best bound and the time spent (in seconds) by each formulation are presented. For clarity and ease of reading purposes, it was decided to present only the 50 instances that yield the biggest differences of performance between the formulations, either in solution quality or CPU time. The complete results are available in a Mendeley repository (see [7]).

The fields with "- " indicate that, due to the time exceeded by one hour of processing, or problems related to lack of memory, it was not possible to solve the respective instance. The values in bold indicate the best results in processing time and the best solution found.

From Table 1, it can be seen that, among the most prominent results, it is shown that the Multi-flow formulation [30] outperforms the Flow formulation [19] in terms of processing time and quality of the solution obtained. Considering all the 144 instances and the solution quality achieved, the Multi-Flow model wins in 46, ties in 82, and loses in 16 instances. Concerning the CPU time spent, the formulation of Silva [30] is faster for 48 instances, slower for just one instance, and ties in 95 cases, where the time limit is reached in both models.

4.3. Heuristics

The GRASP heuristic includes a probabilistic behavior by setting, in the construction phase, a restricted candidate list (RCL) with the best candidates. In our proposal, the RCL is composed of all elements $v \in CL$ whose incremental cost is inferior to $c^{min} + \alpha^{*}(c^{max} - c^{min})$.

As we made use of a reactive GRASP, there was no need to tune the α parameter, so the tuning experiments were made just for the $maxIt$ parameter. We have conducted the tuning experiment on a subset of 37 instances that best represent the whole set of instances. By running ten executions for each one of the 37 instances, and for each value of $maxIt \in \{50, 60, 70, 80, 90, 100\}$, we realized that $maxIt = 70$ fits better our approach.

Next, we conducted experiments with the remaining 107 instances, which were not used for tuning. Both approached were run ten times with the same ten different seeds, and the results are presented in Table 2. As in the section above, we have decided to present only the 50 instances in which R-GRASP had the greatest impact over ILS-RDM-CI, and the complete results are available in a Mendeley repository (see [7]).

In Table 2, the first column indicates the name of the instance, and the results obtained by the MIP solver Gurobi through the formulation of Silva [30], are presented on the second and third columns. The remaining columns show for each method, respectively, the best solution, the average solution, and the average time spent. The last column shows the percentage time difference between the compared heuristics according to equation (30). Moreover, the symbol "-" indicates that no solution was found by Gurobi within a one-hour CPU time, and the best results are bold-faced.

$$Diff_T = 100 * \frac{(T_{grasp} - T_{ils})}{T_{ils}} \tag{30}$$

From Table 2, one can observe that, regarding the best solution, the proposed approach outperforms the ILS-based heuristic in five and ties in the remaining instances. Concerning the average solution, the performance of the R-GRASP is even better, improving the results of the literature in 14 cases. The major improvement obtained by our heuristic was in terms of computational time spent, in which a reduction of time above 96% was applied for all 50 instances.

The summary of the complete results, with all the 107 instances, are presented in the Table 3, which is shown how many times the R-GRASP was superior (or not) to the ILS-RDM-CI and how many ties occurred, in terms of the best solution, average solution, and average time in ten executions. To better compare the results from Table 3, we have applied the Non-parametric Friedman's test [29] which presents, among brackets, the number of R-GRASP wins and losses that have statistical significance with a certain confidence interval.

Friedman's test is normally used to evaluate heuristics that make use of randomness, by identifying whether or not the difference between their averages were due to the superiority of some of the heuristics, or just due to the randomness of the methods. We have used the implementation provided by the [25] package, and we have set the p-value equal to 0.05. Two hypotheses were made:

- The null hypothesis (H_0): There are no significant differences between the average solutions found by the compared heuristics; and
- the alternative hypothesis (H_1): There are significant differences between the average solutions found by the compared heuristics.

Hence, H_0 can be rejected with 95% of confidence if, for each instance, Friedman's test outputs a value smaller than or equal to the p-value. If H_0 is rejected, the alternative hypothesis H_1 is considered.

From Table 3, we can observe that regarding the best solution, our approach is capable of improving them in 11 cases. Concerning the average solution, the achievements of our approach are more significant. The R-GRASP reached better average solutions for 48 instances out of 107, in which there is statistical significance for 37 of them. On the other hand, the ILS-RDM-CI performed better in only 19 cases, of which seven are statistically significant. Regarding the processing time, a huge improvement was achieved by our approach, which was faster in 96 instances.

5. Conclusions

Both GENIUS and VND heuristics have been successfully applied to different combinatorial problems. In this work, we have proposed a Reactive GRASP heuristic, combining both GENIUS and VND to find good quality solutions for the PCCTP. The GENIUS heuristic was used to create an initial solution with the α periodically updated over the iterations, while the VND procedure was used to perform the local searches, randomly exploring the solution space with insertions, removals and swap movements.

In order to verify the contribution of our proposal, experiments were performed in instances from literature and the results were compared to the state-of-the-art heuristic ILS-RDM-CI. The computational results showed that our proposal can produce, on average, 82.24% better or equal solutions than the ILS-based heuristic. In terms of computational time spent, the proposed heuristic was far faster than ILS-RDM-CI, employing a huge average time reduction of at least 80%, on 90% of the instances. Also, the R-GRASP results seem particularly interesting in terms of the intermediate instances, where exact resolution is not possible within reasonable time constraints, if the well-known commercial software, Gurobi, is used.

However, we must admit that even R-GRASP needs quite a lot of time when dealing with such large instances as rd400_R80_T240_W80_25, pr226_R44_T136_W46_50, and gil262_R51_T53_W158_25. In order to improve R-GRASP performances on PCCT problems, we may reconsider the GENI procedure which represents the major part of the computational time. For the construction phase, we could avoid building the initial solution from scratch by using long-term memory, through data mining or path-relinking [28]. For improving the solution quality, we intend to study the incorporation of a MIP model to the R-GRASP for solving related subproblems of the PCCTP, since it has been proved to be a interesting approach [1, 2, 8].

Finally, we would like to note that the algorithm is quite flexible and could be adapted to accommodate other conditions or constraints, such as the covering tour problem [9] or prize-collecting traveling salesman problem [4].

References

[1] F. Al-Hawari, M. Al-Ashi, F. Abawi, and S. Alouneh. A practical three-phase ilp approach for solving the examination timetabling problem. *International Transactions in Operational Research*, 27(2):924–944, 2020.

[2] C. Almeder. A hybrid optimization approach for multi-level capacitated lot-sizing problems. *European Journal of Operational Research*, 200:599–606, 2010.

[3] B., Sylvain, H., Hideki, V., Michel, and W., Christophe. Un algorithme grasp pour le problème de planification de techniciens et d'interventions pour les télécommunications. *RAIRO-Oper. Res.*, 43(4):387–407, 2009.

[4] J. Bérubé, M. Gendreau, and J. Potvin. A branch-and-cut algorithm for the undirected prize collecting traveling salesman problem. *Networks: An International Journal*, 54:56–67, 2009.

[5] S. A Canuto, M. G. C. Resende, and C. C. Ribeiro. Local search with perturbations for the prize-collecting steiner tree problem in graphs. *Networks: An International Journal*, 38:50–58, 2001.

[6] G. Climaco. Prize-collecting covering tour problem (Data Set, 2020. Mendeley Data, V1, doi: 10.17632/8yjbfgcfvn.1.

[7] G. Climaco. Prize-collecting covering tour problem (Complete Results, 2021. Mendeley Data, V2, doi: 10.17632/8yjbfgcfvn.2.

[8] Glaubos Clímaco, Isabel Rosseti, Luidi Simonetti, and Marcos Guerine. Combining integer linear programming with a state-of-the-art heuristic for the 2-path network design problem. *International Transactions in Operational Research*, 26(2):615–641, 2019.

[9] J. R. Current and D. A. Schilling. The covering salesman problem. *Transportation science*, 23:208–213, 1989.

[10] A. D. Ebrahimi and R. Sahraeian. The maximal backup covering tour problem. In *6th International Conference on Industrial Engineering and Industrial Management*, pages 367–374, Vigo, 2012.

[11] D. A. Flores-Garza, M. A. Salazar-Aguilar, S. U. Ngueveu, and G. Laporte. The multi-vehicle cumulative covering tour problem. *Annals of Operations Research*, 258:761–780, 2017.

[12] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40:1086–1094, 1992.

[13] M. Gendreau, G. Laporte, and F. Semet. The covering tour problem. *Operations Research*, 45:568–576, 1997.

[14] M. Hachicha, M. J. Hodgson, G. Laporte, and F. Semet. Heuristics for the multi-vehicle covering tour problem. *Computers & Operations Research*, 27:29–42, 2000.

[15] M. Hamidi, K. Farahmand, S. Reza S., and K. E. Nygard. A hybrid grasp-tabu search metaheuristic for a four-layer location-routing problem. *International Journal of Logistics Systems and Management*, 12:267–287, 2012.

[16] N. Jozefowiez, F. Semet, and E. Talbi. The bi-objective covering tour problem. *Computers & operations research*, 34:1929–1942, 2007.

[17] İ. Karaoğlan, G. Erdoğan, and Ç Koç. The multi-vehicle probabilistic covering tour problem. *European Journal of Operational Research*, 271:278–287, 2018.

[18] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In Jean-Yves Potvin Michel Gendreau, editor, *Handbook of metaheuristics*, pages 320–353. Springer, Boston, MA., 2003.

[19] A. R. de Lyra. O problema de recobrimento de rotas com coleta de prêmios: regras de redução, formulação matemática e heurísticas (in portuguese). Master's thesis, Programa de Pós-Graduação em Computação, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, 2004.
[20] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997.
[21] Gurobi Optimization. Gurobi optimizer reference manual, 2019. Last accessed on December 10, 2019.
[22] C. Park and J. Seo. A grasp approach to transporter scheduling for ship assembly block operations management. *European Journal of Industrial Engineering*, 7:312–332, 2013.
[23] T. Anh Pham, M. Hà, and X. H. Nguyen. Solving the multi-vehicle multi-covering tour problem. *Computers & Operations Research*, 88:258–278, 2017.
[24] M. Prais and C. C. Ribeiro. Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. *INFORMS Journal on Computing*, 12(3):164–176, 2000.
[25] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017.
[26] G. Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3:376–384, 1991.
[27] M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2003.
[28] Ítalo Santana, Alexandre Plastino, and Isabel Rosseti. Improving a state-of-the-art heuristic for the minimum latency problem with data mining. *International Transactions in Operational Research*, 2020.
[29] Sidney Siegel. *Nonparametric statistics for the behavioral sciences*. McGraw-hill, 1956.
[30] M. S. A. Silva. Problema de Recobrimento de Rotas com Coleta de Prêmios (in portuguese). Master's thesis, Programa de Pós-Graduação em Computação, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, 2009.
[31] D. Trachanatzi, M. Rigakis, M. Marinaki, and Y. Marinakis. A firefly algorithm for the environmental prize-collecting vehicle routing problem. *Swarm and Evolutionary Computation*, page 100712, 2020.
[32] F. Tricoire, A. Graf, and W. J. Gutjahr. The bi-objective stochastic covering tour problem. *Computers & operations research*, 39:1582–1592, 2012.

Table 1. Results for the mathematical formulations.

	Flow			Multi-flow		
Instance	Sol	Best Bound	T (s)	Sol	Best Bound	T (s)
bier127_R43_T42_W42_75	251062	83756	3600.0	-	96950	3600.0
brg180_R36_T36_W108_25	142850	180	3600.0	-	-	3600.0
pr76_R14_T46_W16_75	219818	66871	3600.0	**90862**	90862	**523.3**
lin105_R21_T63_W21_50	58498	6833	3600.0	-	11573	3600.0
pr107_R35_T36_W36_75	57482	20295	3600.0	-	37040	3600.0
pr144_R28_T29_W87_50	82642	11123	3600.0	**30294**	30294	**408.0**
bier127_R24_T26_W77_50	125855	62064	3600.0	**74745**	74745	**495.2**
kroA100_R34_T33_W33_75	47550	10137	3600.0	-	14549	3600.0
kroB200_R40_T40_W120_50	45214	7306	3600.0	-	-	3600.0
gr96_R18_T58_W20_25	-	33028	3600.0	**42746**	42746	**1670.9**
pr76_R24_T26_W26_50	116859	52704	3600.0	**74539**	74539	**1012.3**
gr96_R32_T32_W32_50	74785	28599	3600.0	**35461**	35461	**993.1**
kroC100_R20_T60_W20_25	45872	12478	3600.0	**16785**	16785	**3369.8**
pr136_R26_T28_W82_25	76646	34449	3600.0	**47620**	47620	**243.8**
kroE100_R34_T33_W33_75	27385	10255	3600.0	-	14202	3600.0
kroC100_R34_T33_W33_50	40281	8004	3600.0	**13235**	13235	**1796.3**
pr124_R24_T25_W75_75	62995	15408	3600.0	**36719**	33585	3600.0
kroB100_R34_T33_W33_75	25197	11115	3600.0	-	14301	3600.0
kroB100_R34_T33_W33_50	24846	9217	3600.0	-	12506	3600.0
pr152_R29_T31_W92_75	70971	19577	3600.0	**49167**	40798	3600.0
u159_R31_T32_W96_50	47566	24433	3600.0	**28613**	28613	**606.4**
kroB150_R30_T30_W90_25	18533	5384	3600.0	-	7196	3600.0
kroE100_R20_T60_W20_25	-	12235	3600.0	**17199**	17199	**3091.3**
d198_R39_T40_W119_25	14140	6432	3600.0	-	10911	3600.0
gr120_R40_T40_W40_25	17438	3514	3600.0	**4301**	4301	**1929.8**
pr107_R20_T22_W65_50	47333	23618	3600.0	**35869**	35869	**65.7**
lin105_R35_T35_W35_75	-	6340	3600.0	**11090**	10549	3600.0
rd100_R34_T33_W33_25	15716	4168	3600.0	**5970**	4989	3600.0
brazil58_R11_T35_W12_50	32029	16750	3600.0	**23312**	23312	**109.6**
ch150_R30_T30_W90_75	10981	3876	3600.0	**4292**	4292	**513.7**
ch130_R26_T78_W26_75	5443	3352	3600.0	-	-	3600.0
gr120_R24_T24_W72_75	9702	3563	3600.0	**4275**	4275	**139.7**
si175_R35_T35_W105_25	9941	3593	3600.0	**4845**	4845	**1960.3**
gr96_R18_T20_W58_75	34379	24392	3600.0	**30015**	30015	**467.1**
ch130_R26_T26_W78_50	-	3408	3600.0	**4139**	4139	**296.0**
kroC100_R20_T20_W60_75	15900	7078	3600.0	**12582**	12582	**78.4**
rat195_R39_T39_W117_50	2345	1034	3600.0	-	1147	3600.0
lin105_R21_T21_W63_25	11121	4853	3600.0	**8893**	8893	**90.9**
brazil58_R18_T20_W20_25	21198	15786	3600.0	**19417**	19417	**798.9**
kroE100_R20_T20_W60_50	12260	6632	3600.0	**10586**	10586	**551.7**
eil101_R19_T61_W21_50	1547	451	3600.0	-	503	3600.0
gr137_R26_T28_W83_25	**59114**	25048	3600.0	60630	39167	3600.0
rd100_R20_T20_W60_75	6705	4834	3600.0	**5421**	5421	**143.4**
kroA100_R20_T20_W60_25	11980	5566	3600.0	**10848**	10848	**104.6**
eil101_R33_T34_W34_75	1367	407	3600.0	**462**	462	3600.0
hk48_R9_T29_W10_25	10557	8557	3600.0	**9708**	9708	**50.5**
gr48_R9_T29_W10_75	5135	4097	3600.0	**4415**	4415	**0.4**
rat99_R19_T20_W60_25	1362	737	3600.0	**792**	792	**12.8**
pr76_R14_T16_W46_25	66533	50625	3600.0	**66007**	66007	**10.4**
kroA150_R30_T30_W90_50	**19405**	6544	3600.0	19898	9778	3600.0

Table 2. Comparison between R-GRASP and ILS-RDM-CI heuristics

		ILS-RDM-CI			R-GRASP			
Instance	Multi-flow	Best Sol.	Avg. Sol.	Avg. T (s)	Best Sol.	Avg. Sol.	Avg. T (s)	$Diff_T$
pr226_R76_T75_W75_25	-	40994	41014.00	1627.04	40994	**40994.00**	**4.37**	-99.70
rd400_R136_T132_W132_75	-	35465	36107.90	1497.27	35465	**35465.00**	**6.30**	-99.50
pr124_R42_T41_W41_25	-	61068	61077.70	2463.89	61068	**61068.00**	**15.24**	-99.30
lin318_R108_T105_W105_50	3571	3571	3571.00	8.89	3571	3571.00	**0.05**	-99.30
kroA150_R50_T50_W50_25	2650	2650	2650.00	27.96	2650	2650.00	**0.19**	-99.20
gr229_R77_T76_W76_75	19417	19417	19417.00	55.64	19417	19417.00	**0.44**	-99.10
pr299_R101_T99_W99_75	35869	35869	35869.00	22.79	35869	35869.00	**0.17**	-99.10
d198_R66_T66_W66_75	36719	34359	34359.00	201.94	34359	34359.00	**1.52**	-99.10
kroB150_R50_T50_W50_50	5970	5552	5552.00	169.72	5552	5552.00	**1.32**	-99.10
ts225_R75_T75_W75_75	792	792	792.00	74.65	792	792.00	**0.65**	-99.00
pr299_R59_T180_W60_25	13235	13235	13238.40	821.52	13235	**13235.00**	**7.55**	-98.90
gil262_R88_T87_W87_50	863	847	847.00	217.58	847	847.00	**2.09**	-98.90
a280_R94_T93_W93_25	248	248	248.00	3.68	248	248.00	**0.04**	-98.80
lin318_R63_T191_W64_75	338	338	338.00	122.08	338	338.00	**1.26**	-98.80
si175_R59_T58_W58_75	5421	5421	5421.00	35.57	5421	5421.00	**0.37**	-98.80
tsp225_R75_T75_W75_50	273	273	273.00	67.94	273	273.00	**0.75**	-98.70
pr299_R59_T60_W180_50	4275	4275	4275.00	57.88	4275	4275.00	**0.63**	-98.70
pr144_R48_T48_W48_75	90862	90862	90862.00	90.38	90862	90862.00	**1.09**	-98.60
rat195_R39_T39_W117_50	-	1199	1202.80	3308.10	1199	**1199.00**	**42.48**	-98.50
gil262_R51_T158_W53_75	60630	39305	39305.00	101.86	39305	39305.00	**1.34**	-98.50
gr137_R45_T46_W46_50	-	10916	10918.30	1187.35	10916	**10916.00**	**15.92**	-98.40
si175_R35_T35_W105_25	6916	6916	6916.00	115.10	6916	6916.00	**1.69**	-98.30
pr264_R88_T88_W88_50	4301	4301	4301.00	82.83	4301	4301.00	**1.23**	-98.30
kroB200_R40_T40_W120_50	356	356	356.00	241.04	356	356.00	**3.90**	-98.10
rat195_R39_T117_W39_75	-	4383	4385.50	1680.15	4383	**4383.00**	**30.41**	-97.90
kroB150_R30_T90_W30_75	12582	12582	12582.00	14.77	12582	12582.00	**0.27**	-97.90
gr229_R45_T138_W46_50	-	35442	35442.00	1721.60	35442	35442.00	**31.58**	-97.90
ts225_R45_T135_W45_50	-	550	551.00	552.07	550	**550.00**	**10.66**	-97.80
kroA200_R68_T66_W66_50	419	419	419.00	10.82	419	419.00	**0.20**	-97.80
bier127_R43_T42_W42_75	554	554	554.00	92.55	554	554.00	**1.73**	-97.80
eil101_R33_T34_W34_75	4845	4845	4863.60	4252.27	4845	**4847.40**	**84.02**	-97.70
pr264_R52_T159_W53_75	3524	3524	3524.00	3.31	3524	3524.00	**0.07**	-97.60
brg180_R36_T36_W108_25	-	15197	15201.00	2460.12	15197	**15197.00**	**52.49**	-97.50
gr229_R45_T46_W138_25	-	14606	14613.00	892.98	14606	**14608.50**	**19.44**	-97.50
kroB200_R68_T66_W66_75	10848	10848	10848.00	52.14	10848	10848.00	**1.12**	-97.50
rat195_R65_T65_W65_25	540	505	505.00	53.65	505	505.00	**1.14**	-97.50
rd400_R80_T80_W240_50	74745	74745	74745.00	42.41	74745	74745.00	**0.98**	-97.30
ch130_R26_T78_W26_75	7593	7593	7593.00	14.01	7593	7593.00	**0.37**	-97.00
ch150_R30_T90_W30_50	66007	66007	66007.00	5.87	66007	66007.00	**0.15**	-97.00
kroA150_R30_T90_W30_75	35461	35461	35461.00	254.77	35461	35461.00	**6.85**	-96.90
gr202_R39_T41_W122_75	-	46122	46262.30	3410.33	46122	**46122.00**	**94.63**	-96.80
d198_R39_T119_W40_50	387	387	387.00	56.11	387	387.00	**1.57**	-96.80
tsp225_R75_T75_W75_25	10586	10586	10586.00	16.16	10586	10586.00	**0.44**	-96.80
kroA200_R40_T120_W40_25	-	25905	25934.40	1805.06	25905	**25905.00**	**52.32**	-96.60
tsp225_R45_T45_W135_75	-	5747	5754.60	31233.58	5.747	**5747.00**	**950.23**	-96.50
pr107_R35_T36_W36_75	2824	2824	2824.00	16.47	2824	2824.00	**0.52**	-96.40
a280_R56_T168_W56_50	4292	4292	4292.00	107.75	4292	4292.00	**3.34**	-96.40
kroB100_R34_T33_W33_50	5186	5186	**5186.00**	45.80	5186	5191.80	**1.44**	-96.30
ch130_R44_T43_W43_25	-	506	506.20	1058.25	506	**506.00**	**36.40**	-96.00
a280_R56_T56_W168_75	-	46940	46954.40	2720.76	46940	**46940.00**	**92.66**	-96.00

Table 3. Summary of the results for the 107 instances: number of wins, ties, and losses of the R-GRASP.

	Best Sol.	Avg. Sol.	Avg. T. (s)
Wins	11	48 (37)	96
Ties	86	40	0
Losses	10	19 (7)	11