GRASP with evolutionary path-relinking for the capacitated arc routing problem

Fábio Luiz Usberti a,*, Paulo Morelato França b, André Luiz Morelato França a

a School of Electrical and Computer Engineering, Campinas State University, Av. Albert Einstein 400, 13083-852 Campinas, SP, Brazil
b Department of Mathematics, Statistics and Computing, São Paulo State University, Av. Roberto Simonsen 305, 19060-900 Presidente Prudente, SP, Brazil

1. Introduction

The Undirected Capacitated Arc Routing Problem (CARP or UCARP), proposed by Golden and Wong [1], is a combinatorial optimization problem defined in a connected undirected graph \( G(V,E) \) where non-negative costs \( c_e \) and demands \( d_e \) are assigned to each edge \( e = \{v_i,v_j\} \). All edges with positive demand (required edges, \( E_R \leq E \)) must be serviced by a fleet of identical vehicles with limited capacity \( D \). While traversing the graph, a vehicle might (i) service an edge, which deducts the demand from the vehicle capacity and increases the solution cost, or (ii) deadhead an edge, which only increases the solution cost. A tour is defined feasible when it starts and ends at a distinguished node \( v_0 \), called depot, and the sum of the demands serviced by that vehicle is less than or equal to \( D \). A feasible solution is formed by a family of feasible tours, which services all required edges. The number of vehicles \( M \) is a decision variable with no related costs. The CARP objective is to search for a minimum cost feasible solution.

Many real world applications have been related to CARP, such as street sweeping, garbage collection, mail delivery, school bus routing, meter reading, etc., and estimates on the expenditure involved in these services reaches billions of dollars, thus revealing a substantial savings potential. Details on these applications are provided in [2–4].

Other problems related to CARP are the Directed CARP or DCARP (directed graph), the Mixed CARP or MCARP (mixed graph), the Capacitated Chinese Postman Problem or CCPP (\( E_R = E \)), the CARP with Time Windows or CARP-TW (required edges must be serviced within a given time interval), and, more recently, the Open CARP or OCARP, explained in the following paragraph.

The OCARP is a new arc routing problem, introduced by Usberti et al. [5], where tours are not constrained to form cycles, and therefore both open and closed tours are possible. Two applications from the literature that can be modeled as an OCARP are the Meter Reader Routing Problem [6] and the Cutting Path Determination Problem [7]. A reactive path-scanning heuristic, which adapts its own parameters for every instance, was developed to solve the OCARP.

The CARP belongs to the class of NP-hard problems, and it has been shown that even the \( \frac{1}{2} \)-approximation for the CARP is already NP-hard [1]. Attempts were made toward solving CARP to optimality, including a branch-and-bound algorithm [8], and a CARP reduction into the capacitated vehicle routing problem (CVRP), which is then solved by a branch-and-cut-and-price algorithm [9]. These exact approaches, however, can only solve
relatively small size instances. In terms of approximate algo-
rithms, the current best approximation factor is $\frac{7}{4} -\frac{1}{4} [10]$, where $D$ is the vehicle capacity. Furthermore, there are algorithms specialized in determining lower bounds for the CCP [11], DCARP [12], CARP [13–16], and MCARP [17,18].

Due to the CARP complexity, many real world instances are intractable for exact algorithm, hence opening research for heuristics which, despite being unable to guarantee optimality, perform well in most cases, providing high-quality solutions on average. Examples of heuristics for the CARP are path-scanning [19,20], augment-merge [1], and augment-insert [21]. Better CARP solutions were obtained through metaheuristics such as tabu search [22–24], genetic algorithm [25], hybrid tabu-scatter search [26], guided local search [27], variable neighborhood search [28–30] (this last one [30] also solves the DCARP), and ant colony optimization [31]. Two GRASPs with path-relinking [32,33] were developed for the CARP (and CARP-TW in the case of [33]). The solution quality of these two GRASPs, however, was outperformed by the three most recent metaheuristics [24,29,31].

For an overview on the CARP complexity, polyhedral results, exact, approximate and heuristic algorithms, we refer to [2,4,34–36].

The remainder of this paper is organized as follows. Section 2 reviews the general structure of a Greedy Randomized Adaptive Search Procedure (GRASP) and gives a thorough description of the proposed GRASP to solve the CARP, including the constructive phase, parameters reactive adjustments, local search, and the statistical solution filtering. To strengthen the search for high-quality solutions, a path-relinking was coupled to the GRASP, mirroring several successful experiences in the literature, which are referred to in Section 3. Still on this section, the detailed modus operandi of the proposed path-relinking is provided, with special attention to the metric used to measure the distance between a pair of CARP solutions, the operator used to progressively transform an initial solution toward a guiding solution, the way the GRASP and path-relinking were jointed. Computational experiments were conducted and the results presented in Section 4. Conclusions close this paper in Section 5.

2. Greedy randomized adaptive search procedure

A Greedy Randomized Adaptive Search Procedure (GRASP) [37] is a memoryless multi-start metaheuristic, where each iteration consists of two phases:

- **construction phase**: initial solutions are built, one element at a time, with a greedy randomized heuristic. At each construction iteration, the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function that estimates the benefit of selecting each element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, not always the top best.

- **local search**: the neighborhood of the initial solutions is explored. The solutions generated by a GRASP construction are not guaranteed to be locally optimal. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better one from its neighborhood. It terminates when there are no better solutions in the neighborhood.

The best solution over all GRASP iterations is returned as the result. Success for a local search algorithm depends on an efficient neighborhood search technique and a good starting solution provided by the construction phase. A GRASP can be seen as a metaheuristic which captures good features of pure greedy algorithms (intensification) and also of random construction procedures (diversification).

Competitive results have been reported in the literature using GRASP-based metaheuristics in different routing problems such as the vehicle routing problem [38], the truck and trailer routing problem [39], and the CARP-TW [33]. According to Resende and Ribeiro [40], the performance of GRASP can be enhanced by using reactive parameter tuning mechanisms, multiple neighborhoods, and path-relinking. These features were incorporated in this work proposed GRASP, whose components and the general structure follows.

2.1. Constructive phase

The GRASP constructive phase was developed based on Santos et al. [20] path-scanning heuristic with ellipse rule. This heuristic was adapted to include a restricted candidate list, responsible for holding a set of good and diversified elements to embody the solution under construction. The parameters which control the balance of good and diversified have their values reactively adjusted according to the average solution cost these values provide.

2.1.1. Path-scanning heuristics

The path-scanning heuristics developed for CARP construct each solution by adding to a path starting at the depot, one required edge at a time. To determine the next edge to add, an edge-selection rule $\psi(v_i,e)$ is used (1), where $e = [v_i,v_j]$ is a candidate for the next required edge to be visited starting from $v_i$ to $v_j$. $v_i$ is the last node visited by the tour, and $SP$ represents the shortest path cost between two nodes. Every unserviced required edge whose demand $d_{\hat{e}}$ is less than the vehicle remaining capacity is a possible candidate, and the heuristic will select the one which minimizes $\psi(v_i,e)$:

\[
\psi(v_i,e) = SP(v_i,v_j)
\]

There are cases where more than one candidate edge minimizes $\psi(v_i,e)$, specially when they are incident to $v_i$. In these situations, a tie breaking rule is considered, and this rule represents the major difference between CARP path-scanning heuristics. Golden et al. [1] have used five criteria to break ties:

1. minimize $c_{ij}/d_{\hat{e}}$
2. maximize $c_{ij}/d_{\hat{e}}$
3. minimize the cost back to depot
4. maximize the cost back to depot
5. criterion 3 if the vehicle has used more than half of its capacity; criterion 4, otherwise.

A problem instance is solved five times, using a different criterion each time, and the best of the five solutions is taken. Pearn [41] modified this approach by selecting one of the five criteria at random, with equal probability, whenever a tie occurs. Belenguer et al. [17] simplified the tie breaking rule by randomly selecting one tied edge. This was copied by Santos et al. [20], in their path-scanning heuristic with ellipse rule, explained in the following paragraphs.

Recently, Santos et al. [20] developed a path-scanning heuristic which makes use of an ellipse rule. When a vehicle is near its full capacity, this rule enforces the vehicle to service only edges near the shortest path between the last serviced edge and the depot, following the rationale that a heavily loaded vehicle should stay closer to the depot in order to reduce its returning cost. These authors define $ned = |E_k|$, $td$ is the total demand to be serviced, $tc$ the total cost from edges with positive demand, $v_0$ the depot node, $[v_h,v_i]$ the last serviced edge on the tour, and $\beta$ a real parameter. If the remaining vehicle capacity is less than or equal to $|h|/|ned|$, then the next edge to be serviced $[v_i,v_j]$ must be the nearest edge to $[v_h,v_i]$ if the
If no candidate edge satisfies (2) then the vehicle returns to the depot. Through the ellipse rule, the authors obtained 44% reduction in overall average deviation from lower bounds with little or no increase in solution time, compared to previous path-scanning heuristics.

2.1.2. Constructive heuristic

The path-scanning heuristic with ellipse rule was adapted into a GRASP constructive heuristic by replacing the edge-selection rule with the restricted candidate list (RCL) (3), which is filled with the best candidate edges according to the edge-selection rule (1), limited by a threshold parameter \( \alpha \).

\[ e \in \text{RCL} \Rightarrow \psi(v_1, e) \leq 2(\psi_{\max} - \psi_{\min}) + \psi_{\min} \]

where \( \psi_{\min} = \min_{e \in E_k} \psi(v_1, e) \) and \( \psi_{\max} = \max_{e \in E_k} \psi(v_1, e) \).

The constructive heuristic pseudo-code is presented in Algorithm 1.

**Algorithm 1.** constructivePhase \((G, D, \alpha, \beta)\).

**Input:** \( C \)- instance graph, \( D \)- vehicle capacity, \( \alpha \)-RCL parameter, \( \beta \)-ellipse rule parameter

**Output:** \( S \)- feasible CARP solution

1: \( t_d = \sum_{e \in E_k} d(e), \text{ned} = |E_k| \)
2: \( S = \emptyset \)
3: \( t \leftarrow t - 1 \) // tour index
4: \( \text{tour}_i \leftarrow \emptyset \) // ordered set of edges representing a tour
5: \( r\text{vc} \leftarrow \emptyset \) // remaining vehicle capacity
6: \( v_i = 0 \) // starting tour on depot
7: \( (i = 1 \text{ to ned) do} \)
8: \( \text{RCL} \leftarrow \emptyset \)
9: \( \psi_{\min} = \min_{e \in E_k} \psi(v_1, e), \psi_{\max} = \max_{e \in E_k} \psi(v_1, e) \)
10: \( \text{for (} \forall e \in E_k \text{ do} \)
11: \( \text{if } \psi(v_1, e) \leq 2(\psi_{\max} - \psi_{\min}) + \psi_{\min} \text{ then} \)
12: \( \text{RCL} \subseteq \text{RCL} \cup \{e\} \)
13: \( \text{end if} \)
14: \( \text{end for} \)
15: \( \text{if } \text{rvc} \leq \beta \frac{m}{|E_k|} \text{ then} \)
16: \( \text{RCL} \leftarrow \text{RCL} \setminus \{ \text{edgesvioletellipserule(2)} \} \)
17: \( \text{end if} \)
18: \( \text{if } \text{RCL} = \emptyset \text{ then} \)
19: \( // \text{ starting new tour} \)
20: \( S \leftarrow S \cup \{ \text{tour}_i \} \)
21: \( t = t + 1 \)
22: \( \text{tour}_i = \emptyset \)
23: \( \text{rvc} = \text{D} \)
24: \( v_i = 0 \)
25: \( \text{else} \)
26: \( e = [v_i, v_j] \leftarrow \text{randomEdge(RCL)} \) // randomly selects an edge from set RCL
27: \( \text{tour}_i = \text{tour}_i \cup \{e\} \)
28: \( \text{rvc} = \text{rvc} - d(e) \)
29: \( v_i = v_j \)
30: \( \text{end if} \)
31: \( \text{end for} \)
32: \( \text{return } S \)

2.1.3. Reactive parameters

The proposed constructive heuristic has two parameters, \( \alpha \) and \( \beta \), that directly affect the heuristic performance and ergo must be properly adjusted. The RCL parameter \( \alpha \) controls the greediness of the candidate edge selection (\( \alpha = 0 \) pure greedy; \( \alpha = 1 \) pure random). The ellipse rule parameter \( \beta \) is responsible for controlling the ellipse shape, or in other words, how active is this rule (\( \beta = 0 \), inactive; \( 0 < \beta = \text{Dned}/(t_d) \), always active).

A reactive parameter adjustment, based on the work of Prais and Ribeiro [42], was implemented to select the values for \( \alpha \) and \( \beta \) at each iteration of the constructive heuristic from a discrete set of possible values. This strategy was successfully used by Usberti et al. [5] to adjust the ellipse rule parameter \( \beta \) for the OCARP. Let \( \Pi = \{\pi_1, \ldots, \pi_n\} \) be the set of possible values for a given parameter \( \pi \). The probabilities associated with the choice of each value are all initially made equal to \( p_i = 1/m \), \( i = 1, \ldots, m \). Furthermore, let \( c_{\text{best}} \) be the cost of the incumbent best solution and \( \tau_i \) the average cost of all solutions obtained by using \( \pi = \pi_i \). In Prais and Ribeiro [42], the selection probabilities are periodically reevaluated through (4).

\[ p_i = \frac{q_i}{\sum_{j=1}^{n} q_j}, \quad q_i = c_{\text{best}} / \tau_i \quad (i = 1, \ldots, m) \]

It is intended that the values of \( \pi_i \) producing good solutions on average will generate larger \( q_i \), which in turn increases the probabilities \( p_i \) associated to them. However, it turned out that through Eq. (4), the probabilities are not expressing well the relative differences between their associated average costs. For some CARP instances, these probabilities would hardly differ in more than 1%. An alternative reactive scheme is proposed (5), which preserves the main idea of the previous one, but amplifies the effect of the average costs in their associated probabilities

\[ q_i = 1 - \left( \frac{m - 1}{m} \right) \left( \frac{\tau_{\text{cmax}} - \tau_{\text{cmin}}}{\tau_{\text{cmax}} - \tau_{\text{cmin}}} \right) \quad (i = 1, \ldots, m) \]

where \( \tau_{\text{cmin}} \) and \( \tau_{\text{cmax}} \) are the minimum and maximum average costs, and \( p_i \) is calculated the same way as before (4).

Let \( p_{\text{max}} \) and \( p_{\text{min}} \) be the probabilities associated to the best \( \tau = \tau_{\text{cmax}} \) and worse \( \tau = \tau_{\text{cmin}} \) parameters, respectively. Then, through Eq. (5), \( p_{\text{max}} = m p_{\text{min}} \), giving a much better probability distribution, in the sense that the best parameter will have \( m \) times better chance to be chosen than the worse parameter.

**Algorithm 2** describe the pseudo-code for the reactive parameter adjustment.

**Algorithm 2.** reactiveChoice \((CN)\).

**Input:** \( C = \{\tau_1, \ldots, \tau_m\}\)-average solution costs for each parameter value

\( N = \{n_1, \ldots, n_m\}\)-number of solutions obtained for each parameter value

**Output:** \( i \in \{1, \ldots, m\}\)-index of the parameter value

1: \( q_{\text{sum}} \leftarrow 0, \quad \tau_{\text{cmin}} \leftarrow \min \tau_i, \quad \tau_{\text{cmax}} \leftarrow \max \tau_i \)
2: \( \text{for } (i = 1 \text{ to } m) \text{ do} \)
3: \( q_i \leftarrow 1 \)
4: \( \text{if } n_i > 0 \text{ and } \tau_{\text{cmin}} \neq \tau_{\text{cmax}} \) then \( q_i \leftarrow q_i - \left( \frac{m - 1}{m} \right) \frac{\tau_{\text{cmax}} - \tau_{\text{cmin}}}{\tau_{\text{cmax}} - \tau_{\text{cmin}}} \)
5: \( q_{\text{sum}} \leftarrow q_{\text{sum}} + q_i \)
6: \( \text{end if} \)
7: \( q_{\text{sum}} \leftarrow q_{\text{sum}} + q_i \)
8: \( \text{end for} \)
9: \( n_{\text{rand}} \leftarrow \text{randNumber}(0, 1) \) // real random number between [0, 1]
10: \( p_{\text{sum}} \leftarrow 0 \)
11: \( \text{for } (i = 1 \text{ to } m) \text{ do} \)
12: \( p_{\text{sum}} \leftarrow p_{\text{sum}} + \frac{q_i}{q_{\text{sum}}} \)
13: \( \text{if } n_{\text{rand}} \leq p_{\text{sum}} \) then \( \text{break for} \)
14: \( \text{end if} \)
15: \( \text{end for} \)
16: \( \text{return } i \)
2.2. Local search phase

After an initial solution is generated by the constructive phase, the local search tries to improve it by exploring neighbor solutions defined by a set of moves which operate on the required edges order and orientation. The solutions are encoded as a list of required edges with implicit shortest paths between them, following the ideas in [25,27].

2.2.1. Neighborhood moves

Four types of moves were considered, all of them applicable for inter-routes and intra-routes.

- **single-insertion + reversal**: a required edge is removed from its current position and placed in another one, reversed or not.
- **double-insertion + reversal**: two adjacent required edges are removed from their current positions and placed in another ones, both reversed or not.
- **swap + reversal**: two required edges switch their current positions, reversing or not one or both required edges.
- **block-insertion**: a block of adjacent required edges is removed from its current position and placed in another one.

The local search phase uses the first three moves, while block-insertion is used as the path-relinking operator. To achieve a local optimal solution, the best improvement scheme was adopted, where the selected move in each local search iteration is the one which achieves the greatest reduction in solution cost, preserving feasibility, i.e., the vehicle capacity constraints (Algorithm 3).

**Algorithm 3. localSearch (S).**

**Input:** S - CARP solution

**Output:** $S_{ls}$ - locally optimal CARP solution

1: $c_{lsp} \leftarrow \infty$, $S_{ls} \leftarrow S$
2: while $(\text{cost}(S_{ls}) < c_{lsp})$ do
3: $c_{lsp} \leftarrow \text{cost}(S_{ls})$
4: $S_{ls} \leftarrow \text{applyBestSingleInsert}(S_{ls})$
5: $S_{ls} \leftarrow \text{applyBestDoubleInsert}(S_{ls})$
6: $S_{ls} \leftarrow \text{applyBestSwap}(S_{ls})$
7: if cost($S_{ls}$) < cost($S_{ls}$) then
8: $S_{ls} \leftarrow S_{ls}$
9: end if
10: if cost($S_{ls}$) < cost($S_{ls}$) then
11: $S_{ls} \leftarrow S_{ls}$
12: end if
13: if cost($S_{ls}$) < cost($S_{ls}$) then
14: $S_{ls} \leftarrow S_{ls}$
15: end if
16: end while
17: return $S_{ls}$

2.2.2. Infeasible local search

A diversification strategy was incorporated into local search by allowing capacity infeasible moves. Since an integer linear programming problem optimal solution must reside on the boundary of the feasible convex hull, then this optimal solution is adjacent to the infeasible space, making search techniques which explore the infeasible solution space an interesting field of investigation. Glover [43] draws some light on the importance of exploring feasible/infeasible boundaries in the solution space of combinatorial optimization problems. This work proposes an infeasible local search, which receives as input a feasible solution, and probably returns a better cost solution, infeasible with respect to the vehicles capacities (Algorithm 4). It works mostly like the normal local search, except for two differences: (i) on each iteration, the search ignores if a move violates a vehicle capacity; (ii) the search is interrupted after a given number of infeasible moves, preventing the solution going too deep in the infeasible space, and possibly harming its way back. The infeasible local search provides these infeasible solutions as initial solutions to the path-relinking, with means to explore paths traversing the infeasible/feasible boundaries of the solution space.

**Algorithm 4. infeasibleLocalSearch ($S, n_{in}$).**

**Input:** $S$ - CARP solution, $n_{in}$—number of moves to execute

**Output:** $S_{ls}$ - CARP solution, likely infeasible

1: $c_{lsp} \leftarrow \infty$, $S_{ls} \leftarrow \text{localSearch}(S)$, $S_{ls} \leftarrow S_{ls}$
2: for $i = 1$ to $n_{in}$ do
3: $c_{lsp} \leftarrow \text{cost}(S_{ls})$
4: $S_{ls} \leftarrow \text{applyBestInfeasibleSingleInsert}(S_{ls})$
5: $S_{ls} \leftarrow \text{applyBestInfeasibleDoubleInsert}(S_{ls})$
6: $S_{ls} \leftarrow \text{applyBestInfeasibleSwap}(S_{ls})$
7: if cost($S_{ls}$) < cost($S_{ls}$) then
8: $S_{ls} \leftarrow S_{ls}$
9: end if
10: if cost($S_{ls}$) < cost($S_{ls}$) then
11: $S_{ls} \leftarrow S_{ls}$
12: end if
13: if cost($S_{ls}$) < cost($S_{ls}$) then
14: $S_{ls} \leftarrow S_{ls}$
15: end if
16: if (cost($S_{ls}$) $\geq c_{lsp}$) then
17: break
18: end if
19: end for
20: return $S_{ls}$

2.3. Statistical filter

In general, good solutions uncovered by local search comes from good initial solutions found in the constructive phase. Besides, local search is often the most demanding phase of a GRASP in terms of computational effort. Therefore, it seems unwise and computationally expensive to explore the neighborhood of all initial solutions, including low-quality ones. Instead, poor quality initial solutions should be rather discarded, and with the computational time saved, other more promising solution space regions should be explored. This strategy is called GRASP filtering [44]. Prais and Ribeiro [42] propose a filtering by storing the average value ($\mu$) of the ratio between initial ($c_{ini}$) and local search ($c_{ls}$) solutions costs. After the first 100 iterations, they use this information to decide whether each constructed solution will be submitted to local search or not. Their idea is based on the rationale that if some reasonable threshold applied to the cost of the constructed solution leads to a value much higher than the cost of the best solution already found, it is unlikely that local search could produce a better solution than the current best. Their threshold is determined by (6), where an initial solution passes through the filter only if 90% of the ratio $c_{ini}/c_{best}$ is less than or equal to the average ratio ($\mu$)

$$0.9c_{ini} \leq \mu c_{best}$$

This work addresses GRASP filtering with a different approach, where a statistically meaningful filter is proposed. This filter is able to classify bad solutions within a certain confidence interval. For this, an additional variable is needed to determine the threshold, the standard deviation ($\sigma$) of the ratio between initial and local search solutions costs. A solution is considered good, and passes through the filter, when it satisfies the following
condition:
\[ c_{i_{\text{ini}}} \leq (\mu + 2\sigma)c_{\text{best}} \]  
(7)

The filter accepts an initial solution to undergo local search when the ratio \(c_{i_{\text{ini}}}/c_{\text{best}}\) is less than the average ratio plus two times its standard deviation, which gives a confidence interval of slightly more than 95% probability that a rejected solution could not be improved by local search further than \(c_{\text{best}}\), assuming of course that \(c_{i_{\text{ini}}}/c_{\text{best}}\) is an independent random variable with normal distribution.

### 3. Path-relinking

Path-relinking (PR) was introduced by Glover [45], in the context of tabu and scatter searches, as a mechanism to combine intensification and diversification by exploring trajectories connecting high-quality (elite) solutions previously produced during the search. These elite solutions often share a significant portion of their attributes, for example the nodes and edges of a graph. Paths between a pair of solutions \((S_1, S_2)\) in the search space traverse other solutions that share these attributes contained in \(S_1\) and \(S_2\). Such paths may be generated by applying neighborhood moves to the initial solution \(S_1\), which progressively introduces attributes from the guiding solution \(S_2\). This generates a sequence of intermediate solutions, often not locally optimals, however improbable by local search and possibly better than \(S_1\) and \(S_2\).

Labadi et al. [33] observed that, despite GRASP simplicity and speed, it is often less effective than its counterparts metaheuristics, like tabu search, and they explain this may be due to the independent (memoryless) GRASP iterations, using no information to sample good regions of the solution space. This may be remedied hybridizing PR with GRASP, as Resende and Ribeiro [40] suggest, in order to improve the performance of the latter by tackling the memoryless criticism faced by the basic GRASP scheme.

The use of path-relinking within a GRASP procedure can be done as an intensification strategy to each local optimum obtained after the local search phase, and/or as a post-optimization strategy to all pairs of elite solutions. Labadi et al. [33] use both strategies separately to solve the CARP-TW, and conclude that the intensification strategy provides a better average deviation from lower improvements in quality are observed on the elite set [39,46].

To explore these intermediate solutions, the local search phase is applied repeatedly once the current solution is four units closer to the guiding solution. The four units of distance is not arbitrary, but recommended by Ribeiro and Resende [48] as the minimum number of differing components between pairs of solutions to find a better local minimum.

### Algorithm 5: solutionRelinking \((S_i, S_j, c_{\text{filter}})\).

**Input:** \(S_i, S_j\) – pair of initial-guiding solutions, \(c_{\text{filter}}\) – local search filter threshold

**Output:** \(S_{\text{best}}\) is the lowest cost solution obtained on the path between \(S_i\) and \(S_j\)

1. \(S_{\text{best}} \leftarrow S_i\)
2. \(\delta_j \leftarrow \text{distance}(S_i, S_j) // \text{distance between solutions}\)
3. \(\delta_{\text{next}} \leftarrow \delta_j - 4\)
4. **while** \((\delta_j \geq 1)\) **do**
5. \(\text{tour} \leftarrow \text{most loaded tour in } S_i \text{ containing an incorrectly positioned required edge}\)
6. \([e_{\text{pred}}, e_{\text{end}}] \leftarrow \text{less demanding incorrectly positioned block of required edges in tour}\)
7. \(e_{\text{pred}} \leftarrow \text{predecessor edge of } e_{\text{ini}} \text{ in } S_j\)
8. \(S_j \leftarrow \text{blockInsert}(S_j, e_{\text{pred}}, [e_{\text{ini}}, e_{\text{end}}]) // \text{move block to its correct relative position}\)
9. \(\delta_j \leftarrow \text{distance}(S_i, S_j)\)
10. **if** \(\delta_j \leq \delta_{\text{next}}\) **then**

---

11: if (isFeasible(S_j) and cost(S_j) < c_filter) then
12: \( S_b \leftarrow \text{localSearch}(S_j) \)
13: if (cost(S_{new}) < cost(S_{best})) then
14: \( S_{\text{best}} \leftarrow S_{\text{new}} \)
15: end if
16: end if
17: \( \delta_{\text{new}} \leftarrow \delta_{\text{old}} - 4 \)
18: end if
19: end while
21: return \( S_{\text{best}} \)

3.3. Elite solutions pool

The elite solutions pool represent a set of the best solutions found by the metaheuristic that still preserve some diversity among them. An invariant of this pool \( P = \{S_1, S_2, \ldots, S_n\} \) is that for all pairs \((i,j)\) with \(i \neq j\), then \(\delta_{ij} \geq \delta_{\text{min}}\), where \(\delta_{\text{min}}\) is a diversity parameter which sets the minimum distance between solutions belonging to the pool.

In order to enter the elite pool, a candidate solution \( S_k \) must satisfy one of the following conditions:

- pool is not full, and there are no elite solutions \( S_i \) such that \(\text{cost}(S_i) \leq \text{cost}(S_k)\) and \(\delta_{ik} < \delta_{\text{min}}\).
- pool is full, there are no elite solutions \( S_i \) such that \(\text{cost}(S_i) \leq \text{cost}(S_k)\) and \(\delta_{ik} < \delta_{\text{min}}\), and there is at least one elite solution \( S_j \) such that \(\text{cost}(S_j) > \text{cost}(S_k)\).

Once the candidate solution \( S_k \) is admitted in the pool, every elite solution \( S_i \) \((i \neq k)\) with \(\delta_{ik} < \delta_{\text{min}}\), if any, are excluded from the pool. If still the pool size remains above its capacity, then the worst elite solution is excluded from the pool.

3.4. GRASP and path-relinking coupling

The path-relinking proposed in this work was implemented as an intensification strategy for the GRASP, combined with the concepts of evolutionary path-relinking. At every GRASP iteration, the solution generated after the local search phase is tested for membership of the elite pool, and relinked with the five best elite solutions (iterative PR). The best solution obtained from each path is tested for membership of the elite pool, and relinked with the five best elite solutions. At every 100 iterations, an evolutionary PR is executed, where each solution from the pool is relinked with the five best solutions from the same pool. The rationale of this strategy is to initially fill the elite pool with high-quality and diverse solutions generated by the iterative PR. The quality of the pool is then improved with the evolutionary PR, and in order to maintain diversity, another 100 iterations of the GRASP with iterative PR are executed. This is repeated for 10,000 iterations or while the average cost of the elite solutions is improved.

The path between two solutions is always explored in both directions, i.e., each solution acts as initial and guiding. To sum up some diversity in the path-relinking, the solution space exploration is not restrained to the feasible space between two solutions, but also to promising unfeasible regions. Given a pair of solutions, one of them acts as initial PR solution, after going through the infeasible local search (Section 2.2), while the other acts as the guiding solution, unchanged. This strategy leads to an alternative path traversing the feasible-infeasible boundary between the initial and guiding solutions.

Algorithms 6 and 7 give the pseudo-code for the iterative PR and evolutionary PR. For simplicity, it is considered that the elite solutions in the pool are sorted by costs in increasing order.
The standard set of CARP instances\(^1\) was referred to, which includes 23 gdb (7–27 nodes, 11–55 edges) \[19\], 34 val (24–50 nodes, 34–97 edges) \[14\], 24 egl (77–140 nodes, 98–190 edges) \[49\], totaling 81 instances. The solutions for these instances were compared with the lower bounds identified by Longo et al. \[9\]. All tests were executed in a Intel Core 2 Quad 3.0 GHz with 4 GB of RAM, using Linux 64 bits as the operating system. Algorithms were implemented in C language, and compiled with the GNU compiler collection (GCC). Table 1 lists the GRASP parameters and their values used in the computational experiments.

\(^{1}\) http://www.uv.es/~belenque/carp.html

Table 1
<table>
<thead>
<tr>
<th>GRASP parameters.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(k_{end} = 10000)</td>
<td>Maximum number of GRASP iterations</td>
</tr>
<tr>
<td>(k_{filter} = 100)</td>
<td>Number of iterations to calibrate the filter threshold</td>
</tr>
<tr>
<td>(k_{PR} = 100)</td>
<td>Number of GRASP iterations between evolutionary path-relinking executions</td>
</tr>
<tr>
<td>(A = (0.0, 0.5, 1.0, 1.5, 2.0))</td>
<td>Possible values for the RCL parameter (a)</td>
</tr>
<tr>
<td>(B = (1.0, 1.25, 1.5, 1.75, 2.0))</td>
<td>Possible values for ellipse rule parameter (\beta)</td>
</tr>
<tr>
<td>(n_{inf} = 4)</td>
<td>Maximum number of infeasible moves to execute with the infeasible local search</td>
</tr>
<tr>
<td>poolSize = 100</td>
<td>Size of the elite solutions pool</td>
</tr>
<tr>
<td>(\delta_{min} = 0.4/E_k)</td>
<td>Minimum distance between solutions in the elite solutions pool</td>
</tr>
</tbody>
</table>

4. Computational experiments

4.1. Constructive heuristics comparison

In order to show the effectiveness of the reactive parameter tuning scheme, described in Section 2.1, the path-scanning heuristic with ellipse rule (PS\(_{ER}\)) \[20\] was implemented with a fixed \(\beta = 1.5\), and compared with the proposed GRASP constructive heuristic (GCH). Fifteen runs of the PS\(_{ER}\) and GCH with 10,000 iterations were executed for each of the 81 instances.

Table 2 shows that practically with the same computational effort, GCH was able to reduce the average deviation from lower bound from every instance set, and all but one (val set) maximum average deviation from lower bound. In addition, due to the restricted candidate list, GCH provides a much more diverse set of initial solutions, which is an important diversification ingredient for the local search and path-relinking.

4.2. Time-to-target plots

Run time distributions or time-to-target (TTT) plots display the probability that an algorithm will find a solution at least as good as a given target value within a given running time. Time-to-target plots were first used by Feo et al. \[44\], and give subsidy to characterize the running times of stochastic algorithms for combinatorial optimization problems. Such plots are very useful in the comparison of different algorithms for solving a given problem. Basically, to plot the empirical run time distribution of a given stochastic algorithm, a solution target value is fixed and each algorithm is executed \(N\) times, recording the instant \(t_i\) when a solution with cost at least as good as the target value is found. For each algorithm, the ith sorted running time \(t_i\) is associated to probability \(p_i = (i−1/2)/N\). The TTT plot represents the points \((t_i,p_i)\), for \(i = 1, \ldots, N\). In this work, a sample of \(N = 200\) runs were collected for each evaluated algorithm.

4.3. GRASP filtering effect on runtime

To establish the effect of filtering in the GRASP run time, TTTs were drawn (Fig. 2) for two basic GRASP heuristics, with (G) and
without (Gnf) filtering, applied to the hardest instance (egl-s4-c). Three targets were selected, in order that the heuristics would not take too long to hit. Still, when the heuristics reached a limit of 2000 s, they were interrupted.

Fig. 2 reveals that for all three targets Gf has improved run time, which is minor for the highest target, but increases substantially for harder targets. For instance, with 50% probability, Gf hits the 21,500 target in less than 500 s, while Gnf takes almost 900 s. Lower targets reflect better the filter effect on the TTTs, once high-quality solutions require many GRASP iterations to appear, and the filter safely eliminates unpromising initial solutions, which in turn saves plenty of the heuristic computational time.

4.4. Evolutionary PR effect on runtime

To quantify the evolutionary path-relinking contribution on the solution space search, two GRASP heuristics were compared using TTTs (Fig. 3). The first one (EvPR) is the complete proposed GRASP, as described in Algorithm 8, while the second (PR) is the same heuristic except by the evolutionary PR (step 29), which was removed, and only the iterative PR is executed. Hence, this comparison tries to verify if the additional computational effort of EvPR is only an extra weight for the metaheuristic, or it effectively helps finding better solutions in reduced execution times.

Table 3
GRASP results for gdb instances.

<table>
<thead>
<tr>
<th>Instance</th>
<th>LB</th>
<th>UB</th>
<th>GRASP</th>
<th>Best cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Median</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cost</td>
<td>CPU</td>
<td>iter</td>
<td>Cost</td>
</tr>
<tr>
<td>gdb1</td>
<td>316</td>
<td>0.01</td>
<td>2.40</td>
<td>316</td>
</tr>
<tr>
<td>gdb2</td>
<td>339</td>
<td>0.25</td>
<td>16.33</td>
<td>339</td>
</tr>
<tr>
<td>gdb3</td>
<td>275</td>
<td>0.02</td>
<td>3.67</td>
<td>275</td>
</tr>
<tr>
<td>gdb4</td>
<td>287</td>
<td>0.00</td>
<td>2.47</td>
<td>287</td>
</tr>
<tr>
<td>gdb5</td>
<td>377</td>
<td>0.13</td>
<td>10.27</td>
<td>377</td>
</tr>
<tr>
<td>gdb6</td>
<td>298</td>
<td>0.03</td>
<td>5.13</td>
<td>298</td>
</tr>
<tr>
<td>gdb7</td>
<td>325</td>
<td>0.00</td>
<td>1.80</td>
<td>325</td>
</tr>
<tr>
<td>gdb8</td>
<td>348</td>
<td>348</td>
<td>480.5</td>
<td>350</td>
</tr>
<tr>
<td>gdb9</td>
<td>303</td>
<td>303</td>
<td>752.20</td>
<td>303</td>
</tr>
<tr>
<td>gdb10</td>
<td>275</td>
<td>0.02</td>
<td>3.53</td>
<td>275</td>
</tr>
<tr>
<td>gdb11</td>
<td>395</td>
<td>0.87</td>
<td>9.53</td>
<td>395</td>
</tr>
<tr>
<td>gdb12</td>
<td>458</td>
<td>458</td>
<td>24.40</td>
<td>458</td>
</tr>
<tr>
<td>gdb13</td>
<td>536</td>
<td>0.15</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>gdb14</td>
<td>100</td>
<td>0.01</td>
<td>2.87</td>
<td>100</td>
</tr>
<tr>
<td>gdb15</td>
<td>58</td>
<td>58</td>
<td>0.71</td>
<td></td>
</tr>
<tr>
<td>gdb16</td>
<td>127</td>
<td>127</td>
<td>127.00</td>
<td>127</td>
</tr>
<tr>
<td>gdb17</td>
<td>91</td>
<td>91</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>gdb18</td>
<td>164</td>
<td>0.08</td>
<td>2.80</td>
<td>164</td>
</tr>
<tr>
<td>gdb19</td>
<td>55</td>
<td>55</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>gdb20</td>
<td>121</td>
<td>121</td>
<td>0.71</td>
<td></td>
</tr>
<tr>
<td>gdb21</td>
<td>156</td>
<td>156</td>
<td>155.00</td>
<td>156</td>
</tr>
<tr>
<td>gdb22</td>
<td>200</td>
<td>200</td>
<td>0.69</td>
<td></td>
</tr>
<tr>
<td>gdb23</td>
<td>233</td>
<td>233</td>
<td>234.7</td>
<td></td>
</tr>
</tbody>
</table>

Reported GRASP results were obtained with 15 runs. LB—best known lower bound. UB—best known feasible solution cost. CPU—execution time in seconds. Median CPU refers to the execution time for the median cost solution. iter—number of GRASP iterations executed. Best solutions between metaheuristics in bold.

Table 4
GRASP results for val instances.

<table>
<thead>
<tr>
<th>Instance</th>
<th>LB</th>
<th>UB</th>
<th>GRASP</th>
<th>Best cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Median</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cost</td>
<td>CPU</td>
<td>iter</td>
<td>Cost</td>
</tr>
<tr>
<td>val1A</td>
<td>173</td>
<td>0.12</td>
<td>3.07</td>
<td>173</td>
</tr>
<tr>
<td>val1B</td>
<td>173</td>
<td>190</td>
<td>227.00</td>
<td>173</td>
</tr>
<tr>
<td>val1C</td>
<td>245</td>
<td>0.49</td>
<td>21.87</td>
<td>245</td>
</tr>
<tr>
<td>val1D</td>
<td>457</td>
<td>3.96</td>
<td>205.87</td>
<td>457</td>
</tr>
<tr>
<td>val2A</td>
<td>81</td>
<td>0.12</td>
<td>3.27</td>
<td>81</td>
</tr>
<tr>
<td>val2B</td>
<td>87</td>
<td>0.36</td>
<td>8.07</td>
<td></td>
</tr>
<tr>
<td>val3A</td>
<td>138</td>
<td>0.89</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>val4A</td>
<td>400</td>
<td>0.48</td>
<td>8.40</td>
<td></td>
</tr>
<tr>
<td>val4B</td>
<td>412</td>
<td>11.97</td>
<td>21.67</td>
<td></td>
</tr>
<tr>
<td>val4C</td>
<td>428</td>
<td>98.02</td>
<td>774.53</td>
<td></td>
</tr>
<tr>
<td>val4D</td>
<td>530</td>
<td>158.00</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>val5A</td>
<td>423</td>
<td>4.72</td>
<td>8.80</td>
<td></td>
</tr>
<tr>
<td>val5B</td>
<td>446</td>
<td>8.13</td>
<td>4.39</td>
<td></td>
</tr>
<tr>
<td>val5C</td>
<td>427</td>
<td>98.33</td>
<td>1353.33</td>
<td></td>
</tr>
<tr>
<td>val5D</td>
<td>583</td>
<td>155.55</td>
<td>2200</td>
<td></td>
</tr>
<tr>
<td>val6A</td>
<td>223</td>
<td>0.53</td>
<td>4.33</td>
<td></td>
</tr>
<tr>
<td>val6B</td>
<td>233</td>
<td>50.72</td>
<td>846.67</td>
<td></td>
</tr>
<tr>
<td>val6C</td>
<td>317</td>
<td>52.86</td>
<td>906.67</td>
<td></td>
</tr>
<tr>
<td>val6D</td>
<td>379</td>
<td>3.56</td>
<td>8.77</td>
<td></td>
</tr>
<tr>
<td>val7A</td>
<td>283</td>
<td>1.31</td>
<td>6.00</td>
<td></td>
</tr>
<tr>
<td>val7B</td>
<td>334</td>
<td>33.40</td>
<td>219.47</td>
<td></td>
</tr>
<tr>
<td>val7C</td>
<td>386</td>
<td>2.04</td>
<td>5.67</td>
<td></td>
</tr>
<tr>
<td>val7D</td>
<td>395</td>
<td>10.69</td>
<td>30.07</td>
<td></td>
</tr>
<tr>
<td>val8A</td>
<td>521</td>
<td>115.43</td>
<td>1908.67</td>
<td></td>
</tr>
<tr>
<td>val8B</td>
<td>521</td>
<td>0.91</td>
<td>25.92</td>
<td></td>
</tr>
<tr>
<td>val8C</td>
<td>332</td>
<td>65.91</td>
<td>305.53</td>
<td></td>
</tr>
<tr>
<td>val8D</td>
<td>326</td>
<td>51.00</td>
<td>326</td>
<td></td>
</tr>
<tr>
<td>val9A</td>
<td>332</td>
<td>159.90</td>
<td>914.30</td>
<td></td>
</tr>
<tr>
<td>val9B</td>
<td>381</td>
<td>246.30</td>
<td>1833.33</td>
<td></td>
</tr>
<tr>
<td>val9C</td>
<td>428</td>
<td>73.04</td>
<td>26.00</td>
<td></td>
</tr>
<tr>
<td>val9D</td>
<td>466</td>
<td>211.84</td>
<td>888.87</td>
<td></td>
</tr>
<tr>
<td>val10A</td>
<td>446</td>
<td>446</td>
<td>333.33</td>
<td>446</td>
</tr>
<tr>
<td>val10B</td>
<td>532</td>
<td>530.6</td>
<td>2093.33</td>
<td>531</td>
</tr>
</tbody>
</table>

Reported GRASP results were obtained with 15 runs. LB—best known lower bound. UB—best known feasible solution cost. CPU—execution time in seconds. Median CPU refers to the execution time for the median cost solution. iter—number of GRASP iterations executed. Best solutions between metaheuristics in bold.
Fig. 3 clearly shows that EvPR is very effective in finding high-quality solutions in less time. For example, it has 50% chance to hit the 21,100 target, for instance egl-s4-c, in less than 500 s. With the same probability, it requires 30% additional time in average to hit the 21,100 target, for instance egl-s1-C.

4.5. Comparison of metaheuristics

The GRASP was compared to three high-performance metaheuristics, based on their average deviation from lower bounds reported in literature, which are:

- TS—Tabu Search proposed by Brand and Eglese [24], which executed a single run for each instance on a Pentium Mobile at 1.4 GHz.
- VNS—Variable Neighborhood Search proposed by Polacek et al. [29], which executed ten runs for each instance (except for the gdb set) on a Pentium IV at 3.6 GHz.
- ACO—Ant Colony Optimization proposed by Santos et al. [31], which executed fifteen runs for each instance on a Pentium III at 1.0 GHz.

Tables 3–5 report the results for sets gdb, val, and egl after 15 runs of the GRASP with evolutionary PR (Algorithm 8), and compare the best solutions obtained by each metaheuristic. The best lower bounds (LB) [9] and upper bounds (UB) [31] for each instance were also reproduced. It should be noticed that UB can be lower than the best solution reported for some instances (e.g., val10D). The values of UB were generated after additional experiments with CARP metaheuristics, for example by particular parameter tuning for each instance. Thus UB values are used only as an information of the current best known solution cost for each instance, and are not comparable with the metaheuristics best results. A similar study was made with the GRASP, and through it, five new best upper bounds (in italics) were discovered for instances egl-e4-c (UB = 11559), egl-s2-B (UB = 13088), egl-s3-C (UB = 17189), egl-s4-B (UB = 16267), and egl-s4-C (UB = 20484).

Table 6 summarizes CARP metaheuristics results on computational effort and solution quality. The average execution times reported are multiples of the metaheuristics original times, where the factor was determined by the processor frequency ratio between the original machine and the machine used in this work. The intention was to make a reasonably fair execution time comparison, despite distinct programming languages, operating systems, and other particular configurations of each machine. On the one hand, the GRASP is the most computer demanding metaheuristic, reaching over 4 min of CPU time per instance, on

![Table 5](image)

<table>
<thead>
<tr>
<th>Instance</th>
<th>LB</th>
<th>UB</th>
<th>GRASP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cost</td>
<td>iter</td>
<td></td>
</tr>
<tr>
<td>egl-e1-A</td>
<td>3548</td>
<td>3548</td>
<td>3548.0</td>
</tr>
<tr>
<td>egl-e1-B</td>
<td>4506.6</td>
<td>72.16</td>
<td>1286.67</td>
</tr>
<tr>
<td>egl-e1-C</td>
<td>5542</td>
<td>5595</td>
<td>5613.5</td>
</tr>
<tr>
<td>egl-e2-A</td>
<td>5011</td>
<td>5018</td>
<td>5018.0</td>
</tr>
<tr>
<td>egl-e2-B</td>
<td>6307.7</td>
<td>318.16</td>
<td>6466.67</td>
</tr>
<tr>
<td>egl-e2-C</td>
<td>8234</td>
<td>8335</td>
<td>8335.8</td>
</tr>
<tr>
<td>egl-e3-A</td>
<td>5898</td>
<td>5989</td>
<td>5989.0</td>
</tr>
<tr>
<td>egl-e3-B</td>
<td>7697</td>
<td>7775</td>
<td>7775.3</td>
</tr>
<tr>
<td>egl-e3-C</td>
<td>10,163</td>
<td>10,292</td>
<td>10,292.5</td>
</tr>
<tr>
<td>egl-e4-A</td>
<td>6395</td>
<td>6444</td>
<td>6444.1</td>
</tr>
<tr>
<td>egl-e4-B</td>
<td>8884</td>
<td>8983</td>
<td>8937.1</td>
</tr>
<tr>
<td>egl-e4-C</td>
<td>11,427</td>
<td>11,559</td>
<td>11,560.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
<th>LB</th>
<th>UB</th>
<th>GRASP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cost</td>
<td>CPU</td>
<td>iter</td>
</tr>
<tr>
<td>egl-s1-A</td>
<td>5014</td>
<td>5018</td>
<td>5038.9</td>
</tr>
<tr>
<td>egl-s1-B</td>
<td>6379</td>
<td>6388</td>
<td>6388.4</td>
</tr>
<tr>
<td>egl-s1-C</td>
<td>5014</td>
<td>5018</td>
<td>5018.0</td>
</tr>
<tr>
<td>egl-s2-A</td>
<td>5011</td>
<td>5018</td>
<td>5018.0</td>
</tr>
<tr>
<td>egl-s2-B</td>
<td>12,968</td>
<td>13,088</td>
<td>13,088.4</td>
</tr>
<tr>
<td>egl-s2-C</td>
<td>16,353</td>
<td>16,425</td>
<td>16,425.8</td>
</tr>
</tbody>
</table>

average. On the other hand, this additional time is highly compensated by presenting the best overall results in both average deviation from lower bounds (\(\Delta LB = 0.33\%\)) and number of best solutions (\(n_{best} = 72\)). In additional, GRASP was the only metaheuristic to achieve the optimal solution for every gdb instance. It is also worth noticing the GRASP excellent performance in the hardest set egl, where compared to the ACO, it reduced \(\Delta LB\) from 0.9% to 0.79% and found five more best solutions.

5. Conclusions

This work contribution resides on a high-end metaheuristic to solve the capacitated arc routing problem (CARP), grounded on a greedy randomized adaptive search procedure (GRASP) with evolutionary path-relinking.

The GRASP constructive heuristic was based on the Santos et al. [20] path-scanning with ellipse rule heuristic. The restricted candidate list parameter \(x\) and the ellipse rule parameter \(\beta\) were made reactive, or in other words, had their values selected from a set of possible values based on the average solution cost induced by each of them. This scheme has been shown successful on reducing the initial solutions average and maximum deviations from lower bounds with almost none additional computational effort.

In the GRASP local search phase, not all initial solutions have their neighborhood explored. A filter prevents low-quality solutions going through local search by defining a statistical threshold, which gives more than 95% probability of not throwing away an initial solution that would otherwise outperform the incumbent best. The proposed filter was demonstrated by time-to-target plots (TTT) to improve GRASP run time in average.

A path-relinking, whose elite solution pool progressively improves itself (evolutionary), was proposed based on the work of Resende et al. [47]. The proposed metaheuristic alternates GRASP iterations with the evolutionary path-relinking, in an attempt to intensify the search, while preserving some diversity. As recommended by Glover [45], this work does not constrain the search in the feasible solution space, but also explores paths traversing the feasible/infeasible boundaries. This is accomplished by an infeasible local search, which reduces the cost of a locally optimum feasible solution through capacity infeasible moves. The resulting solutions are then used as initial solutions for the path-relinking. The effectiveness of evolutionary path-relinking in the metaheuristic run time was demonstrated by TTT plots.

In the computational experiments 81 instances from the literature (gdb [19], val [14], and egl [49]) were solved by the GRASP, and compared with a tabu search [24], variable neighborhood search [29], and ant colony optimization [31] metaheuristics. Results show that the GRASP outperformed all other metaheuristics with respect to the overall average deviation from lower bound and number of best solutions found, in spite of additional execution time.

Acknowledgments

This work was supported by CNPq (Brazilian Research Agency). Special thanks for both anonymous referees for their valuable comments.

References


