Journal of the Operational Research Society (2014) 65, 1800–1813

www.palgrave-journals.com/jors/

GRASP with path relinking for the orienteering problem

Vicente Campos[1], Rafael Martí [1]*, Jesús Sánchez-Oro [2] and Abraham Duarte [2]

[1]*Universitat de València, València, Spain; and* [2]*Universidad Rey Juan Carlos, Mostoles, Spain*

In this paper, we address an optimization problem resulting from the combination of the well-known travelling salesman and knapsack problems. In particular, we target the orienteering problem, originated in the context of sport, which consists of maximizing the total score associated with the vertices visited in a path within the available time. The problem, also known as the selective travelling salesman problem, is NP-hard and can be formulated as an integer linear program. Since the 1980s, several solution methods for this problem have been developed and applied to a variety of fields, particularly in routing and tourism. We propose a heuristic method—based on the Greedy Randomized Adaptive Search Procedure (GRASP) and the Path Relinking methodologies—for finding approximate solutions to this optimization problem. We explore different constructive methods and combine two neighbourhoods in the local search of GRASP. Our experimentation with 196 previously reported instances shows that the proposed procedure obtains high-quality solutions employing short computing times.

Journal of the Operational Research Society (2014) **65**(12), 1800–1813. doi:10.1057/jors.2013.156

Published online 13 November 2013

Keywords: metaheuristics; GRASP; path relinking; orienteering problem

1. Introduction

The orienteering problem (OP), originated in the context of sport, consists of determining a path from an origin to a destination in a graph (directed or undirected), through a subset of locations in order to maximize the sum of the scores of the visited locations. Not all locations (vertices in the graph) can be visited since the available time (or total distance) is limited. The OP belongs to the well-known family of routing problems with many different applications (see, eg, Pacheco *et al*, 2009) and variants, including multi-objective approaches (Jozefowiez *et al*, 2008; Schilde *et al*, 2009, Martí *et al*, 2011). Different applications for this problem can be found, for example, in tourism, where we want to plan a tourist route in a large city, giving scores to the locations (in terms of their cultural interest for instance), and the tour visiting the selected vertices cannot exceed a maximum length or time (Tsiligirides, 1984).

Let $G(V,A)$ be a complete directed graph where V ($|V|=n$) and A ($|A|=m$) represent, respectively, the set of vertices and arcs. Each vertex $v_i \in V$ has a non-negative score S_i for $i=1,\ldots,n$, and each arc $(i,j) \in A$ has an associated non-negative travel time t_{ij} for $i,j=1,\ldots,n$. The OP consists of determining a path from v_1 to v_n, visiting some of the vertices in V in a way that the total travel time does not exceed a pre-established limit $T_{\max}$, maximizing the sum of the associated scores.

*Correspondence: Rafael Martí, Departamento de Estadística e Investigación Operativa, Universitat de València, c/ Dr. Moliner 50, Valencia, Burjassot 46100, Spain.
E-mail: rafael.marti@uv.es

We can formulate the OP as a linear integer problem (Vansteenwegen *et al*, 2011) by defining the binary variables $x_{ij}=1$ if vertex i is visited followed by vertex j, and $x_{ij}=0$ otherwise, for $i,j=1,\ldots,n$. In this formulation, originally proposed by Miller *et al* (1960) in the context of the TSP, we also need the integer variables u_i with the position of vertex i in the path, $i=1,\ldots,n$. In mathematical terms:

$$\text{Max} \sum_{i=2}^{n-1}\sum_{j=2}^{n} S_i x_{ij},$$

s.t.:

$$\sum_{j=2}^{n} x_{1j} = \sum_{i=1}^{n-1} x_{in} = 1, \tag{1}$$

$$\sum_{j=2}^{n} x_{kj} = \sum_{i=1}^{n-1} x_{ik} \leqslant 1, \quad k = 2, \ldots, n-1, \tag{2}$$

$$\sum_{i=1}^{n-1}\sum_{j=2}^{n} t_{ij} x_{ij} \leqslant T_{\max}, \tag{3}$$

$$2 \leqslant u_i \leqslant n, \quad i = 2, \ldots, n, \tag{4}$$

$$u_i - u_j + 1 \leqslant (n-1)\left(1 - x_{ij}\right), \quad i,j = 2, \ldots, n, \tag{5}$$

$$x_{ij} \in \{0,1\}\ 1 \leqslant i,j \leqslant n,\ u_i \in \mathrm{Z}, \quad i = 2, \ldots, n.$$

In the formulation above, constraint (1) forces the path to start at vertex 1 and to end at vertex n, while constraint (2)

guarantees that every vertex in the graph is visited at most once. The time limit constraint (3) and the sub-tour elimination constraints (4) and (5) complement the formulation.

We do not include a discussion of previous work on the OP because fairly complete reviews have appeared in recent publications, including Vansteenwegen *et al* (2011). They clearly stated that the methods proposed in the most recent solution approach for the OP (Schilde *et al*, 2009), Ant Colony Optimization and VNS, improve upon all the previous approximate methods, including the classic five-step heuristic of Chao *et al* (1996). A path relinking (PR) approach was applied for the Team Orienteering Problem in Souffriau *et al* (2010); in this variant of the OP, a set of routes have to maximize the total score of the visited vertices, and each route cannot exceed a common limit time. On the other hand, we have also identified an exact branch-and-cut algorithm to optimally solve this problem (Fischetti *et al*, 1998). It must be noted that this method includes the computation of a lower bound in the root node of the branch-and-cut search tree. Our main contribution is the development of a solution method for the OP based on the Greedy Randomized Adaptive Search Procedure (GRASP) and Path Relinking (PR) methodologies (Resende and Ribeiro, 2003), using several constructive methods. We compare our different designs and test them over a collection of instances with optimum known and also with a wider set of instances (in which we consider the best-known solutions up to now). The experimentation shows that our methods compete with the best heuristics reported in the literature.

2. Constructive methods

In this section, we propose four constructive methods for the OP based on GRASP (Resende and Ribeiro, 2003).

Given a graph $G(V,A)$, constructive method C1 starts with a path of length one in which we directly go from 1 to n through the arc $(1, n)$. The set $P = \{1, n\}$ represents the partial solution under construction and T_P, its associated travel time (initially $T_p = t_{1n}$). The candidate list CL is formed with all the vertices not present in the path that can be added to the current path within the time limit $T_{\max}$. Specifically, in the first step

$$CL = \{i \in V \setminus P : t_{1i} + t_{in} \leqslant T_{\max}\}.$$

At each step, C1 selects a candidate element $i \in CL$ to be included in the partial solution. C1 implements a typical GRASP construction in which first each candidate element is evaluated with a greedy function to construct the restricted candidate list $RCL1$ and then an element is selected at random from the $RCL1$. In particular, we compute the maximum score $S_{\max}$ of the elements in CL as

$$S_{\max} = \max_{i \in CL} S_i,$$

then we construct the $RCL1$ with all the candidate elements with a score value within a fraction α (the so-called *greediness parameter*) of the maximum score. In mathematical terms:

$$RCL1 = \{i \in CL : S_i \geqslant \alpha S_{\max}\}.$$

Finally, C1 randomly selects an element of the $RCL1$ and the selected element is inserted in the best position in the path P (ie, the one that produces the minimum path travel time). C1 performs iterations reconstructing the CL as long as new vertices can be added to the partial path (ie, as long as the CL is not empty). When no element out of the path can be inserted in it within the time limit, C1 stops and returns the path as the solution.

The randomization in C1 permits running it several times, say *Max_iter* iterations. Performing multiple GRASP iterations may be interpreted as a means of strategically sampling the solution space. On the basis of empirical observations, it has been found that the sampling distribution generally has a mean value that is inferior to the one obtained by a deterministic construction (that one with the same GRASP elements but replacing the random selection with the best available one), but the best overall trial dominates the deterministic solution with a high probability. The intuitive justification of this phenomenon is based on the ordering statistics of sampling. It implements a way of independently sampling the solution space, and each construction consists of an independent algorithm. In this sense, GRASP is a memory-less method since no information is recorded from one construction to the next.

We now consider C2, an alternative construction procedure introduced in Resende and Werneck (2004) as random plus greedy construction, which has been successfully applied in, for example, Resende *et al* (2010), Pantrigo *et al* (2012), or Duarte *et al* (2011). At each step in C2, we first construct CL, as in the case of the constructive method C1. The restricted candidate list $RCL2$ is constructed, determining first its cardinality from a fraction $0 < \alpha < 1$ of the elements in CL ($|RCL2| = \alpha|CL|$) with no repetitions, then a random sample of size $|RCL2|$ is taken from CL, and the element $i \in RCL2$ with the maximum score S_i is selected. Like C1, C2 inserts the selected element in the best position (ie, the one that produces the least increase in the path travel time) in the path P. It performs iterations as long as new vertices can be added to the partial path (ie, while CL is not empty). Note that the role of α is not the same in C1 and in C2. In C1, the greater α is, the lesser random the method of selection, while in C2 α computes the fraction of the candidates independently of the element quality. In other words, any element in $RCL2$ can be selected at any iteration, while in C1 the best elements of $RCL1$ have a greater probability of being selected.

Constructive method C3 also implements a typical GRASP construction, as C1, in which first each candidate element is evaluated by a greedy function to construct RCL, from which an element is randomly selected. The candidate set of elements CL is computed in the same way as in C1 and C2, but the greedy evaluation e_i of element i consists now on the quotient between the score S_i and the smallest increment in the time,

Δt_i, when the element is inserted in the path. In mathematical terms:

$$e_i = \frac{S_i}{\Delta t_i}.$$

As is customary in GRASP, we construct *RCL*3 with the elements in *CL* with an evaluation within a fraction α of the maximum value. In mathematical terms:

$$RCL3 = \{i \in CL : e_i \geqslant \alpha e_{\max}\},$$

where

$$e_{\max} = \max_{i \in CL} e_i.$$

Finally, C4 differs from C3 in the way the random and the greedy choices are made (as C2 differs from C1). In particular, C4 first constructs the restricted candidate list *RCL*4 with a random sample of the elements in *CL* of size $|RCL4| = \lceil \alpha |CL| \rceil$ where no repetitions are allowed. Then, it evaluates all the elements in *RCL*4, computing e_i for all $i \in RCL4$, and selects the best one. In other words, the element with maximum quotient between the score S_i and the time increment Δt_i. Like the three previous methods, C4 inserts the selected element in the best position (ie, the one that produces the least increase in the path travel time) in the path under construction.

2.1. Comparison of methods

When we design a constructive method as a part of a larger solving method, it has to be able to produce good starting points for the master method. This is especially true in the case of GRASP with PR in which we apply first the local search and then the PR to the constructed solutions. In this context, we want the constructive method to obtain good solutions in terms of the objective function, but also diverse to reach different regions of the solution space.

Considering the four constructive methods proposed above, a way of comparing them is to generate a set of solutions with each one and compare their quality and diversity. Since the quality is directly measured by the objective function, we now propose a measure of diversity. Given two solutions, $A = \{1, a_1, a_2, \ldots, a_k, n\}$ and $B = \{1, b_1, b_2, \ldots, b_t, n\}$, we compute their diversity, $div(A,B)$, as the number of elements (vertices in the graph) in A not present in B, plus the number of elements in B not present in A. To make this value independent of the size of the problem and of the maximum time limit $T_{\max}$, we divide it by the total number of elements, excluding origin and destination, present in both solutions (ie, $k+t$).

To illustrate, suppose two solutions in a graph with $n = 21$, $A = \{1, 7, 10, 4, 8, 3, 12, 20, 21\}$ with seven elements (without computing the origin 1 and destination 21) and $B = \{1, 12, 3, 9, 11, 7, 21\}$ with five elements. The number of elements in A not present in B is 4 and the number of elements in B not present in A is 2. Then, the diversity value between A and B is:

$$div(A, B) = \frac{4+2}{7+5} = 0.5.$$

We then generate 100 solutions with each constructive method and compute the average diversity value between all pairs of solutions. On the other hand, we compute the objective function value of each solution and its associated relative deviation from the optimal solution value. The averages of these deviation and diversity values provide an overall evaluation of the method.

To test this point with our four constructive methods and the different values of their associated parameter, we consider 33 instances with optimum known in Fischetti *et al* (1998). Table 1 shows for each method, C1, C2, C3, and C4, and each value of the parameter α tested, 0.2, 0.4, 0.6, and 0.8, the average across the 33 instances of the average deviation value (Dev) and the average diversity value (Div).

Table 1 shows that the best method in terms of quality is C2 (with $\alpha = 0.2$), since it is able to obtain a 7.2% deviation, which compares favourably with the rest of the methods. We also observe that the best method in terms of diversity is C1 with an average value of 0.57. Given their different nature and range, it is difficult to directly compare both measures (Div and Dev) in order to establish the best constructive method overall. Figure 1 depicts a point for each of the 16 methods reported in Table 1 (the four methods with the four values of α). The x-axis represents the diversity value (Div) and the y-axis the deviation value in the range [0, 1] (ie, we represent the 1-Dev value). In this way, for both values in the diagram, we have that the larger the value the better the method.

If we analyse the points in Figure 1 from a bi-objective perspective, in which an objective is to maximize the quality (1-Dev) and the other to maximize diversity (Div), we conclude that there are only five non-dominated points (methods): C1(0.2), C1(0.4), C1(0.6), C2(0.2), and C3(0.2). We say that a point is non-dominated if there is no other point with a better value in one objective and a better or equal value in the other.

Table 1 Average deviation from optimal values and diversity of constructive methods

	α	0.2	0.4	0.6	0.8
C1	Dev	19.3%	13.5%	12.1%	13.0%
	Div	0.57	0.49	0.43	0.32
C2	Dev	7.2%	7.9%	8.4%	9.9%
	Div	0.28	0.20	0.15	0.11
C3	Dev	11.8%	13.1%	13.8%	14.2%
	Div	0.41	0.33	0.27	0.21
C4	Dev	7.4%	7.3%	7.9%	9.2%
	Div	0.25	0.18	0.14	0.11

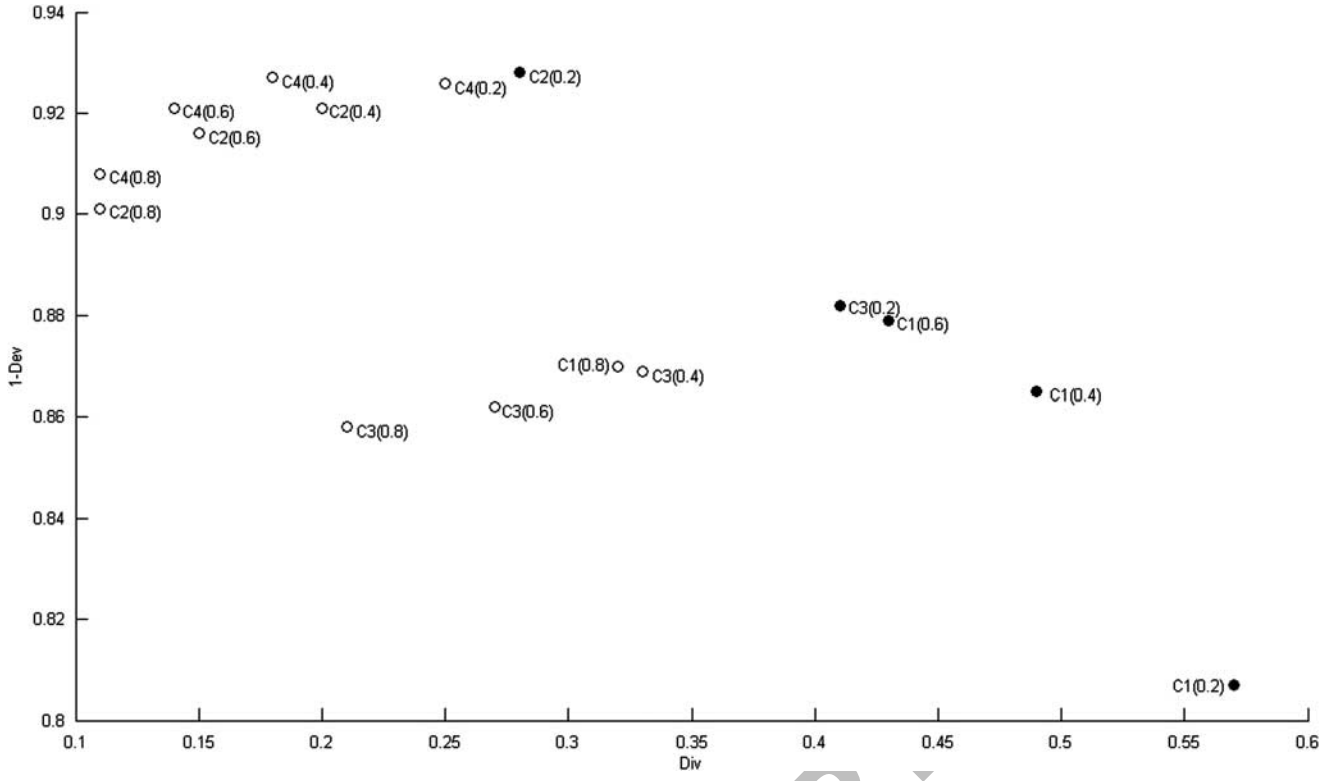

Figure 1 Quality and diversity of constructive methods.

It is difficult to determine whether the quality is more or less important than the diversity in a constructive method. We are not able to anticipate which one can influence more in the application of the local search to the solutions obtained with the construction. We then cannot conclude which of these five methods is the best and will test them with the local search.

3. Improvement method based on local search

We have implemented a two-phase local search procedure. The first phase is based on exchanges, while the second one is based on insertions. Previously, we applied a 2-opt standard improvement mechanism (Lawler *et al*, 1985) to the constructed solutions, before submitting them to our improvement method. The 2-opt will be applied again to the improved solutions.

The neighbourhood of the first phase is based on the exchange between a vertex v in the path P and another vertex w not in P (without exceeding the maximum time limit T_{max}). The difference between the scores of both vertices ($S_w - S_v$) provides the associated move value. We examine the vertices in the path and, for each vertex v in P, we consider to exchange it with each vertex out of the path. We compute the value of each combination and perform the exchange with the largest improvement found. If for a vertex v in P no associated move qualifies to be performed (all of them are non-improving moves), we try to reduce the length of the path without decreasing the total score. Specifically, we explore again the vertices w not in P but now we focus on those with the same score value than v and check whether the exchange reduces the length of the path. In this case, we perform the move; otherwise, we resort to the next vertex v in the path P.

When a one-to-one exchange is performed, we try an insertion move in which a vertex not present in the current path is considered to be added to it. Note that in this problem the insertion of a new point into the path could not necessarily increment its length (some points are in the same location). It could even reduce the total length because the matrix distance does not necessarily satisfy the triangular inequality in some instances, and therefore after we add a vertex to the path we have to check the addition of more vertices. This is why after an exchange we consider insertions as long as we can add vertices in the path without exceeding T_{max}. The added vertices are inserted in the best position without changing the relative order of the other vertices in the path.

The local search procedure examines the vertices in their order in the path and tries to perform exchanges and insertions for all of them as described above. If a move has been performed for any vertex in the path, we explore again all the vertices in the current path until no further improvement is possible.

3.1. Comparison of methods

In the previous section, we tested 16 constructive methods (four algorithms with four values of parameter α) on 33 instances.

We identified five of them, C1(0.2), C1(0.4), C1(0.6), C2(0.2), and C3(0.2) as the best ones in terms of quality and diversity. Now, we are going to see their performance when the local search is applied to the 100 solutions constructed with each one. For each method and each instance, we compute the best solution found over the 100 constructions plus the local search described above. Table 2 shows the average deviation with respect to the optimum value (Dev) and the number of best solutions that each method is able to match (#best).

Table 2 shows that the best solutions are obtained with the C1(0.2)+LS, which is able to achieve a 3.42% deviation and 17 best solutions out of the 33 instances. Anyway, all the methods perform very well considering that their running times are extremely short (below 0.1 s on each instance on an Intel Core i5 650 at 3.20 GHz). If we check the best solutions that each method is able to identify, we find out that some of them are obtaining different best solutions. This is especially true when we compare two methods based on different constructive algorithms. For example, if we compare the 16 best solutions obtained with C2(0.2)+LS and the 17 obtained with C1(0.2)+LS, we realize that only 10 of them are the same and both together are able to match 23 best solutions. As a matter of fact, these are the two methods sharing the least number of best solutions and therefore they are suitable to be combined for improved outcomes.

Table 2 Average deviation and number of best solutions of constructive methods plus local search

	$C1$ $(0.2)+LS$	$C1$ $(0.4)+LS$	$C1$ $(0.6)+LS$	$C2$ $(0.2)+LS$	$C3$ $(0.2)+LS$
Dev	**3.42%**	3.97%	5.65%	3.96%	7.76%
#best	**17**	13	7	16	7

The best algorithm is shown in bold.

We have performed a further preliminary experiment to compare the value of the constructive method with the value of the construction coupled with the local search. In particular, we have generated 100 solutions on instance gil262 with C1(0.2) computed their value, improved them with LS and computed the value of the improved solution. Figure 2 depicts a scatter-plot with 100 points, where the coordinates of each one is the pair of values of each solution (the score of the constructed solution on the x-axis and the score of the improved solution on the y-axis).

The points in Figure 2 range in the x-axis from 6691 to 8813, while in the y-axis they go from 9998 to 10 940. We observe two effects, the first one is that, as expected, the values in the y-axis are larger than those in the x-axis. The second one is that the range in the y-axis is narrower (942) than the range in the x-axis (2122). Moreover, the best solution found with the LS with value 10 940 does not come from the best construction value 8813, which can be interpreted that diversity is as important as quality when applying the local search. However, the correlation coefficient between both variables is r=0.284, which indicates that the correlation is weak, but significantly positive as stated by a t-test. Therefore, we cannot conclude with a high confidence that the good solutions of the local search come from the good constructions, and the performance of the entire method is also explained by its diversification power. For the sake of simplicity, we only depict here an

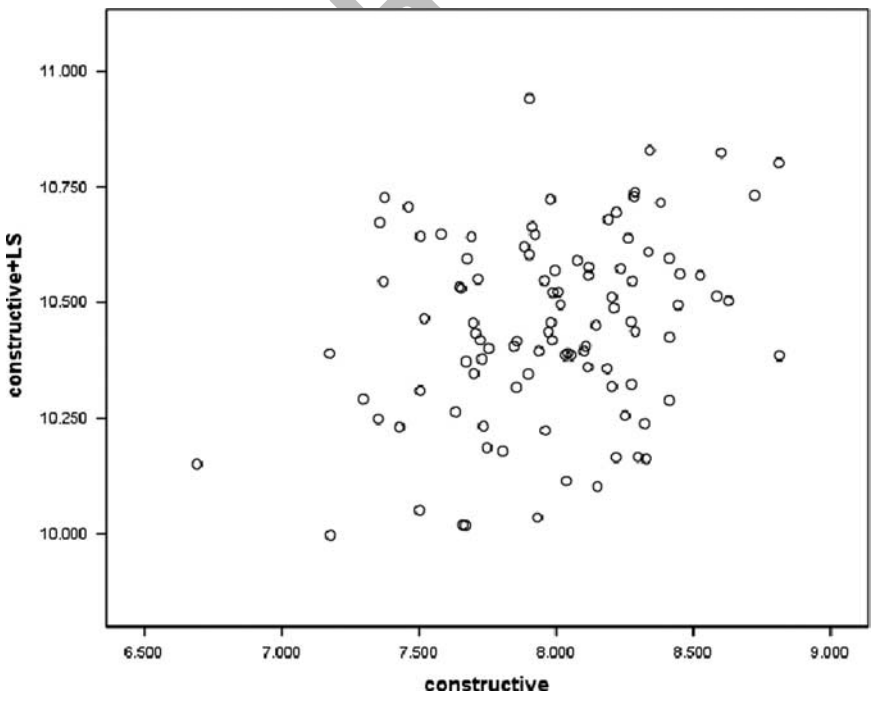

Figure 2 Objective function on *gil*262 instance with C1(0.2).

example, although we have empirically found that this behaviour is representative over the instances tested.

4. Path relinking

PR was suggested as an approach to integrate intensification and diversification strategies in the context of tabu search (Glover and Laguna, 1997). This approach generates new solutions by exploring trajectories that connect high-quality solutions—by starting from one of these solutions, called the initiating solution, and generating a path in the neighbourhood space that leads towards the other solutions, called guiding solutions. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions, and incorporating them in an intermediate solution initially originated in the initiating solution.

Laguna and Martí (1999) adapted PR in the context of GRASP as a form of intensification. The relinking in this context consists in finding a path between a solution found with GRASP and a chosen elite solution. Therefore, the relinking concept has a different interpretation within GRASP, since the solutions found in different GRASP iterations are not linked by a sequence of moves (as in the case of tabu search). Resende and Ribeiro (2003) present numerous examples of GRASP with PR.

Let $P = \{1, u_1, u_2, \ldots, u_k, n\}$ and $Q = \{1, w_1, w_2, \ldots, w_s, n\}$ be two solutions (paths from 1 to n) of the OP. The PR procedure PR(P,Q) starts with the first solution P, and gradually transforms it into the second one Q, by swapping out elements in P with elements in Q. The elements in both solutions P and Q remain in the intermediate solutions generated in the path between them. Let V_{Q-P} be the set of vertices in Q and not present in P and symmetrically let V_{P-Q} be the set of vertices in P and not present in Q.

To apply PR(P,Q), we order the vertices in V_{Q-P} according to their score, where the vertex with the largest score comes first. In the first step of the path, we insert the first vertex from V_{Q-P} in P in the best position according to the total time of the path. If the obtained path is feasible (its total time does not exceed $T_{\max}$), we have obtained a better solution than P, which is considered the first intermediate solution; otherwise, we remove from this path vertices of V_{P-Q} consecutively until the solution becomes feasible (vertices in V_{P-Q} are selected by increasing score). In further steps of the method, we insert into the intermediate solution the next vertex in V_{Q-P} and remove, if necessary, vertices in V_{P-Q} to generate a sequence of feasible solutions until we finally reach Q.

Our implementation of PR has two phases. In the first one, a set of different solutions is generated with the GRASP method (ie, we remove from the set those solutions that are identical). Instead of retaining only the best solution overall when running GRASP, this phase stores all the different solutions obtained with the method. In the second phase, we first apply a 2-opt improvement method (Lawler *et al*, 1985) to the GRASP solutions in this set, and then we apply the relinking process to each pair of them. Given the pair (P,Q), we consider two paths: from P to Q (where P is the initiating solution and Q the guiding one), and from Q to P (where they interchange their roles). In short, we apply PR(P,Q) and PR(Q,P).

We have experimentally found that in most cases this relinking process by itself does not produce better solutions than the initiating and guiding solutions. It is convenient to add a local search exploration from some of the visited solutions in order to produce improved outcomes. These results are in line with those reported in Laguna and Martí (1999) for the arc crossing problem. Specifically, we have applied the local search method introduced in the previous section to the best solution generated in the path if it improves either the initiating or the guiding solution, or both.

4.1. Comparing GRASP and GRASP with path relinking

In this section, we compare GRASP and GRASP with PR on the 33 instances used in Sections 2 and 3 to evaluate the contribution of the PR post-processing. Table 3 shows the objective function (Value), the average deviation with respect to the optimum value (Dev), the number of optimal solutions (#opt) and best solutions (#best) that each of both methods is able to match (comparing only the results of both methods), and the CPU time in seconds (Time). We run GRASP for 500 constructions and then apply, in GRASP with PR, the PR to all pairs of different solutions.

The results in Table 3 clearly indicate that the PR post-processing significantly improves the GRASP method. This is especially true with the number of best solutions, since GRASP only obtains 14 best values by itself and when it is coupled with PR the combined method is able to match 33 best values. However, PR consumes much longer running times than GRASP.

To complement the results above, we study now the search profile of our two methods. Figure 3 shows the progression of the best solutions found by GRASP and GRASP with PR, for a representative problem instance (cmt200vrpC), during 325.71 s of search time. This figure shows how most improvements on the best solution are achieved early in the search (ie, within 10% of the total search time, corresponding to 30 s approximately). After that point, GRASP stagnates, while GRASP with PR exhibits an improving trajectory throughout the entire search horizon. In this experiment, the best solution found has a score of 2852, while the optimal value reported in Fischetti *et al* (1998) is 2881 with a CPU time of 17 389.9 s on an HP Apollo 9000/720 at 80 MHz, with 58 MIPS, and 18 MFlops. We have

Table 3 Comparison over 33 instances

	GRASP	*GRASP with PR*
Value	2078.30	2099.82
Dev.	1.75%	0.84%
#best	14	33
#opt	14	18
Time	1.72	26.87

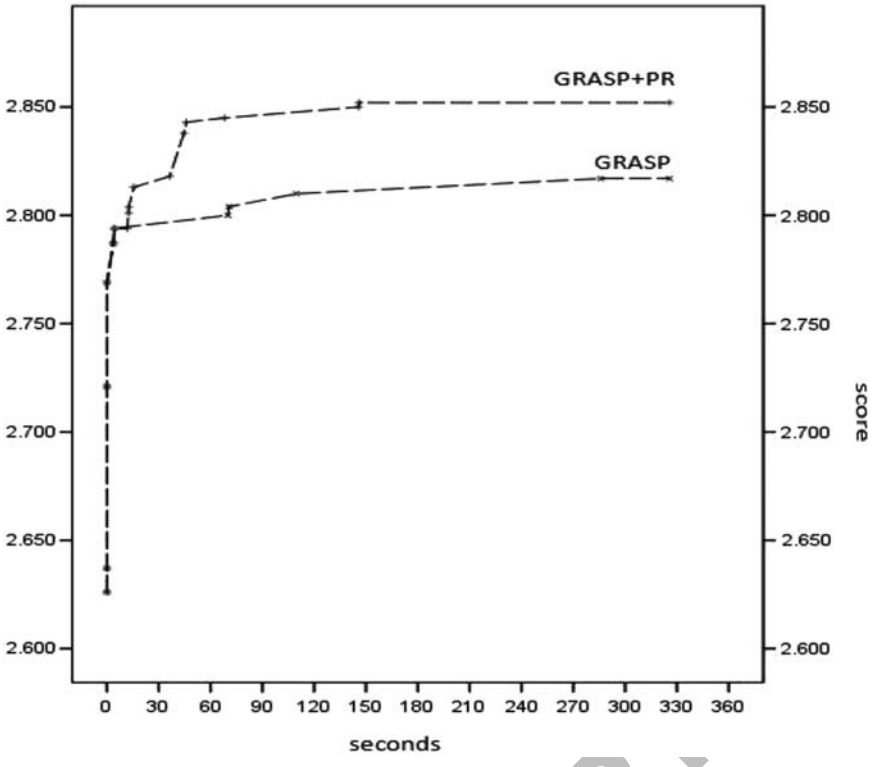

Figure 3 Search profile on instance cmt200vrpC.

performed the same experiment with different instances and the profile follows the same pattern depicted in this figure: GRASP with PR outperforms GRASP over the long search horizon.

5. Comparison with previous methods

In this section, we report our computational experiments to compare the methods proposed in the sections above with the best previous heuristics. Specifically, we compare our GRASP and GRASP with PR with the following four methods:

- CGW by Chao *et al* (1996)
- GLS by Vansteenwegen, *et al* (2009)
- ACO by Schilde *et al* (2009)
- VNS by Schilde *et al* (2009)

The first benchmark for this problem was proposed by Tsiligirides (1984), which consists of 49 instances with vertices ranging from 21 to 33. These small instances have optimum known, and as described in Schilde *et al* (2009) most of the current methods are able to match these optima. More recently, Chao *et al* (1996) proposed a benchmark with 40 larger instances in two sets: the first set, called p64, has 14 of them with 64 vertices, and the second set, called p66, the other 26 with 66 vertices each one. All the instances in a set are based on the same graph and they only differ in the time limit value ($T_{\max}$). We employ these 40 instances to compare our method with the four previous ones.

Table 4 Comparison of previous best methods on p64 instances

T_{max}	*CGW*	*GLS*	*ACO*		*VNS*	
			value	*time*	*value*	*time*
15	96	96	96	0.007	96	0.000
20	294	294	294	0.017	294	0.002
25	390	390	390	0.025	390	0.022
30	474	474	474	0.034	474/468	1.217
35	570	552	576/570	0.508	576	0.148
40	714	702	714	0.409	714	2.453
45	816	780	816/804	7.013	816	0.587
50	900	888	900/894	4.492	900/894	3.872
55	984	972	984/978	8.323	984/966	3.011
60	1044	1062	1062/1056	0.991	1062/1050	1.606
65	1116	1110	1116	0.711	1116	8.268
70	1176	1188	1188	2.281	1188/1182	0.975
75	1224	1236	1236	0.721	1236/1230	1.158
80	1272	1260	1284/1278	2.109	1284/1278	12.593
Average	790.7	786.0	795.0/792.0	1.970	795.0/790.7	2.570

Table 4 shows the best values obtained with the four methods referenced above (CGW, GLS, ACO, and VNS) on the p64 set of instances. These values are taken from Table 1 in Schilde *et al* (2009), and correspond to an Intel Pentium 4D at 3.2 Ghz. Note that the running times are missing for CGW and GLS methods. On the other hand, ACO and VNS are replicated 10 times on each instance and they reported the best and worst

values of the 10 runs (and only one value if both are the same). The running times correspond to the elapsed seconds to reach the solution found in the best run out of the 10. These results indicate that the ACO method is the best one and therefore we compare in Table 5 our GRASP and GRASP with PR with ACO on the p64 set of instances. To perform a fair comparison, we replicate our methods 10 times and report the elapsed time as Schilde *et al* (2009) did with ACO. However, running times of our two methods are computed on an Intel Core i5 650 at 3.20 GHz, so a direct comparison of running times with ACO is not possible.

Table 5 Comparison with the previous best method on p64 instances

T_{max}	*ACO*		*GRASP*		*GRASP + PR*	
	value	*time*	*value*	*time*	*value*	*time*
15	96	0.007	96	0.015	96	0.015
20	294	0.017	294	0.046	294	0.062
25	390	0.025	390	0.062	390	0.140
30	474	0.034	468	0.062	468	0.171
35	576/570	0.508	576	0.078	576	0.280
40	714	0.409	714/708	0.093	714	0.280
45	816/804	7.013	816/804	0.093	816	0.296
50	900/894	4.492	900	0.109	900	0.421
55	984/978	8.323	984/978	0.171	984/978	0.296
60	1062/1056	0.991	1062/1044	0.124	1062/1044	0.249
65	1116	0.711	1116	0.109	1116	0.202
70	1188	2.281	1188	0.093	1188	0.187
75	1236	0.721	1236	0.093	1236	0.171
80	1284/1278	2.109	1284/1278	0.093	1284/1278	0.140
Average	795.0/792.0	1.970	794.6/791.1	0.089	795.0/792.9	0.208

Table 5 clearly shows that our methods are competitive with the best published heuristic for this problem. In particular, GRASP obtains a best average value of 794.6 and a worst average value across the 10 replications of 791.1, and it exhibits an average CPU time of 0.089 to reach the best values. GRASP with PR marginally improves these results and has an average best and worst values across the 10 runs of 795.0 and 792.9, respectively, which are slightly better than the 795.0 and 792.0 obtained with ACO. Moreover, the average running time of GRASP + PR on an Intel Core i5 650 at 3.20 GHz is 0.208, while the CPU time required by ACO is 1.97 on an Intel Pentium 4D at 3.2 Ghz. It is difficult to perform indirect comparisons of running times taken from different computers, but the ratio between the average CPU times of ACO and GRASP with PR is 1.97/0.21 = 9.47, which seems to indicate that GRASP + PR is faster than ACO.

Table 6 shows the best values obtained with the two best previous heuristics, ACO and VNS, and our two methds,

Table 6 Comparison of best methods on p66 instances

T_{max}	*ACO*		*VNS*		*GRASP*		*GRASP with PR*	
					value	*time*	*value*	*time*
5	10	0.01	10	0.00	10	0.00	10	0.00
10	40	0.01	40	0.00	40	0.00	40	0.00
15	120	0.01	120	0.00	120	0.02	120	0.02
20	205	0.06	205	0.01	205	0.02	205	0.02
25	290	0.02	290	0.01	290	0.03	290	0.03
30	400	0.02	400	0.01	400	0.05	400	0.05
35	465	0.03	465	0.13	465	0.06	465	0.06
40	575	0.04	575	1.94	575/555	0.06	575	0.28
45	650	0.05	650	0.03	650	0.08	650	0.08
50	730	0.05	730	0.13	730	0.09	730	0.09
55	825	0.06	825	3.04	825	0.09	825	0.09
60	915	0.13	915	0.04	915	0.11	915	0.11
65	980	0.08	980	1.63	980	0.11	980	0.11
70	1070	0.08	1070	0.40	1070	0.11	1070	0.11
75	1140	0.09	1140/1135	0.06	1140	0.11	1140	0.11
80	1215	1.27	1215/1195	2.61	1215	0.12	1215	0.12
85	1270	0.22	1270/1265	0.97	1270	0.12	1270	0.12
90	1340	0.48	1340/1330	0.61	1340	0.12	1340	0.12
95	1395	0.39	1395/1390	0.85	1395	0.11	1395	0.11
100	1465	1.41	1465/1445	11.1	1465/1455	0.11	1465	0.41
105	1520	0.15	1520/1495	0.33	1520/1510	0.11	1520	0.51
110	1560	0.32	1560/1550	1.33	1560	0.09	1560	0.09
115	1595	0.16	1595/1585	8.46	1595	0.09	1595	0.09
120	1635	1.22	1635/1625	1.18	1635/1625	0.08	1635	0.22
125	1670	0.19	1670/1665	1.07	1670/1655	0.08	1670	0.19
130	1680	0.19	1680	0.02	1680/1640	0.00	1680	0.02
	952.3	0.26	952.3/947.5	1.38	952.3/948.3	0.07	952.3	0.12

GRASP and GRASP with PR, on the p66 set of instances. As in the previous experiment, the four methods are run 10 times on each instance and we report the best and worst value across the 10 replications. Note that we only report both values when they differ. The average best/worst values across the 10 runs of VNS and GRASP are, respectively, 952.3/947.5 and 952.3/948.3. On the other hand, ACO and GRASP with PR are able to match in the 10 runs the best-known value for each of the 26 instances in the p66 set. This may indicate that these instances, p66, are easier to solve than those reported in Table 5 (p64 instances). GRASP only requires an average running time of 0.07seconds, while GRASP with PR needs 0.12 seconds on average. ACO and VNS require 0.26 and 1.38 s, respectively. However, they were run on an older computer than that used for GRASP and GRASP with PR, so we can conclude that GRASP has a performance similar to that of VNS, and GRASP with PR a performance similar to that of ACO on this set of instances.

Fischetti *et al* (1998) proposed an exact procedure (a branch-and-cut algorithm) and compute the optimal value for some instances (their method requires about 5 h of CPU time to obtain the optimum in some of their largest instances). It must be noted that this method includes the computation of a lower bound in the root node of the branch-and-cut search tree. As shown in the previous sections, we have used these instances to calibrate our algorithm (find the best values for key search parameters) and compare different search strategies. We also use them in this section to compare the results of our two methods with the optimal solution and the lower bound.

Note that in the VRP instances reported in Fischetti *et al* (1998), called Class I instances, the customer demand is interpreted as the vertex score. Table 7 reports the average optimal value (opt) and the corresponding CPU time to obtain it with the branch and cut in Fischetti *et al* (1998) for the 33 instances in this set. This table also shows the average lower bound computed by this method in the root node of the search tree, its average percent deviation with respect to the optimum, and its associated running time. Finally, it also shows the average value of GRASP and GRASP with PR with their average deviations and running times in seconds. The individual results for each instance are shown in Table A1. Note that the branch and cut and its lower bound were run on a HP Apollo 9000/720 at 80 MHz with 18 MFlops and Cplex 3, while the GRASP and GRASP with PR on an Intel Core i5 650 at 3.20 GHz. Therefore, as mentioned above, we have to keep in mind that it is difficult to perform indirect comparisons of running times taken from different computers. It seems, however, that the branch and cut requires a running time longer than that of the heuristic methods: 1847.5 s on average *versus* 1.8 for GRASP and 32.6 for GRASP with PR, although it obtains the optimum in 32 out of the 33 instances, while the heuristics approximate it, as can be shown in the average percent deviations. On the other hand, the lower bound computed by the branch and cut is very accurate (it exhibits a 0.6% deviation), and is obtained in a relatively short running time (167.8 s on average). GRASP with PR performs remarkably well considering that it has an average deviation of 0.9% from the optimal solution value, achieved on 32.6 s.

Fischetti *et al* (1998) also considered a second set of 123 instances (Class II) derived from the TSPLIB 2.1 involving up to 400 nodes. For these instances, the scores were generated according to three different ways, called generations. Generation 1 contains 41 instances with the goal of covering as many nodes as possible (all the nodes have a score equal to 1). Generation 2 contains 41 instances with pseudo-random scores in the range [1, 100]. Finally, generation 3 contains the most difficult instances in which large prices are assigned to the nodes far away from the depot. Tables 8–10 summarize the results of each method (GRASP, GRASP with PR, Branch and Cut, and Lower Bound of the Branch and Cut) on each type of instance (generation), respectively. The results of each method on each instance are shown in Tables A2–A4, respectively, for each generation class.

Tables 8–10 clearly show that our both heuristics obtain high-quality solutions in short computing times. In particular,

Table 7 Comparison with optimal values in Class I instances

	Branch and cut	*LB*	*GRASP*	*GRASP with PR*
Value	2132.2	2094.2	2075.8	2095.3
CPU time	1847.5	167.8	1.8	32.6
Optimal deviation	0.0%	0.6%	1.7%	0.9%

Table 8 Comparison with optimal values in Class II (Generation 1) instances

	Branch and cut	*LB*	*GRASP*	*GRASP with PR*
Value	85.2	85.0	83.7	84.3
CPU time	1768.0	480.4	12.9	231.1
Optimal deviation	0.0%	0.1%	1.1%	0.5%

Table 9 Comparison with optimal values in Class II (Generation 2) instances

	Branch and cut	*LB*	*GRASP*	*GRASP with PR*
Value	4649.1	4645.4	4507.3	4563.2
CPU time	2729.6	975.2	4.8	91.5
Optimal deviation	0.0%	0.1%	2.1%	0.9%

Table 10 Comparison with optimal values in Class II (Generation 3) instances

	Branch and cut	*LB*	*GRASP*	*GRASP with PR*
Value	4509.7	4427.8	4371.8	4447.6
CPU time	8588.6	1943.3	5.5	179.8
Optimal deviation	0.0%	1.8%	2.1%	0.7%

GRASP with PR is able to obtain solutions below 1% of average deviation from optimality in less than 4 min on average. Moreover, the lower bound of the branch-and-cut method performs remarkably well, since it is able to obtain very accurate values (below 1.8% overall, with many instances matching the optimal solution). The tables in the appendix show the individual results of these methods on each instance. Note that according to Fischetti *et al* (1998), generation 3 contains the most difficult instances, and our GRASP with PR obtains in these instances its best values, with an average percentage deviation from the optimal solution of 0.7%.

6. Conclusions

The OP is a difficult combinatorial optimization problem and a perfect platform to study the effectiveness of search mechanisms. Of particular interest in our work has been testing the effect of combining two different neighbourhoods within the local search of GRASP, as well as studying the effect of a post-processing based on PR. Through extensive experimentation, we have been able to determine the benefits of adding enhanced search strategies to basic procedures. We have compared our methods with the best identified in previous studies. The comparison clearly shows that our proposals are competitive with the state-of-the-art methods.

Acknowledgements—This research has been partially supported by the Ministerio de Educación y Ciencia of Spain (Grant Refs. TIN2009-07516 and TIN2012-35632-C02) and the Generalitat Valenciana (Prometeo 2013/049). The authors thank Profs Schilde, Doerner, Hartl, and Kiechle for sharing their results with them. The authors also thank Profs. Fischetti, Salazar, and Toth for sharing their branch-and-cut code with them.

References

Chao IM, Golden BL and Wasil EA (1996). A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research* **88**(3): 475–489.

Duarte A, Martí R, Resende MGC and Silva RMA (2011). GRASP with path relinking heuristics for the antibandwidth problem. *Networks* **58**(3): 171–189.

Fischetti M, Salazar JJ and Toth P (1998). Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing* **10**(2): 133–148.

Glover F and Laguna M (1997). *Tabu Search*. Kluwer Academic Publishers: Boston, MA.

Jozefowiez N, Glover F and Laguna M (2008). Multi-objective meta-heuristics for the traveling salesman problem with profits. *Journal of Mathematical Modelling and Algorithms* **7**(2): 177–195.

Laguna M and Martí R (1999). GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing* **11**(1): 44–52.

Lawler AEL, Lenstra JK, Rinooy Kan AHG and Shwoys DB (1985). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley Series in Discrete Mathematics & Optimization: New York.

Martí R, Campos V, Resende MGC and Duarte A (2011). *Multi-objective GRASP with Path Relinking*. Technical Report, Universidad de Valencia: Valencia.

Miller CE, Tucker AW and Zemlin RA (1960). Integer programming formulation of traveling salesman problems. *Journal ACM* **7**(4): 326–329.

Pacheco J, Alvarez A, Casado S and González-Velarde JL (2009). A tabu search approach to an urban transport problem in Northern Spain. *Computers and Operations Research* **36**(3): 967–979.

Pantrigo JJ, Duarte A, Martí R and Pardo EG (2012). Scatter search for the cutwidth minimization problem. *Annals of Operations Research* **199**(1): 285–304.

Resende MGC, Martí R, Gallego M and Duarte A (2010). GRASP and path relinking for the max-min diversity problem. *Computers and Operations Research* **37**(3): 498–508.

Resende MGC and Ribeiro CC (2003). Greedy randomized adaptive search procedures. In: Glover F and Kochenberger G (eds). *State-of-the-Art Handbook in Metaheuristics*. Kluwer Academic Publishers: Boston, MA, pp. 219–250.

Resende MGC and Werneck RF (2004). A hybrid heuristic for the p-median problem. *Journal of Heuristics* **10**(1): 59–88.

Schilde M, Doerner KF, Hartl RF and Kiechle G (2009). Metaheuristics for the bi-objective orienteering problem. *Swarm Intell* **3**(3): 179–201.

Souffriau W, Vansteenwegen P, Vanden Berghe G and Oudheusden DV (2010). A path relinking approach for the team orienteering problem. *Computers and Operations Research* **37**(11): 1853–1859.

Tsiligirides T (1984). Heuristic methods applied to orienteering. *Journal of the Operational Research Society* **35**(9): 797–809.

Vansteenwegen P, Souffriau W and Oudheusden DV (2011). The orienteering problem: A survey. *European Journal of Operational Research* **209**(1): 1–10.

Vansteenwegen P, Souffriau W, Vanden Berghe G and Oudheusden DV (2009). A guided local search metaheuristic for the team orienteering problem. *European Journal of the Operational Research* **196**(1): 118–127.

Appendix

Table A1 Comparison with optimal values in Class I instances

Instance	*Branch and cut*		*LB*		*GRASP*		*GRASP with PR*	
	Optimal	*time*	*value*	*time*	*value*	*time*	*value*	*time*
att48vrpA	17	2.6	17	2.6	17	0.1	17	0.1
att48vrpB	30	0.8	30	0.8	30	0.1	30	1.4
att48vrpC	39	1.1	39	1.1	39	0.2	39	2.7
cmt101vrpA	530	57.8	530	57.8	530	0.1	530	3.8
cmt101vrpB	1030	55.2	1019	36.9	1010	0.4	1020	14.1
cmt101vrpC	1480	130.7	1480	130.7	1460	0.6	1480	14.9
cmt121vrpA	412	612.5	405	347.9	412	0.7	412	20.6
cmt121vrpB	715	1525.6	715	411.1	699	0.8	707	16.3
cmt121vrpC	1134	50.9	1134	50.9	1120	1.4	1125	46.0
cmt151vrpA	824	128.4	824	128.4	815	1.2	824	21.6
cmt151vrpB	1537	167.3	1535	131.8	1482	2.2	1528	30.5
cmt151vrpC	2003	380.3	1998	114.6	1965	2.3	1996	47.9
cmt200vrpA	1205	299.1	1203	130.7	1145	2.9	1181	34.1
cmt200vrpB	2198	596.3	2198	147.4	2073	5.6	2105	51.1
cmt200vrpC	2881	17389.9	2644	127.9	2791	5.8	2824	133.8
eil101vrpA	572	189.7	572	177.3	566	0.3	572	10.0
eil101vrpB	1049	5.6	1049	5.6	1024	0.7	1032	14.6
eil101vrpC	1336	4.7	1336	4.7	1295	0.6	1302	14.3
eil30vrpA	2650	2.8	2650	2.8	2650	0.0	2650	0.0
eil30vrpB	7600	5.2	7600	5.2	7600	0.0	7600	0.0
eil30vrpC	11550	0.9	11550	0.9	11550	0.0	11550	0.0
eil51vrpA	264	2.2	264	2.2	264	0.2	264	0.4
eil51vrpB	508	2.4	508	2.4	508	0.3	508	2.0
eil51vrpC	690	77.6	690	60.0	690	0.3	690	2.0
eil76vrpA	490	48.7	490	48.7	490	5.5	490	3.6
eil76vrpB	907	3.1	907	3.1	896	13.4	904	8.8
eil76vrpC	1186	457.6	1181	56.6	1172	13.2	1184	7.4
gil262vrpA	4466	18000.0	4403	2541.6	3916	0.1	4050	138.4
gil262vrpB	8456	3252.7	8439	365.8	7946	0.1	8074	103.7
gil262vrpC	11195	17512.0	10288	436.5	10938	0.1	11046	328.4
op33A	250	0.8	250	0.8	250	0.1	250	0.1
op33B	500	1.8	500	1.8	500	0.1	500	0.6
op33C	660	1.1	660	1.1	660	0.1	660	1.1
Average	2132.2	1847.5	2094.2	167.8	2075.8	1.3	2095.3	32.5

Table A2 Comparison with optimal values in Class II (Generation 1) instances

Instance	*Branch and cut*		*LB*		*GRASP*		*GRASP with PR*	
	optimal	*time*	*value*	*time*	*value*	*time*	*value*	*time*
att48	31	0.7	31	0.7	31	0.1	31	1.51
gr48	31	1.1	31	1.1	31	0.2	31	1.92
hk48	30	1.7	30	1.7	30	0.2	30	3.24
eil51	30	1.2	30	1.2	30	0.2	30	0.31
brazil58	46	3.2	46	3.2	46	0.3	46	12.37
st70	44	5.3	44	5.3	44	0.5	44	2.26
eil76	47	5.7	47	5.7	47	0.7	47	1.34
pr76	49	50.9	49	50.9	48	0.6	49	44.09
gr96	64	121.1	64	113.8	64	1.4	64	93.38
rat99	53	53.5	53	53.5	52	1.3	53	5.62
kroA100	57	27.1	57	27.1	56	1.4	57	54.57
kroB100	58	326.9	57	156.7	58	1.4	58	54.40
kroC100	56	50.7	56	50.7	56	1.3	56	40.47
kroD100	59	32.0	59	32.0	59	1.4	59	65.96
kroE100	57	776.0	57	82.9	57	1.4	57	54.35
rd100	61	30.2	61	30.2	61	1.7	61	44.26
eil101	66	7.1	66	7.1	65	1.7	66	3.59
lin105	66	83.4	66	78.8	66	1.6	66	141.44
pr107	54	86.3	54	86.3	54	0.6	54	25.13
gr120	75	17.5	75	17.5	74	3.3	75	74.05
pr124	75	41.2	75	41.2	75	2.5	75	145.22
bier127	103	73.7	103	73.7	102	3.5	103	213.50
pr136	71	214.2	71	214.2	70	2.9	70	50.22
gr137	81	178.6	81	178.6	81	4.5	81	380.63
pr144	77	240.3	77	240.3	77	3.1	77	311.31
kroA150	86	4669.0	86	582.2	85	5.1	86	94.49
kroB150	87	145.6	87	145.6	86	5.7	87	107.42
pr152	77	204.6	77	204.6	77	3.1	77	365.31
u159	93	497.6	93	497.6	91	6.2	93	242.47
rat195	104	331.9	104	331.9	101	11.7	102	36.04
d198	124	716.3	124	716.3	122	15.1	123	424.13
kroA200	118	395.0	118	395.0	116	16.0	117	245.68
kroB200	119	683.6	119	683.6	116	14.0	118	321.59
ts225	126	18000.0	126	9.7	124	17.9	124	146.91
pr226	126	18000.0	126	1955.3	124	14.2	126	902.20
gil262	164	120.6	164	120.6	158	34.5	160	183.21
pr264	132	2860.2	132	2860.2	132	11.8	132	1267.89
pr299	162	14224.0	160	5726.3	156	63.2	158	933.33
lin318	206	3169.9	206	2558.0	196	82.8	201	1703.69
rd400	243	4272.5	238	874.0	228	176.2	228	444.24
Average	85.2	1768.0	85.0	480.4	83.7	12.9	84.3	231.1

Table A3 Comparison with optimal values in Class II (Generation 2) instances

Instance	*Branch and cut*		*LB*		*GRASP*		GRASP with PR	
	optimal	*Time*	*value*	*time*	*value*	*time*	*value*	*time*
att48	1634	3.9	1634	3.9	1634	0.1	1634	2.6
gr48	1469	18.0	1469	18.0	1469	0.1	1469	3.6
hk48	1535	7.1	1535	7.1	1524	0.1	1535	3.2
eil51	1778	30.7	1778	30.7	1778	0.1	1778	3.8
brazil58	2326	7.8	2326	7.8	2326	0.2	2326	13.1
st70	2302	181.0	2302	147.2	2294	0.3	2302	13.9
eil76	2591	7.2	2591	7.2	2525	0.4	2591	12.0
pr76	2666	62.0	2666	58.6	2626	0.3	2666	14.6
gr96	3506	453.1	3506	399.3	3447	0.7	3501	35.8
rat99	3042	125.4	3042	125.4	2976	0.7	3031	21.7
kroA100	3181	67.8	3181	67.8	3135	0.8	3181	29.1
kroB100	3195	481.4	3195	259.0	3183	0.7	3191	31.4
kroC100	3044	316.2	3040	207.6	3044	0.7	3044	21.7
kroD100	3226	334.2	3226	237.2	3152	0.7	3212	33.1
kroE100	3310	1433.9	3263	285.5	3260	0.7	3310	36.5
rd100	3470	27.8	3470	27.8	3449	0.7	3453	33.7
eil101	3668	296.5	3668	51.5	3596	0.8	3645	28.3
lin105	3577	163.6	3577	151.9	3577	0.9	3577	83.1
pr107	2681	99.4	2681	99.4	2681	0.5	2681	20.5
gr120	4223	650.0	4223	3471.0	4138	1.3	4201	43.9
pr124	3840	79.4	3840	79.4	3840	1.3	3840	83.3
bier127	5376	245.0	5376	73.2	5154	1.6	5264	100.6
pr136	4223	194.3	4223	194.3	4170	1.7	4213	36.8
gr137	4291	3193.0	4291	797.9	4255	1.9	4284	126.9
pr144	3994	1409.0	3994	668.0	3902	1.8	3994	114.9
kroA150	4919	3950.6	4889	460.1	4768	2.4	4915	52.8
kroB150	5017	1018.1	5017	735.7	4967	2.2	5001	66.5
pr152	4196	188.6	4196	188.6	4094	1.8	4175	166.7
u159	5044	1772.4	5023	518.0	4809	2.8	4987	100.1
rat195	5936	2498.6	5936	1750.1	5693	5.3	5693	63.8
d198	6539	2517.1	6532	1337.8	6347	5.8	6476	241.6
kroA200	6616	805.1	6616	515.2	6447	6.1	6551	111.5
kroB200	6597	3522.8	6590	1240.7	6357	5.6	6409	103.9
ts225	6812	1195.5	6812	763.3	6701	8.9	6784	93.6
pr226	6691	18000.0	6684	3973.9	6375	6.9	6614	265.5
gil262	9159	5574.6	9149	2783.0	8847	14.6	8941	132.8
pr264	6666	4253.3	6666	4253.3	6666	9.6	6666	343.7
pr299	9107	18000.0	9107	10803.8	8645	19.7	8689	400.9
lin318	10962	18000.0	10962	1370.0	10074	27.3	10339	339.7
rd400	13555	18000.0	13541	837.6	12365	54.0	12365	229.2
Average	4649.1	2729.6	4645.4	975.2	4507.3	4.8	4563.2	91.5

Table A4 Comparison with optimal values in Class II (Generation 3) instances

Instance	*Branch and cut*		*LB*		*GRASP*		*GRASP with PR*	
	optimal	*time*	*value*	*time*	*value*	*time*	*value*	*time*
att48	1050	251.8	1050	180.2	1050	0.1	1050	4.0
gr48	1481	27.5	1481	27.5	1481	0.1	1481	4.5
hk48	1765	3.5	1765	3.5	1765	0.1	1765	2.7
eil51	1465	19.5	1465	19.5	1465	0.1	1465	5.3
brazil58	1703	2.8	1703	2.8	1703	0.2	1703	12.8
st70	2182	70.6	2182	70.6	2182	0.3	2182	14.6
eil76	2526	49.6	2526	49.6	2469	0.4	2523	17.5
pr76	2431	34.0	2431	34.0	2421	0.4	2431	25.4
gr96	3183	416.6	3179	189.1	3165	0.9	3183	52.5
rat99	2950	487.4	2950	481.1	2929	0.9	2949	38.3
kroA100	3227	248.8	3227	144.8	3181	0.7	3227	51.1
kroB100	2807	138.9	2807	134.1	2762	0.7	2807	61.0
kroC100	3158	228.1	3158	228.1	3114	0.8	3152	40.7
kroD100	3173	230.3	3173	167.8	3168	0.8	3173	60.0
kroE100	3053	184.4	3053	184.4	3015	0.8	3053	69.6
rd100	2957	1032.3	2957	644.9	2940	0.9	2943	40.0
eil101	3427	186.7	3427	120.0	3341	0.9	3389	29.5
lin105	2995	1121.1	2953	325.9	2942	1.2	2995	101.7
pr107	1878	17609.0	1876	632.4	1876	0.6	1878	47.7
gr120	3780	145.5	3780	145.5	3687	1.7	3753	56.1
pr124	3558	11487.2	3337	1377.8	3547	1.3	3558	98.5
bier127	2366	2001.2	2351	139.3	2242	1.6	2336	116.5
pr136	4391	958.5	4391	861.6	4363	2.1	4381	124.4
gr137	3980	4958.7	2169	2401.9	3844	2.2	3946	100.4
pr144	3747	180000.0	3747	1573.8	3688	1.9	3747	176.6
kroA150	5050	3828.9	5044	269.7	5006	2.5	5043	157.1
kroB150	5337	1363.9	5337	1112.5	5203	2.4	5318	113.0
pr152	3910	13736.7	3882	1099.0	3871	2.3	3910	238.2
u159	5273	1447.2	5273	1308.4	5198	3.2	5273	232.0
rat195	6296	3975.4	6289	3672.2	5970	7.2	6164	124.7
d198	6369	8635.7	6369	1810.0	6136	8.8	6222	384.7
kroA200	6140	6548.9	6140	3116.5	5989	5.9	6089	205.1
kroB200	6274	783.7	6274	642.6	6024	5.2	6213	254.0
ts225	7618	5821.8	7618	3437.6	7477	11.8	7576	263.2
pr226	6994	7923.2	5874	3379.5	6802	7.0	6971	816.1
gil262	9547	9574.0	9527	6177.4	8911	15.9	9277	244.5
pr264	8138	4011.3	8138	4011.3	7752	12.8	8013	901.2
pr299	10356	18000.0	10356	14699.4	9848	26.0	10145	959.9
lin318	10430	18000.0	10430	8597.5	9664	29.1	9876	434.9
rd400	13422	18000.0	13422	14257.1	12681	56.4	12743	512.6
Average	4509.7	8588.6	4427.8	1943.3	4371.8	5.5	4447.6	179.8

Received 17 April 2012;
accepted 7 October 2013 after two revisions