GRASP with Path Relinking for the Orienteering Problem

VICENTE CAMPOS
Departamento de Estadística e Investigación Operativa
Universitat de València, Spain
vicente.campos@uv.es

RAFAEL MARTÍ
Departamento de Estadística e Investigación Operativa
Universitat de València, Spain
rafael.marti@uv.es

JESÚS SÁNCHEZ-ORO
Departamento de Ciencias de la Computación
Universidad Rey Juan Carlos, Spain
jesus.sanchezoro@urjc.es

ABRAHAM DUARTE
Departamento de Ciencias de la Computación
Universidad Rey Juan Carlos, Spain
abraham.duarte@urjc.es

ABSTRACT

In this paper, we address an optimization problem resulting from the combination of the well-known traveling salesman and knapsack problems. In particular, we target the orienteering problem, originated in the context of sports, 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 and the Path Relinking methodologies – for finding approximate solutions to this optimization problem. We explore different constructive methods and combine two neighborhoods in the local search of GRASP. Our experimentation with 73 previously reported instances shows that the proposed procedure obtains high quality solutions employing short computing times.

Keywords: Metaheuristics, GRASP, Path Relinking, Orienteering problem.

Original version: February 29, 2012
Revised version: March 21, 2013

1. Introduction

The orienteering problem (OP), originated in the context of sports, 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 for example Pacheco et al., 2009) and variants, including multi-objective approaches (Jozefowiez et al., 2008; Schilde et al., 2009). Different applications for this problem can be found, for example in tourism, 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 $i = 1, \dots, n$, and each arc $(i,j) \in A$ has an associated travel time t_{ij} for $i,j = 1, \dots, n$. 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.

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 $i,j = 1, \dots, n$. 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, $i = 1, \dots, n$. In mathematical terms:

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

s.t.:

$$\sum_{j=2}^{n} x_{1j} = \sum_{i=1}^{n-1} x_{in} = 1 \qquad (1)$$

$$\sum_{j=2}^{n} x_{kj} = \sum_{i=1}^{n-1} x_{ik} \leq 1 \qquad k = 2, \dots, n-1 \qquad (2)$$

$$\sum_{i=1}^{n-1} \sum_{j=2}^{n} t_{ij} x_{ij} \leq T_{max} \qquad (3)$$

$$2 \leq u_i \leq n \qquad i = 2, \dots, n \qquad (4)$$

$$u_i - u_j + 1 \leq (n-1)(1 - x_{ij}) \qquad i,j = 2, \dots, n \qquad (5)$$

$$x_{ij} \in \{0,1\} \quad 1 \leq i,j \leq n\,, \;\; u_i \in \mathrm{Z} \quad i = 2, \dots, n$$

In the formulation above, constraints (1) force the path to start at vertex 1 and to end at vertex n, while constraints (2) guarantee 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 orienteering problem 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 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, 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). 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, 2001), 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 orienteering problem 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 within the time limit T_{max}. Specifically, in the first step

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

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:

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

Finally, C1 randomly selects an element of the $RCL1$ and the selected element is inserted in the best position in the path P (i.e., the one that produces the least increase in the path travel time). C1 performs iterations reconstructing the CL as long as new vertices can be added to the partial path (i.e., 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. Based on 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). 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 ($|RCL2| = \lceil \alpha |CL| \rceil$) 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 (i.e., 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 (i.e., 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, whilst 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 $RCL3$ with the elements in CL with an evaluation within a fraction α of the maximum value. In mathematical terms:

$$RCL3 = \{i \in CL : \ e_i \geq \alpha \, e_{max}\} \quad \text{where} \quad e_{max} = max_{i \in CL} \, e_i$$

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 $RCL4$ 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 $RCL4$, computing e_i for all $i \in RCL4$, and selects the best one, i.e. the element with maximum quotient between the score S_i and the time increment Δt_i. As the three previous methods, C4 inserts the selected element in the best position (i.e., 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, $A = \{1, a_1, a_2, \dots, a_k, n\}$ and $B = \{1, b_1, b_2, \dots, b_t, n\}$ 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 (i.e., $k + t$).

To illustrate, suppose two solutions in a graph with $n = 21$, $A = \{1,7,10,4,8,3,12,20,21\}$ with 7 elements (without computing the origin 1 and destination 21) and $B = \{1,12,3,9,11,7,21\}$ with 5 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).

	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

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

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 favorably 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] (i.e., 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.

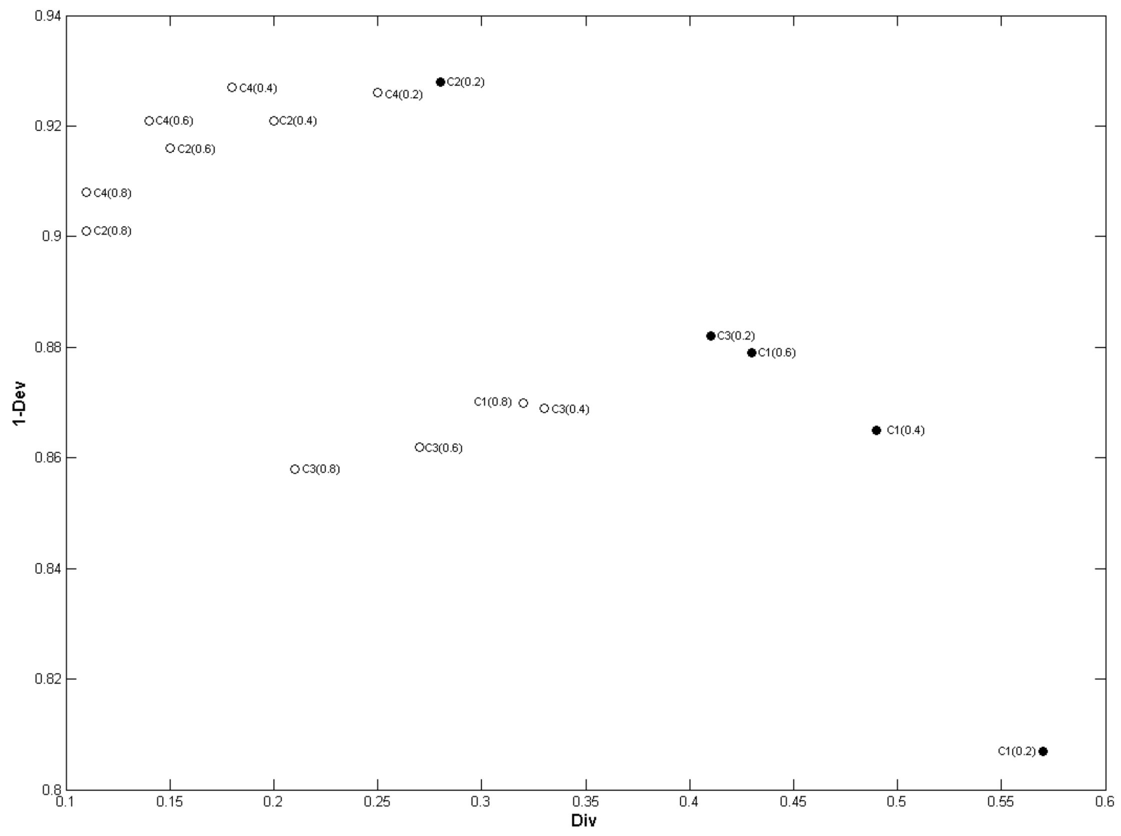

Figure 1. Quality and diversity of constructive methods.

If we analyze 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 method is non-dominated if there is no other method with a better value in one objective and a better or equal value in the other.

It is difficult to say if 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.

The neighborhood 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 if 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.

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 sixteen 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).

	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

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

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

seconds 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.

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).

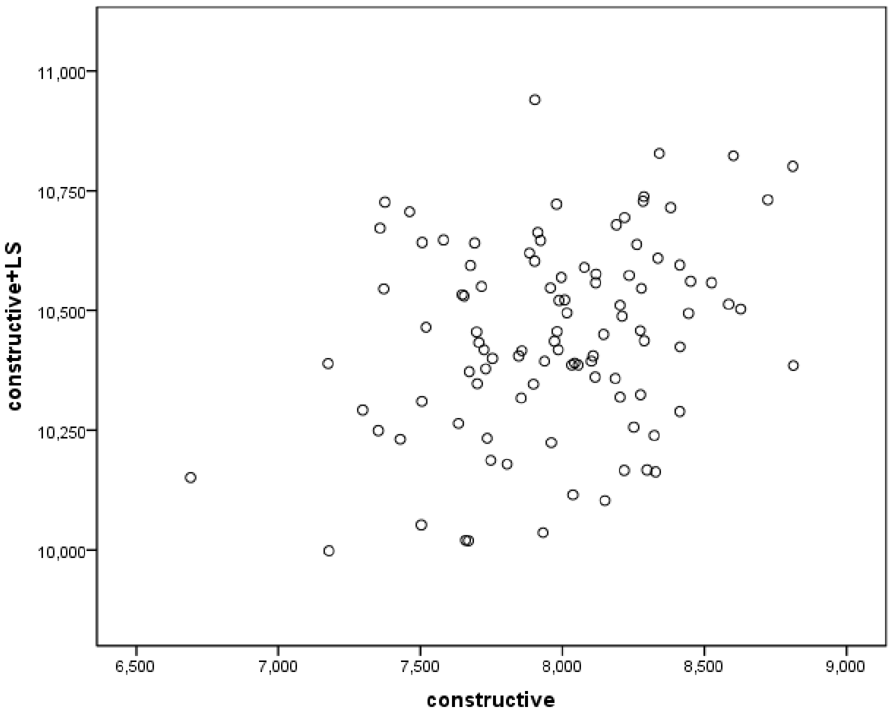

Figure 2. Objective function on *gil*262 instance with C1(0.2)

The points in Figure 2 range in the x-axis from 6,691 to 8,813 while in the y-axis they go from 9,998 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 (2,122). Moreover, the best solution found with the LS with value 10,940 does not come from the best construction value 8,813, which can be interpreted that diversity is as important as quality when apply 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 example although we have empirically found that this behavior is representative over the instances tested.

4. Path Relinking

Path Relinking 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 neighborhood space that leads toward 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 $P = \{1, u_1, u_2, \dots, u_k, n\}$ and $Q = \{1, w_1, w_2, \dots, w_s, n\}$ be two solutions (paths from 1 to n) of the orienteering problem. The path relinking procedure $\mathrm{PR}(P, Q)$ 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 $\mathrm{PR}(P, Q)$ 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 path relinking has two phases. In the first one a set of different solutions is generated with the GRASP method (i.e., we remove from the set those solutions that are equal). 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 apply the relinking process to each pair of solutions in the set. 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 $\mathrm{PR}(P, Q)$ and $\mathrm{PR}(Q, P)$.

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 path relinking 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 path relinking to all pairs of different solutions.

	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

Table 3. Comparison over 33 instances.

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, Path Relinking consumes much longer running times than GRASP.

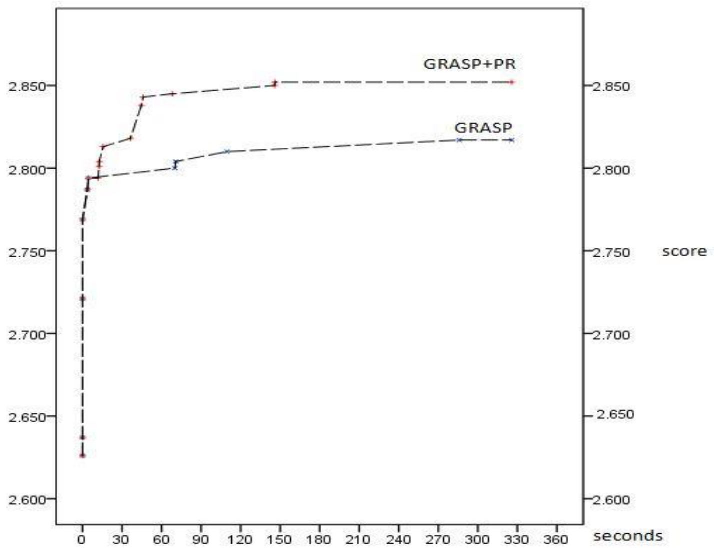

Figure 3. Search profile on instance cmt200vrpC

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 seconds of search time. This figure shows how most improvements on the best solution are achieved early in the search (i.e., within 10 percent of the total search time, corresponding to 30 seconds 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 2,852 while the optimal value reported in Fischetti et al. (1998) is 2,881 with a CPU time of 17,389.9 seconds on an HP Apollo 9000/720 at 80 MHz, with 58 MIPS, and 18 MFlops. We have 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.

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
Avg.	790.7	786.0	795.0 / 792.0	1.970	795.0 / 790.7	2.570

Table 4. Comparison of previous best methods on p64 instances

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 ten. 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.

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.

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
Avg.	795.0 / 792.0	1.970	794.6 / 791.1	0.089	795.0 / 792.9	0.208

Table 5. Comparison with the previous best method on p64 instances

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 Path Relinking 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 GRASP with PR and ACO 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 methods, 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. GRASP only requires an average running time of 0.07, while GRASP with PR needs 0.12 seconds on average. ACO and VNS require 0.26 and 1.38 seconds respectively. However, they were run on an older computer than GRASP and GRASP with PR, so we can conclude that GRASP has a similar performance than VNS, and GRASP with PR a similar performance than ACO in this set of 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

Table 6. Comparison of best methods on p66 instances

Fischetti et al. (1998) proposed an exact procedure (a branch-and-cut algorithm) for the special case in which the origin and destination are in the same location, and compute the optimal value for some instances (their method requires about 5 hours of CPU time to obtain the optimum in some of their larger instances). 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 solutions when solving this particular case.

Note that in the VRP instances reported in Fischetti et al. (1998), the customer demand is interpreted as the vertex score. Table 7 reports, for each of the 33 instances in this set, the optimal value (opt) and the corresponding CPU time to obtain it with the branch and cut in Fischetti et al. (1998), and the value of GRASP and GRASP with PR with the associated running time in seconds. These three methods were run on the same computer (an Intel Core i5 650 at 3.20 GHz). As expected, the branch and cut requires a running time much longer than the heuristic methods: 1060.46 seconds on average versus 3.43 for GRASP and 90.57 for GRASP with PR, although it obtains the optimum in the 33 instances while the heuristics approximate it. GRASP with PR performs remarkably well considering that it is able to match 19 optima and its average percent deviation from the optimal solution of 1.42%.

	Branch and Cut		GRASP		GRASP with PR	
instance	opt	time	value	time	value	time
op33A	250	0.08	250	0.06	250	0.30
att48vrpA	13	0.35	13	0.12	13	0.19
eil30vrpA	2650	0.13	2650	0.03	2650	0.03
eil51vrpA	264	0.17	264	0.12	264	0.84
eil76vrpA	490	0.77	490	0.06	490	7.30
eil101vrpA	572	733.61	572	0.12	572	23.85
cmt101vrpA	530	0.87	520	0.03	530	5.01
cmt121vrpA	412	4.73	408	0.12	412	50.19
cmt151vrpA	824	6.60	814	0.39	824	58.05
cmt200vrpA	1205	0.51	1162	0.80	1182	113.49
gil262vrpA	4466	4745.02	4167	0.36	4342	284.88
op33B	500	0.08	500	1.40	500	2.59
att48vrpB	30	0.07	30	2.51	30	6.18
eil30vrpB	7600	0.51	7600	6.07	7600	0.09
eil51vrpB	508	0.14	508	11.31	508	4.60
eil76vrpB	907	0.19	897	0.11	907	24.04
eil101vrpB	1049	0.33	1020	0.34	1032	34.59
cmt101vrpB	1030	0.73	990	0.05	1020	28.75
cmt121vrpB	715	11194.4	702	0.22	707	44.12
cmt151vrpB	1537	123.76	1501	0.70	1526	86.31
cmt200vrpB	2198	5217.42	2076	1.39	2103	147.75
gil262vrpB	8456	1370	7896	0.86	8004	341.86
op33C	660	0.09	660	1.73	660	4.32
att48vrpC	39	0.09	39	4.52	39	9.19
eil30vrpC	11550	0.20	11550	11.17	11550	0.34
eil51vrpC	690	0.15	690	22.74	690	5.13
eil76vrpC	1186	0.60	1180	0.09	1181	15.21
eil101vrpC	1336	967.49	1295	0.36	1304	31.70
cmt101vrpC	1480	0.41	1450	0.09	1480	41.06
cmt121vrpC	1134	218.24	1108	0.25	1119	116.84
cmt151vrpC	2003	623.05	1973	0.73	1992	137.37
cmt200vrpC	2881	1031.19	2794	1.33	2852	318.99
gil262vrpC	11195	8752.81	10922	1.15	11019	1043.63
Average	**2132.12**	**1060.46**	**2081.55**	**3.43**	**2101.58**	**90.57**

Table 7. Comparison with optimal values

6. Conclusions

The orienteering 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 neighborhoods within the local search of GRASP as well as studying the effect of a post-processing based on Path Relinking. 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.

Acknowledgments

This research has been partially supported by the Ministerio de Educación y Ciencia of Spain (Grant Refs. TIN2009-07516 and TIN2012-35632-C02). 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, I.M., Golden, B.L. and Wasil, E.A. (1996) A fast and effective heuristic for the orienteering problem, European Journal of Operational Research 88, 475-489.

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

Glover, F. , Laguna, M. (1997) Tabu Search, Kluwer Academic Publishers, Boston.

Jozefowiez, N., F. Glover, and M. Laguna (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, 11:44–52.

Miller, C. E., A. W. Tucker, and Zemlin, R. A. (1960) Integer programming formulations and traveling salesman problems, Journal ACM, 7, 326–329.

Pacheco, J., A. Alvarez, S. Casado and J. L. González-Velarde (2009) A Tabu Search Approach to an Urban Transport Problem in Northern Spain, Computers and Operations Research 36, pp. 967-979.

Resende, M.G.C. and Werneck, R.F. (2004) A hybrid heuristic for the p-median problem, Journal of Heuristics, 10:59–88.

Resende, M.G.C. and Ribeiro, C.C. (2003) Greedy randomized adaptive search procedures. F. Glover, G. Kochenberger, eds. State-of-the-art Handbook in Metaheuristics, Kluwer Academic Publishers, Boston, MA. 219–250.

Resende, M.G.C., R. Martí, M. Gallego and A. Duarte (2010), GRASP and Path Relinking for the Max-Min Diversity Problem, Computers and Operations Research 37(3), 498-508

Schilde, M., Doerner, K.F., Hartl, R.F.and Kiechle, G. (2009), Metaheuristics for the bi-objective orienteering problem, Swarm Intell., 3, 179 -201.

Souffriau, W., Vansteenwegen, P., Vanden Berghe, G. and Oudheusden, D.V. (2010) A path relinking approach for the team orienteering problem, Computers and Operations Research, 37, 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, D.V. (2011) The orienteering problem: A survey, European Journal of Operational Research 209, 1-10.

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