INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 24 (2017) 1307–1323
DOI: 10.1111/itor.12419

A GRASP heuristic using path-relinking and restarts for the Steiner traveling salesman problem

Ruben Interian and Celso C. Ribeiro

Institute of Computing, Universidade Federal Fluminense, Niterói, RJ 24210-346, Brazil
E-mail: rinterian@ic.uff.br [Interian]; celso@ic.uff.br [Ribeiro]

Received 31 July 2016; received in revised form 31 December 2016; accepted 28 February 2017

Abstract

The traveling salesman problem (TSP) is one of the most studied problems in combinatorial optimization. Given a set of nodes and the distances between them, it consists in finding the shortest route that visits each node exactly once and returns to the first. Nevertheless, more flexible and applicable formulations of this problem exist and can be considered. The Steiner TSP (STSP) is a variant of the TSP that assumes that only a given subset of nodes must be visited by the shortest route, eventually visiting some nodes and edges more than once. In this paper, we adapt some classical TSP constructive heuristics and neighborhood structures to the STSP variant. In particular, we propose a reduced 2-opt neighborhood and we show that it leads to better results in smaller computation times. Computational results with an implementation of a GRASP heuristic using path-relinking and restarts are reported. In addition, ten large test instances are generated. All instances and their best-known solutions are made available for download and benchmarking purposes.

Keywords: Steiner traveling salesman problem; traveling salesman problem; GRASP; path-relinking; restarts

1. Introduction

The traveling salesman problem (TSP) is one of the fundamental combinatorial optimization problems (Garey and Johnson, 1979; Zhang et al., 2015) and has numerous real-life applications in transportation, logistics, vehicle routing, genome sequencing, and other areas. In its undirected version it consists in, given a set of nodes and the distances between them, find the shortest route that visits each node exactly once and returns to the first city. Mathematically, the TSP can be defined as follows (Cornuéjols et al., 1985). Given a graph $G = (V, E)$ and a function $w : E \to \mathbb{R}$ that associates a weight $w(e)$ to each edge $e \in E$, the goal is to find a Hamiltonian cycle of minimum total weight (or cost). The TSP is NP-hard, since its decision version is proven to be NP-complete by a simple reduction from the Hamiltonian cycle problem (Garey and Johnson, 1979).

However, in many practical applications it is more frequent to find the following variant of the TSP. A set $V_R \subseteq V$ of required nodes is given. Instead of searching for a Hamiltonian cycle visiting

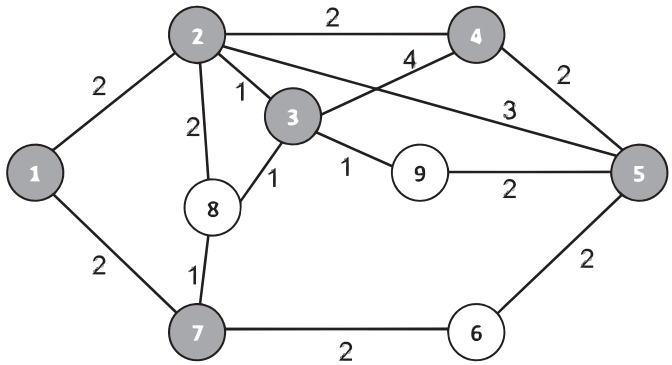

Fig. 1. Instance of Steiner TSP with nine nodes: the six required nodes are darkened.

all nodes, a minimum-weight closed walk is requested that visits only the required nodes. Since only a walk is sought, nodes can be visited more than once and edges may be traversed more than once. The so-called Steiner TSP (STSP, for short) was first proposed in Cornuéjols et al. (1985) and Fleischmann (1985), where its NP-hardness is also proved. The Steiner TSP is specially suitable to model network design (Borne et al., 2013), package delivery (Zhang et al., 2015, 2016), and routing (Letchford et al., 2013) problems. All of them are typically modeled using sparse graphs. Figure 1 illustrates an instance of the Steiner TSP with nine nodes, six of which are required.

Most studies on the Steiner TSP focus on integer programming formulations and valid inequalities. The STSP is solved efficiently (in linear time) for series-parallel graphs in Cornuéjols et al. (1985). Compact, polynomial size integer programming formulations of the TSP are extended to the STSP in Letchford et al. (2013). An extension of the Steiner TSP that adds penalties to the nodes not visited by the cycle is proposed in Salazar-González (2003). A network design problem consisting of multiple Steiner TSPs with order constraints is studied in Borne et al. (2013), using an integer linear programming formulation and a branch-and-cut algorithm. An extension of the STSP in which the edge traversal costs are stochastic and correlated is studied in Letchford and Nasiri (2015). An online algorithm is proposed in Zhang et al. (2015, 2016) to solve another extension of the STSP considering real-time edge blockages.

This paper is organized as follows. In the next section, adaptive greedy constructive heuristics for the Steiner TSP are presented. Section 3 reports local search strategies that are explored by the GRASP with path-relinking heuristic presented in Section 4. Computational experiments are reported in Section 5 and extended in Section 6, where an improved strategy exploring periodical restarts is developed. In addition, ten large test instances are generated and their best-known solutions are made available for download and benchmarking purposes in Section 7. Concluding remarks are drawn in the last section.

2. Greedy algorithms for the Steiner TSP

The following strategy can be applied as a heuristic for the Steiner TSP (Letchford et al., 2013). First, the instance of the STSP is reduced to a TSP instance in a complete graph defined by the set of required nodes, in which the new distances correspond to the shortest paths between every

pair of required nodes in the original graph. Next, any exact or heuristic algorithm is used to solve the TSP in this new, complete graph. Finally, the solution of the TSP is converted into an STSP solution by expanding every edge by the corresponding shortest path between the two consecutive required nodes. However, if the original STSP instance is a sparse graph, the conversion to a standard TSP instance significantly increases the number of edges, some of which may never be used.

Therefore, instead of using a complete graph formed by the required nodes, we will use the original graph for searching a minimum-weight closed walk. We use a straightforward adaptation of the nearest neighbor TSP adaptive greedy heuristic (see, e.g., Resende and Ribeiro, 2016, Chapter 3) to the STSP described in Algorithm 1, which builds the solution greedily by choosing at each iteration the closest required node to the last node added to the walk.

The algorithm starts in line 1 by arbitrarily selecting any initial node $i \in V_R$ to start the walk. The set of required nodes $\mathcal{N}$ already visited by the walk is initialized in line 2. The partially built walk $\mathcal{P}$ is initialized in line 3. The currently visited required node *current* is set in line 4. The loop in lines 5–11 is performed until all required nodes have been visited. At each iteration, the node *next* to be visited is set to the closest among all yet unvisited required nodes. The shortest path $\mathcal{P}'$ from *current* to *next* is computed in line 7. The partially built walk $\mathcal{P}$ is updated in line 8 by appending the shortest path $\mathcal{P}'$ to it. The set of already visited required nodes $\mathcal{N}$ and the current node are updated in lines 9 and 10, respectively. Finally, after completing the loop, the shortest path from *current* to the initial node *i* is appended to the walk in lines 12 and 13. The result is returned in line 14.

Algorithm 1. Nearest neighbor adaptive greedy heuristic for STSP

1: Select initial required node $i \in V_R$;
2: $\mathcal{N} \leftarrow \{i\}$;
3: $\mathcal{P} \leftarrow \{i\}$;
4: *current* $\leftarrow i$;
5: **while** $\mathcal{N} \neq V_R$ **do**
6: *next* $\leftarrow$ closest node to *current* among all those in $V_R \setminus \mathcal{N}$;
7: $\mathcal{P}' \leftarrow$ shortest path from *current* to *next*;
8: $\mathcal{P} \leftarrow \mathcal{P} \oplus \mathcal{P}'$;
9: $\mathcal{N} \leftarrow \mathcal{N} \cup \{next\}$;
10: *current* $\leftarrow$ *next*;
11: **end while**;
12: $\mathcal{P}' \leftarrow$ shortest path from *current* to initial node *i*;
13: $\mathcal{P} \leftarrow \mathcal{P} \oplus \mathcal{P}'$;
14: **return** $\mathcal{P}$.

In the case of the Steiner TSP, the greedy criterion is the choice of the nearest required node to be visited. Algorithms that add randomization to a greedy or adaptive greedy algorithm are called *semigreedy* or *randomized greedy* algorithms. Randomization is an important feature in the implementation of effective heuristics. Semigreedy algorithms act by replacing the deterministic greedy choice of the next element to be incorporated into the solution under construction by the random selection of an element from a restricted set of best candidate elements, called the restricted candidate list (RCL).

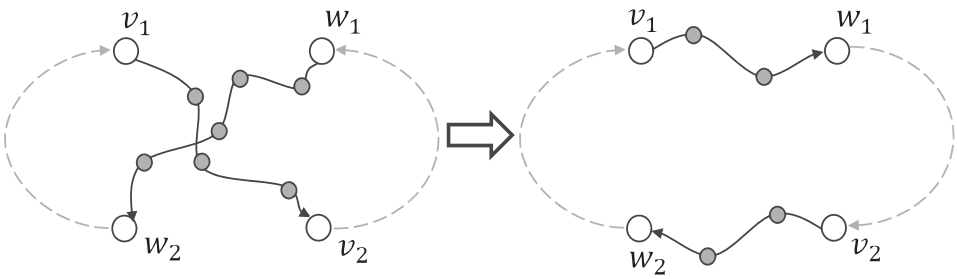

Fig. 2. 2-Opt move for STSP.

A simple quality-based scheme is used to define an RCL. Let $g_{\min} = \min\{g_i : i \in V_R \setminus \mathcal{N}$ and g_i is the shortest path from *current* to node $i\}$ and $g_{\max} = \max\{g_i : i \in V_R \setminus \mathcal{N}$ and g_i is the shortest path from *current* to node $i\}$. Furthermore, let α be such that $0 \leq \alpha \leq 1$. The RCL is formed by all yet unselected required nodes $i \in V_R \setminus \mathcal{N}$ satisfying $g_{\min} \leq g_i \leq g_{\min} + \alpha(g_{\max} - g_{\min})$.

3. Local search

Local search procedures are used to iteratively improve the quality of an initial solution, usually obtained by a constructive heuristic. First-improving (FI) and best-improving (BI) strategies are proposed and compared in terms of their performance. Efficient objective function updates are used, without the need of recalculating the objective function values from scratch: the previous walk weight is used in order to find the weight of the walk obtained after the changes performed during each local search iteration.

3.1. Neighborhood structure

The 2-opt neighborhood is the most commonly used neighborhood structure for the TSP problem and consists in replacing any pair of nonadjacent edges of the current solution by the unique pair of new edges that recreates a cycle.

The following property holds: Let $W = (v_1, \ldots, v_i, \ldots, v_j, \ldots, v_m)$ be any optimal solution of the Steiner TSP. Then, the subpath $(v_i, \ldots, v_j)$ is also a shortest path between the required nodes v_i and v_j. This is true because if this subpath was not the shortest, then W would not be optimal. Therefore, it is not necessary to investigate moves that involve changes in the order in which the nonrequired nodes are visited. Then, the problem amounts to determining the order in which the required nodes should be visited and then finding the shortest path between any pair of consecutive required nodes in the walk.

In consequence, we explore a 2-opt neighborhood for the STSP that is formed by all moves that replace the paths between two pairs of consecutive required nodes in the walk by the two unique pairs of shortest paths that reconnect a closed walk, as illustrated in Figure 2.

3.2. Reduced 2-opt neighborhood

A reduced 2-opt neighborhood can be defined in order to take advantage of the problem structure. In fact, convergence can be faster if only a few, promising moves in the neighborhood are considered.

We implement this idea in the following way. For each required node v, let $I(v)$ be the set formed by all required nodes that are reachable from v by a shortest path that does not visit any other required node. In other words, $I(v)$ represents the set of required nodes that are closer to v, in the sense that they necessarily belong to paths to farther nodes.

Using this auxiliary data structure, we restrict the 2-opt moves to pairs of consecutive required nodes (v_1, v_2) and (w_1, w_2) satisfying the condition that $w_1 \in I(v_1)$; see Figure 2.

3.3. First-improving versus best-improving local search

In a first-improving local search strategy, the algorithm moves from the current solution to any neighbor with a better value for the objective function. In contrast, in a best-improving local search strategy, the algorithm always moves from the current solution to the best of its neighbors. This implies that the time consumed by each iteration will be longer, but also that a better solution will result at the end of the iteration. Both strategies will be considered and compared using the 2-opt and restricted 2-opt neighborhoods.

4. GRASP with path-relinking heuristic

GRASP (which stands for *greedy randomized adaptive search procedures*) is a multistart metaheuristic in which each iteration consists of two main phases: construction and local search. The first phase is the construction of a feasible solution, usually by a greedy randomized algorithm. Once a feasible solution is obtained, its neighborhood is investigated until a local minimum is found during the second phase of local search. The best overall solution is kept as the result. The reader is referred to Resende and Ribeiro (2016) for a complete account of GRASP. Annotated bibliographies of GRASP appeared in Festa and Resende (2009a, 2009b). A recent application of GRASP appeared in Ferone et al. (2016).

We used the adaptive greedy randomized heuristic presented in Section 2 and the local search strategies described in Section 3 to customize a GRASP with path-relinking heuristic for the Steiner TSP.

Path-relinking is an intensification strategy that explores trajectories connecting elite solutions produced by metaheuristics. Path-relinking is usually carried out between two solutions: one is the initial solution S_i, while the other is the guiding solution S_g. A path that connects these solutions is constructed in the search for better solutions. Local search may be applied to the best solution in the path, since there is no guarantee that this solution is locally optimal. In the context of GRASP, path-relinking may be used to connect solutions obtained after the local search step with elite solutions produced during previous iterations, providing a sort of memory mechanism.

More specifically, in the context of the STSP, path-relinking attempts to preserve common characteristics of good walks, that is, common subpaths. As explained below, path-relinking matches the positions of the largest common subpath to the initial and guiding solutions and then swaps the positions of nodes that do not belong to this common subpath.

We first observe that any solution of the STSP has no unique representation as a sequence of the visited required nodes, since any closed walk can start from different nodes and can be traversed in two directions (forward and backward). Therefore, the representation of the initial and guiding solutions must be adjusted to facilitate the operation of relinking them. With this purpose, before applying path-relinking, we adjust the representations of the initial and guiding solutions by detecting the largest common subpath $w_l = (v_i \dots v_j)$ between them.

In our implementation, we choose to detect the largest (or longest) common subpath, instead of the longest common subsequence, in order to prioritize consecutive sequences of nodes in both solutions. This problem is known as the largest (or longest) common substring problem and can be solved in $O(n)$ time and space (Hui, 1992).

The guiding solution S_g and the initial solution S_i are oriented in the same direction according with w_l. Next, the initial nodes of the walks associated with S_g and S_i are made to coincide with the initial node v_i of w_l.

To move from the initial to the guiding solution, path-relinking considers a restricted neighborhood. Each move in this restricted neighborhood involves the swap of two required nodes in the walk corresponding to the current solution that are not in the same positions as they are visited in the guiding solution. In addition, each move should place at least one of the two involved nodes in the appropriate position corresponding to the order in which it will be visited in the guiding solution. After two required nodes are swapped, the shortest paths from their predecessors and to their successors are updated. Since at least one node is placed in the appropriate position of the guiding solution at each iteration, path-relinking will take at most as many iterations as the number of required nodes that were misplaced in the initial solution with respect to the guiding solution.

Algorithm 2 presents the pseudocode of path-relinking from the initial solution S_i to the guiding solution S_g. The current solution S and the best solution S^* are initialized in line 1. The cost f^* of the best solution found by path-relinking is initialized in line 2. The loop in lines 3–10 is performed until the current solution reaches the guiding solution. S' is set to the best solution in the restricted neighborhood of the current solution S in line 4. The best solution S^* found by path-relinking and its cost f^* are updated in lines 6 and 7, respectively, if the new solution S' improved the previous best. The current solution is updated in line 9 and a new path-relinking iteration resumes. The best solution found by path-relinking is returned in line 11.

Algorithm 2. Path-relinking algorithm for STSP

```
1: S, S* ← S_i;
2: f* ← cost(S_i);
3: while S ≠ S_g do
4:       S' ← best solution in the restricted neighborhood of S;
5:       if cost(S') < f* then
6:             S* ← S';
7:             f* ← cost(S');
8:       end if;
9:       S ← S';
10: end while;
11: return S*.
```

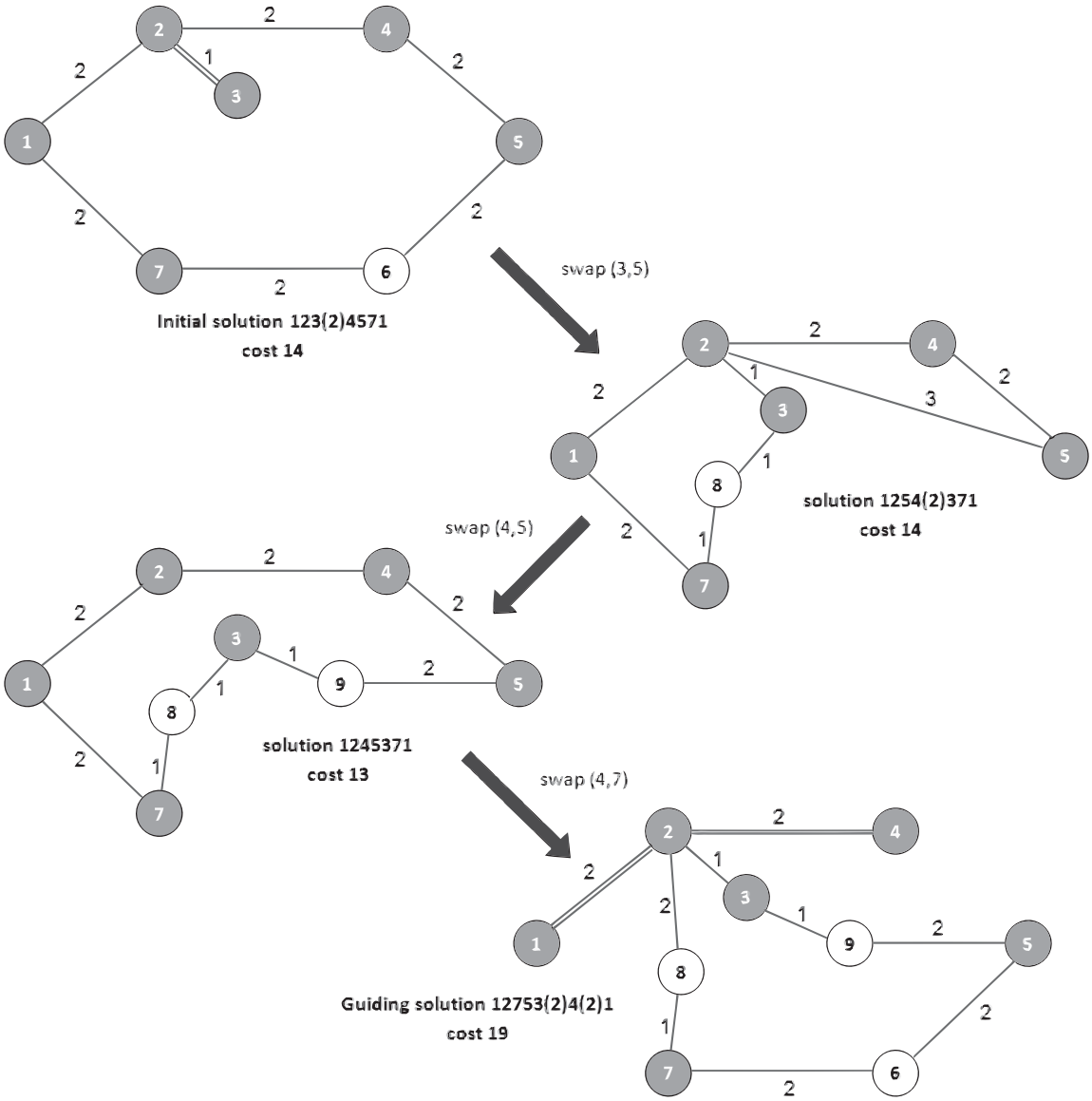

Fig. 3. Three iterations of path-relinking applied to initial and guiding solutions of the Steiner TSP instance in Figure 1: the six required nodes are darkened.

Figure 3 illustrates the application of path-relinking to the Steiner TSP instance described in Figure 1. The graph has six required nodes and three nonrequired nodes. The cost of the initial solution is 14, while that of the guiding solution is 19. The guiding solution is reached after three path-relinking iterations and the cost of the best solution found along them is 13.

The pseudocode in Algorithm 3 summarizes the main steps of the proposed GRASP with path-relinking (GRASP + PR) heuristic, following the same structure proposed in (Resende and Ribeiro,

2016, Section 9.3). The set of elite solutions is initialized in line 1. The loop in lines 2–12 is performed until some stopping criterion is satisfied. An initial solution is built in line 3 by the greedy randomized constructive heuristic described in Section 2. A local search procedure is used in line 4 to improve the solution obtained at the end of the construction phase. Except for the first iteration, when the elite set is still empty, lines 6–9 amount to the application of path-relinking. Line 6 randomly selects an elite solution S' from the elite set $\mathcal{E}$. The representation of solution S is adjusted considering the selected elite solution S'. Backward path-relinking is applied from the initial solution $S_i = S'$ to the guiding solution $S_g = S$. Local search is applied in line 9 to the solution obtained by path-relinking. The elite set $\mathcal{E}$ is updated with the new solution S in line 11. The best elite solution is returned in line 13.

Algorithm 3. GRASP+PR algorithm for STSP

```
 1: E ← ∅;
 2: while stopping criterion not satisfied do
 3:     S ← RandomizedGreedy;
 4:     S ← LocalSearch(S);
 5:     if |E| > 0 then
 6:         Select solution S' at random from E;
 7:         S ← AdjustRepresentation(S, S')
 8:         S ← PathRelinking(S, S');
 9:         S ← LocalSearch(S);
10:     end if;
11:     UpdateEliteSet(S, E);
12: end while;
13: Return the best solution S in E.
```

5. Computational experiments

Several experiments were performed to assess the performance of the algorithms presented above and their variants. The algorithms were implemented in C# programming language and compiled by Roslyn, a reference C# compiler, in an Intel Core i5 machine with a 2.9 GHz processor and 8 GB of random-access memory, running under the Windows 10 operating system.

We considered the same test problems used by Letchford et al. (2013), as well as Letchford and Nasiri (2015) and Zhang et al. (2015, 2016), created by a random generator described in Letchford et al. (2013). This generator was designed to create graphs that resemble real-life road networks. It creates connected sparse graphs and a fraction of required nodes is specified for each instance. In addition to graphs from Letchford et al. (2013), we considered some larger instances with up to 300 nodes. Altogether, ten sparse weighted graphs with 50 to 300 nodes were used to assess the performance of the heuristics. Each graph generated two instances: one with $\lfloor \frac{N}{3} \rfloor$ required nodes and another with $\lfloor \frac{2 \cdot N}{3} \rfloor$ required nodes, where N is the total number of nodes, corresponding to 20 different instances. We observe that individual optimal values for each of these instances have not been previously reported in Letchford et al. (2013).

Table 1
Greedy randomized heuristic: average and best value over 100 runs

	1/3 of required nodes				2/3 of required nodes			
	$\alpha = 0.10$		$\alpha = 0.05$		$\alpha = 0.10$		$\alpha = 0.05$	
Nodes	Average	Best	Average	Best	Average	Best	Average	Best
50	944.63	**813**	947.48	814	1207.51	**979**	1213.34	1031
75	1068.80	**848**	1055.51	**848**	1308.21	1125	1309.49	**1094**
100	1070.02	959	1070.26	**947**	1607.99	1375	1599.44	**1299**
125	1436.82	1275	1425.84	**1228**	1929.31	1627	1911.41	**1623**
150	1410.76	**1151**	1403.36	1230	2110.56	**1875**	2102.40	1899
175	1642.85	**1379**	1615.89	1411	2248.23	1970	2265.40	**1913**
200	1759.45	1512	1765.95	**1445**	2615.95	2369	2614.78	**2354**
225	1826.48	1610	1845.74	**1604**	2817.25	2524	2844.12	**2522**
250	2045.90	**1769**	2041.03	1804	3022.70	2748	2988.79	**2723**
300	2186.87	**1907**	2226.41	1991	3242.32	**2952**	3264.87	2977

5.1. Selecting the quality measure for the RCL

In order to compare the effect of the value of α in the quality-based scheme used to define an RCL, we ran the randomized nearest neighbor constructive heuristic with $\alpha = 0.1$ and $\alpha = 0.05$. The greedy randomized algorithm was applied to all instances. Average and best values over 100 runs are presented in Table 1. As for all tables that follow, the best solution values found for each instance are depicted in boldface.

Although for the instances with one-third of required nodes quality-based constructive heuristic with $\alpha = 0.10$ found slightly more better solutions, the randomized heuristic with $\alpha = 0.05$ found significantly more better solutions for the instances with two-thirds of required nodes. We observe that the use of a better constructive method for building the initial solutions is likely to improve the quality of the solutions produced by the GRASP heuristic.

5.2. Comparing the local search variants

The solutions obtained by the greedy randomized construction algorithm can be refined either by first-improving (FI) or by best-improving (BI) local search strategies. In this experiment, we considered 100 iterations of the pure GRASP (without path-relinking) heuristic using the 2-opt neighborhood structure. The results can be seen in Table 2.

GRASP with best-improving LS clearly outperformed GRASP with the first-improving LS strategy, since substantially better solution values were reached.

5.3. Reduced 2-opt neighborhood

We now address the benefits of using the reduced 2-opt neighborhood, designed specifically for this problem. Preliminary computational experiments have shown that the use of this reduced

Table 2
First-improving versus best-improving local search, 100 GRASP iterations

	1/3 of required nodes		2/3 of required nodes	
Nodes	FI	BI	FI	BI
50	**789**	**789**	979	**978**
75	**830**	**830**	1049	**1035**
100	**931**	934	1280	**1239**
125	1188	**1171**	**1489**	1496
150	1152	**1110**	1699	**1643**
175	1325	**1286**	1759	**1743**
200	1375	**1338**	2051	**1976**
225	1470	**1442**	2202	**2094**
250	1589	**1515**	2313	**2224**
300	1798	**1717**	2603	**2409**

FI, first-improving; BI, best-improving.

Table 3
2-opt versus reduced 2-opt neighborhoods, 100 GRASP iterations

	1/3 of required nodes				2/3 of required nodes			
	2-Opt neighborhood		Reduced 2-opt		2-Opt neighborhood		Reduced 2-opt	
Nodes	Value	Seconds	Value	Seconds	Value	Seconds	Value	Seconds
50	789	0.204	789	0.171	978	0.484	978	0.36
75	830	0.469	830	0.375	**1035**	1.078	1045	0.985
100	934	0.766	**919**	0.672	1239	2.359	**1208**	1.610
125	1171	1.313	**1166**	1.093	1496	4.454	**1469**	2.921
150	**1110**	2.015	1121	1.703	1643	7.468	**1615**	4.703
175	1286	3.016	**1272**	2.281	1743	11.313	**1719**	6.687
200	1338	4.343	**1304**	3.235	1976	17.469	**1925**	9.187
225	1442	5.610	**1415**	4.140	2094	24.891	**2047**	12.828
250	**1515**	7.656	1539	5.578	2224	32.546	**2170**	15.297
300	1717	11.531	**1687**	7.328	2409	55.718	**2285**	26.578

neighborhood led to some target objective function values much faster than the use of the entire 2-opt neighborhood. These empirical observations were explored by the implementation of an alternative VND (variable neighborhood descent) local search procedure. First, only moves in the reduced 2-opt neighborhood are applied. The full neighborhood is explored only after a local minimum is obtained in the reduced 2-opt neighborhood.

Table 3 illustrates the efficiency of the VND approach when compared with the classic use of the full 2-opt neighborhood. It presents the solution values and computation times in seconds for 100 iterations of the pure GRASP (without path-relinking) heuristic using both pure 2-opt neighborhood and VND approach for local search. In both cases, local search was implemented following a best improvement strategy.

The VND local search strategy starting by the reduced 2-opt neighborhood led to the best solutions for 17 out of the 20 test problems. In addition, its computation times have been significantly

Table 4
GRASP versus GRASP+PR, 200 pure GRASP iterations, instances with one-third of required nodes

	Greedy ($\alpha = 0$)	GRASP (200 iterations)		GRASP + PR (same running time)		
Nodes	Value	Value	Seconds	Value	Seconds	Iterations
50	906	**789**	0.359	**789**	0.359	207
75	1181	**830**	0.782	**830**	0.797	191
100	1030	**919**	1.406	**919**	1.406	191
125	1306	**1145**	2.343	1148	2.344	188
150	1429	1121	3.469	**1108**	3.469	191
175	1606	**1272**	4.828	**1272**	4.828	193
200	1595	**1295**	6.500	**1295**	6.500	187
225	1625	1416	8.594	**1401**	8.609	195
250	1956	1531	10.859	**1491**	10.860	186
300	2030	1693	15.750	**1657**	15.797	191

smaller for all instances. As an example, in the case of the largest instance with two-thirds of required nodes, the time taken by 100 GRASP iterations using the VND local search strategy amounted to only 47.8% of the time taken when exclusively the complete 2-opt neighborhood was used. The GRASP heuristic using the VND local search strategy performed better both in terms of solution quality and computation times.

5.4. Probabilistic choice of α

We have already shown in Section 5.1 that although choosing $\alpha = 0.05$ most often leads to better results than $\alpha = 0.10$ for the greedy randomized heuristic proposed for the STSP, for some instances the latter was a better choice than the former. Different probabilistic strategies were considered in Prais and Ribeiro (2000) for the choice of the RCL parameter α, in contrast with the commonly used choice of fixing its value (see also Resende and Ribeiro, 2016). It was shown that a randomly chosen α from a decreasing nonuniform discrete probability distribution offers a good compromise between the running time of the algorithm and the quality of the solutions produced by the randomized heuristic. We relied on this work to sustain that it could be a good choice, in addition to the previously used appropriate value $\alpha = 0.05$, to consider also other higher values for this parameter with smaller probabilities of being chosen at each iteration. Therefore, in the following experiments, we used $\alpha = 0.05$ with a probability 70% and the values $\alpha = 0.10$ and $\alpha = 0.20$ with probabilities 20% and 10%, respectively.

5.5. Path-relinking

In this section, we address the impact of path-relinking in the search process. Tables 4 and 5 show the lengths of the solutions produced by the nearest neighbor adaptive greedy heuristic for the pure GRASP heuristic running for 200 iterations (together with its running time in seconds), and by the GRASP with backward path-relinking running by the same time taken by 200 pure GRASP

Table 5
GRASP versus GRASP + PR, 200 pure GRASP iterations, instances with two-thirds of required nodes

	Greedy ($\alpha = 0$)	GRASP (200 iterations)		GRASP + PR (same running time)		
Nodes	Value	Value	Seconds	Value	Seconds	Iterations
50	1356	**978**	0.750	**978**	0.750	179
75	1204	1031	1.781	**1029**	1.782	183
100	1290	1211	3.484	**1193**	3.485	182
125	1915	1450	6.297	**1427**	6.313	196
150	1847	1615	9.812	**1567**	9.844	184
175	2085	1704	14.640	**1667**	14.641	191
200	2484	1952	19.828	**1895**	19.828	177
225	2549	2024	27.266	**1973**	27.281	191
250	2523	2165	35.078	**2080**	35.172	182
300	2759	2308	62.328	**2219**	62.609	193

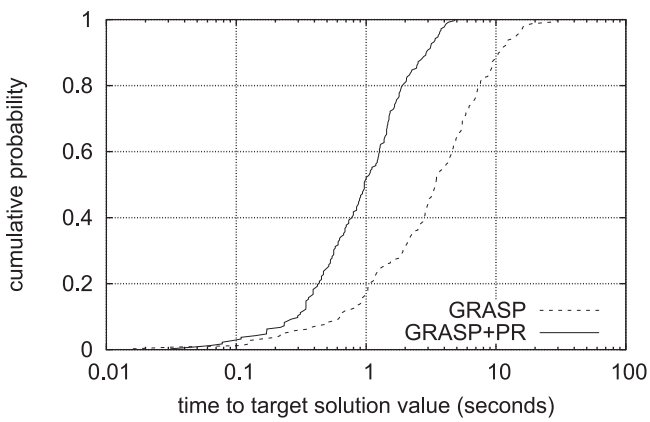

Fig. 4. Time-to-target plot for 200-node instance with one-third of required nodes and target value set to 1330.

iterations. The randomized heuristic used in the construction phase of both pure GRASP and GRASP with path-relinking algorithms uses the probabilistic criterion for the choice of α discussed in Section 5.4. The local search phase of both pure GRASP and GRASP with path-relinking algorithms was implemented using the best improvement VND strategy starting by the reduced 2-opt neighborhood. Path-relinking used elite sets formed by at most ten elements.

Path-relinking considerably improved GRASP performance, leading to better solutions for all but one instance in the same running time and fewer iterations than the pure GRASP heuristic. Time-to-target plots for GRASP and GRASP with path-relinking (GRASP+PR) algorithms for 200-node instances are shown in Figures 4 and 5. The target values are 1330 and 2000 for the instances with one-third and two-thirds of required nodes, respectively. Each algorithm was run 200 times. The plots in these figures provide empirical evidence that algorithm GRASP + PR outperforms GRASP for these instances and target values.

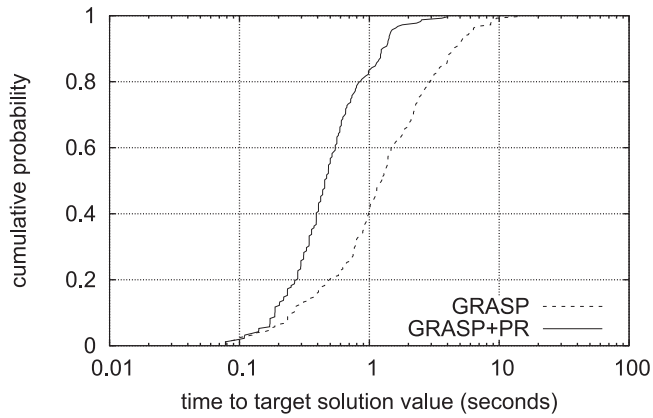

Fig. 5. Time-to-target plot for 200-node instance with two-thirds of required nodes, and target value set to 2000.

Table 6
Restart strategies for 1000 iterations, instances with one-third of required nodes

	No restarts		Restart(100)		Restart(200)	
Nodes	Value	Seconds	Value	Seconds	Value	Seconds
50	**789**	1.75	**789**	1.71	**789**	1.70
75	**830**	4.03	**830**	4.06	**830**	4.04
100	**919**	7.23	**919**	7.20	**919**	7.18
125	**1143**	12.37	**1143**	12.26	**1143**	12.20
150	**1105**	19.07	**1105**	18.51	**1105**	18.32
175	**1272**	26.73	**1272**	27.01	**1272**	27.78
200	**1295**	36.28	**1295**	35.62	**1295**	35.59
225	**1377**	47.03	1384	48.95	1389	47.71
250	1489	62.23	**1487**	59.00	1498	60.50
300	1648	88.51	**1629**	85.95	1645	86.46

6. Restart strategies for GRASP with path-relinking

Resende and Ribeiro (2011) have shown that restart strategies are able to reduce the running time to reach a target solution value for many problems. We apply the same type of restart(κ) strategy in which the elite set is emptied and the heuristic restarted from scratch after κ consecutive iterations have been performed without improvement in the best solution found. We evaluate the performance of the restart strategies for the Steiner TSP for $\kappa = 100$ and $\kappa = 200$. Computational results for the restart strategies for STSP are displayed in Tables 6 and 7, showing that they contribute to find better solutions in the same number of iterations, mainly when the problem size increases.

Figure 6 depicts time-to-target plots for the restart(200) strategy, compared to the original strategy without restarts, for the 200-node instance with two-thirds of required nodes and the target value set to 1900.

As previously observed in Resende and Ribeiro (2011, 2016), the effect of restart strategies can be mainly noted in the longest runs. Considering the 200 runs for the 200-node instance with the

Table 7
Restart strategies for 1000 iterations, instances with two-thirds of required nodes

	No restarts		Restart(100)		Restart(200)	
Nodes	Value	Seconds	Value	Seconds	Value	Seconds
50	**978**	4.03	**978**	3.96	**978**	3.95
75	**1029**	9.46	**1029**	9.45	**1029**	9.26
100	**1193**	18.95	**1193**	18.73	**1193**	18.43
125	1421	33.04	**1417**	33.93	1420	33.09
150	1565	53.39	1564	53.04	**1562**	52.23
175	1657	77.25	1665	76.84	**1652**	76.23
200	1883	105.53	1885	105.68	**1867**	105.20
225	1941	148.96	1953	144.68	**1928**	142.26
250	2054	173.68	2073	175.01	**2035**	176.04
300	2203	304.40	**2192**	310.03	2205	296.76

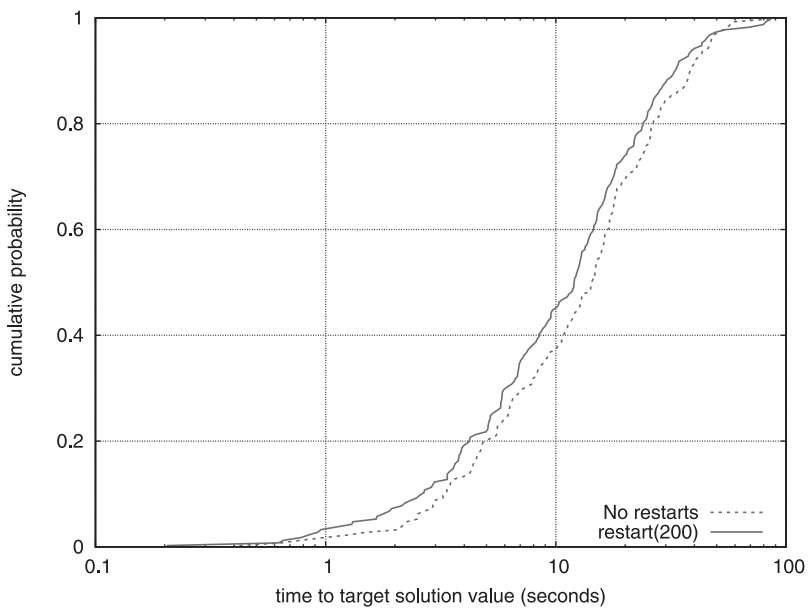

Fig. 6. Time-to-target plot for 200-node instance with two-thirds of required nodes and target value set to 1900.

target value set to 1900, they are associated with the column corresponding to the fourth quartile of Table 8. Entries in this quartile correspond to those in the heavy tails of the runtime distributions. The restart strategies in general do not affect too much the other quartiles of the distributions, which is a desirable characteristic. Compared to the no restart strategy, the restart(200) strategy was able to reduce not only the average running time in the fourth quartile, but also in the third and second quartiles. Consequently, strategy restart(200) performed the best among those tested, with the smallest average running times over the 200 runs.

Table 8
Summary of computational results for each restart strategy for the 200-node instance: 200 independent runs were executed for each strategy. Each run was made to stop when a solution as good as the target solution value 1900 was found. For each strategy, the table shows the distribution of the running times by quartile. For each quartile, the table gives the average running times in seconds over all runs in that quartile. The average running times over the 200 runs are also given for each strategy.

	Average running times in quartile (seconds)				
Strategy	1st	2nd	3rd	4th	Average
Without restarts	3.648	9.915	17.952	37.355	17.218
Restart(100)	2.933	8.466	17.067	37.509	16.494
Restart(200)	2.955	**8.093**	**15.410**	**34.878**	**15.334**

Table 9
Larger instances: description and numerical results for 1000 iterations of the GRASP heuristic using restarts and path-relinking

Instance name	Nodes	Edges	Required nodes	Solution value	Time (seconds)
euroroad05	1174	1417	58	31,571	147.5
euroroad10	1174	1417	117	49,975	354.9
euroroad15	1174	1417	176	58,016	684.6
euroroad20	1174	1417	234	71,967	1162.0
isprouters05	2113	6632	105	20,258	1030.5
isprouters10	2113	6632	211	37,486	3360.7
isprouters15	2113	6632	316	51,571	6850.6
rome05	3353	8870	167	481,048	1893.5
rome10	3353	8870	335	622,491	5196.2
rome15	3353	8870	502	795,535	8494.4

7. Results for larger instances

We have also created a set of ten substantially larger test instances for future benchmarking purposes. These instances were created from tree sparse graphs:

- street network of the city of Rome (Storchi et al., 1999);
- main roads between European cities (Rossi and Ahmed, 2015); and
- network of main Internet service providers at global level (Spring et al., 2002).

For each of the above graphs, we created several different instances with different number of required nodes, randomly chosen from the set of vertices of the graph. The number of nodes, edges, and required nodes, together with the walk lengths and the running times in seconds obtained by the GRASP heuristic using restarts and path-relinking, are presented in Table 9. All data are available from http://www2.ic.uff.br/∼rinterian/instances/allinstances.html.

8. Concluding remarks

In the STSP, one seeks a minimum-weight closed walk that visits a subset of required nodes. Since only a walk is sought, nodes can be visited more than once and edges may be traversed more than once.

In this paper, we developed a GRASP with path-relinking and restarts for solving the STSP. The algorithm used in the construction phase is a randomized extension of the nearest neighbor heuristic for the TSP. A VND strategy exploring a reduced 2-opt neighborhood is used to optimize a best-improving local search scheme. Path-relinking and restart strategies are used to improve the efficiency of the GRASP algorithm.

Extensive computational results for a set of instances previously used in the literature are reported. In addition, we also considered a set of larger test instances derived from real-life graphs. Since neither optimal values nor even upper bounds have been previously reported for these instances, the solutions obtained by the GRASP with path-relinking and restarts heuristic proposed in this work cannot be directly compared to other solutions.

As a step toward avoiding this difficulty and facilitating the research on this problem, we made all test instances considered in this paper available at the URL http://www2.ic.uff.br/~rinterian/instances/allinstances.html together with their best-known solution values. This website will be updated with information provided by other researchers working on this problem with optimal values, lower and upper bounds for these and other benchmarking instances.

References

Borne, S., Mahjoub, A.R., Taktak, R., 2013. A branch-and-cut algorithm for the multiple Steiner TSP with order constraints. *Electronic Notes in Discrete Mathematics* 41, 487–494.

Cornuéjols, G., Fonlupt, J., Naddef, D., 1985. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming* 33, 1–27.

Ferone, D., Festa, P., Resende, M.G.C., 2016. Hybridizations of GRASP with path relinking for the far from most string problem. *International Transactions in Operational Research* 23, 481–506.

Festa, P., Resende, M.G.C., 2009a. An annotated bibliography of GRASP, Part I: Algorithms. *International Transactions in Operational Research* 16, 1–24.

Festa, P., Resende, M.G.C., 2009b. An annotated bibliography of GRASP, Part II: Applications. *International Transactions in Operational Research* 16, 131–172.

Fleischmann, B., 1985. A cutting plane procedure for the travelling salesman problem on road networks. *European Journal of Operational Research* 21, 3, 307–317.

Garey, M.R., Johnson, D.S., 1979. *Computersand Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

Hui, L.C.K., 1992. *Color set size problem with application to string matching. Proceedings of the Third Annual Symposium on Combinatorial Pattern Matching*, Springer-Verlag, London, pp. 230–243.

Letchford, A.N., Nasiri, S.D., 2015. The Steiner travelling salesman problem with correlated costs. *European Journal of Operational Research* 245, 62–69.

Letchford, A.N., Nasiri, S.D., Theis, D.O., 2013. Compact formulations of the Steiner traveling salesman problem and related problems. *European Journal of Operational Research* 228, 83–92.

Prais, M., Ribeiro, C.C., 2000. Parameter variation in GRASP procedures. *Investigación Operativa* 9, 1–20.

Resende, M.G.C., Ribeiro, C.C., 2011. Restart strategies for GRASP with path-relinking heuristics. *Optimization Letters* 5, 467–478.

Resende, M.G.C., Ribeiro, C.C., 2016. *Optimization by GRASP*. Springer, Boston.

Rossi, R.A., Ahmed, N.K., 2015. The network data repository with interactive graph analytics and visualization. Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, pp. 4292–4293.

Salazar-González, J.J., 2003. The Steiner cycle polytope. *European Journal of Operational Research* 147, 3, 671–679.

Spring, N., Mahajan, R., Wetherall, D., 2002. Measuring ISP topologies with Rocketfuel. SIGCOMM'02, Vol. 32, Pittsburgh, pp. 133–145.

Storchi, G., Dell'Olmo, P., Gentili, M., 1999. 9th DIMACS Implementation Challenge—Shortest Paths, Piscataway. Available at http://www.dis.uniroma1.it/challenge9/download.shtml (accessed 31 December 2016).

Zhang, H., Tong, W., Xu, Y., Lin, G., 2015. The Steiner traveling salesman problem with online edge blockages. *European Journal of Operational Research* 243, 30–40.

Zhang, H., Tong, W., Xu, Y., Lin, G., 2016. The Steiner traveling salesman problem with online advanced edge blockages. *Computers & Operations Research* 70, 26–38.