Expert Systems with Applications 39 (2012) 3526–3536

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A GRASP-based approach to the generalized minimum spanning tree problem

Cristiane S. Ferreira[a], Luis Satoru Ochi[a], Víctor Parada[b,*], Eduardo Uchoa[c]

[a] Instituto de Computaçao, Universidade Federal de Fluminense, Rua Passo da Pátria 156, Bloco E, 3° andar, São Domingos, Niterói, RJ, CEP: 24210-240, Brazil
[b] Departamento de Ingeniería Informática, Universidad de Santiago de Chile, Av. Ecuador 3659, Santiago, Chile
[c] Departamento de Engenharia de Produção, Universidade Federal Fluminense, Rua Passo da Patria 156, Bloco E, 4° andar, CEP: 24210-240, Brazil

ARTICLE INFO

Keywords:
GRASP
Generalized minimum spanning tree
Constructive heuristics

ABSTRACT

Given a multipartite graph G the generalized minimum spanning tree problem is to find a tree of minimal cost that includes a vertex from each part. This paper proposes several versions of the GRASP metaheuristic for the problem. The GRASP approach is based on constructive heuristics as well as on additional improvement mechanisms such as path-relinking and iterated local search. Several computational experiments are performed over a set of existing instances. A cut generation algorithm is proposed that is able to find lower bounds, based on a formulation for Steiner's problem in directed graphs. The computational results show that the best versions of the GRASP approach use improvement mechanisms. The solutions found are better than most of the known solutions in the literature and require significantly less computer time. Furthermore, a set of rules is defined for pre-processing the instances, based on the *Bottleneck* distance concept. Using those rules, it was possible to reduce the size of the instances to an average of 14% of the number of edges in relation to the original graphs.

© 2011 Elsevier Ltd. All rights reserved.

1. Introduction

Consider an undirected multipartite graph $G(V,E)$ whose vertices are partitioned into m subsets $\{V_1,V_2,\ldots,V_m\}$, with $|V| = n$, such that $V = V_1 \cup V_2 \cup,\ldots,\cup V_m$ and $V_l \cap V_k = \emptyset \forall\ l,k \in \{1,\ldots,m\}, l \neq k$. G only has edges between vertices of different subsets, and assumes that each edge has an associated nonnegative cost. We define the generalized minimum spanning tree problem (GMSTP) as the problem of finding a tree of minimal cost that spans exactly one vertex from each part. In contrast with the minimum spanning tree problem (MSTP), which has been extensively studied in the literature and solved efficiently in polynomial time (Kruskal, 1956), the GMSTP is classified as NP-hard (Myung, Lee, & Tcha, 1995). A variant of this problem, which has also been extensively studied, consists of finding a tree that spans at least one vertex from each group; this problem is called L-GMSTP.

It is known that when an instance is defined with a complete graph that respects the triangular inequality, the optimum solution of L-GMSTP always has only one vertex in each group. Under this condition, both generalizations of the MSTP have the same optimum solution, and a direct consequence is that GMSTP is a particular case of L-GMSTP (Feremans, Labbe, & Laporte, 2001). It has also been shown that L-GMSTP is a special case of the Steiner problem, inspiring numerical solutions with the use of heuristics and some metaheuristics (Shyu, Yin, Lin, & Haouari, 2003). In the former case, instances with up to 500 nodes and 50 edges have been solved (Dror, Haouari, & Chaouachi, 2000; Feremans, 2001; Ihler, Reich, & Widmayer, 1999).

The GMSTP appears in the field of telecommunications (Myung et al., 1995), in the location of facilities such as distribution centers, warehouses, or stores (Shyu et al., 2003). Dror et al. (2000) describe an interesting application in the field of agricultural watering, while Kansal and Torquato (2001) extend the field of applications to physics. Considering a different objective function that contains prices associated with the nodes, Golden, Raghavan, and Stanojević (2008) study an application on the design of submarine cable networks.

The GMSTP has been addressed by integer programming approaches, giving rise to several mathematical formulations, some of which are equivalent (Feremans, Labbe, & Laporte, 2002; Myung et al., 1995; Pop, 2009).

From an initial solution of the problem generated by a tabu search algorithm, Feremans et al. (2001) used a *branch and cut* algorithm to solve randomly generated instances and adapted instances of the traveling salesman problem (Reinelt, 1991). The algorithm solved all the Euclidian instances with up to 160 vertices and all the instances with random costs with up to 200 vertices, and it also solved 150 of the 169 instances of the TSPLIB that were tested, given a limiting time of two hours of computer time. The results compare favorably with the other extant proposals in the literature, both in terms of the size of the solved problems and of the computation time required. In a second stage of this research,

* Corresponding author. Tel.: +56 2 7180900; fax: +56 5627795892.
E-mail addresses: satoru@ic.uff.br (L.S. Ochi), victor.parada@usach.cl (V. Parada), uchoa@producao.uff.br (E. Uchoa).

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.09.043

Feremans, Lodi, Toth, and Tramotani (2005) modified the algorithm to incorporate Chvátal–Gomory cuts (Letchford & Lodi, 2002). In addition, the algorithm's branching strategy was modified. In this way, they could compare four *branch and cut* versions for both the GMSTP and the L-GMSTP using the same set of instances considered in Dror et al. (2000), and they generated a set of results showing an effective improvement in the behavior of the algorithm.

To make numerical comparisons among different algorithmic approaches, one must obtain the lower bounds of the instances. Golden, Raghavan, and Stanojevic (2005) introduced an algorithm for calculating the lower bounds of the GMSTP to compare some constructive algorithms and two heuristics. The constructive algorithms were iterated versions of the Kruskal's, Prim's and Solin's algorithms. The proposed heuristics consisted of local search and a genetic algorithm. On the average, the distances between the solutions of the local search procedure and of the genetic algorithm with respect to the optimum solution were 0.07% and 0.01%, respectively.

Haouari and Chaouachi (2006) proposed and compared several algorithms for calculating lower bounds of the L-GMSTP, all of them taking as a point of reference the formulations for the Steiner problem. The authors also proposed an adaptation of the PROGRES method, which they had previously developed, and improvements of an existing genetic algorithm (Dror et al., 2000).

Öncan, Cordeau, and Laporte (2008) proposed a procedure for determining lower bounds with a tabu search algorithm for the GMSTP and also presenting new, larger instances of the problem. Those instances were generated from instances of the TSPLIB using the grouping techniques of Fischetti, Gonzalez, and Toth (1995). The lower bounds are calculated on the basis of the linear relaxation of the formulation proposed by Myung et al. (1995). With their proposed method, the authors found much better limits than those of Golden et al. (2005), but they could not find limits for all the instances presented. The computational experiment was performed with 101 instances proposed by the authors; the tabu search algorithm was compared with the two previously described Golden algorithms and with an improved version of a genetic algorithm. In all the instances, the method proposed by Öncan et al. (2008) achieved results that were better than or equal to those of Golden et al. (2005). The described algorithm was also adapted to the L-GMSTP, and that adaptation was compared with the work of Haouari and Chaouachi (2006) on the instances proposed by Dror et al. (2000). The tabu search algorithm did not surpass the results of the genetic algorithm, but the authors consider the results to be promising.

Greedy Randomized Adaptive Search Procedure (GRASP) is a hybrid metaheuristic that in each iteration considers two phases: one for constructing a solution and another one for improving the current solution (Talbi, 2009). The common solution is updated when the improvement phase generates a better solution. The process is repeated until the established number of iterations is completed or until a stopping criterion is satisfied. GRASP has been applied successfully to a large variety of optimization problems (Resende & Ribeiro, 2005). Several elements can be added to the metaheuristic to take advantage of the results of the previous iterations to orient the search. Some of these techniques are path-relinking, data mining, among others. Also, different construction algorithms and local search strategies can be considered, giving rise to a large variety of methods.

Although recent work on the GMSTP constitute a major contribution to its solution, still to find high quality solutions with low computational times, even for known instances, remains a challenge. This paper is concerned with the computational performance of GRASP considering several constructive algorithms, a local search procedure, and two additional mechanisms: path-relinking and iterated local search (*ILS*), in solving GMSTP. To that end, several algorithms are designed (Section 2) with which a computational experiment is performed whose results are described in Section 3. The conclusions are presented in Section 4.

2. Solution procedures

2.1. Constructive algorithms

This section describes five heuristic procedures used in the construction phase of GRASP:

(a) Random construction (C1). One vertex in each group is chosen at random, and the minimum spanning tree on the m chosen vertices is generated by means of Kruskal's algorithm (1956).

(b) Modified Kruskal method (C2 and RC2). The modification proposed initially by Feremans et al. (2001) prevents the insertion of a second node from each group in the tree (C2). We propose a second version of the C2 algorithm, which we call RC2, by considering a random selection of each new edge to be included. For that purpose, at each iteration we define a subset of edges called the restricted candidate list (RCL), before choosing the next edge. Let $c_{\min}$ and $c_{\max}$ be the least and the most costly of the edges that have not yet been evaluated during the construction process; RCL is composed of all the edges whose cost is less than or equal to $c_{\min} + \alpha(c_{\max} - c_{\min})$, where α is in the interval $[0,1]$ and indicates the level of greediness that is used for choosing the edges. Note that if $\alpha = 0$, the algorithm proceeds in the same way as the adaptation of Kruskal's method.

(c) Prim's modified method (C3 and RC3). The adaptation of Prim's algorithm (C3) does not differ much from the original algorithm, with the simple modification that now the procedure is applied to groups instead of to vertices. We define the distance between two groups as the cost of the shortest edge that joins the vertices of both groups. At first, the algorithm gives a partial solution T composed of merely one group V_i, and at each step a new edge is inserted into T, adding to the solution a new group V_j not yet included. As in the construction of RC2, we generate RC3. The selection of the vertices in each iteration is also made based on the restricted candidate list, which is constructed in the same way as in RC2. In this case $c_{\min}$ and $c_{\max}$ are the shortest and longest distance between T and the vertices of the groups that have not been covered yet during the construction process.

(d) Heuristics of the average distance (C4 and RC4). This heuristic procedure chooses a vertex $\gamma[i] \in V_i$, $\forall\, i = 1,\ldots,m$, using the average distance from the vertices in V_i to an auxiliary set $S_i \subseteq V/V_i$, namely the set of vertices that are at a maximum distance $D_{\max}$ from the group V_i. The average distance d_v from a vertex v to S_i is obtained as $d_v = \frac{\sum_{u \in S_i} c(u,v)}{|S_i|}$. The vertex $\gamma[i]$ with the smallest value of d_v is chosen. To carry out this process in the initial stage, an order is defined randomly in which the groups V_i will be visited. RC4 is defined, by choosing vertices from RCL at random. In this case, each visited group V_i has RCL composed of all the vertices whose distance $d_v \leqslant d_{\min} + \alpha(d_{\max} - d_{\min})$, where $d_{\min}$ and $d_{\max}$ are the shortest and the longest average distance from the vertices of V_i, respectively. Note that if $\alpha = 1$, the behavior of C4 is identical to that of C1.

(e) Clustering method (RC5). The heuristic C5 consists of the initial clustering of the groups of nodes into n_s disjoint subsets, considered as initial seeds. Each non-seed group is associated with the nearest seed group in order to include each group in a cluster. The value of n_s is selected in two ways: choosing the most distant groups (C5) and randomly (RC5). In each cluster of groups, the constructive heuristic C2 is applied locally, determining a tree connecting the groups in the cluster. Then a minimum spanning tree is constructed, considering the nodes chosen in the previous stage as fixed. To construct the initial clusters in C5, the distance between two groups is taken to be the cost of the shortest edge that joins them. In the construction process, a group θ_1 is first chosen randomly among the m groups of the graph; the second group θ_2 is chosen as the most distant from θ_1; the third group is that which is the most distant from $\{\theta_1$ and $\theta_2\}$ and so on.

2.2. Local search

Starting from an initial solution T, a set of neighboring solutions T' is generated and the best of the neighboring solutions is selected, following the method proposed by Golden et al. (2005). The procedure includes the following stages:

(i) The order in which the groups are visited is defined randomly.
(ii) For the visited group V_i, all the neighbors of T in relation to V_i are determined. The current solution is replaced by the best neighbor T' if it is better than T.
(iii) Steps *i* and *ii* are repeated until the m groups have been visited without improvement in the solution.

Let γ be the vector of vertices associated with solution T, and γ' the vector associated with a neighboring solution T'; the neighborhood of T in relation to group V_i is composed of all solutions T' such that $\gamma'[i] \neq \gamma[i]$ and $\gamma'[j] = \gamma[j]$, $\forall j \in \{1, \ldots, m\}/i$; therefore, in relation to the group V_i, the current solution T has $|V_i| - 1$ neighboring solutions. Each solution T' is constructed from the minimum spanning tree that covers the vertices at γ'. Algorithm 1 describes the procedure in pseudo-code. The function *define-order*() defines the order in which the groups will be visited. At each iteration, the *next-group*() function returns the next group to be visited according to the pre-established order. The neighbors of T are calculated by the *neighbor*() function, which returns a solution that differs from T only by vertex v according to the neighborhood model.

Algorithm 1. Local search

```
Define-order();
Iter ← 1;
While iter ⩽ m do
  V_i ← next-group();
  For all v ∈ V_i do
    T' ← neighbor(T, v);
    If c(T') < c(T) then T ← T'; Iter ← 1;
  iter ← iter + 1;
Return T';
```

2.3. Path-relinking

The path-relinking mechanism was proposed originally for tabu search or *scatter search* (Glover, Laguna, & Marti, 2000), and its objective is to find intermediate solutions between two good solutions. From a base solution T_0 and a guiding solution T_f for GMSTP, we construct a path of intermediate solutions by replacing vertices of T_0 by those of T_f, until a solution T_{f-1} is reached after $f - 1$ steps. Typically, the GRASP versions that use path-relinking consider elite set ES that contains the |ES| best solutions found during the execution. The set is updated when the solution generated by the local search is better than the worst solution in ES. If the algorithm reaches $It_{\max}$ iterations without updating the elite set, then all the solutions of this set are removed, except the best one.

Path-relinking is activated after the $It_{\min}$ iterations, if the solution T resulting from the local search is at most $p\%$ worse than the best solution in the elite set. The procedure is applied between the base solution and the most different solution in ES. As an intensification strategy, we select p to be proportional to the number of iterations without updating ES, i.e., the harder it is to find a good solution, the more probable it is that path-relinking will occur.

2.4. Iterated local search

The basic idea of the iterated local search is to apply perturbations to the current solution with the purpose of escaping from a local optimum. Den Besten, Stützle, and Dorigo (2001), mention four basic components of an *ILS*: an initial solution, a local search procedure, a perturbation strategy, and a stopping criterion. A perturbation is applied to the current solution and then a local search is made on the perturbed solution, updating the solution when convenient. The process is repeated until a stopping criterion is reached. An appropriate choice of the size of the perturbation is essential for the proper operation of the procedure proposed in Algorithm 2.

Algorithm 2. *ILS*

```
Define-order();
Iter ← 1;
While iter ⩽ m do
  V_i ← next-group();
  T' ← perturbation(T, V_i, D_max);
  T'' ← local_search(T');
If c(T'') < c(T) then T ← T''; Iter ← 0;
iter ← iter + 1;
Return T';
```

We propose a perturbation that selects one vertex $v \in \{V_i \setminus \gamma[i]\}$ at random to be incorporated into a solution. The same is applied to all those groups that are found at a distance $D_{\max}$ from V_i. The solution arising from that perturbation goes through a local search process, giving rise to a new solution T'. The procedure stops when a solution T' is found whose cost is less than that of the original solution. Algorithm 1 is used as the local search procedure.

2.5. GRASP procedure

We have designed several versions of GRASP with the purpose of verifying the contribution of the additional incorporated mechanisms into the quality of the solutions. The versions differ from one another in the use or non-use of path-relinking, of *ILS*, and in the constructive algorithm used.

RC4 is the procedure that achieved the best results in preliminary tests compared to the other heuristics, so some versions of GRASP use only this procedure in the construction phase. On the other hand, other versions of GRASP use the C1, RC2, RC3, RC4 and RC5 procedures. In these versions, each constructive algorithm is allotted a number of iterations proportional to its performance. Therefore, each constructive heuristic i is executed by a number β_i of consecutive GRASP iterations. If during the β_i iterations of the heuristic the elite set is updated, the value of β_i is increased by one unit. In this way, during the execution there is a tendency for those heuristics that deliver the best solutions to be executed over more iterations. In those versions that use path-relinking,

Table 1
Proposed GRASP algorithms.

Version	Constructive algorithms	Uses path-relinking	With *ILS*
G1	RC4	NO	NO
G2	RC4	YES	NO
G3	RC4	YES	YES
G4	All	NO	NO
G5	All	YES	NO
G6	All	YES	YES

the value of β_i is restarted for all the heuristics every time the elite set is restarted. Table 1 presents the six versions of GRASP that were studied.

At the beginning of the execution, each constructive algorithm generates a solution s_0 to which a local search is applied $m/10$ times. Since the search starts from a randomly constructed group permutation, different solutions can be generated from T_0.

As an illustration, the pseudo-code of G3 is described in Algorithm 3. Each iteration is essentially composed of three phases: construction (by means of the RC2 heuristic), improvement (local search), and path-relinking. After the local search, the *update*() function verifies whether T can be part of ES. Path-relinking occurs after $It_{\min}$ iterations have gone by, provided the quality of s in relation to the ES condition is satisfied. After the algorithm has carried out $It_{\max}$ iterations without updating, an *ILS* is applied to every solution of ES by means of *ils*(). The *restart*(ES) function removes all the solutions of the elite set when this has not been updated during the *ILS* process.

Algorithm 3. GRASP G3

```
$Iter \leftarrow 1$;
While no condition_stopped() do
  $T_0 \leftarrow$ constructs-solution(RC4);
  $T \leftarrow$ local_search($T_0$);
  If updates($ES, T$) then $Iter \leftarrow 0$;
  If ($iter > It_{\min}$) and (minimum_quality ($ES, T$)) then
    Path-relinking ($ES, T$);
  If $iter > It_{\max}$ then
    For all $T \in ES$ do ils($T$);
    restart($ES$);
  $Iter \leftarrow Iter + 1$;
Return better-solution($ES$);
```

3. Computational experiments

3.1. Instances

The instances of the experiment were divided into two groups:

Group 1. It consists of 169 instances $48 \leqslant |V| \leqslant 226$ (Feremans, Labbe, & Laporte, 2004; Fischetti et al., 1995), of which 150 instances have the optimum known value, and for only 19 instances, the lower bound is known (Öncan et al., 2008).

Group 2. It consists of 101 instances $229 \leqslant |V| \leqslant 783$ generated by Öncan et al. (2008). No lower bounds are known for these instances. They generated the instances from the TSPLIB by following two vertex grouping techniques: *Cluster Centering* and *Grid Clusterization*. In the former, the vertices were grouped into $m = \lceil |V|/5 \rceil$ groups. The second technique was applied in those instances where the coordinates were known. The Cartesian plane is divided by generating a mesh in which each cell corresponds to a group of vertices. The groups contain at least one and at most $|V|/\mu$ vertices, where μ denotes the approximate number of vertices per group. Table 2 shows the details of the instances.

Table 2
Number of instances generated by each grouping technique.

Group	Cluster centering	Grid Clusterization				Total
		$\mu = 3$	$\mu = 5$	$\mu = 7$	$\mu = 10$	
1	36	32	32	32	32	169
2	21	20	20	20	20	101
Total	57	52	52	52	52	270

3.2. Pre-processing of the instances

The objective of the pre-processing of the instances is to decrease their size by removing the number of edges that have been shown not to be part of at least one optimum solution. To this end, the *Bottleneck* Distance (Uchoa, 2006) is taken as a basic concept. Let $P(u,v)$ be the set of all the paths from u to v composed only of vertices present in the optimum solution, and let $C(P)$ be the cost of the most costly edge of path P; then we define for the GMSTP the distance $B(u,v)^{-(u,v)}$ between two nodes u and v without considering the path composed only for the edge (u,v) as follows:

$$B(u,v)^{-(u,v)} = \min\{C(P)|P \in P(u,v) \notin P\}. \tag{1}$$

Theorem 1. *Given an instance of the GMSTP,* $G = (V,E)$, *whose optimum solution is a minimum spanning tree* $T = (V',E')$, *if* $B(u,v)^{-(u,v)} \leqslant c(u,v)$ *then* $(u,v) \notin E'$.

Proof. Assume that $B(u,v)^{-(u,v)} \leqslant c_{(u,v)}$ and that $(u,v) \in E'$. Since $B(u,v)^{-(u,v)} \leqslant c_{(u,v)}$, there is an edge (w,z) in a path between u and v such that $c_{(w,z)} \leqslant c_{(u,v)}$ and $w, z \in V'$. Therefore, replacing (u,v) by (w,z) in E', we get a tree with a lower cost than T, which is a contradiction. Therefore, if $B(u,v)^{-(u,v)} \leqslant c_{(u,v)}$, then $(u,v) \notin T$. □

Following this result, the removal of edges (u,v) was performed by confirming the existence of at least one path $P \in P(u,v)$ such that $C(P) \leqslant c_{(u,v)}$. If P exists, then (u,v) is redundant because $B(u,v)^{-(u,v)} \leqslant C(P) \leqslant c_{(u,v)}$.

The computation time for the detection of P increases exponentially with the number k of intermediate groups between u and v. For that reason, our reduction algorithm considers only those cases in which $k = 1$ and $k = 2$. First, we apply the rules to $k = 1$. Each edge (u,v) is redundant if there is a group V_i, $i \in \{1,\ldots,m\}$ in which all the edges (u,v_i) and (v_i,v) cost at most $c_{(u,v)}$ $\forall$ $v_i \in V_i$. After reducing the instances with the first phase, the rules associated with $k = 2$ are applied to the remainder. In this stage, an edge (u,v) is said to be redundant if there are two groups V_i and V_j such that $c_{(u,v)}$ is greater than or equal to the cost of each edge contained in $\{(u,v_i), (v_i,v_j), (v_j,v)\}$ $\forall$ $v_i \in V_i$ and $v_j \in V_j$.

The reduction tests were applied to all the instances considered in this paper, and they led to the reduction of the original number of edges by an average of 85.29%. The application of the rules considering $k = 1$ resulted in the removal of 84.73% of the edges, while the application of the following stage with $k = 2$ led to the removal of 0.56% of the original number of edges. Of the 270 instances, the number of edges in 115 instances was reduced to less than 10% of the original number, and only 13 instances retained more than 40% of their edges. The execution time for the pre-processing step was 2.04 s on average, and reached a maximum of 20.45 s.

Fig. 1 shows an example of the impact of the application of the reduction test to the instance 95gil262, whose vertices had been

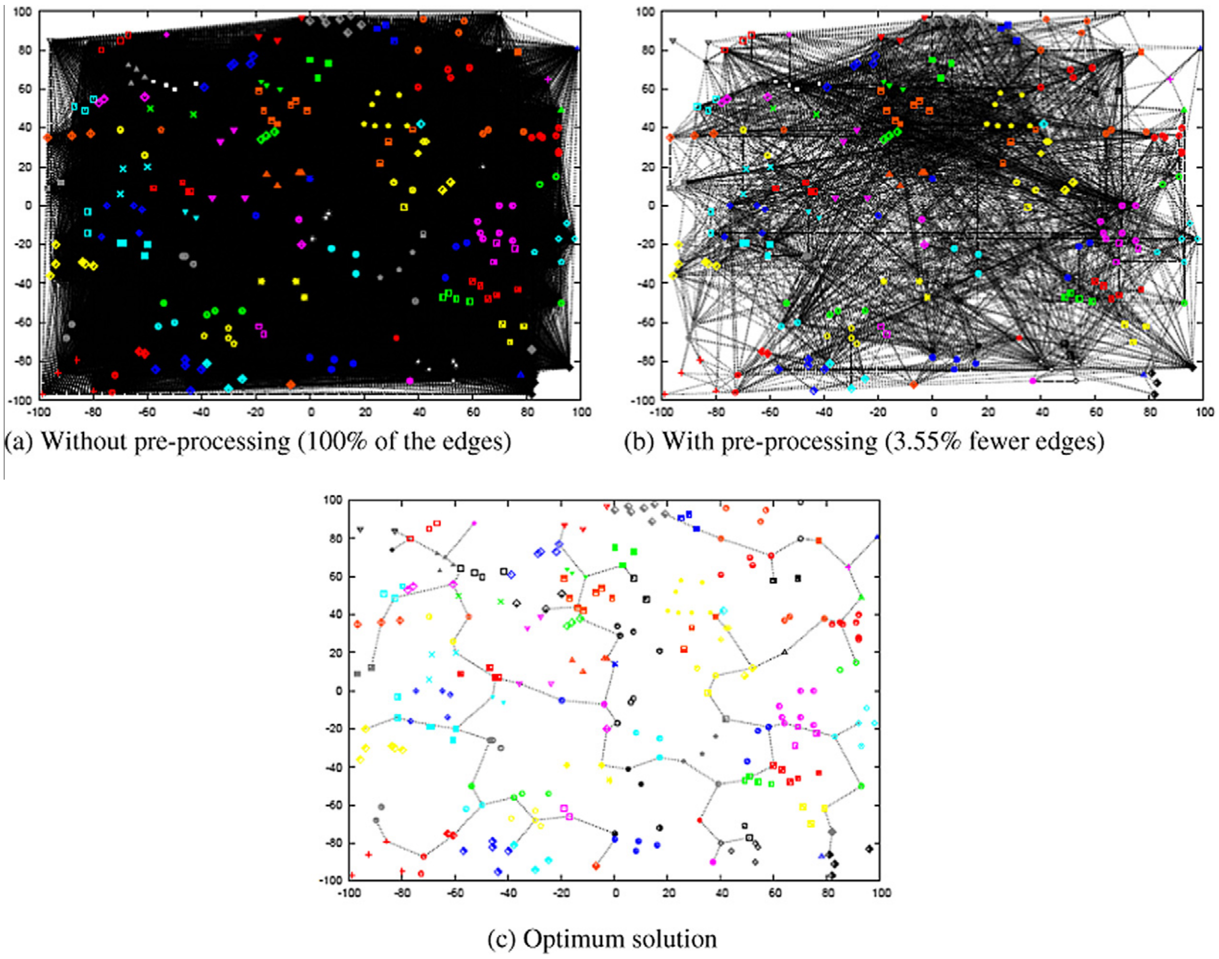


Fig. 1. Instance 95gil262 generated by *Grid Clusterization* with $\mu = 3$.

grouped by the *Grid Clusterization* technique with $\mu = 3$. Fig. 1(a) and (b) represent the instances before and after pre-processing, respectively. In this case the pre-processing led to a 3.55% reduction in the original number of edges. Fig. 1c presents the optimum solution for this instance.

3.3. Obtaining lower bounds

To obtain the lower bounds of an instance of the GMSTP problems, an algorithm is used for generating cuts by solving Steiner's problem in directed graphs represented as the *Directed Cut* formulation (2a), (2b), where x_a is a binary variable associated with arc a (Duin, Volgenant, & Voss, 2004; Haouari & Chaouachi, 2006). The constraints guarantee that for every cut $[W, V/W]$ there is at least one edge in the solution:

$$\text{Minimize} \quad \sum_{a \in E} c_a x_a, \tag{2a}$$

$$\text{Subject to}: \quad \sum_{a(u,v): v \in W} x_a \geq 1; \quad \forall\ W; \quad x_a \in \{0, 1\}. \tag{2b}$$

A solution x is feasible if and only if for all W, at least one edge of the cut $[W, V/W]$ is present in the solution. In numerical terms, the number of sets W grows exponentially as a function of $|V|$, hindering the solution of the linear relaxation of the problem; for that reason an algorithm for the generation of cuts was implemented by solving the linear problems only with a subset of the constraints, and then searching for unsatisfied constraints in the linear solution generated. First the linear problem π is solved considering only a trivial set of constraints, getting a solution x_0. Then a separation problem is solved by looking for constraints not present in π that are unsatisfied in the solution x_0. Such constraints are included in π and the linear problem is solved again, giving rise to a new solution x_1. This procedure is repeated until no more constraints are unsatisfied.

The problem of finding a set W that corresponds to the most commonly unsatisfied constraints corresponds to a minimum cut problem, which we solved using the algorithm proposed by Hao and Orlin (1994), that has the advantage of finding all the cuts referring to each terminal vertex in an execution.

In Algorithm 4, the function *solve*() finds the solution of the linear problem. The function *separation*() solves the minimum cut problem generated from x. The set R contains the sets W that correspond to the most commonly unsatisfied cuts. The constraints that correspond to R are added to the linear problem through the *add*() function and the new problem generated is solved. The process is repeated until no more constraints are unsatisfied.

Algorithm 4. Cut generation

```
$x \leftarrow solve\ (\pi)$;
**Repeat**
  $R \leftarrow separation\ (x)$;
  $Add(\pi, R)$
  $x \leftarrow solve\ (\pi)$;
**Until** $|R| = 0$
**Return** $x$;
```

Algorithm 4 was implemented in C^{++}, using the CPLEX tool version 10.1 to solve the linear problem that was compiled with g^{++}

version 4.02. The numerical tests on the pre-processed instances described in the previous section were executed on a Pentium IV 3.2 GHz computer with 2 Gb RAM. For the instances of the first group, the algorithm found the optimum solution of the 150 instances whose optimum values were known, in a maximum time of 170 s. The same values as in Öncan et al. (2008) were found. For 16 of the 19 instances, the gap between the cost of the best known solution and the lower bound presented is zero, which means that the value of the limit corresponds to the value of the optimum solution of the problem. The cut-generation algorithm found lower bounds for 82 out of 101 instances of the second group, but the time needed to get this result is longer than the time for the instances of the first group. In 65 of these instances, the limit found is equivalent to the cost of the best known solution.

3.4. Calibration of parameters

The numerical results presented in this section were obtained on the same computer described in the previous section. To calibrate the parameters of the algorithms we used 10 instances of group 2, selecting two for every type of grouping. The instances were selected for diversity in terms of the number of vertices and of the degree of difficulty.

The value of α in RC2, RC3 and RC4 was defined by means of 100 executions evaluating values in the range $0.05 \leqslant \alpha \leqslant 0.9$. After running the experiment we found that there is no preference for a specific α for the three heuristics; however, the best average results were found when $0.05 \leqslant \alpha \leqslant 0.3$ for RC3 and RC4, so α was selected randomly in $[0.05, 0.3]$. For RC2, the best performance among all the instances was obtained with $\alpha = 0.05$.

To determine $D_{\max}$ in the constructive algorithms C4 and RC4, an experiment was run with $D_{\max} = \{c_m, c_m/2, c_m/3, c_m/4\}$. The best results were obtained with $c_m/3$.

The number of sets considered in the constructive methods C5 and RC5 was determined by means of 100 executions of each instance, testing the values $m/3$, $m/5$, $m/7$ and $m/10$, where m is the number of groups of each instance. The best solutions were obtained with $n_s = m/7$ in almost all cases. Therefore, we set the random selection of $n_s \in [\lceil m/7 \rceil - 2; \lceil m/7 \rceil + 2]$ before each execution of the algorithm.

The parameters $It_{\min}$, $|ES|$, $It_{\max}$ and p were established by executing G5, subject to a maximum execution time.

$|ES|$ was defined by numerical tests with values between 4 and 8. For each value, 30 executions were carried out. The results that used the values 4 and 5 surpassed the others slightly. It was also verified that the smaller the value of $|ES|$, the shorter the execution time of the algorithm. In this way we established $|ES| = 4$. It was also verified that ES begins to produce good solutions after the twentieth iteration, so $It_{\min} = 20$ was set. The number of updates of ES tends to decrease over its execution time; after 50 iterations without updating, ES starts to stabilize and updates occur very rarely; for that reason we set $It_{\max} = 50$.

When the algorithms find good quality solutions easily, path-relinking tends to occur successively, reducing the efficiency of the algorithm. On the other hand, when the average quality of the solution is low, path-relinking is rarely applied. For that reason it was determined that the value of p must be a function of *iter*, the number of iterations without updating ES, establishing that $p = 1 + iter$. In this way, path-relinking is made to occur more frequently when the algorithm has difficulties updating the elite set.

3.5. Comparison of constructive heuristics

In order to study the computational performance of the non-greedy constructive heuristics, an experiment was performed in two stages. In the first stage, each constructive algorithm was executed 100 times for each instance, while in the second, a local search was performed for each of the solutions generated in the previous stage. The experiment was run on the instances of groups 1 and 2. Specifically, the constructive heuristics C1, RC2, RC3, RC4 and RC5 were considered. The results are shown in Table 3. The third column shows the average gap between the cost of the solutions and the best known cost, and the fourth column shows the run times required to run all the instances; for each instance is reported the average over the 100 time the algorithm was executed. Finally, the last column gives the percentage of instances in which each algorithm achieved the best average cost.

RC2 finds the best solution for many of the instances, yielding the lowest average cost as well as the lowest average gap value; this task requires the longest time of all the constructive heuristics. Compared to the constructive algorithms, no difference in behavior or in computational performance was identified among the instances of groups 1 and 2. In general, the constructive heuristics that show good performance in instances with few nodes also perform well with larger instances.

Using the local search, the differences in the computational performance of the algorithms decrease. The constructive heuristics RC2 + LS, RC3 + LS and RC5 + LS had the best costs for a majority of the instances, and the average gap for the RC4 + LS algorithm is slightly higher than the others. The execution times are also close to one another; however, the RC2 + LS and RC3 + LS algorithms require the longest computer times.

In this set of numerical results, it was detected that the grouping of the vertices has more influence on the quality of the solutions than the number of vertices. The gap in the average cost of the generated solutions compared to the best known cost tends to be greater in the instances with the largest number of vertices per group (Fig. 2). In each graph, each point represents one of the 270 instances considered. The *x*-axis represents the average gap of the solutions generated with C1. In Fig. 2(a), the *x*-axis indicates the number of vertices, while in Fig. 2(b) it indicates the number of vertices per group. In Fig. 2(b) the gap tends to grow with the increase in the number of vertices per group, specifically in instances with less than three vertices per group the gap does not exceed

Table 3
Comparison of constructive algorithms.

Heuristic	Average cost	Average gap	Total time (s)	Percentage of instances (%)
C*1*	24509.09	28.47	0.01	0.00
RC2	**21930.4**	**9.86**	**26.83**	**71.11**
RC3	22392.24	12.12	25.46	18.8
RC4	23013.92	16.21	0.10	2.96
RC5	22216.49	11.94	0.07	7.03
C1 + LS	20798.45	2.27	15.97	8.88
RC2 + LS	20788.64	2.26	41.64	29.25
RC3 + LS	20773.00	2.04	41.24	24.81
RC4 + LS	**20769.45**	**1.81**	**15.00**	**29.62**
RC5 + LS	20804.17	2.27	14.56	13.33

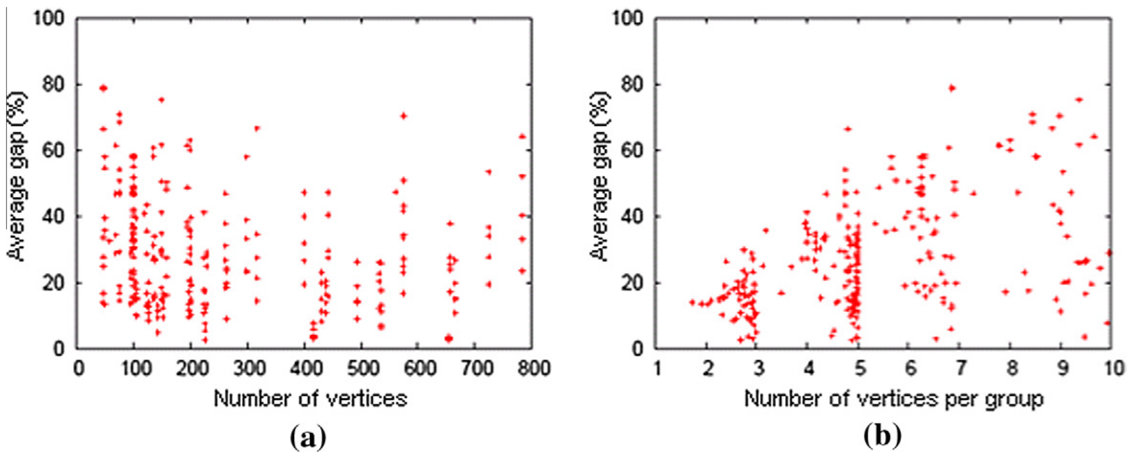


Fig. 2. Relation between the solution characteristics and qualities generated by C1.

40%. We have also tested the other heuristics numerically under this approach and found similar behavior.

3.6. Computational performance of the GRASP versions

To evaluate the computational performance of the different versions of GRASP, we performed an experiment with the instances of groups 1 and 2. Since many of the instances of group 1 have a known optimum solution, the algorithms were executed taking as the stopping criterion either a target cost or a limit on the execution time (300 s). In the instances of group 2, the stopping criterion was only an execution time limit. The results were obtained by executing each algorithm 10 times for each instance (Table 4).

The computational performance of all the algorithms G1 through G6 was similar on the instances of group 1. All the gaps were very close to zero, and all the algorithms succeeded in finding the target costs of the instances whose optimum values were known.

The importance of path-relinking in the algorithms can be seen, in fact, in that in those cases in which path-relinking is not used (G1 and G4) it was more difficult to find the target, even with longer execution times. The algorithms that used path-relinking found the target in all the instances and in all the executions, with the algorithms using the *ILS* being slightly faster. The target was achieved for most of the instances in each of the 10 executions.

For the instances of group 2, a larger difference in the computational performance of the GRASP versions was detected. In this case, the algorithms with the best performance were the versions that applied the iterative search (G3 and G6), which obtained the lowest cost for more than 80% of the instances. These results also confirm the importance of path-relinking, since those algorithms that did not use it (G1 and G4) did not achieve a better cost than the rest of the algorithms for any instance.

3.7. Comparison with results from the literature

Öncan et al. (2008) compared a tabu search algorithm (TS) with the best results from other reports in the literature, using as reference the instances of group 2 (Table 5). Local search (LS) and the genetic algorithms GA1 and GA2 were proposed by Golden et al. (2005). To make this comparison, we considered G6, because it is the one that gave the best results for a greater number of group 2 instances compared to the other versions.

Although the execution time of TS was not the lowest among all the methods, the solutions found present the lowest gap compared to the best known solutions. An important aspect is that this algorithm achieved the best results for all the instances of group 2. For that reason, we compared G6 with TS. As the TS code was not available, we first executed G6 using the same execution time as TS. Note that both TS and G6 can eventually find its best solution before to reach such time. Second, we determined the time needed by G6 to achieve the same solution quality as TS. Although the computer used by Öncan et al. (2008) did not have the same

Table 4
Comparison of GRASP versions for the instances of groups 1 and 2.

Version	Average cost	Average gap (%)	Average time (s)	% Success (%)	Average cost	Average gap (%)	Instances with better cost (%)
	Group 1				Group 2		
G1	14679.29	0.002	6.81	98.2	30321.07	0.244	30.69
G2	14679.02	0.000	0.70	100.0	30292.92	0.028	65.34
G3	14679.02	0.000	0.35	100.0	30288.91	0.014	82.17
G4	14679.33	0.001	5.58	98.8	30315.21	0.209	48.51
G5	14679.02	0.000	0.52	100.0	30290.16	0.032	70.29
G6	14679.02	0.000	0.33	100.0	30289.31	0.017	84.15

Table 5
Comparison of the main algorithms from the literature.

Algorithm	Average cost	Average gap (%)	Time (s)	Instances with better solution (%)
LS	30328.11	0.33	548.14	33.33
GA1	30305.55	0.17	754.41	44.11
GA2	30292.88	0.04	578.12	67.64
TS	30289.49	0.01	736.85	100

Table 6
Comparison of *G6* and *TS* in instances generated by *Cluster Centering*.

Instance	Time (s)	Lower bound	TS		G6	
			Cost	Gap (%)	Cost	Gap (%)
107ali535	683	114289	114303	0.01	114303	0.01
107att532	597	12001	12001	0.00	12001	0.00
99d493	587	16493	16493	0.00	16493	0.00
132d657	1056	19427	19427	0.00	19427	0.00
84fl417	233	7935	7935	0.00	7935	0.00
53gil262	74	887	887	0.00	887	0.00
87gr431	233	–	86885	–	86885	–
134gr666	1365	–	**144756**	–	144773	–
64lin318	130	18471	18471	0.00	18471	0.00
131p654	1045	–	22208	–	**22207**	–
113pa561	702	861	864	0.35	864	0.35
89pcb442	266	19571	19571	0.00	19571	0.00
53pr264	72	21872	21872	0.00	21872	0.00
60pr299	94	20290	20290	0.00	20290	0.00
88pr439	574	51749	51760	0.02	**51749**	0.00
115rat575	762	2168.5	2170	0.07	2170	0.09
157rat783	1916	3009	3017	0.27	**3012**	0.11
80rd400	208	5868	5868	0.00	5868	0.00
107si535	573	12791	12791	0.00	12791	0.00
115u574	517	15027	15037	0.07	**15032**	0.04
145u724	1290	15904	15905	0.01	**15904**	0.00

characteristics as the one used in our experiment, the versions of the different processors are very similar. The results obtained in the executions that used the same computation time as TS are presented in Tables 6–8. In each table, the first column shows the name of the instance, the second corresponds to the time reported to TS, that is, the result of the best run for each instance. The third column shows the lower bound determined by the Algorithm 4. From the fourth to the seventh column, are showed the results obtained by TS and G6.

G6 and TS achieved the same performance in 72 instances, while in 21 instances the average cost of G6 was better, and in 8 instances G6 did not reach the cost obtained by TS. Considering only those instances in which a lower limit to the solution is known, the solutions obtained by G6 are on the average closer to the lower bound than the solutions obtained by TS, and the average gap was 0.016% for G6 and 0.031% for TS. Even in those instances in which G6 did not reach the cost of the solution obtained by TS, it did actually obtain that cost in some of the 10 executions. The average gap between the best solutions obtained by G6 and the lower bound was 0.013%.

It was more difficult for G6 to find the targets for the instances with greater numbers of vertices per group. In instances generated by *Grid Clusterization* with $\mu = 3$, G6 found many solutions better

Table 7.
Comparison of *G6* and *TS* in instances by *Grid Clusterization* with $\mu = 3$ and $\mu = 5$.

Instance	Time (s)	Lower bound	TS		G6	
			Cost	Gap (%)	Cost	Gap (%)
Grid Clusterization with $\mu = 3$						
181ali535	1576		134362		134362	
182att532	1569	15652	15652	0.00	15652	0.00
171d493	1104	20269	20269	0.00	20269	0.00
221d657	3027	25703	**25703**	0.00	25704	0.00
142fl417	681	8353	8353	0.00	8353	0.00
95gil262	204	1255	1255	0.00	1255	0.00
81gr229	67	74792	74792	0.00	74792	0.00
149gr431	1697	–	103844	–	103844	–
224gr666	3105	–	174671	–	**174655**	–
108lin318	206	24092	24092	0.00	24092	0.00
230p654	4127	23547	23553	0.03	**23547**	0.00
156pcb442	810	27513	27513	0.00	27513	0.00
101pr264	102	29199	29199	0.00	29199	0.00
102pr299	399	23096	23096	0.00	23096	0.00
163pr439	932	64953	64953	0.00	64953	0.00
196rat575	2080	2880	2884	0.14	**2880**	0.00
285rat783	7180	4203	4211	0.19	**4203**	0.00
135rd400	548	7632	7632	0.00	7632	0.00
198u574	1340	19696	19696	0.02	19696	0.00
266u724	5201	21739	**21739**	0.1	21747	0.04
Grid Clusterization with $\mu = 5$						
108ali535	632	–	108201	–	108201	–
110att532	641	11896	11896	0.00	11896	0.00
102d493	725	16132	16132	0.00	16132	0.00
137d657	1249	–	19846	–	**19826**	–
93fl417	230	7952	7952	0.00	7952	0.00
63gil262	63	984	984	0.00	984	0.04
47gr229	43	54305	54305	0.00	54305	0.00
87gr431	244	81856	84856	0.00	84856	0.00
139gr666	2192	–	144077	–	144077	–
64lin318	116	17667	17667	0.00	17667	0.00
134p654	1316	22377	22379	0.01	**22377**	0.00
95pcb442	357	19383	19411	0.14	**19383**	0.00
55pr264	67	21351	21351	0.00	21351	0.00
69pr299	105	18582	18582	0.00	18582	0.00
96pr439	587	54230	54230	0.00	54230	0.00
121rat575	705	2014	2018	0.2	**2014**	0.00
169rat783	2424	2823	2832	0.32	**2823**	0.00
81rd400	284	5433.5	**5434**	0.01	5434.8	0.02
127u574	1037	15240	15255	0.1	**15252**	0.08
166u724	2064	15435	15458	0.15	**15435**	0.00

Table 8
Comparison of G6 and TS *Grid Clusterization* in instances with $\mu = 7$ and $\mu = 10$.

Instance	Time (s)	Lower bound	TS		G6	
			Cost	Gap (%)	Cost	Gap (%)
Grid Clusterization with $\mu = 7$						
83ali535	1012	–	94149	–	94149	–
80at532	992	10204	10206	0.02	**10204**	0.00
78d493	212	14152	14152	0.00	14152	0.00
96d657	440	–	**16542**	–	16542.6	–
61fl417	99	7446	7446	0.00	7446	0.00
49gil262	63	802	**802**	0.00	802.6	0.07
34gr229	13	45989	46049	0.13	46049	0.13
64gr431	111	–	71415	–	71415	–
96gr666	452	–	119271	–	119271	–
49lin318	44	14909	14909	0.00	14909	0.00
100p654	462	–	21770	–	21770	–
64pcb442	117	14639,5	**14644**	0.03	14645.2	0.04
43pr264	28	20438	20438	0.00	20438	0.00
47pr299	37	15238	15238	0.00	15238	0.00
74pr439	140	47101	47101	0.00	47101	0.00
100rat575	601	1734,5	**1735**	0.03	1735.6	0.06
121rat783	850	2228	2230	0.09	**2229**	0.04
64rd400	97	4576	4581	0.11	4581	0.11
92u574	304	12186	12186	0.00	12186	0.00
11u724	710	–	**13101**	–	13109	–
Grid Clusterization with $\mu = 10$						
57ali535	213	–	73359	–	73359	–
57att532	187	8494,84	8497	0.03	8497	0.03
52d493	180	11121	11121	0.00	11121	0.00
73d657	386	–	13495	–	13495	–
42fl417	75	6986	6986	0.00	6986	0.00
36gil262	33	639	639	0.00	639	0.00
23gr229	28	39793	39793	0.00	39793	0.00
52gr431	110	62722	62722	0.00	62722	0.00
70gr666	315	–	97198	–	97198	–
36lin318	31	10119	10119	0.00	10119	0.00
69p654	274	20736	20739	0.00	20739	0.00
48pcb442	135	11941	11941	0.00	11941	0.00
27pr264	28	16546	16546	0.00	16546	0.00
35pr299	63	11624	11624	0.00	11624	0.00
48pr439	117	40518	40518	0.00	40518	0.00
64rat575	284	1235	1235	0.00	1235	0.00
81rat783	468	–	1682	–	1682	–
49rd400	99	3825	3825	0.00	3825	0.00
64u574	234	9755	9755	0.00	9755	0.00
80u724	365	–	9608	–	9608	–

Table 9
G6 and TS algorithms.

Algorithm	Average cost	Gap (%)	Best cost	Best gap (%)	Average time (s)
TS	30289.49	0.0112	30289.49	0.0112	736.85
G6	30288.65	0.0062	30287.93	0.0007	59.13

than TS; in several of the instances generated with $\mu = 7$ the execution time required by TS was not sufficient for GRASP to find the best known solution in all the executions.

In the second part of the experiment, G6 was executed with a maximum time of 3 h, with the objective of achieving the same solution quality obtained by TS (Table 9). The fourth and fifth columns present the best cost and gap of the 10 executions. These results verify that the G6 algorithm succeeds in finding the same solutions as the TS algorithm.

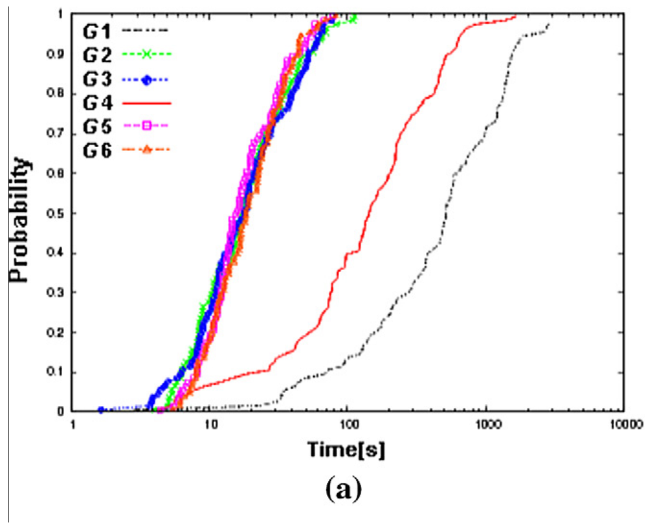
3.8. Probabilistic analysis of the algorithms

To examine the robustness of the proposed algorithms, a computational experiment has been performed to generate the distribution probability function of obtaining a solution in a given computation time (Aiex, Resende, & Ribeiro, 2002). As an example, we will illustrate the results of this experiment for the 107ali535 instance.

To carry out this experiment, each algorithm was used with a stopping criterion that corresponds to finding a target or an execution time limit. A total of 100 executions were carried out for each version of GRASP; the time limit considered was 3600 s. Two targets were identified, one easy and one difficult. The former was the cost of the solution found by LS (Golden et al., 2005), while the latter was the cost of the best known solution for the corresponding instance.

To obtain the probability distribution function, the ith shortest execution time t_i among the 100 executions of the algorithm was associated with the probability $p_i = (i - 0.5)/100$, producing a set of points of the form (t_i, p_i) for $i = 1, \ldots, 100$.

Figs. 3(a) and (b) show the probability distribution functions for both targets, easy and difficult respectively. In both graphs, the



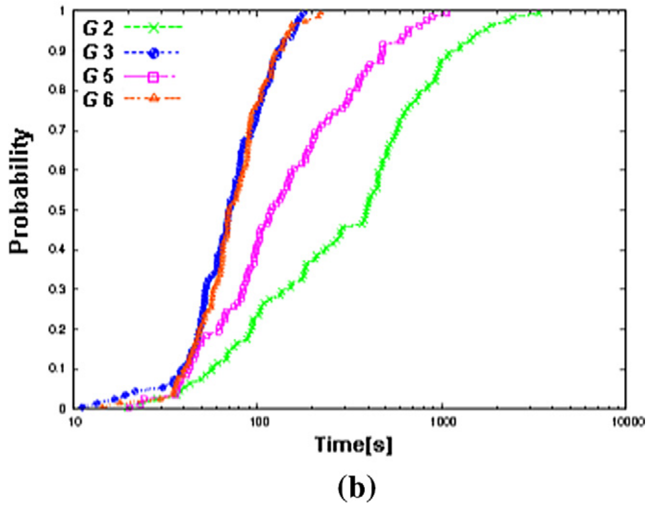


Fig. 3. Probability distribution function, instance 107ali535.

curve further to the left indicates that the algorithm converges more rapidly toward the objective value. In the first case, it is seen that algorithms G1 and G4, which do not apply path-relinking, show the worst performance, and G2, G3, G5 and G6, which do apply it, have similar performance. In the case of the difficult target, the algorithms that use *ILS* (G3 and G6) have better performance. In general, it is seen that the convergence of the algorithm ensures a good quality solution, but it is more effective with the G2, G3, G5 and G6 proposals.

4. Conclusions

This paper describes, and evaluates computationally, several algorithmic approaches to solving the generalized minimum spanning tree problem based on GRASP. Several constructive heuristics are proposed to be incorporated into GRASP. A path-relinking mechanism and an *ILS* were implemented. The computational performance of each constructive heuristic was studied, comparing the results by solving a set of instances from the literature. From those experiments, six GRASP versions are proposed, and evaluated with the same set of instances. The experiments indicate that the use of additional mechanisms, namely path-relinking and *ILS*, significantly improves the performance of the GRASP heuristic, regardless of the constructive heuristic used. The versions that do not include any of these mechanisms yield worse results than the others in practically all the experiments performed. Also, the GRASP versions that consider the *ILS* find the best costs for more than 80% of the tested instances.

The use of more than one constructive heuristic tends to cause the algorithms to find good solutions for a greater number of instances. Even though the execution time is slightly longer, those versions show more uniform behavior than the others. Compared to existing algorithms in the literature, adaptive GRASP using path-relinking and *ILS* is capable of finding better costs for 22 of the 101 instances, while in only 8 instances its average cost does not achieve the best result from the literature.

This paper also proposes a cut generation algorithm for calculating lower bounds based on a formulation of Steiner's problem in directed graphs. The limit found by the algorithm corresponds to the optimum value of all the instances with known optimum value. On the other hand, the algorithm found the limit for 82% of the 101 instances for which a lower bound is not known.

An adaptation of the *bottleneck* distance concept used in the pre-processing of instances of Steiner's problem in graphs to be used in the GMSTP is also presented in this work. Thanks to this adaptation it was possible to reduce the size of the instances to an average of 14% of their original size.

Acknowledgement

The third author thanks to the Complex Engineering Systems Institute (ICM: P-05-004-F, CONICYT: FBO16)

References

Aiex, R., Resende, M., & Ribeiro, C. (2002). Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics, 8*(3), 343–373.
Den Besten, M., Stützle, T., & Dorigo, M. (2001). Design of iterated local search algorithms: An example application to the single machine total weighted tardiness problem. In *Proceedings of the Evo workshops. Lectures notes in computer science* (vol. 2037, pp. 441–451). Berlin: Springer-Verlag.
Dror, M., Haouari, M., & Chaouachi, J. (2000). Generalized spanning trees. *European Journal of Operational Research, 120*(3), 583–592.
Duin, C., Volgenant, A., & Voss, S. (2004). Solving group Steiner problems as Steiner problems. *European Journal of Operational Research, 154*(1), 323–329.
Feremans, C. (2001). *Generalized spanning trees and extensions*. PhD Thesis. Université Libre de Bruxellas.
Feremans, C., Labbe, M., & Laporte, G. (2001). On generalized minimum spanning trees. *European Journal of Operational Research, 134*(2), 457–458.
Feremans, C., Labbe, M., & Laporte, G. (2002). A comparative analysis of several formulations for the generalized minimum spanning tree problem. *Networks, 39*(1), 29–34.
Feremans, C., Labbe, M., & Laporte, G. (2004). The generalized minimum spanning tree problem: Polyhedral analysis and branch-and-cut algorithm. *Networks, 43*(2), 71–86.
Feremans, C., Lodi, A., Toth, P., & Tramotani, A. (2005). Improving on branch-and-cut algorithms for generalized minimum spanning trees. *Pacific Journal of Optimization, 1*, 491–508.
Fischetti, M., Gonzalez, J. J., & Toth, P. (1995). The symmetric generalized traveling salesman polytope. *Networks, 26*(2), 113–123.
Glover, F., Laguna, M., & Marti, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics, 39*(3), 653–684.
Golden, B., Raghavan, S., & Stanojevic, D. (2005). Heuristic search for the generalized minimum spanning tree problem. *INFORMS Journal on Computing, 17*(3), 290–304.
Golden, B., Raghavan, S., & Stanojević, D. (2008). The prize-collecting generalized minimum spanning tree problem. *Journal of Heuristics, 14*(1), 69–93.
Hao, J., & Orlin, J. (1994). A faster algorithm for finding the minimum cut in a directed graph. *Journal of Algorithms, 17*(3), 424–446.
Haouari, M., & Chaouachi, J. (2006). Upper and lower bounding strategies for the generalized minimum spanning tree problem. *European Journal of Operational Research, 171*(2), 632–647.
Ihler, E., Reich, G., & Widmayer, P. (1999). Class Steiner trees and VLSI-design. *Discrete Applied Mathematics, 90*(1–3), 173–194.
Kansal, A., & Torquato, S. (2001). Globally and locally minimal weight spanning tree networks. *Physica A: Statistical Mechanics and its Applications, 301*(1–4), 601–619.
Kruskal, J. B. Jr., (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 48–50.
Letchford, A., & Lodi, A. (2002). Strengthening Chvátal–Gomory cuts and Gomory fractional cuts. *Operations Research Letters, 30*(2), 74–82.

Myung, Y., Lee, C., & Tcha, D. (1995). On the generalized minimum spanning tree problem. *Networks, 26*(4), 231–241.

Öncan, T., Cordeau, J., & Laporte, G. (2008). A tabu search heuristic for the generalized minimum spanning tree problem. *European Journal of Operational Research, 191*(2), 306–319.

Pop, P. (2009). A survey of different integer programming formulations of the generalized minimum spanning tree problem. *Carpathian Journal of Mathematics, 25*(1), 104–118.

Reinelt, G. (1991). TSPLIB – A traveling salesman problem library. *INFORMS Journal on Computing, 3*(4), 376–384.

Resende, M., & Ribeiro, C. (2005). GRASP with path-relinking: Recent advances and applications. In *Metaheuristics: Progress as real problem solvers* (pp. 29–63). New York: Springer.

Shyu, S., Yin, P., Lin, B., & Haouari, M. (2003). Ant-tree: An ant colony optimization approach to the generalized minimum spanning tree problem. *Journal of Experimental & Theoretical Artificial Intelligence, 15*(1), 103–112.

Talbi, E. (2009). *Metaheuristics: From design to implementation*. Hoboken, NJ: Wiley.

Uchoa, E. (2006). Reduction tests for the prize-collecting Steiner problem. *Operations Research Letters, 34*(4), 437–444.