MOST – Multiple Objective Spanning Trees Repository Project[♣]

(Technical Report)

Pedro Cardoso[♠] and Mário Jesus[♡] and Alberto Márquez[♢]

Abstract

This article presents the Multiple Objective Spanning Trees repository – MOST – Project. As the name suggests, the MOST Project intends to maintain a repository of tests for the MOST related problems, mainly addressing real-life situations.

MOST is motivated by the scarcity of repositories for the problems in the referred field. This entails difficulty in test and classify the proposed algorithms based on their performance.

At present, the problems are expressed as networks classified according to their intrinsic properties, namely nodes, edges, and weights. Different generators were developed, which upon combinations, allow a large number of problems with distinct features like, large sets of solutions, concave fronts, and fronts with gaps.

This repository is open to the general participation of the interested communities, mainly in terms of original contributions, approximation improvements, and problem variations (besides the unconstrained cases).

MOST Project can be found at http://est.ualg.pt/adec/csc/most.

Keywords: Multiobjective optimization, minimum cost spanning tree, libraries.

[*]

♣ – The authors thank the *Centro de Simulação e Cálculo* (CsC) from Department of Civil Engineering, EST, University of Algarve, for the computational resources made available.

♠ – Department of Electrical Engineering
EST, University of Algarve, Faro, Portugal
pcardoso@ualg.pt
♡ – Department of Civil Engineering
EST, University of Algarve, Faro, Portugal
mjesus@ualg.pt
♢ – Department of Applied Math I
University of Seville, Seville, Spain almar@us.es

1 Introduction

The use of heuristics and meta-heuristics in optimization requires that those methods are tested with known problems and solutions to predict their overall performance. The use of large sets of problems reduces the risk of overfitting in the tested methods although, even in an extensive collection, some classes of features may be rare or absent (Gent and Walsh, 1999; Deb et al., 2002).

When we try to focus on some specific problem often a number of difficulties arise, like, the need to find repositories for the problems in context (or possibly similar), the need to know the optimums or quasi-optimums of the problems and, in the majority of the cases, the lack of diversity of the so far studied instances. This situation implies that generally the proposed algorithms are first tested with classical problems, as for example, some libraries of functions for the continuous cases, or the travelling salesman problem for the discrete ones. This proves to be a touchstone to many meta-heuristics (Johnson and McGeoch, 1997; Cirasella et al., 2001).

It should be noted that this lack of choices is even more pertinent in the multiple objective combinatorial optimization, where only a few disperse sets of results are available. Therefore, in this article we propose the Multiple Objective Spanning Tree repository – MOST – Project. This project will serve as an archive for the large number of multidisciplinary applications problems of the multiple objective spanning trees. The objective of the MOST Project is to establish, maintain, and successively improve an intuitive and large set of problems along with their optimal or quasi-optimal solutions, on which different algorithms can quickly be tested and classified relative to their performance, with respect to processing time and accuracy.

Therefore, MOST Project is meant to be a dynamic platform that the interested community can make use of and contribute as well to maintain a helpful repository for the types of problems already mentioned. MOST can be accessed via Internet at http://est.ualg.pt/adec/csc/most.

Relative to the groundwork of the MOST Project, the spanning trees optimization problem is one of the most well-studied areas in the combinatorial optimization, with practical applications in distinct areas like, VLSI layout, electricity, water, telecommunication or traffic networks (Sack and Urrutia, 2000), biology and medicine (such as cancer detection – medical imaging(An et al., 2000)), dependency parsing (Hajic et al., 2005), and also as a basic step in some other graph problems.

In the multiple objective minimum spanning tree problems, similar to the generalized multiple objective optimization cases, improving one of the weights of a good solution will worsen at least one of the others, which implies that there rarely exists an over performing response that is preferable to all others. Therefore, the answer to these problems is a set of trade-off compromises between all concurring weights, resulting in a set of solutions. For example, in a telecommunication network, where the optimization of several problems is based on spanning trees, factors like the maintenance and communication costs, communication delays, reliability, bandwidth, and profitability, compete among themselves (Pinto et al., 2005). This intricacy implies that the multiple objective minimum spanning trees problem is NP-complete (Camerini et al., 1984), and Hamacher and Ruhe proved that it is also $NP-\#$(Hamacher and Ruhe, 1994).

Problems in NP-hard class of complexity have attracted for the attention of several researchers in that they used ever-increasing computational capacities to shape and develop a large number of heuristics and meta-heuristics, like the Genetic Algorithms (Vose, 1999; Aarts and Lenstra, 1997; Deb, 2001), Tabu Search (Glover and Laguna, 1997), Simulated Annealing (Kirkpatrick et al., 1983), and Ant Colony Optimization (Dorigo et al., 1999; Dorigo and Stutzle, 2004). These methods proved to be excellent working tools and are being adopted as first-choice procedures to obtain suitable approximations to many solutions of problems. Although some of these algorithms have the guaranty of convergence through the optimum solutions (see for example (Dorigo and Stutzle, 2004) for the Ant Colony Optimization or (Coello-Coello et al., 2005) for the Simulated Annealing, an Artificial Immune System and a General Evolutionary Algorithm for multiple objective optimization problems), most of the times they cannot exactly predict the quality of the approximations computed within a limited time. Therefore, if with these techniques we look for an extensive problem solver, then we should at least test them in greatest possible number of different problems, to estimate their performance in several effective measures and rank the algorithms' utility as practical tools. This has been mentioned in the beginning of the article: it is exactly here that the MOST Project has to prove its usefulness by providing a variety of problems and their exact solutions or known good approximations.

In the present implementation stage of MOST Project, problems are classified according to three types of generators: nodes, edges, and weights generators. For each of the types, we have defined a set of instances, as follows:

- For the nodes we consider five generators: the Grid Nodes Generator, the Triangular Nodes Generator, the Uniform Nodes Generator, the Normal Nodes Generators, and the Clusters Nodes Generator.
- For the edges, six generators are considered: the Grid Edges Generator, the Delaunay Edges Generator, the k-Convex Edges Generator, the Complete Edges Generator, the Voronoi Edges Generator, and the Clusters Edges Generator.
- Finally, for the edge weights, we consider three generators: the Random Weights Generator, the ρ-Correlated Weights Generator, and the Concave Weights Generator.

Therefore, this article is organized as follows. In the next section, some preliminary concepts about multiple objective optimization and networks are introduced. Section 3 describes the methods to generate the networks, namely: the nodes, the edges, and the weights generators. In sections 4 and 5, we suggest the use of three reference metrics and discuss the methods used to compute the approximations (in small instances the exact solutions). Some examples along with conclusions and future work are presented in the last sections.

2 Preliminaries

Mathematically, a multiple objective or criteria optimization problem can be stated as

$$\min_{s \in S} \mathcal{W}(s), \tag{1}$$

where S is the feasible set and

$$\begin{array}{rccl} \mathcal{W}: & S & \to & \mathbb{R}^m \\ & s & \mapsto & \mathcal{W}(s) = (w_1(s), w_2(s), \ldots, w_m(s)) \end{array} \tag{2}$$

is the m-objective (vector) function, for which all objectives are to be minimized. Note that if some of the objectives was to be maximized, we could use the dual principle and therefore still use (1) (Deb, 2001).

The optimization process requires the definition of an order relation. The prevailing relation, in the multiple objective optimization area, is defined as follows: in a minimization process, a vector $x = (x_1, x_2, \ldots, x_m)$

is said to dominate a vector $y = (y_1, y_2, \ldots, y_m)$, and we write $x \prec y$, if $x_i \leq y_i$ for all $i \in \{1, 2, \ldots, m\}$, and for at least one $j \in \{1, 2, \ldots, m\}$ we have $x_j < y_j$. A solution is said to be Pareto optimal if it is not dominated by any other solution of the feasible set. The set of all the Pareto optimal solutions is called Pareto(-optimal) set or efficient set.

A network is a structure defined as

$$\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{Z}), \tag{3}$$

where $\mathcal{V}$ is the set of nodes or vertices, $\mathcal{E}$ is the set of edges and $\mathcal{Z} : \mathcal{E} \to \mathbb{R}^m$ is the weight vector function associating to each edge, $e \in \mathcal{E}$, values $\mathcal{Z}(e) = (z_1(e), z_2(e), \ldots, z_m(e))$. Throughout this article, we will consider that $\mathcal{N}$ is undirected.

For a sub-network T of $\mathcal{N}$, it is considered the sum weight function as

$$\begin{aligned} \mathcal{W}(T) &= (w_1(T), w_2(T), \ldots, w_m(T)) = \\ &= \left(\sum_{e \in T} z_1(e), \ldots, \sum_{e \in T} z_m(e) \right) = \\ &= \sum_{e \in T} \mathcal{Z}(e), \end{aligned} \tag{4}$$

where $e \in T$ means that e is an edge that belongs to T.

3 Network Generators

The network-generating process is divided into three phases that we will describe in this section. The method starts by generating a set of nodes, followed by the computation of a set of edges. The network construction is concluded by setting the edges weights. There will be a single exception to this sequence, as we will see in Section 3.2.5, when a Voronoi diagram is used to set up the edges, since the original set of nodes is replaced by the one induced by the diagram.

3.1 Nodes generators

One of the main objectives of the MOST Project is to start with a number of networks that could reproduce practical problems, as much as possible. After a previous analysis, we decided to start with five types of nodes generators, namely: Grid, Triangular, Uniform, Normal, and

Cluster Nodes Generator. The first two generators can be classified as regular since they return regular clouds of nodes. The third and forth cases are examples of a generator that uses pseudo-random numbers associated with statistical distributions. In particular, we use the Uniform and Normal distributions. Finally, the previous generators are applied to create a set of nodes clusters. First, we calculate a set of centers, which is followed by the generation of a set of nodes located in the neighborhood of those centers. Next, we give a detailed description of each one of these methods.

3.1.1 Grid Nodes Generator (GNG)

The simplest case is that of a Grid Node Generator. The set of nodes produced here is distributed over an $m \times n$ lattice. To simplify the text, $GNG_{m,n}$ will represent an instance of that generator, with m lines and n columns of nodes,

$$\mathcal{V} = GNG_{m,n} = \{(x, y) : x \in \{1, 2, \ldots, n\}, y \in \{1, 2, \ldots, m\}\}.$$

In Figure 1(b) we can see an example of a $GNG_{10,10}$.

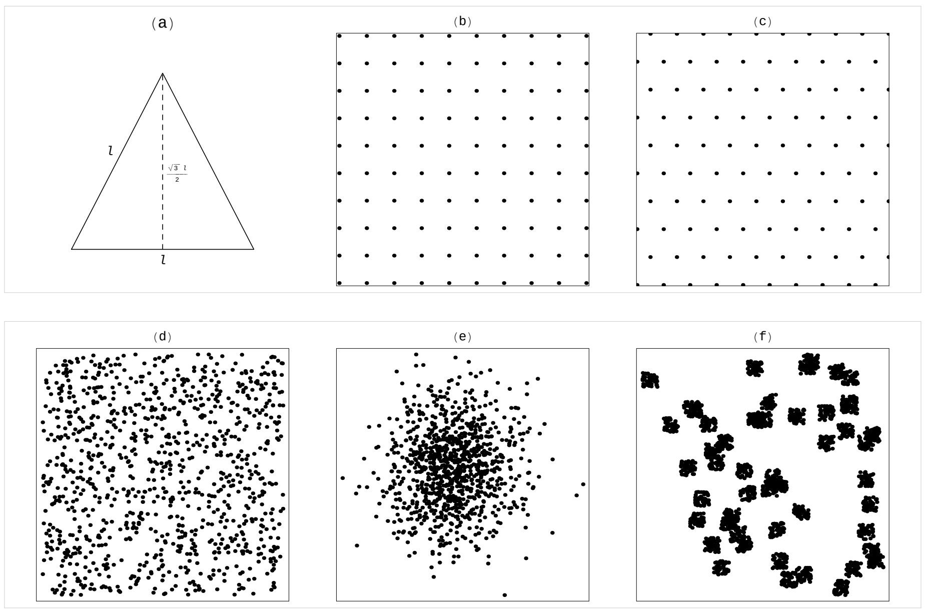

Figure 1: (a) Height of an equilateral triangle with side l. Graphical representation of nodes distribution according to the nodes generator: (b)$GNG_{10,10}$; (c) $TNG_{10,10}$; (d) UNG_{1000}; (e) NNG_{1000}; and (f) $ClNG_{50,50}$.

3.1.2 Triangular Nodes Generator (TNG)

Like GNG, the Triangle Nodes Generator also sets m lines with n nodes per line, $TNG_{m,n}$, distributed according to Algorithm 1. In Figure 1(c), we can see the nodes distribution for a $TNG_{10,10}$.

Algorithm 1 Triangular Nodes Generator

```
input: $m$, $n$, $l$, $h = \frac{l\sqrt{3}}{2}$    ▷ $m, n$ – Number of lines and number of nodes
    by line; $l$ – distance between neighbor nodes; $h$ - distance between lines (see
    Figure 1(a))
output: $\mathcal{V}$    ▷ set of nodes
1: $\mathcal{V} = \emptyset$
2: for all $(x, y) \in \{1, 2, ..., n\} \times \{1, 2, ..., m\}$ do
3:     if $x$ is odd then
4:         $\mathcal{V} = \mathcal{V} \cup \{(x \times l, y \times h)\}$
5:     else
6:         $\mathcal{V} = \mathcal{V} \cup \{(x \times l + \frac{l}{2}, y \times h)\}$
7: return $\mathcal{V}$
```

3.1.3 Statistical Nodes Generator

The Statistical Nodes Generator uses (pseudo) random number generators defined by two statistical distributions, Z_X and Z_Y (associated to x and y coordinates, respectively), to yield as much nodes as necessary. Algorithm 2 describes the process, where $random(Z)$ is a function that returns a (pseudo) random number with specified distribution Z.

We considered two instances of the Statistical Nodes Generator obtained with the Uniform and the Normal distributions.

Uniform Nodes Generator (UNG) As stated, the Uniform Nodes Generator is a particular case of the Statistical Nodes Generator and UNG_n represents an instance of UNG with n nodes. In this case, $Z_X \sim$

Algorithm 2 Statistical Nodes Generator

```
input: $Z_X, Z_Y, n$    ▷ $Z_X, Z_Y$ – statistical distributions; $n$ – number of nodes
output: $\mathcal{V}$    ▷ set of nodes
1: $\mathcal{V} = \emptyset$
2: for $i = 1, 2, ..., n$ do
3:     $\mathcal{V} = \mathcal{V} \cup \{(random(Z_X), random(Z_Y))\}$
4: return $\mathcal{V}$
```

$U[x_{min}, x_{max}]$ and $Z_Y \sim U[y_{min}, y_{max}]$, where $U[a, b]$ is the Uniform distribution over the interval $[a, b]$, and $x_{min}, x_{max}, y_{min}$, and y_{max} are generator parameters. This produces a cloud of nodes uniformly distributed over the rectangle $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$.

In Figure 1(d) we can see an example of the distribution for $n = 1000$ nodes, UNG_{1000}, with $x_{min} = y_{min} = 0$, and $x_{max} = y_{max} = 1000$.

Normal Nodes Generator (NNG) The Normal Nodes Generator is also a particular case of the Statistical Nodes Generator and NNG_n represents an instance with n nodes. Here, $Z_X \sim N(\mu_X, \sigma_X)$ and $Z_Y \sim N(\mu_Y, \sigma_Y)$, where $N(\mu, \sigma)$ is the Normal distribution with mean μ and standard deviation σ, and μ_X, μ_Y, σ_X and σ_Y are generator parameters.

In this case, the cloud of nodes is distributed around (μ_X, μ_Y) with horizontal and vertical deviations associated with the values of σ_X and σ_Y, respectively.

In Figure 1(e) we can see the cloud of nodes for NNG_{1000}, with $\mu_x = \mu_y = 10000$ and $\sigma_x = \sigma_y = 100$.

3.1.4 Clusters Nodes Generator ($ClNG$)

The last nodes generator that we will consider is the Clusters Nodes Generator. The $ClNG$ uses, in two steps, the previous generators, to produce a set of m clusters with n nodes each. In the first phase, m nodes are generated, $c = (c_x, c_y)$, around which, in the second phase, the clusters are completed by the addition of the remaining $n - 1$ nodes. An instance with m clusters of n nodes each will be represented by $ClNG_{m,n}$. In this case, that obviously depends on the location of the centers and the dimensions of the clusters, we usually obtain a set of separated sub-clouds of nodes.

Figure 1(f) shows an example of 50 clusters with 50 nodes each, $ClNG_{50,50}$. We used a UNG_{50} to generate the centers with $x_{min} = y_{min} = 0$ and $x_{max} = y_{max} = 1000$. Then each cluster is completed using a UNG_{49} in the neighbors of the centers defined by the parameters $x_{min} = c_x - \delta_x$, $x_{max} = c_x + \delta_x$, $y_{min} = c_y - \delta_y$, and $y_{max} = c_y + \delta_y$, where $\delta_x = \delta_y = \sqrt{1000}$.

Although, in the example the result is a set of perfectly balanced clusters with similar complexity among all, other combinations of the generators can produce very distinct cases.

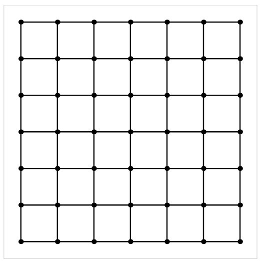

Figure 2: Graphical representation of a network obtained with $GEG_{7,7}$

3.2 Edge generators

In this section, we describe the six methods used to generate the networks' edges, $\mathcal{E}$. The first, the Grid Edges Generator, is only applicable to GNG instances and returns a Manhattan-like topology. The next two are called Delaunay and k-Convex Edge Generator since they use a Delaunay triangulation and an onion triangulation, respectively. This one is based on layered k-convex hulls, to generate the edges. Fourth, we consider the complete networks case. It follows the definition of the Voronoi Edges Generator, based on the Voronoi diagram, which has the pecularity of replacing the set of nodes by the ones induced by the diagram. Finally, the Cluster Edge Generator uses a set of nodes obtained with the $ClNG$ and uses some of the previous methods to generate the edges. Next, we outline each of these generators.

3.2.1 Grid Edges Generator (GEG)

The Grid Edges Generator uses the nodes obtained with the GNG. An edge e_{pq} defined by nodes p and q with coordinates (p_x, p_y) and (q_x, q_y), respectively, belongs to $\mathcal{E}$ if

$$(|p_x - q_x| = 1 \land p_y = q_y) \lor (p_x = q_x \land |p_y - q_y| = 1)\,. \qquad (5)$$

This network topology appear, for instance, in problems related to Manhattan-like cities networks and VLSI circuits problems where many times the components must be connect over a grid arrangement. Figure 2 shows an example for GEG over a $GNG_{7,7}$.

3.2.2 Delaunay Edges Generator (DEG)

The Delaunay Edges Generator uses a Delaunay triangulation to define the edges. Starting from a set of nodes $\mathcal{V}$, the Delaunay triangulation is

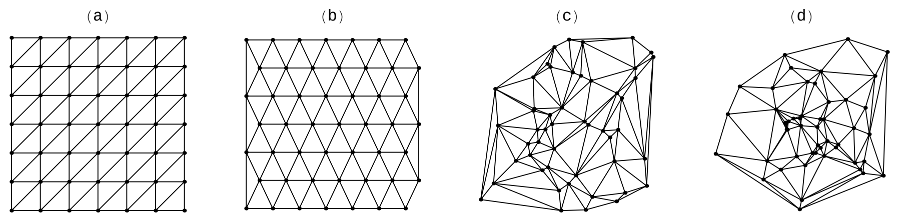

Figure 3: Graphical representation of networks obtained with the DEG from the set of nodes obtained with: (a) $GND_{7\times7}$; (b) $TNG_{7\times7}$; (c) UNG_{50}; and (d) NNG_{50}.

the decomposition of the convex-hull defined by $\mathcal{V}$ in triangles, such that two nodes are connected if and only if they lie on a circle whose interior contains no other node of $\mathcal{V}$ (see for example (de Berg et al., 1997) for more details). This triangulation is the one that maximizes the minimum angle of the triangles that compose it and is used in various problems like mesh generation and finite elements. It is also known that the Euclidean minimum spanning tree of a set of nodes is a subset of the Delaunay triangulation for these same set of nodes.

In Figure 3(a)$\sim$(d), we see examples of four networks obtained with a $GNG_{7,7}$, a $TNG_{7,7}$, a UNG_{50}, and an NNG_{50}, respectively. It is obvious that for the first two cases the triangulation is not unique. For the other two cases, it depends on whether the points are in general position (no three nodes are in the same line and no four in the same circle).

Triangle Edges Generator (TEG) The Triangle Edges Generator is a particular case of DEG when applied to TNG. Figure 3(b) shows an example of the TEG for a $TNG_{7,7}$.

3.2.3 k-Convex Edges Generator ($k-CEG$)

A convex hull of a set of nodes is the boundary of the smallest convex set that contains those nodes (see, for example, (de Berg et al., 1997) for more details). If after the computation of a convex hull some unused interior nodes still remain then the process can be recursively repeated, to obtain a set of k-convex hulls. This is known as onion peeling or onion layer and has been studied in areas like computational geometry (Sack and Urrutia, 2000), search algorithms (Chang et al., 2000), and pattern recognition (Poulos et al., 2004). In Figure 4, we can see the resulting k-convex hulls for an UNG_{50}.

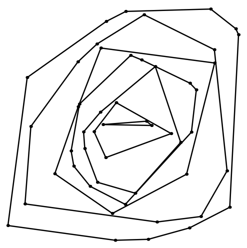

Figure 4: k-convex hulls or onion peeling of a UNG_{50}.

Any triangulation that includes the k-convex hulls is called an onion triangulation. We specifically used a very simple and elegant method that uses the rotating callipers to complete the triangulation (Toussaint, 1983; Pirzadeh, 1999).

The k-Convex Edges Generator ($k - CEG$) uses this onion triangulation, described above, to naturally define the edges of the network. Figure 5 depicts four networks obtained with $k - CEG$.

This k-hull kind of topology can also be found in many city traffics networks.

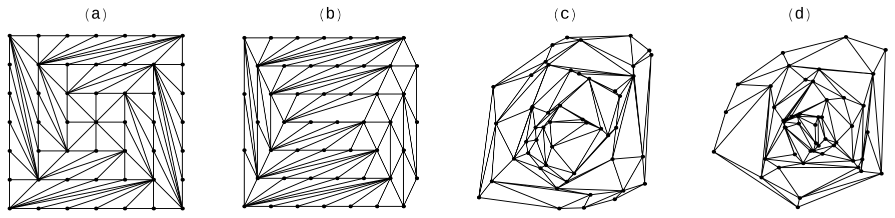

Figure 5: Graphical representations of networks obtained with $k - CEG$ for nodes generated with: (a) $GNG_{7,7}$; (b) $TNG_{7,7}$; (c) UNG_{50}; and (d) NNG_{50}.

3.2.4 Complete Edges Generator (CEG)

As the name suggests, the Complete Edges Generator (CEG) defines the $\binom{|\mathcal{V}|}{2}$ edges between each pair of nodes. Most of the real-life problems do not allow every node to be connected to each other. However, from a more academic point of view, it is usual to consider and test this kind of network as a worse case scenario(Knowles and Corne, 2001).

3.2.5 Voronoi Edges Generator (VEG)

The Voronoi Edges Generator (VEG) uses the Voronoi diagram, dual of the Delaunay triangulation, to generate a network (see, for example, (Sack and Urrutia, 2000)). In this case, we can use any of the nodes generators to start the process. However, as we can see from example in Figure 6(a), the final set of nodes does not coincide with the starting set $\mathcal{V}$, since the edges of the Voronoi diagram are not directly defined by $\mathcal{V}$. Therefore, in this case, $\mathcal{V}$ is replaced by the set of nodes induced by the Voronoi diagram. Note that this set need not necessarily have the same cardinality of the original $\mathcal{V}$.

In Figures 6(b)$\sim$(c) we can see examples of networks obtained with a UNG_{50} and a NNG_{50}. The Voronoi diagram is associated with problems like the Post Office problem (see, for example, (de Berg et al., 1997)) and this kind of topology is very often present as the frontiers of geopolitical maps.

Hexagonal Edges Generator (HEG) The Hexagonal Edges Generator is particular case of VEG when applied to a TNG. In Figure 6(d), we can see an example for a $TNG_{7,7}$.

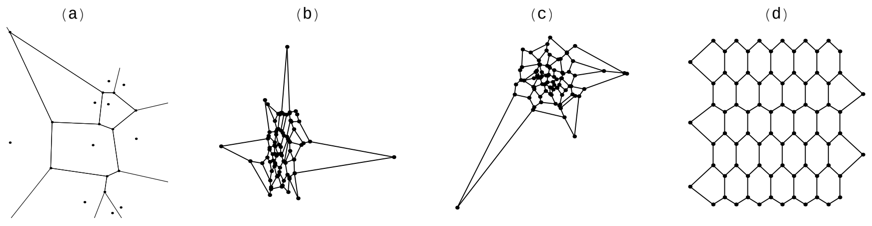

Figure 6: (a) Voronoi diagram for a set of ten points; Graphical representation of the networks obtained with the Voronoi Generator for: (b) NNG_{50}; (c) UNG_{50}; and (d) $TNG_{7,7}$.

3.2.6 Clusters Edges Generator ($ClEG$)

The Clusters Edges Generator uses the previous edges generators in two phases. First, the center nodes of the clusters (see Section 3.1.4) are connected. Then, the process is repeated now over the nodes of each cluster (including their respective centers). In Figure 7, we see two examples for the same set of nodes, $ClNG_{10,15}$. For Figure 7(a), DEG was used both to connect the centers and the nodes of the clusters. In the case of Figure 7(b), $k - CEG$ was adopted to connect the centers and the CEG for the clusters. It is easy to identify real applications where this topology appears, like power, phone or cable distribution networks, and country traffic maps (where each cluster may represent a village and each village is connected to neighbor villages).

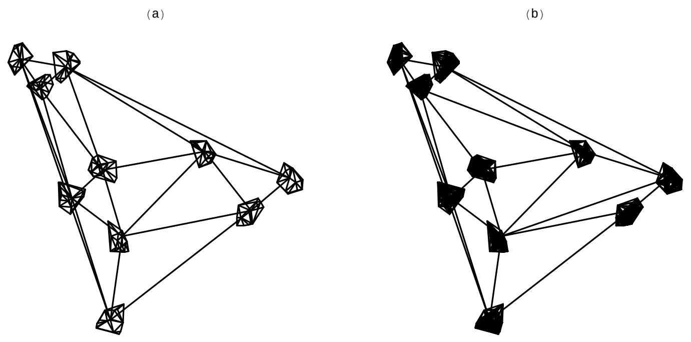

Figure 7: Graphical representation of networks obtained with the $ClEG$ for $ClNG_{10,15}$ with: (a) DEG to connect the centers and the clusters nodes; (b) $k-CEG$ to connect the centers and CEG to generate the edges for the clusters.

3.3 Weights generators

Another important aspect of the networks are the edge weights. In this section we present three types of weights generators: Random Weights Generator, the ρ–Correlated Weights Generator, and the Concave Weights Generator. The first case, for instance, is important in the study of non-Euclidean networks. The second one, has networks with interesting characteristics as in the Pareto front varies from a large cardinality for $\rho \approx -1$ (where for two objectives, the graphical Pareto front representation assumes an almost straight line shape), to a very small cardinality when $\rho \approx 1$. The Concave Weights Generator has other very important characteristic: it can produce large concave Pareto fronts, known to be hard to tackle by classical algorithms like the weighted sum (Deb, 2001).

3.3.1 Random Weights Generator (RWG)

The Random Weights Generator is the simplest case to generate since the weights are set using a discrete integer Uniform random distribution over a predefined set. In Figure 8(a), we see the cloud of weights produced by the RWG for a complete network with 50 nodes.

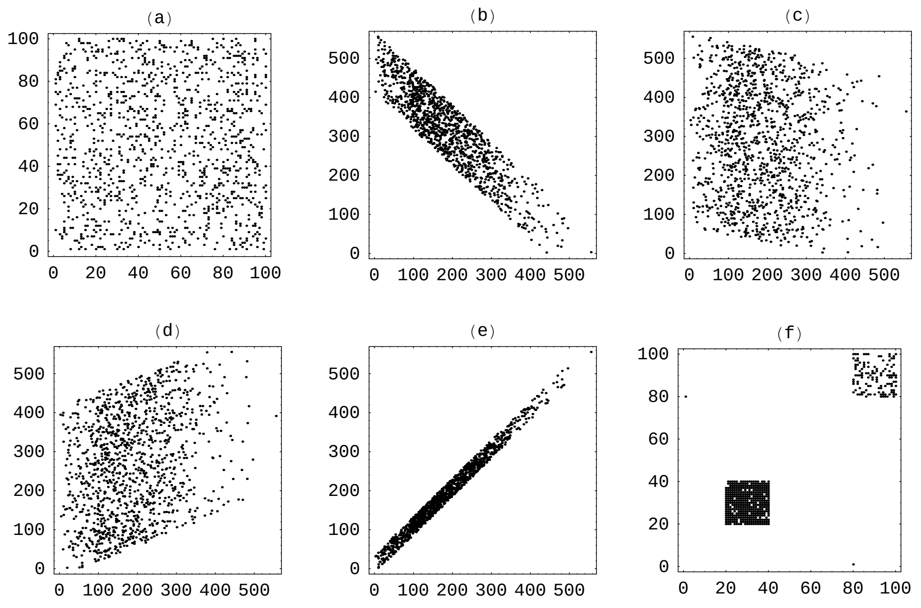

Figure 8: Cloud of weights for a complete network of 50 nodes using: (a) RWG; (b) $(-0.9) - CWG$; (c) $(-0.1) - CWG$; (d) $(0.3) - CWG$; (e) $(0.99) - CWG$; and (f) $CWG_{5,10,100}$

3.3.2 $\rho-$Correlated Weights Generator ($\rho - CWG$)

In the $\rho-$Correlated Weights Generator case, the objective is to set the edges weights, such that the correlation between them is some predefined value $\rho \in [-1, 1]$. Except for the cases where the edges are generated using GEG or TEG, we considered the first weight of the edge e as the (integer) Euclidean length of e. For the exceptions, the first weight is a random integer in $[1, M_d]$, where M_d is the maximum Euclidean distance between every par of nodes. Those two cases have this special treatment since they are regular networks, which imply that the problem could just be reduced to the single objective case (the first weight would be equal in all edges).

In Algorithm 3, we can see a description of the process used to establish the weights vector, where $|e|$ is the Euclidean length of e and $random(U[-1, 1])$ is a function that returns a (pseudo) random number with Uniform distribution in interval $[-1, 1]$.

In Figure 8(b)$\sim$(e), we can see four examples of the clouds of weights obtained with ρ equal to -0.9, -0.1, 0.3, and 0.99 for a complete network

Algorithm 3 Setting edges with ρ-correlated weights

```
input: $\rho, \mathcal{E} = \{e_1, e_2, \dots, e_p\}$    ▷ $\rho$ - desired correlation
output: $\mathcal{Z}$    ▷ $\mathcal{Z} : \mathcal{E} \rightarrow \mathbb{R}^m$
1: Set $W = \begin{bmatrix} 1 & |e_1| & random(U[-1,1]) \\ 1 & |e_2| & random(U[-1,1]) \\ & \vdots & \\ 1 & |e_p| & random(U[-1,1]) \end{bmatrix}$ and $X = \begin{bmatrix} |e_1| \\ |e_2| \\ \vdots \\ |e_p| \end{bmatrix}$
2: Do the $QR$ decompositions of $W$: $W = QR$
3: Set $Y = \begin{bmatrix} 1 & \rho & \sqrt{1-\rho^2} \end{bmatrix} \times Q^T$
4: Normalize $Y$
5: Set $\forall_{e_i \in \mathcal{E}} : z_1(e_i) = \lfloor X_i \rfloor$
6: Set $\forall_{e_i \in \mathcal{E}} : z_2(e_i) = \lfloor Y_i \rfloor$
```

with 50 nodes.

3.3.3 Concave Weights Generator (CWG)

In the Concave Weight Generator(CWG), the weights are generated to produce a large concave Pareto front. This case is an adaptation from the one presented in (Knowles and Corne, 2001) for the k–degree minimum spanning trees problem.

The generator needs two small integers ξ and η, such that $\xi < \eta \ll M - \xi$, where M is the maximum weight admitted. It also needs three special nodes n_1, n_2, n_3. These nodes have an important roll in the generation of the concave front and should have a degree as large as possible. Heuristically, we defined n_1, n_2, and n_3 as follows. The selection process starts by setting n_1 as the node with highest degree. The second node, n_2, is chosen from the nodes adjacent to n_1 with higher degree. Finally, if adjacent nodes exist simultaneously to n_1 and n_2 we choose the one with higher degree to be n_3. Otherwise, the same choice is made from the nodes adjacent to n_1 or n_2. If nodes with the same degree exist in any of the three steps, then one of them is randomly selected.

Finally, the weights of the edges $e \in \mathcal{E}$ are set according to Algorithm 4.

In Figure 8(f), we can see the cloud of weights obtained for a complete network with 50 nodes, $\xi = 20$, $\eta = 40$, and $M = 100$.

Algorithm 4 Setting edges weights with the CWG

```
input: $e_{ij} \in \mathcal{E}$, $M$, $\xi$, $\eta$, $N = \{n_1, n_2, n_3\}$,        ▷ $e_{ij}$
    is the edge defined by nodes $i$ and $j$; $M$ is the maximum weight; $\xi$ and $\eta$ are
    small integers, such that $\xi < \eta \ll M - \xi$; $n_1, n_2, n_3$ are the special nodes;
    and $U_d(a, b)$ - discrete integer Uniform over $\{a, a+1, \ldots, b\}$.
output: $z(e_{ij}) = (z_1, z_2)$        ▷ $e_{ij}$ weight vector
1: if $i, j \notin N$ then return $z(e_{ij}) = (U_d(\xi, \eta), U_d(\xi, \eta))$
2: if $i \in N \ \dot{\vee} \ j \in N$ then return $z(e_{ij}) = (U_d(M - \xi, M), U_d(M - \xi, M))$
3: if $i = n_1 \wedge j = n_2$ then return $z(e_{ij}) = (\xi, \xi)$
4: if $i = n_1 \wedge j = n_3$ then return $z(e_{ij}) = (1, M - \xi)$
5: if $i = n_2 \wedge j = n_3$ then return $z(e_{ij}) = (M - \xi, 1)$
```

3.4 File Format

In Figure 9, we can see the text file format. The class of the problems is implicit in the name that follows the sequence: `[<nodes generator>]<|`$\mathcal{V}$`|>[<edges generators>]<|`$\mathcal{E}$`|>[<weights generator>]<parameters>[NST]<number of spanning trees>.net`. For example, `[ClNG]100[ClEG][DEG][k-CEG]227[ro-CWG]0.5[NST]49.net` is a network with 100 nodes and 227 edges generated using the $ClNG$ for the nodes and $ClEG$ with DEG and $k - CEG$ to connect inter- and intra- clusters, respectively. The weights were set using the $0.5 - CWG$ and the number of distinct spanning trees of $\mathcal{N}$ is of the order of 10^{49}.

10	#$n = \lvert\mathcal{V}\rvert$ – number of nodes
20	#$m = \lvert\mathcal{E}\rvert$ – number of edges
2	#c – number of weights
407 73	#x_1 y_1 – node 1 coordinates
298 758	#x_2 y_2 – node 2 coordinates
⋮	
470 685	#x_n y_n – node n coordinates
0 1 89 35	#i_1 j_1 $z^1_{i_1,j_1} \ldots z^c_{i_1,j_1}$ – $\mathcal{Z}(e_{i_1,j_1}) = \left(z^1_{i_1,j_1}, \ldots, z^c_{i_1,j_1}\right)$
0 4 34 54	#i_2 j_2 $z^1_{i_2,j_2} \ldots z^c_{i_2,j_2}$ – $\mathcal{Z}(e_{i_2,j_2}) = \left(z^1_{i_2,j_2}, \ldots, z^c_{i_2,j_2}\right)$
⋮	
6 8 27 95	#i_m j_m $z^1_{i_m,j_m} \ldots z^c_{i_m,j_m}$ – $\mathcal{Z}(e_{i_m,j_m}) = \left(z^1_{i_m,j_m}, \ldots, z^c_{i_m,j_m}\right)$

Figure 9: MOST file format example.

3.5 Summary

To summarize, the process of generating the networks is divided into three parts: first, generate the nodes; second, generate the edges; and third, generate the weights. In Table 1 we see the possible combinations of the generators. Some of the possible combination obviously take the meaningfulness of the sub-adjacent concept, like for instance, the use of a DEG over a $ClNG$.

	GNG	TNG	UNG	NNG	$ClNG$
GEG	✓	×	×	×	×
DEG	✓	✓	✓	✓	✓
$k-CEG$	✓	✓	✓	✓	✓
CEG	✓	✓	✓	✓	✓
VEG	✓	✓	✓	✓	✓
$ClEG$	×	×	×	×	✓

Table 1: Summary table of the possible combinations of the nodes and edges generators (✓).

4 Performance Metrics

The approximations to the solution of a multiple objective optimization problem should satisfy some desirable aspect, as for example to be as close as possible to the real Pareto front, uniform distribution, and having spread solutions all along the Pareto front. Various metrics for sets of non-dominated solutions have been proposed (Zitzler et al., 2003). As in (Knowles and Corne, 2002) we recommend the use of $R1$, $R2$ and $R3$, three of the metrics suggested in (Jaszkiewicz, 2001) for the multiple objective maximization continuous case. These metrics have the advantage of providing comparisons based on probabilities, are adaptable to various utility functions, are non-cardinal metrics, independent of a reference set (although they induce three other metrics that depend on a reference set – $R1_R, R2_R$, and $R3_R$) and satisfy the complete, the strong, and the weak outperformance relations. On the other hand, the computational overhead can be high and there is the need to set a family of utility functions (which for some cases, like the Tchebycheff utility, can also introduce some other difficulties like the computation of an ideal point).

Next, we shortly describe those metrics adapted for the discrete minimization case.

$R1$ **Metric** For the pair of approximations to the Pareto set $\mathcal{P}_1$ and $\mathcal{P}_2$, $R1$ is defined as

$$R1(\mathcal{P}_1, \mathcal{P}_2, U, p) = \int_{u \in U} C(\mathcal{P}_1, \mathcal{P}_2, u)p(u)du \qquad (6)$$

where U is a set of utility functions, $u : \mathbb{R}^m \rightarrow \mathbb{R}$, which map each approximation to a utility measure, $p(u)$ is the probability of the utility u and

$$C(\mathcal{P}_1, \mathcal{P}_2, u) = \begin{cases} 1 & if \quad u^*(\mathcal{P}_1) < u^*(\mathcal{P}_2) \\ \frac{1}{2} & if \quad u^*(\mathcal{P}_1) = u^*(\mathcal{P}_2) \\ 0 & if \quad u^*(\mathcal{P}_1) > u^*(\mathcal{P}_2) \end{cases}, \tag{7}$$

with

$$u^*(\mathcal{P}') = \min_{q \in \mathcal{P}'} u(q). \tag{8}$$

To define U, we can use a family of Tchebycheff utility function defined as

$$u_\lambda(q, r) = \max_{j=1,2,\ldots,m} \{\lambda_j(q_j - r_j)\}, \tag{9}$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$ is a weight vector, q is a solution for which we want to measure the utility and r is a reference point. Therefore, in formula (6) we set

$$U = \left\{ u_\lambda(q, r) : \lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m) \in]0, 1[^m \wedge \sum_{i=1}^{m} \lambda_i = 1 \right\}. \tag{10}$$

The $R1$ metric measures the probability that $\mathcal{P}_1$ is better that $\mathcal{P}_2$ over the set of utility functions. If $R1(\mathcal{P}_1, \mathcal{P}_2, U, p) > \frac{1}{2}$ then, according to this measure, $\mathcal{P}_1$ is better than $\mathcal{P}_2$ and it will be not worse if $R1(\mathcal{P}_1, \mathcal{P}_2, U, p) \geq \frac{1}{2}$. $R1_R$ is defined by considering one of the sets as a reference set. For example, if $\mathcal{P}_r$ is the reference set, $R1_R(\mathcal{P}_1, U, p) = R1(\mathcal{P}_r, \mathcal{P}_1, U, p)$ and, therefore, the close the value of $R1_R(\mathcal{P}_1, U, p)$ is to 0.5, more is the probability that $\mathcal{P}_1$ is a good approximation.

$R2$ **Metric** The $R2$ metric is defined as

$$R2(\mathcal{P}_1, \mathcal{P}_2, U, p) = \int_{u \in U} (u^*(\mathcal{P}_1) - u^*(\mathcal{P}_2)) p(u) du. \tag{11}$$

$R2$ returns the difference between the expected values of the utilities of two approximations. Therefore, $\mathcal{P}_1$ is expected to be better than $\mathcal{P}_2$ if $R2(\mathcal{P}_1, \mathcal{P}_2, U, p) < 0$ and not worse if $R2(\mathcal{P}_1, \mathcal{P}_2, U, p) \leq 0$.

Similar to the previous case, we can use a reference set $\mathcal{P}_r$ to define $R2_R(\mathcal{P}_1, U, p) = R2(\mathcal{P}_r, \mathcal{P}_1, U, p)$ which implies that $\mathcal{P}_1$ is expected to be a good approximation for values of $R2_R(\mathcal{P}_1, U, p)$ near 0.

$R3$ **Metric** Finally, $R3$ is defined as the ratio

$$R3(\mathcal{P}_1, \mathcal{P}_2, U, p) = \int_{u \in U} \frac{u^*(\mathcal{P}_1) - u^*(\mathcal{P}_2)}{u^*(\mathcal{P}_1)} p(u) du. \tag{12}$$

and $R3_R(\mathcal{P}_1, U, p) = R3(\mathcal{P}_r, \mathcal{P}_1, U, p)$.

In practice, the computation of formulae (6), (11), and (12) can be approximated by replacing the integrals by a Riemann sum over U' where

$$U' = \left\{ u_\lambda(q, r) : \lambda_i \in \left\{ \frac{1}{k}, \frac{2}{k}, \ldots, \frac{k-1}{k} \right\} \wedge \sum_{i=1}^{m} \lambda_i = 1 \right\}, \tag{13}$$

for some large k.

5 Algorithms

In this section, we briefly present the methods used to compute the Pareto front approximations for the multiple objective minimum spanning trees problems. In particular, we used a brute force method that computes the exact Pareto front (applicable only to the smaller networks), the classical weighted sum, and an algorithm called `MONACO`. We need to remember that, in this case an important time-consuming factor is the update of the nondominated set, particularly in large fronts where several comparisons have to be performed for each solution or approximation(Mostaghim et al., 2002).

5.1 Brute Force Method

The first method uses a brute force algorithm to enumerate the exact Pareto set (Shioura et al., 1997). Due to the $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}| + M)$ time complexity to enumerate all spanning trees, where M is the number of spanning trees in $\mathcal{N}$, in practice this method can only be used for the smaller network instances.

5.2 Weighted Sum Method

The weighted sum method transforms the multiple objective optimization problem into a single objective problem that can be stated as

$$\min_{T \in \mathcal{T}} \sum_{i=1}^{m} l_i w_i(T), \tag{14}$$

where $l_1, l_2, \ldots, l_m \in \mathbb{R}^+$ are parameters usually such that $\sum_{i=1}^{m} l_i = 1$, $\mathcal{T}$ is the set of all spanning trees in $\mathcal{N}$, and $w_1, w_2, \ldots, w_m$ are defined in (4). The weighted sum is specially used when a significance can be estimated to each objective (reflected in the ratio between the values of $l_1, l_2, \ldots, l_m$).

Consequently, the solution of (14) is obtained by the application of a single objective algorithm that, for the case in study, is known to have polynomial complexity (see, for example, (Jungnickel, 1999)). The single objective method returns an optimal solution for each set of parameters l_i. Therefore, we can run it with several combinations of those parameters to produce a set of solutions. However, reaching a representative front can, in some cases, be difficult or impossible, since different combinations of the parameter do not necessarily return different solutions (Hamacher and Ruhe, 1994). Difficulties can also arise from the fact that this method cannot return solutions from a concave region of the Pareto front(Deb, 2001).

5.3 MONACO Algorithm

The third method used is the Multiple Objective Network optimizations based on an ACO (MONACO) algorithm(Cardoso et al., 2004). As the name suggests, this method is based on the same paradigm of the Ant Colony Optimization (ACO) metaheuristic (Dorigo et al., 1999; Dorigo and Stutzle, 2004). The ACO mimics the foraging behavior common to almost all ants' colonies. This behavior is supported by a communication process that relies on a chemical pheromone trail, signaling a good path to some supply location. Similar, associated to the possible parts of the solutions, is defined a numerical value that represents the suitability of that element in the construction of good outcomes.

MONACO's basic procedure also uses a set of agents. However, one of the main differences between ACO and MONACO, is the fact that the MONACO process uses a pheromone vector (like if there were several layers of pheromone) associated to each atomic piece. Each element of the vector is associated with a weight and, as described previously, represents how worthy the elements were, for some time-window, in the construction of the solutions relative to the associated weight.

Algorithm 5 describes MONACO's basic procedure. Initially, the process starts by setting the approximation set $\mathcal{P} = \emptyset$ and initializing the pheromone vectors. Then a set of cycles are run. In each cycle, a collection of solutions is built using the pheromone vectors and some possible

Algorithm 5 MONACO's high level description

```
output: $\mathcal{P}$                                  ▷ Approximation to the Pareto set
1: Initialize the pheromone vectors and set $\mathcal{P} = \emptyset$
2: Do pre-computation                                            ▷ optional
3: while stopping criteria is not met do
4:     for all agents do
5:         Construct a new solution using the pheromone vectors
6:         Apply local operators                                 ▷ optional
7:         Evaluate the solution and update $\mathcal{P}$
8:     Update the pheromone vectors.
9: return $\mathcal{P}$
```

heuristics, followed by the update of $\mathcal{P}$ with those new solutions. Each cycle ends with the pheromone vectors update, based on the known solutions. Some extra steps can be performed, namely, doing some pre-computation work before the main cycle or optimizing the solutions obtained using the pheromone vectors with some other local operators. A more detailed description can be found in (Cardoso et al., 2005).

6 Examples

6.1 Computational Environment and Parameters

6.1.1 Brute Force Method

The brute force method was implemented in C++ and the tests were run on a set of LINUX OS workstations with Intel® PIII $533Mhz$ processors and $128Mb$ of RAM. Due to time consumption, the use of this method was restricted to the cases were there were less the 10^{12} distinct networks. To maintain the Pareto set, we used a quadtree data structure(Mostaghim et al., 2002).

6.1.2 Weighted Sum Method

In the weighted sum case, for each problem we considered the distinct solutions of the 1000 runs for the single objective minimum spanning tree problem, defined in (14), with $l_1 = \frac{k}{1001}$ (for $k = 1, 2, \ldots, 1000$) and $l_2 = 1 - l_1$.

6.1.3 `MONACO`

The codification of `MONACO`'s algorithm was made in C++ and tests were run on the same environment of the brute force case. The running time was limited to $|\mathcal{E}| \log |\mathcal{E}|$ seconds, and we also kept the last update time for reference. To maintain the Pareto set, we used a quadtree data structure.

6.2 Network examples

Figures 10 and 11 present the graphical representations of the Pareto fronts (obtained with the brute force method) and the approximations sets of some examples. The Pareto approximations are the nondominated set of the union of the solutions returned by the weighted sum and `MONACO`.

For instance, in Figures 10 (a), (b), (d), (e), (h), and Figures 11 (b), (c), (d) we see sketches of eight concave fronts. In Figures 10 (c), (g), and Figure 11 (g) the solutions of three networks that used the $\rho - CWG$ (with ρ equal to -0.99 and -0.9) are depicted. In these cases, the nondominated set has a relatively large cardinality and the graphical representation is an almost straight line.

7 Conclusions and Future Work

The MOST Project intends to maintain a repository for spanning-tree-related problems. In this installation phase, we considered only the unconstrained minimum spanning trees problem with two objectives. In future, and taking into account the pretension of providing real-life problems, this library must contain some other features like, undirected networks, networks with a larger number of objectives and nodes, different topologies, and different instances of the spanning trees problem.

We have established a set of network generators, that are combinations of three types: nodes, edges, and weights generators. These network generators produce a diversified set of fronts, capable of defining an also diversified number of difficulties in the optimization process. Accompanying the problem instances, the results obtained using a brute force method (whenever the dimension of the problem makes it possible), a weighted sum method or a meta-heuristics are presented. In particular, we used the `MONACO` meta-heuristic.

On our behalf, improvements to the `MONACO`'s algorithm and other

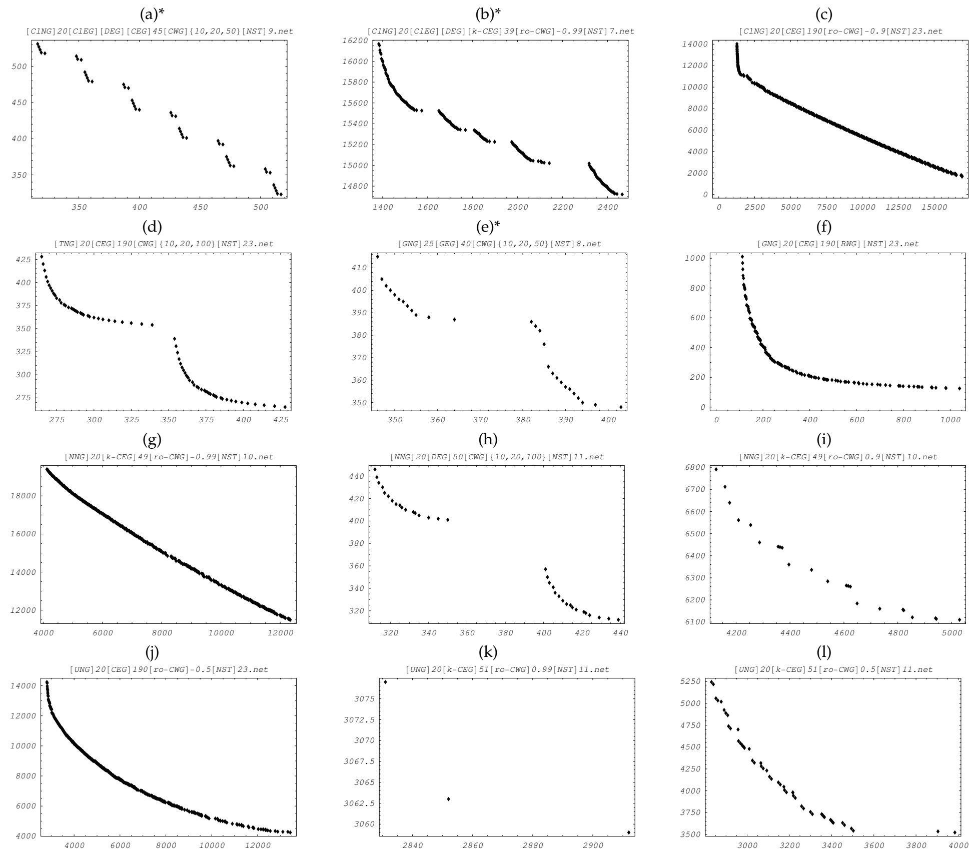

Figure 10: Examples of graphical representation of the approximations to Pareto fronts and exact fronts(*), for some selected networks.

meta-heuristical approaches are being studied in order to compute better Pareto approximations as well as to deal with larger networks.

Finally, we reinforce the ideas:

- MOST Project must be a dynamic venture, open to the free utilization of the enclosed problems;
- MOST is also open to the spanning trees investigators community contributions (new problems, solutions improvements, site improvements,...).

References

Aarts, E. and Lenstra, J. (1997). *Local Search in Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Opti-

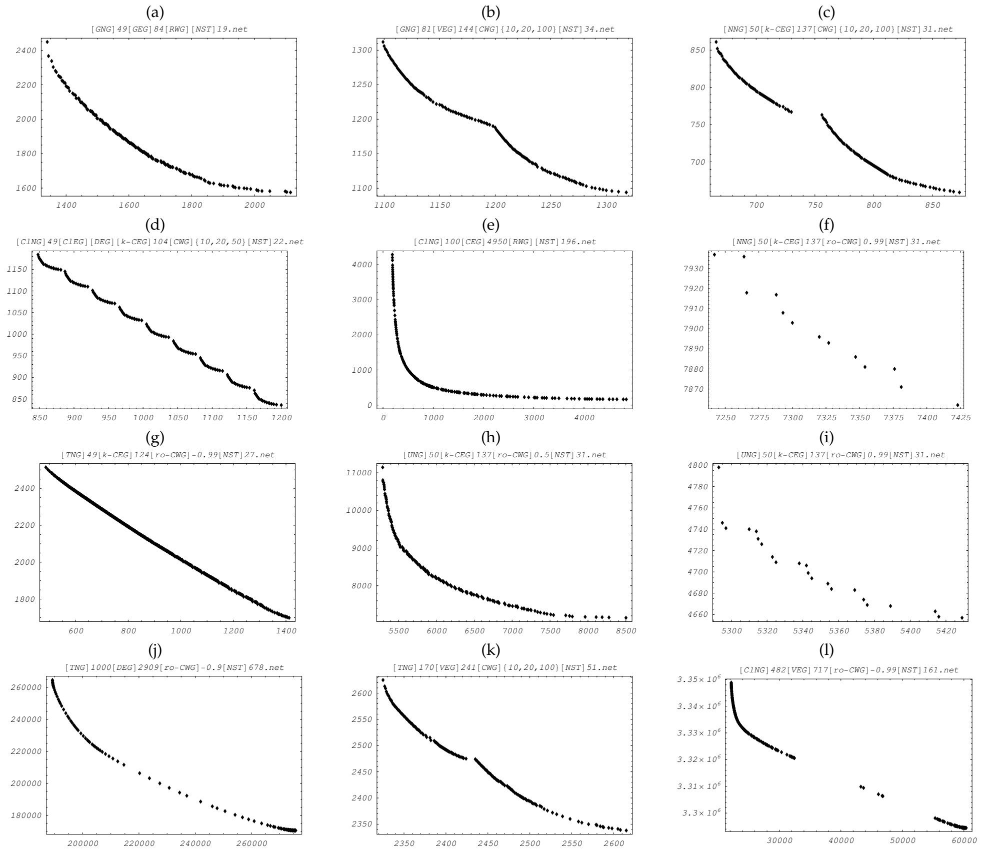

Figure 11: Examples of graphical representation of the approximations to the Pareto fronts of some selected networks.

mization. John Wiley & Sons, Inc.

An, L., Xiang, Q.-S., and Chavez, S. (2000). A fast implementation of the minimum spanning tree method for phase unwrapping. *IEEE, Transactions on Medical Imaging*, 19(8).

Camerini, P., Galbiati, G., and Maffioli, F. (1984). The complexity of weighted multi-constrained spanning tree problems. *Colloquia Mathematica Societais Janos Bolyai*, 44:53–101.

Cardoso, P., Jesus, M., and Márquez, A. (2004). Determinação de minimum spanning trees multi-objectivos baseadas num ACO. In *Congresso de Métodos Computacionais em Engenharia*, Lisboa. LNEC.

Cardoso, P., Jesus, M., and Márquez, A. (2005). Multiple criteria minimum spanning trees. In *XI Encuentros de Geometria Computacional*, Santander, Spain. Universidad de Santander.

Chang, Y., Bergman, L., Castelli, V., Li, C., Lo, M., and Smith, J. (2000). The onion technique: indexing for linear optimization queries. *SIGMOD Rec.*, 29(2):391–402.

Cirasella, J., Johnson, D., McGeoch, L., and Zhang, W. (2001). The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. *Lecture Notes in Computer Science*, 2153.

Coello-Coello, C., Hernández-Lerma, O., and Villalobos-Arias, M. (2005). *Foundations of Genetic Algorithms: 8th International Workshop*. Lecture Notes in Computer Science. Springer.

de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (1997). *Computational Geometry - Algorithms and Applications*. Springer-Verlag, Berlin.

Deb, K. (2001). *Multi-objective Optimization using Evolutionary Algorithms*. John Wiley & sons.

Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. (2002). Scalable multi-objective optimization test problems. In *In Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 825–830, Piscataway, New Jersey.

Dorigo, M., Bonabeau, E., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to artificial Systems*. Oxford University Press.

Dorigo, M. and Stutzle, T. (2004). *Ant Colony Optimization*. MIT Press.

Gent, I. and Walsh, T. (1999). *Principles and Practice of Constraint Programming - CP99*, chapter CSPlib: A Benchmark Library for Constraints. Lecture Notes in Computer Science. Springer.

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

Hajic, J., McDonald, R., Pereira, F., and Ribarov, K. (2005). Non-projective dependency parsing using spanning tree algorithms. In *Human Language Technology Conference*, Vancouver, Canada.

Hamacher, H. and Ruhe, G. (1994). On spanning tree problems with multiple objective. *Annals of operation research*, 52:2309–230.

Jaszkiewicz, A. (2001). *Multiple Objective Metaheuristic Algorithms for Combinatorial Optimization*. PhD thesis, Poznan University of Technology.

Johnson, D. and McGeoch, L. (1997). *Local Search in Combinatorial Optimisation*, chapter The Traveling Salesman Problem: A Case Study in Local Optimization, pages 215–310. E. Aarts and J. Lenstra (editors). John Wiley and Sons Ltd.

Jungnickel, D. (1999). *Graphs, Networks and Algorithms*, volume 5 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220, 4598(4598):671–680.

Knowles, J. and Corne, D. (2001). Benchmark problem generators and results for the multiobjective degree-constrained minimum spanning tree problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 424–431. Morgan Kaufmann Publishers, San Francisco, California.

Knowles, J. and Corne, D. (2002). On metrics for comparing non-dominated sets. In *Congress on Evolutionary Computation (CEC 2002)*, volume 1.

Mostaghim, S., Teich, J., and Tyagi, A. (2002). Comparison of data structures for storing Pareto-sets in MOEAs. In *Evolutionary Computation (CEC'2002)*, volume 1, IEEE Service Center, Piscataway, New Jersey.

Pinto, D., Barn, B., and Fabregat, R. (2005). Multi-objective multicast routing based on Ant Colony Optimization. In *CCIA 2005*, L'Alguer.

Pirzadeh, H. (1999). Computational geometry with the rotating callipers. Master's thesis, McGill University, Canada.

Poulos, M., Evangelou, A., Magkos, E., and Papavlasopoulos, S. (2004). Fingerprint verification based on image processing segmentation using an onion algorithm of computational geometry. In *Sixth Int. Conf. on Mathematics Methods in Scattering Theory and Biomedical Technology (BIOTECH'6)*.

Sack, J.-R. and Urrutia, J. (2000). *Handbook of Computational Geometry*. Elsevier Science, North-Holland.

Shioura, A., Tamura, A., and Uno, T. (1997). An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM Journal on Computing*, 26(3).

Toussaint, G. (1983). Solving geometric problems with the rotating calipers. In *Proceedings of IEEE MELECON'83*, Athens, Greece.

Vose, M. (1999). *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press,.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C., and Fonseca, V. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary computation*, 7(2).