MINIMUM DILATION GEOMETRIC SPANNING TREES

Miguel Francisco Alves de Mattos Gaiowski[1]
Instituto de Computação - UNICAMP
Av. Albert Einstein, 1251 - Cidade Universitária, Campinas/SP - Brasil
miggaiowski@gmail.com

Cid Carvalho de Souza[2]
Instituto de Computação - UNICAMP
Av. Albert Einstein, 1251 - Cidade Universitária, Campinas/SP - Brasil
cid@ic.unicamp.br

ABSTRACT

The *Minimum Dilation Geometric Spanning Tree Problem* (MDGSTP) is $\mathscr{NP}$-hard, which justifies the development of heuristics to it. This paper presents heuristics based on the GRASP metaheuristic paradigm for MDGSTP. The input of this problem is a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in the plane. Let the geometric graph $G(P)$ associated with P be the undirected weighted complete graph of n vertices with edge weights corresponding to the euclidean distance between the points represented by its endpoints. The goal is to find a subgraph T of $G(P)$ with $n-1$ edges (a tree) that minimizes the greatest ratio between the length of the shortest path in T and in $G(P)$ for all pairs of vertices. This measurement is called the *dilation of* T.

KEYWORDS. GRASP, Minimum Dilation, Computational Geometry.

Main area: Metaheuristics

[1]Supported by a grant from PIBITI/CNPq (UNICAMP).
[2]Supported by CNPq grants 301732/2007-8, 472504/2007-0, 473867/2010-9 and FAPESP grant 07/52015-0.

1 Introduction

Consider the following situation: an air company has been authorized to operate $n-1$ flights among n cities. Each flight must be directly between two of the cities, without intermediate stops. The flight times between all pairs of cities are already known. The company is free to choose any flights but, for market reasons, it decided to offer the clients the possibility to travel from each city to any other, possibly having to stop in intermediate cities, traveling only in flights that it operates. A passenger should be able to go from one airport to any other, taking as many flights as needed. That is, no airport should be disconnected from the rest. Considering all pairs of cities, the goal of the company is to minimize the ratio between the total travel time (sum of the times of flights taken) and the direct flight time.

That way, ignoring the waiting times in the airports, the strategy of the company is to try to minimize the biggest delay a client would have (only using flights from that company) to go from one city to another for which the company does not offer a direct flight. For each pair of distinct airports, we define the *dilation* of the pair as the ratio between the shortest path to be traversed in order to go from one airport to the other and the (euclidean) distance between the two airports. From this definition, the goal is now to find a set of flights that minimizes the greater dilation among all the pairs of airports.

This practical situation may be mathematically modeled with a weighted undirected graph, using vertices instead of airports and edges for flights. This problem is called the *Minimum Dilation Geometric Spanning Tree Problem* (MDGSTP). When graphs are used, this means that the resulting graph must be a tree, hence the name of the problem. Figure 1 shows a set of 9 flights connecting 10 Brazilian airports, the dilation of that tree is 2.08. Besides the given example, this problem is of great interest for other study areas, such as communication networks, motion planning, network design and many more (e.g. Cai and Corneil (1995) and Eppstein (2000)).

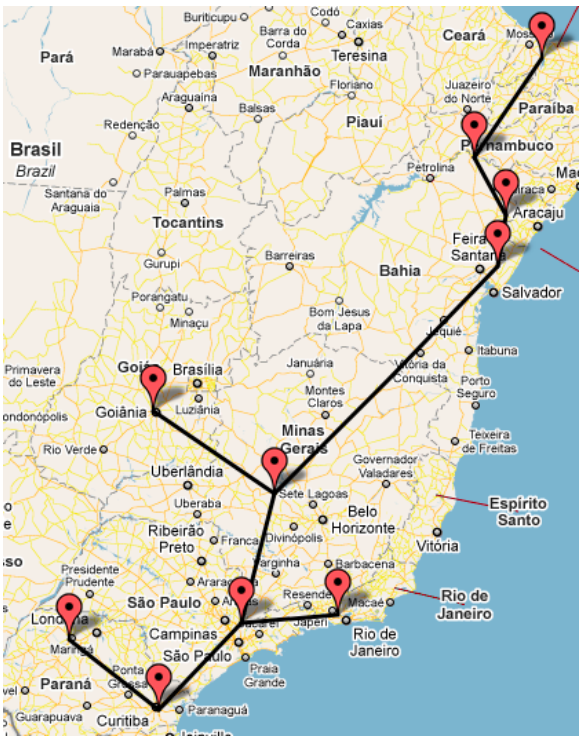

Figure 1: Example of 10 Brazilian airports connected by 9 flights, yielding a dilation of 2.08

The more general version of the problem, when the subgraph does not need to have exactly $n-1$ edges is a problem from the Combinatorial Optimization area, and has been proven to belong

to the class of $\mathscr{NP}$-hard problems by Klein and Kutz (2007). The other version of the problem (i.e. MDGSTP), when the number of connections must be exactly the number of points minus 1, is also in the $\mathscr{NP}$-hard class, as has been proven by Cheong et al. (2008).

Problems from the $\mathscr{NP}$-hard class are unlikely to be solved with a polynomial algorithm, so that other techniques, such as the one developed here, are needed to provide good solutions in a feasible amount of time.

This text is organized in the following way: section 2 has an introduction to the GRASP metaheuristic and section 3 presents a formal definition of the problem with its graph modeling. Next, in section 4 the details of the GRASP used for MDGSTP are shown, with the presentation of the steps to devise a good neighborhood for the local search. After that, in section 5, there is the comparison of the many steps of the algorithm, and how the solutions improve with each technique used. Real instances were used, and figures of the plotted solutions are shown. Finally, section 6 has some concluding remarks about this work.

2 The GRASP method

A metaheuristic is a method that guides simple heuristics so that a good solution may be found. One of the paradigms for metaheuristic development is GRASP. Among the heuristic algorithms, those based on the GRASP paradigm have been a good tool for solving hard combinatorial problems. In spite of not being able to guarantee the quality of the solution for a generic instance, many studies have shown successful applications of this method when solving complex combinatorial problems (e.g. Resende and Ribeiro (2009b)), demonstrating that it is capable of producing high quality solutions within a reasonable amount of computational time.

The term GRASP, from Resende and Ribeiro (2009a), is an acronym for *Greedy Randomized Adaptive Search Procedure*. The procedure is divided in two stages: a construction stage and an improvement or local search stage.

A greedy criteria for the construction of solutions is defined in the first stage. Despite the criteria being fixed, thanks to the introduction of randomness to process, it is not possible to know ahead of time the solution that is being generated. Hence, by repeating this step many times, a greater number of distinct solutions may be obtained, increasing the variety of the solutions created. In the construction of a solution, on each iteration a new element is inserted in the current partial solution using the greedy strategy alongside with the random process.

It is easy to imagine situations in which this greedy randomized construction does not guarantee even the local optimality of the produced solution. That is why GRASP has a second stage, when a local search algorithm is applied to the solution provided by the first step. It is based on the concept of the neighborhood of a solution s, here denoted by $V(s)$.

Once a set of rules is specified for the modification of a solution, $V(s)$ is defined as the set of all solutions obtained from s by applying these rules. The local search of GRASP starts with s as the solution produced by the construction phase. Next, a solution s' with better cost than s is searched for in $V(s)$. If it is found, s is updated with s' and the process is repeated until the search fails.

The definition of $V(s)$ is the most delicate part in the design of the local search stage of GRASP. Simpler neighborhoods tend to make the search faster and, therefore, more solutions may be visited. Yet, these solutions might not be of such a high quality. On the other hand, larger neighborhoods are more expensive to be explored, but may lead to solutions of much better cost. Hence the need of many experiments to evaluate the different alternatives of neighborhood implementation.

Another GRASP characteristic is that the steps of construction and local search are repeated for a pre-specified number of iterations (a parameter of the method that must be stipulated), and the final solution returned by the heuristic is that of best value found during the execution.

There are many ways to improve the basic GRASP method above summarized. One of them is based on the storing of the best solutions found during the iterations, called *elite solutions*. These solutions are used pairwise by the technique known as *path relinking*, which tries to generate solutions of even better quality. A detailed study about *path relinking* was done by Ribeiro, in Ribeiro and Resende (2010).

3 Problem Definition

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points in the plane. We define the *geometric graph* $G(P)$ associated with P as the undirected weighted complete graph of n vertices in which the weight of an edge corresponds to the euclidean distance between the two points represented by its end-points. Given a connected subgraph G, spanner of $G(P)$[3], for any two points, p_u and p_v, of P, we denote by $\pi_G(u,v)$ the length of a shortest path from vertex u to v on the graph G where, for every $i \in \{1,\ldots,n\}$, the vertex i of G (or $G(P)$) represents the point p_i of P. Note that the length of a path is given by the sum of the edge weights on the path.

From the above definitions, the *dilation of* u,v in the subgraph G is defined as:

$$\delta_G(u,v) = \frac{\pi_G(u,v)}{|uv|},$$

where $|uv|$ denotes the euclidean distance between points p_u and p_v of P. The dilation of the graph G is given by:

$$\delta(G) = \max_{u \neq v \in E} \frac{\pi_G(u,v)}{|uv|}$$

The dilation of G in the literature is sometimes called the *stretch factor* of G.

Another way to put it is that, if $G(P) = (V,E)$ and $G = (V,F)$ (F a subset of E), we say that the subgraph G is a *t-spanner* if the graph's dilation if less or equal to t. From this definition it is obvious that if $t' > t$ and G is a $t-spanner$, then G is also a t'-spanner.

Problems involving t-spanners are a recurring topic in Computational Geometry, mainly due to its applications in many fields, as mentioned by Eppstein (2000).

In the *Minimum Dilation Geometric Tree Problem* (MDGSTP), given the geometric graph $G(P)$ of a set P of n points in the plane, the goal is to find a subgraph G of $G(P)$ with $n-1$ edges with minimum dilation. From the number of edges, it is clear that G must be a spanning tree.

By these definitions, it should be clear that MDGSTP is related to problems involving t-spanners of geometric graphs.

4 A GRASP for MDGSTP

The first part of a GRASP is the construction phase. This is when a solution is produced for later further improvement. The main characteristics of the algorithms used for this purpose are that they should be greedy and randomized. Greedy means that it should not take much time to take decisions looking for a optimal solution, it should just take the best local option and continue. This strategy yields solutions for the next stage a lot faster. The second rule is the randomness of the algorithm, which instead of going for the absolute local optimum, also considers pieces within a certain distance from the optimum. Executing this algorithm many times should produce different solutions, delivering the crucial variety for the improvement phase.

A solution should be correct before improvement. That is, it must have all the properties of a solution as required by the problem. For example, the solutions for MDGSTP are *connected*

[3] G is a spanning subgraph of $G(P)$ if G and $G(P)$ have the same set of vertices

graphs, so the final result from the construction stage should be a graph with only one connected component. Basically, the result should be a spanning tree. When the topic is spanning trees, one quickly thinks about Prim's minimum spanning tree algorithm. It is a greedy algorithm, which is exactly what is needed here, (e.g. Cormen et al. (2001)). The problem is that it always generates the same solution, it is not random, so some modifications should be made to randomize the results.

A little more elaboration on the details of the algorithm is necessary to understand the modifications. The basic idea of the algorithm is to maintain a connected component and augment it with new vertices by a greedy decision, taking the vertex whose distance to a vertex from the connected component is minimal. The modification consists of creating a Restricted Candidate List (RCL) of vertices that are eligible to be inserted into the connected component at each step. Among the vertices in the RCL, the decision of which one is taken is arbitrary.

To construct the RCL a parameter α is used. Calling d_m the distance of the vertex that is the closest one to the connected component, the one that the original Prim's algorithm would use, the RCL has all the vertices not yet inserted in the connected component whose distances are in the range $[(1-\alpha)d_m, d_m]$. So for different values of α the size of the RCL may be different. For instance, when $\alpha = 0$ the RCL has only one element, the best one. When $\alpha = 1$, RCL has all the vertices and the algorithm is purely random. Before each iteration of the GRASP construction phase, a new value for α is randomly selected from $\{0.0, 0.1, ..., 1.0\}$ with equal probabilities.

The second part of the GRASP algorithm is the improving stage. That is where the solution just constructed is improved by local search. The neighborhood of solutions to be investigated is defined by the modifications that transform a solution into its neighbors. Such a modification should not cost too much to be applied, allowing more solutions to be visited within the same time limit. The biggest issue with the MDGSTP is that a single edge modification could change the dilation of many vertex pairs and of course the dilation of the whole graph. It is hard to predict which pairs of vertices will have their dilation affected by an arbitrary modification. This leads to the recalculation of the dilations for every pair, which is very time consuming.

A good neighborhood, therefore, should have the property that it will not affect the dilation of every single pair of vertices. Instead, it should provide an easy way to know which pairs are indeed altered, allowing the algorithm to update only the necessary pairs.

A neighborhood was devised keeping these considerations in mind. It is called *Triangle-Search*, and the reason will soon be clear.

Let (u, v) be the pair of vertices with the maximum dilation in the graph. Also, let r be another vertex defined as the tree root, $\pi_G(u, r)$ the shortest path from vertex u to the root r and $\pi_G(v, r)$ the shortest path from vertex v to the root r. We call z the first vertex of the intersection of $\pi_G(u, r)$ and $\pi_G(v, r)$. It might happen that z is the root r. We defined a as the last vertex before z in the path $\pi_G(u, r)$ and b as the analog in $\pi_G(v, r)$. Figure 2 helps locating each vertex.

From this state, two other states (neighbors) are reachable:

- Substitute edge (a, z) by (a, b), as in figure 3
- Substitute edge (b, z) by (a, b), as in figure 4

These permutations can only decrease the dilation of vertices u and v. This is easy to prove as the path goes from using two edges of a triangle to take a shortcut through only one edge. Since the graph is geometric (edge weights are euclidean distances), the triangle inequality holds, and so the final path must be shorter. Because the euclidean distance from u to v does not change, the dilation decreases.

It is obvious that these changes may increase the dilation of other pairs of vertices. Let A be the set of vertices of the subtree with root in a, B the ones of the subtree with root in b and Z all the other vertices that are not in A nor B. So, in the first case, for the permutation of figure 3,

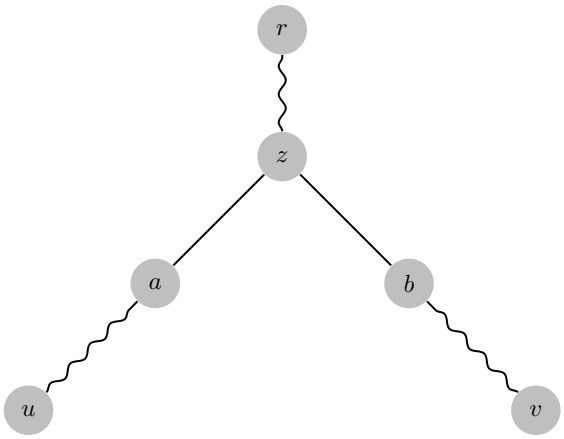

Figure 2: Illustration of the significant vertices for the neighborhood *TriangleSearch*

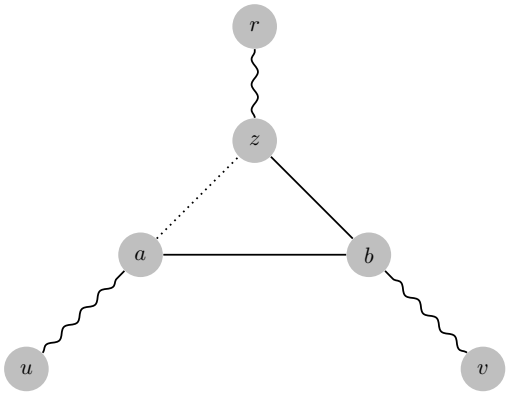

Figure 3: Substitution of edge (a,z) by (a,b) in the graph of figure 2

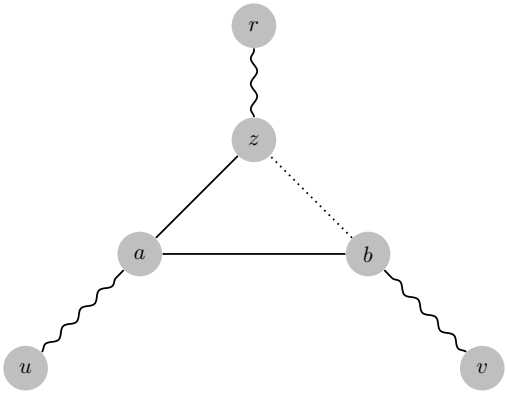

Figure 4: Substitution of edge (b,z) by (a,b) in the graph of figure 2

dilation changes only among pairs of vertices of $A \times Z$ and $A \times B$. Analogously in the second case, for the permutation of figure 4, the changes occur among pair of vertices from $B \times Z$ and $B \times A$.

These observations make it easier to verify how to recalculate the distances between the vertices after the permutation without having to analyze all the vertices pair from the graph again. In the first case, the length of the path between the pairs of vertices from $A \times Z$ and $A \times B$ may be updated using the nested loops of the algorithm 1. The update for the second case is analog.

The dilation between a pair of vertices $i \in A$ and $j \in Z$ always increases, since $dist[a][b] + dist[b][z] > dist[a][z]$. The expectation is that this value may be lower than the dilation of u and v, therefore leading to an improvement of the tree dilation.

A question then rises: which case is better? That is, which permutation creates the lowest dilation? Testing both might be to high of a cost, thus the adopted method is totally greedy. It removes the biggest edge, trying to minimize the increase of dilation between the vertices from set Z and those affected by the change, A or B.

As shown, this neighborhood has the desired property of ease to update the distances after a modification. The problem is that there are only two neighbors, the cases from figure 4 and 3. Variety is an important thing for local searching, so we want to increase the number of neighbors, trying to maintain the good properties.

Suppose now that, instead of finding a single triangle traversing the tree from u and v to the root, we use the path from u to v. Each vertex in the path could be used as if it was vertex z from figure 2 and with its two neighbors a triangle can be defined. If there are n vertices in this path (not

Algorithm 1 Updating the distances between pair of vertices during TriangleSearch

for all $i \in A$ **do**
  for all $j \in Z$ **do**
    $dist[i][j] \leftarrow dist[i][j] - dist[a][z] + dist[a][b] + dist[b][z]$
  end for
  for all $j \in B$ **do**
    $dist[i][j] \leftarrow dist[i][j] - dist[a][z] - dist[b][z] + dist[a][b]$
  end for
end for

counting the endpoints u and v), then there are n triangles. Figure 5 shows an example with $n = 3$, the dotted lines represent the 3 possible shortcuts for the 3 distinct triangles. There is also the result from taking the shortcut from u to b and disconnecting a and b. From now on this neighborhood will be called *PathTriangle*.

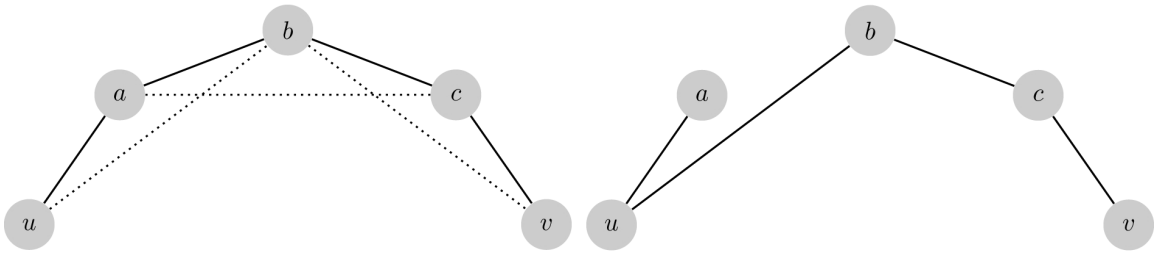

Figure 5: An example of the possible shortcuts for a path with 3 vertices and the result after one move

When faced with many possible moves it is important to have a strategy for choosing the one to go with. Two strategies are readily available, *first improvement* and *best improvement*. The former tests alternatives until it finds one that improves the solution, and then proceeds, while the latter investigates all the neighbors to find the best one, which is then chosen for the next step.

An improvement to the GRASP strategy is *Path Relinking*. The key idea is to use a pair of solutions and transform one into another through a series of moves. The hypothesis is that a better solution may be found while doing this. In order to transform a tree into another one, edges may be added and removed. Let S be the starting tree and X the goal tree. Since they are different, there must be some edge in X that is not in S and inserting this edge into S a cycle is created.There are no cycles in X, therefore some edge of this new cycle is not in X. After the removal of such an edge, the cycle is broken and the new tree S^1 is one step closer to X. After step t, the original tree S will have been transformed into S^t, and each one of these trees S^t are candidates for having a better dilation than both S and X.

5 Computational Results

The randomized construction, when repeated long enough, is expected to produce better solutions than an algorithm that always yields the same result. One should ask how better are the solutions from the randomized Prim algorithm when compared to the non stochastic results. For this purpose, 20 instances of the problem were randomly generated with 10 to 200 vertices. The randomized construction was executed 1000 times, and the best result was reported. The comparison with the dilation of the minimum spanning trees is shown in the graph from figure 6.

The graph also has the α that yielded the best solution for the randomized construction. One should notice that the value that generated the most best solutions is $\alpha = 0.8$. The improvement

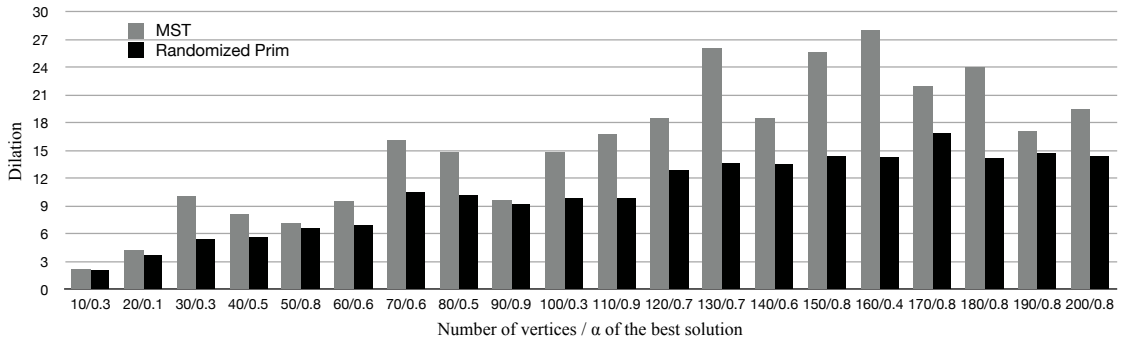

Figure 6: Dilation comparison between the Minimum Spanning Tree and the solution from the randomized algorithm

from the minimum spanning tree to the iterated greedy randomly constructed solutions is easily noticeable. For instance, the test with 160 vertices shows a decrease of dilation of 49%. The average reduction is 28.5%.

Fourteen real instances were created to test the implemented algorithms. The coordinates (latitude and longitude) of the k biggest airports in Brazil were obtained from the website *Tageo.com*, Tageo.com (2010). To sort the airports, we assumed that the runway length would be a good approximation for its size.

Every test case was executed with a thousand iterations of the GRASP algorithm, though some tests only account the construction phase rather than the construction and local search steps.

The first experiment performed is intended to see the improvement from the local search in comparison to the result of the randomized construction. The problem is that, from the earlier discussion, there are two basic ways to conduct the local search, and those are first improvement and best improvement.

The graph from figure 7 allows one to quickly see the improvement that neighborhood PathTriangle, with both the local search methods, produces in comparison with the randomized construction by itself.

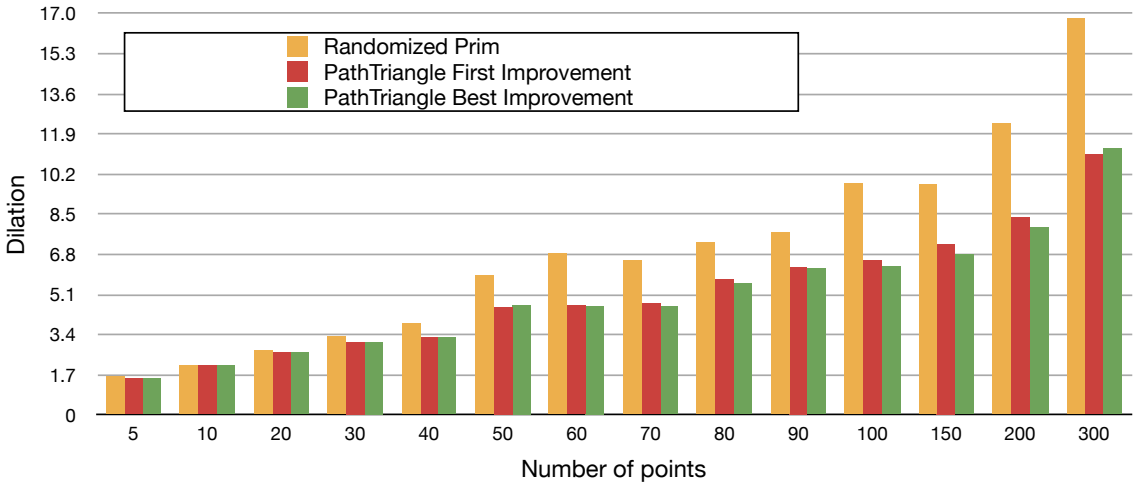

Figure 7: Comparison of the results with the randomized construction only and with the local search PathTriangle

It is easy to notice how better are the solutions after the improvement stage of GRASP,

specially for the larger instances. The biggest decrease in dilation occurs in the instance with 200 airports, with an improvement of almost 40%. The instances with 300 and 100 airports also show an improvement of about 38%. All tests conducted with local search used as starting solution the one from the randomized Prim algorithm.

The best improvement strategy is in most cases a little bit better than first improvement. As we have discussed before, best improvement is also a lot slower, since it evaluates every single neighbor before deciding on which one to go to. In fact, for some cases, running more iterations of the first improvement GRASP would produce the same solution in less time. Figure 8 shows how long each strategy took to execute. The graph is in logarithmic scale, making it easy to see how much slower best improvement is. Although best improvement provides solutions up to 6% better than first improvement, the trade-off is that it might take 6 times longer to produce the solutions.

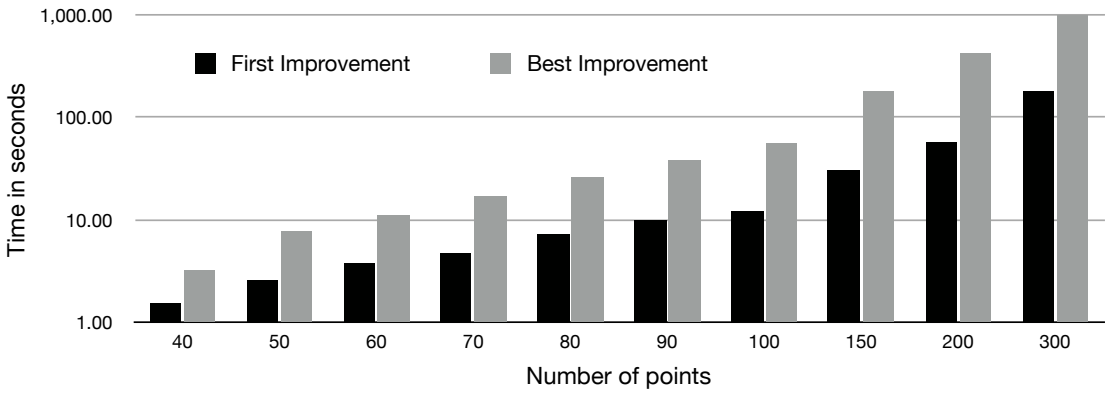

Figure 8: Time comparison of first and best improvement strategies for instances that take more than one second to execute

With the results of the GRASP in hands, many experiments with Path Relinking were conducted, but none of the strategies would yield any improvement to the solutions. This leads to a belief that the local search is very good, finding solutions that are hard to improve upon. With that in mind, a local search was deployed for each step solution. This attempt gave some improvements, as those shown in the graph from figure 9.

The elite solutions used in Path Relinking were the ones that were better than the previous best during the GRASP iterations. In other words, during the 1000 iterations of GRASP there is always a best solution, and when another best is found it goes to the elite list. The number of elite solutions selected this way varied from 4 to 13 in the applied tests. Every pair of solutions was used to generate the intermediate trees S^t.

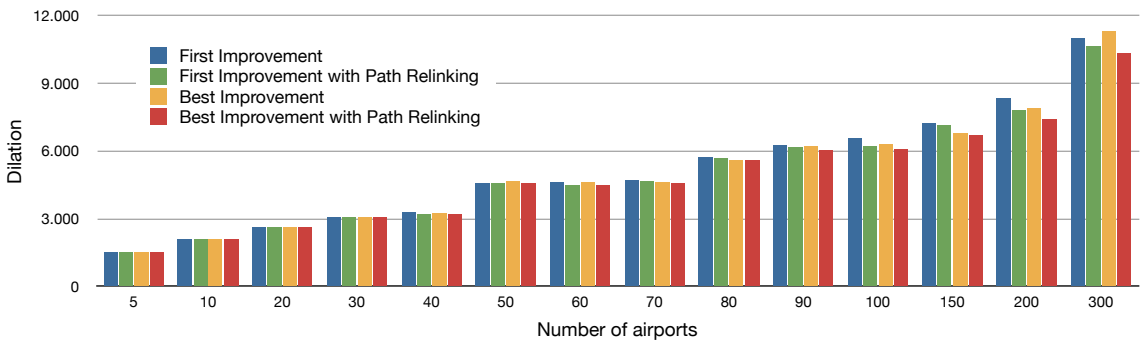

Figure 9: Results from local search and Path Relinking with PathTriangle in each step

A noticeable difference appears in the bigger instances. The one with 100 airports has an improvement of 3.4%, while the 90 points one has a gain in quality of 2.95%. The best improvement provided by path relinking is at the instance with 200 points, and it is of 6.4%.

Although path relinking will slightly improve a solution, it still is very costly. The time increase from simple GRASP to doing the same thing and path relinking is noticeable. Figure 10 has a plotting of the times it took, for both first and best improvement, to run path-relinking. The time axis is in log scale, and only those instances that took more than one second are shown.

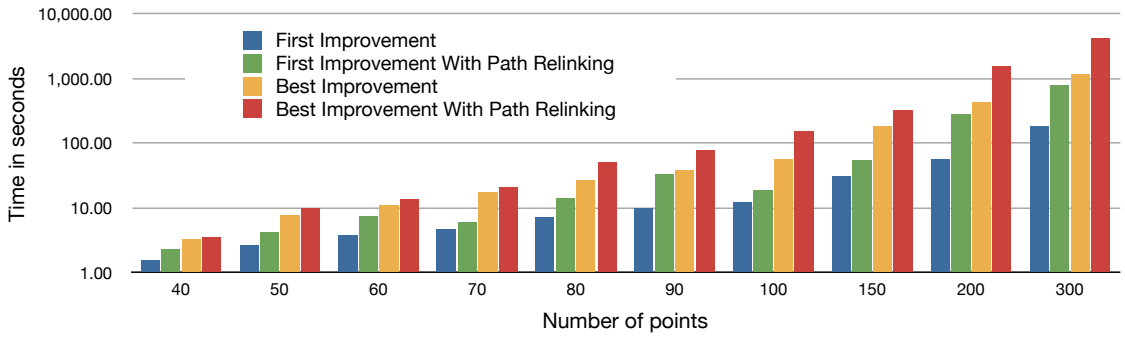

Figure 10: Time comparison of simple GRASP and GRASP with path relinking

Figure 11 shows the Minimum Spanning Tree of the instance with 60 airports, which has a dilation of 9.908. On the other hand figure 12 shows the result of the randomized construction, a modification of Prim's algorithm. This last tree has a dilation of 6.463.

The result from the GRASP algorithm with Path Relinking can be seen in figure 13, and that yielded a dilation of 4.493, an improvement of over 54%. The solutions are plotted on a map image from Google Maps, GoogleMaps (2011).

Another nice result is the largest one, with 300 airports. The minimum spanning tree and the solution obtained with GRASP using best improvement and path relinking can be seen in figures 14 and 15, respectively. The total improvement is of over 42%.

As a last example, figure 16 shows the solution for an instance with the 27 state capitals of Brazil as points. The achieved dilation is 4.206.

6 Concluding Remarks

GRASP can provide good solutions to an $\mathcal{NP}$-hard combinatorial optimization problem such as MDGSTP, though special characteristics of the problem must be explored in order to design good neighborhoods for the local search.

The PathTriangle local search we devised is quite efficient. Not only it allows us to update the data structures easily after exploring each neighborhood, but it also provides improvements of over 30% when compared to the randomized construction of a feasible solution.

Path Relinking did not yield as large improvements to the solution as the local search did, and this might be due to the high quality of the proposed neighborhood.

Although we have not found exact solutions in the literature for the MDGSTP that could be used to assess the results obtained by our heuristic, we are aware that algorithms to produce lower bounds for the problem are currently being developed by Brandt and de Souza (2011). Thus, as a future work, it would be interesting to compare the dual bounds reached by these algorithms with those yielded by the heuristic proposed here.

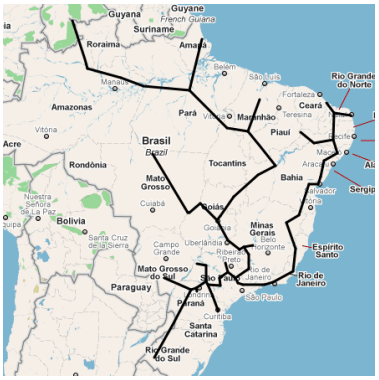

Figure 11: Minimum Spanning Tree for the instance with 60 points, dilation 9.908

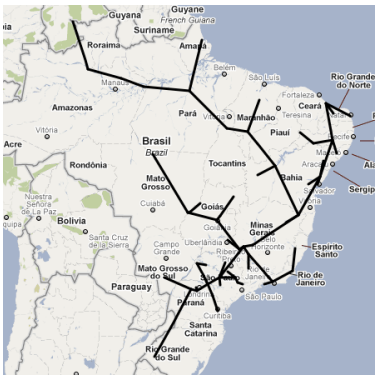

Figure 12: Tree obtained with the randomized construction which has dilation 6.463

Figure 13: Resulting tree from GRASP and Path Relinking, with dilation 4.493

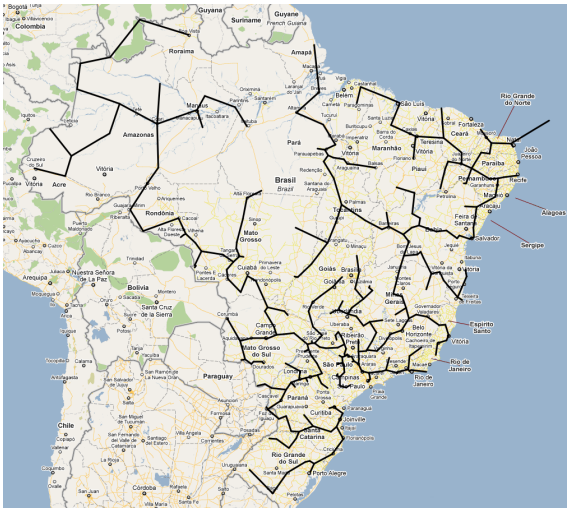

Figure 14: Minimum Spanning Tree for the instance with 300 points, dilation 18.053

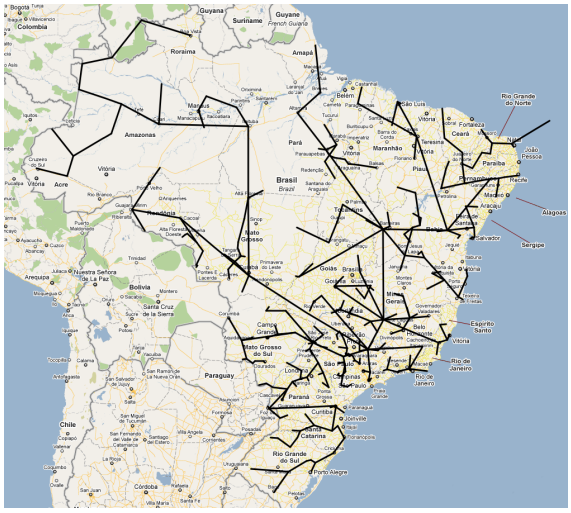

Figure 15: Solution from the GRASP with best improvement and path relinking, dilation 10.332

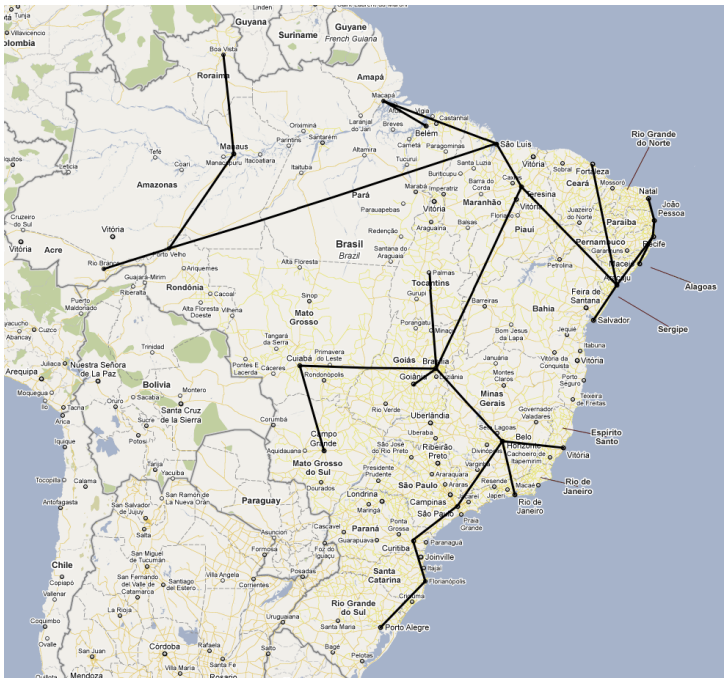

Figure 16: Solution for an instance with all the 27 capitals of Brazil as points. Dilation 4.206

References

Brandt, A. and de Souza, C. C. (2011). Private communication.

Cai, L. and Corneil, D. (1995). Tree spanners. *SIAM Journal on Discrete Mathematics*, 8(3):359–387.

Cheong, O., Haverkort, H., and Lee, M. (2008). Computing a minimum-dilation spanning tree is NP-hard. *Comput. Geom. Theory Appl.*, 41(3):188–205.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press, Cambridge, MA, second edition.

Eppstein, D. (2000). Spanning trees and spanners. In Sack, J.-R. and Urrutia, J., editors, *Handbook of computational geometry*, chapter 9, pages 425–461. North-Holland Publishing Co., Amsterdam, The Netherlands.

GoogleMaps (2011). www.google.com/maps.

Klein, R. and Kutz, M. (2007). Computing geometric minimum-dilation graphs is NP-hard. *Lecture Notes in Computer Science*, 4372:196–207.

Resende, M. G. C. and Ribeiro, C. C. (2009a). GRASP. In Burke, E. and Kendall, G., editors, *Search Methodologies*. Springer, second edition.

Resende, M. G. C. and Ribeiro, C. C. (2009b). Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In Glover, F. and Kochenberger, G. A., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research and Management Science*, chapter 8, pages 219–249. Springer, second edition.

Ribeiro, C. and Resende, M. (2010). Path-relinking intensification methods for stochastic local search algorithms. *Computers and Operations Research*, 37:498–508.

Tageo.com (2010). www.tageo.com.