An Hybrid Genetic Algorithm for Computing (near)Optimal Euclidean Steiner Trees *

Jorge Barreiros [1],[2]
[1] Departamento de Engenharia Informtica e Sistemas,
Instituto Superior de Engenharia de Coimbra
[2] Centro de Informática e Sistemas da Universidade de Coimbra
jmsousa@isec.pt

Abstract

In this paper we present an algorithm for finding an approximation to the Euclidean Steiner Tree for a given set of terminal points. This is defined as the shortest length geometric construction that unites all the terminals. Our algorithm dynamically partitions the point set into multiple, separately optimized subsets. The Steiner tree for these subsets is constructed by running a highly modified genetic algorithm.

1 Introduction

The Euclidean Steiner Tree (EST) problem concerns the construction of the minimum length tree that interconnects a given set of points. Although exact algorithms are known, it has been proved that the EST problem is a NP-hard problem and so heuristics and approximation methods are necessary in practice. In this work, we present an hybrid genetic algorithm that offers a (near)optimal solution for the problem. The preliminary results we obtain are encouraging enough to support further investigation about this approach. The paper is organized as follows. In section 2, we introduce Steiner trees and the Steiner Tree Problem (STP). Next., in section 3, we made a brief reference to genetic algorithms. In section 4, we present our algorithm, while in section 5, we show the results. Finally, in section 6, we draw some conclusions and point for future work.

2 Steiner Trees

The Steiner Tree (ST) of a set of points P is the shortest path that interconnects all points belonging to P. This tree is the Minimum Spanning Tree (MST) of $P \cup S$, where S is a set of points aptly named as Steiner Points (see Figure 1). Therefore, the Steiner problem (the determination of the Steiner tree for a

*This work was partially supported by the Portuguese Ministry of Science and high Education under Program POSI.

given set of points) can be formulated has the search for the set S that minimizes MST(P∪S). Depending on the nature of the space where the points and corresponding connections exist, the problem description can be further refined by distinguishing between the Euclidean and rectilinear Steiner Problems. In this work, we present a novel algorithm based on a heuristic genetic algorithm to find a (near)optimal solution for the Euclidean Steiner Problem (ESP). The ESP is related to many engineering applications, like network or VLSI routing. For further details on Steiner Trees and related issues, consult [4, 3]

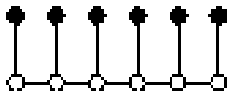

Figure 1: Abstract model of a Steiner Tree. Black circles represent elements od the set P, while white circles are Steiner points.

3 Genetic Algorithms

Genetic Algorithms (GA) are stochastic search procedures inspired by the Darwinian principles of natural selection and the ideas of Mendelian genetics. A GA starts with a randomly initialized population of candidate solutions and implements probabilistic and parallel exploration in the search space using the domain-independent genetic operators of selection, crossover and mutation ([5, 7]). A GA associates each individual candidate in the population with a fitness which measures the quality of a solution. Selection chooses individuals probabilistically, according to their fitness. The higher the fitness, the more likely it is for an individual to be selected. Genetic operators, like crossover and mutation, produce new individuals: for instance, crossover exchanges genetic information between two selected parents; mutation randomly changes one gene value to the generated offspring. The GA searches through an iterative process: the process of one generation involving selection, crossover and mutation is called one cycle of iteration and is repeated until convergence is reached or the number of generations achieves the established limit (see Figure 2).

---

1. **Randomly** initialize population
2. **Do**
   - **a) Evaluate** Population
   - **b) Select** parents
   - **c) Crossover**
   - **d) Mutation**
   - **d) Substitute** old population

   Until (*DONE*)

Figure 2: The simple Genetic algorithm

There are many variants of this general scheme.

4 The Algorithm

Our algorithm has two layers of depth. A top level algorithm, that makes high level partitioning decisions, and a low level algorithm, that solves the ESP for any given partition. The low level algorithm is capable of easily finding optimal or near-optimal solutions for most simple ESP problems of low dimension, while the top level algorithm makes intelligent choices about partitioning of the problem point set in multiple, separately optimized partitions. Partitions are created or recombined whenever deemed necessary, according to the evolution of the search process. It is the combination of these two steps that makes our approach capable of dealing with any type of ST. In the following, and due to lack of space, we only present the main ideas behind each one.

4.1 Low level genetic algorithm for solving the ESP

Let us start by explaining the low level algorithm. This is a modified GA that finds an approximate ST for any given set of points. This algorithm is specially useful for ST with points sets of *small* dimension. We will use the abstract model shown in Figure 1. As we can see number of Steiner points is equal to the number of the problem points or *nodes* [1].

In this algorithm, two consecutive Steiner points, according to the model represented in 1 are always considered to be connected (ie., these points form a "backbone" of the ST) [2]. On the other hand, the connections between nodes and Steiner points are determined dynamically. Every node is always considered to be connected to the nearest Steiner point. The GA task is to find the backbone of Steiner points (from now on referred to only as "backbone") that offers the best solution.

Genetic Operators

The genetic operators are grouped in two categories, according to their capability of disruption. The operators with higher disruption are applied with higher probability. This tries to follow "typical" values of mutation and crossover used in standard GA's (less then 10% for mutations (low disruption operator) and around 80% for crossover (high disruption operator)).

The following genetic operators were used:

First category:

- *Mutation*: a Steiner point is selected randomly and moved to a random location.
- *Crossover*: One point simple crossover of two backbones

[1] Although theoretically it has been shown that only n-2 Steiner points are necessary to create the correct ST for n nodes, using n Steiner points enables us to make some simplifying assumptions on our algorithm. These simplifications do not reduce the potential quality of the solution, because nothing prevents that two pairs of these points occupy the same physical location.

[2] These nodes will be referred to as being *neighbors*

- *Snap*: The Steiner points rotate a random number of positions in a random direction along the backbone.
- *Swap*: Two Steiner points exchange positions along the backbone.

Second category:

- *Nudge*: A random Steiner point is moved randomly a small distance.
- *Slide*: A randomly selected Steiner point moves towards or away from one of it's neighbors (along the backbone). This movement has an random amplitude between –50% and 50% of the distance between those points.
- *Compress*: A Steiner point is selected randomly and moves towards the node that he is attached to. This movement ranges from 0 to 100% of the distance between those two points.

Some of these operators are illustrated in Figures 3 and 4.

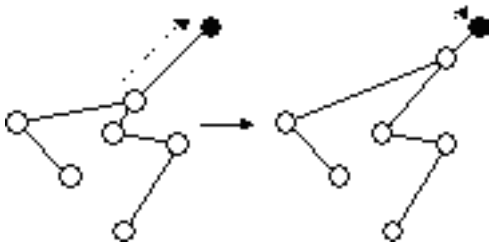

Figure 3: The operator compress

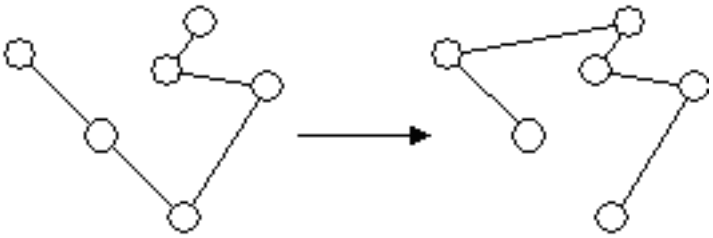

Figure 4: The operator snap

Fitness

The fitness of a solution is given by the length of the ST. The total length is equal to the sum of the length of backbone plus the length of the connections between the Steiner points and nodes.

Selection

Whenever it is necessary to select a solution to generate a new element for a new population, each solution `i`, from the old population, can be selected with probability:

1. **If** the algorithm is in suspended state **then** stop.
2. **Let** the best element from the last population become the first element of the new population
3. **Create** a new element for the new population by selecting one element from the old population and
   a) **Choose** one among *mutation*, *crossover*, *snap* and *swap*. **Apply** that operator with probability 80% **if** *snap* or *crossover* was chosen, **else** apply operator with 10% probability.
   b) **Choose** one among *nudge*, *slide* and *compress*. **Apply** the chosen operator with probability 10%.
4. **If** the total length from the best solution didn't changed within a pre-defined number of generations **then** the algorithm enters in suspended state.

Figure 5: One step of the modified GA

$$\frac{1}{(D_i+1)\sum_k \frac{1}{D_k+1}}$$

where D_n is the total length of the ST represented by the solution $\mathtt{n}$.

The modified GA: one step of iteration

A modified GA algorithm is used to find the backbone that generates the lowest length interconnection tree. The Figure 5 explains one step of the iteration process of the GA. We say that the state of the GA is suspended when the solution's length doesn't change for a predetermined number of iterations [3].

4.2 Top level algorithm

Unfortunately, the simplicity of the model of the previous section has a serious limitation: it can't represent adequately star or tree-shaped backbone configurations. Although it would be possible, in theory, to use more Steiner points to enable more flexible constructs, this approach would make the model less suitable to the incremental changes required by the optimization algorithm.

This limitation is solved by using a top-level algorithm that partitions the node set into smaller subsets whose ST is computed independently. This partitioning, made dynamically during the execution of the algorithm, is based on well known properties of Steiner trees, and will enable the use of a different backbone for every subset. The combination of these multiple backbones can represent the tree or star-shaped structures that are missing from the basic ST model. Additionally, this partitioning also offers advantages concerning the efficiency of the algorithm.

[3] After entering this state, the algorithm won't make any further evolution, but all the information for resuming computation is kept. This is important because the suspended state can be reset by the top level algorithm.

The algorithm

The top level algorithm is presented in Figure 6. Due to lack of space we cannot enter into many details and explanations and concentrate only on the most important aspects.

Let P be the set of nodes for which the Steiner Tree should be computed:

---

1. **Determine** the processing order of the points in P [4].
2. **Let** G be an empty set of low level GAs.
3. **Create** low level GA, with first 4 points in P, and add it to G.
4. **If** a pre-defined number of generations is reached **or** the result converge **then stop**
5. **If** the number of solutions is multiple of a pre-defined constant **then add** next point in P to the closest GA in G.
6. **If** the number of generations is multiple of a pre-defined constant **then check** the GAs in G for the possibility/necessity of recombining 2 or more distinct GAs in a single GA. At the same time, **check** if the suspended state of any GA in G should be reset [5].
7. **If** the number of generations is superior to a certain constant, **then for each** element of G, with a pre-defined probability , **consider** the possibility of dividing it in two subproblems or partitions.
8. **Do** a single iteration in all GAs in G.
9. **Goto** 4.

Figure 6: The top level algorithm

Partitioning

After an initial transient phase is passed, the problem point set is divided in multiple subsets that are optimized independently. This partitioning, is a crucial step, and is achieved by applying algorithm in Figure 7, with a predefined probability, to every low-level GA (LLGA).

For a given *low-level* GA (LLGA), search for the best individual and do the following:

1. **Choose** randomly two neighboring Steiner points A and B.
2. **Let** PA be the set of nodes connected to Steiner points closer (along the backbone) to B then A
3. **Let** N_d be the node in PA closest to A
4. **Let** N be the set of all nodes
5. **If** the distance between A and PA is smaller than the distance between A and B, **then Divide** the GA into two new GA that optimize the following point sets: PA **and** $(N|PA) \cup \{N_d\}$, respectively [6].
6. **One individual** in each one of these two new GAs is **initialized** with the corresponding Steiner points from LLGA.
7. **Destroy** LLGA

Figure 7: The partioning algorithm

Figure 8: Step 1

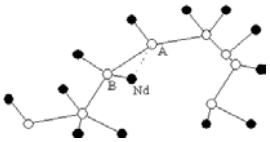

Figure 9: steps 2 and 3

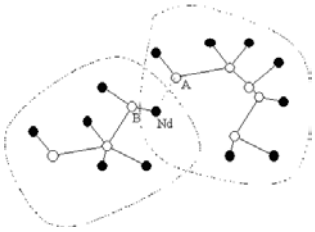

Figure 10: Steps 4, 5 and 6

The Figures 8, 9 and 10 ilustraste the behavior of the algorithm.

Recombining GAs

Two low-level GAs that solve the ESP for the point sets P_1 and P_2 will be recombined, when necessary, simply by creating a new low level GA that solves the problem for the point set $P_1 \cup P_2$.

5 Results

To test the algorithm, some test sets from the OR-library (see [1]) test suite were used. These test sets are compiled from [2, 6]. We present the results for the full 10 and 100 points sets described in [2] and some results from higher dimension sets, as well the results for the Soukup and Chow test problems described in [6].

In the following tables we display the reduction in length when compared to the length of the MST (a known upper bound for the length of the ST), both for the exact ST and the GA computed ST. The *Performance* column displays the ratio between the previous two values.

The Table 1 shows the results for the test set with 10 points. The algorithm was executed a variable number of times for each case, 5 times or less on most cases and up to 10 on the hardest problems. In some of these cases performance rises above 100% (which should be impossible), likely because of roundoff errors

(1/400) introduced when converting the normalized float coordinates in which the test set is stored into the integer coordinates used by our implementation [7].

Problem Nr.	**CPU (s)**	**Red. Best**	**Red. GA**	**Performance**
1	5	4.23%	3.56%	**84.1%**
2	5	0.47%	0.46%	**98.7%**
3	5	4.34%	4.34%	**100.0%**
4	5	1.13%	1.12%	**99.7%**
5	6	2.47%	2.47%	**100.0%**
6	4	4.70%	4.70%	**99.9%**
7	5	7.84%	7.84%	**100.0%**
8	6	6.76%	6.75%	**100.0%**
9	4	2.39%	2.08%	**87.1%**
10	9	2.03%	2.03%	**100.0%**
11	5	5.47%	5.46%	**99.8%**
12	8	0.56%	0.56%	**100.2%**
13	6	6.09%	6.09%	**100.1%**
14	9	5.64%	5.69%	**100.8%**
15	4	3.04%	2.79%	**91.8%**

Table 1: Results for 10 point set (esteiner10)

In the following figures we can observe, for problem 14, the Steiner tree referred to as optimal in the test set description and the slightly improved version found by the algorithm (probably due do rounding errors, as it has already been discussed).

The results for the 100 points set are presented in Table 2. The algorithm was executed twice for each test and the best result is shown here.

The following figures show, for the 100 points problem number 2, the optimal solution and the solution found by the GA.

The results for the 46 Soukup and Chow tests are summarized in Table 3. we used 5 to 10 runs per test case, depending on the difficulty of the problem.

To test the algorithm's behavior with the test sets with higher dimension, one single test was run for the first problem of the 250, 500 and 1000 point sets. The data for the exact ST was taken from [3]. The results are presented in Table 4.

These results are encouraging, because the algorithm offers reasonable performance even with higer dimension sets. The algorithm is faster than exact ESP algorithms (see [3]) without considerable loss of quality of the solution.

6 Conclusions

We presented a hybrid genetic algorithm based into a decomposition into two layers of depth. The algorithm achieved good results and is reasonably efficient. We hope that with better tuning and additional testing we will be able to increase performance and reduce execution time even further. Additional tests

[7] All tests were run on a Pentium 3 processor at 866MHz running MS Windows 98.

Problem Nr.	**CPU (s)**	**Red. Best**	**Red. GA**	**Performance**
1	60	3.20%	2.92%	**91.4%**
2	78	3.40%	2.88%	**84.8%**
3	80	3.39%	3.05%	**90.0%**
4	73	3.19%	2.84%	**89.0%**
5	103	3.29%	3.11%	**94.6%**
6	105	3.36%	2.75%	**81.6%**
7	107	4.00%	3.43%	**85.7%**
8	88	3.49%	3.08%	**88.1%**
9	80	3.49%	3.13%	**89.8%**
10	84	3.32%	2.93%	**88.3%**
11	77	2.78%	2.58%	**92.8%**
12	70	2.59%	2.42%	**93.5%**
13	101	2.61%	2.21%	**84.5%**
14	64	3.55%	2.91%	**81.9%**
15	109	2.71%	2.01%	**74.1%**

Table 2: Results for 100 point set (esteiner100)

Average Performance	92.0 %
% of tests where performance reaches 100%	78.6%

Table 3: Summary of results for Soukup and Chow test set

Problem Nr.	**CPU (s)**	**Red. Best**	**Red. GA**	**Performance**
250-1	450	3.08%	2.25%	**72.96%**
500-1	2024	3.42%	2.76%	**80.77%**
1000-1	13728	4.45%	2.64%	**76.40%**

Table 4: Results for higher dimension sets

can also be included for improving the early detection of erroneous constructs (e.g. the luna and/or bottleneck tests). Some of the test cases seemed to be extremely sensitive to minor variations caused by rounding errors, so that issue must be addressed also in the future.

References

[1] J. E. Beasley. Or-library: distributing test problems by electronic mail. *Journal of Operational Research*, 41, 1990.

[2] J. E. Beasley. A heuristic for euclidean and rectilinear steiner problems. *European journal of Operational Research*, 58:284–292, 1992.

[3] M. Zachariasen D. Warme, P. Winter. Exact algorithms for plane steiner tree problems: a computational study. Technical Report DIKU-TR-98/11, University of Copenhagen, 1998.

[4] P. Winter F. K. Hwang, D. S. Riochards. *The Steiner tree problem.* Elsevier Science Publishers B. V., 1992.

[5] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison Wesley, 1989.

[6] W. F. Chow J. Soukup. Set of test problems for the minimum length connectoin networks.

[7] M. Mitchell. *An introduction to Genetic Algorithms.* MIT Press, 1996.