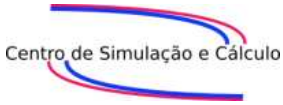

Steiner Trees Optimization using Genetic Algorithms

(Technical Report)

Mário Jesus mjesus@ualg.pt
Department of Civil Engineering, University of Algarve, Faro, PORTUGAL

Sérgio Jesus sjesus@ualg.pt
Department of Electrinics and Computer Science, University of Alagrve, Faro, PORTUGAL

Alberto Márquez almar@us.es
Department of Applied Mathmatics I, University of Seville, Seville, SPAIN

Abstract

We present a practical method to approximate good solutions to one of the most studied and difficult optimization problems, the Euclidean Steiner Tree Problem for a large number of points. This method is developed using a genetic algorithm, as the main optimization tool, which was enhanced by some specific heuristics originated from Computational Geometry for the most part.

The intrinsic complexity of the problem and the large amount of points that we propose to deal with, requires a specific approach in order to obtain good solutions considering its practical use. To fulfill this objective, our genetic algorithm operates on a previously subdivided input instance. A clustering technique and the use of some geometric operators, both contribute to bound the error of the approximations obtained by our technique.

Keywords

Genetic algorithms, geometric optimization, Steiner trees, clusterization.

1 Introduction

Space and time requirements are two fundamental characteristics to classify computational problems. On top of this classification we can find among others the NP-hard problems, those that possess an intrinsic complexity that only allows approximated solutions. Particularly if the problem in hands proves to be very difficult to be solved, it is acceptable to establish a bound for a solution to be admitted as suitable.

The Euclidean Steiner Tree Problem (ESTP) belongs to that list and it is an NP-hard geometric combinatorial optimization kind of problem (Garey et al., 1977). It can be stated as follows:

"Given a set of N points in the Euclidean plane, find out a minimum overall distance to connect all of them, considering the addition of some extra points."

In this article we present a new approach for the ESTP when a large number of points are required to be processed, considering that a useful (in practical terms) answer will be produced in an acceptable amount of processing time. To reach our objective, we propose the use of a genetic algorithm (GA) adjusted to the problem by some geometric operators.

The ESTP has been broadly studied since its formulation, and many work has been published reporting several heuristics or methods used to derive 'good' solutions. Those methods can be divided in two classes: deterministic and random ones. Important insights to the problem using geometric heuristics or algorithms are presented by (Melzak, 1961; Winter, 1985; Hwang, 1986; Kou et al., 1981; Beasley, 1989; Rayward-Smith and Clare, 1986; Dreyer and Overton, 1996; Warme et al., 1998). Also some special techniques were developed by (Smith, 1992) using numerical methods or by Lundy who used a simulated annealing algorithm as stated in (Hwang et al., 1992). Previous works involving genetic algorithms can also be reported, as in (Hesser et al., 1989; Kapsalis et al., 1993; Esbensen, 1995). In general, all the previous mentioned works use a fixed small amount of points or they know the number of Steiner points in advance. However, in many practical situations these assumptions are not valid, thus a new approach must be proposed.

The new approach presented here is based in a two-step strategy. In the first step, we speed up a Genetic Algorithm (see (Goldberg, 1989; Holland, 1992; Michalewicz, 1996)) using some results borrowed from Computational Geometry (see (Preparata and Shamos, 1985; de Berg et al., 1997; O'Rourke, 1998; Tucker Jr., 1997)) in order to compute a good approximation of the MStT for a medium-size set of points. It must be pointed out this first step provides a very competitive output for that kind of set of sites. In the second step, we show how to use a clustering technique in order to fulfill our final goal.

In spite of its effectiveness, no previous work is known that uses an *a priori* domain's partition, performed according to the ESTP instance. The outcome of this preprocessing phase will be a set of clusters of points. Each cluster will originate a sub-Steiner tree that will belong to the final answer, due to the appliance of a final merge process. This is an important achievement, since it allows a substantial reduction of the overall processing time for the ESTP. It is known that instances of the problem with some hundreds of points requires hours of processing time, even using 'good computers'. We must emphasize also that the processing time is not our only concern, since some accuracy is also needed for answers to become useful even for practical use.

Thus, the processing time and the accuracy of the answers for the instances of the ESTP with large number of points will be the two guidelines that supports the development of a computational model, the cGa model. Prior to its use with large instances we will prove its robustness and precision with smaller instances of ESTP. A suitable balance between time and accuracy is achieved due to the development of special adaptations for the genetic algorithm to the ESTP.

Most of this article is devoted to describe and prove the robustness of the computational model cGa. This is the main concern for the next two sections, where a special attention was taken with the adaptation of the genetic algorithm to the ESTP, by means of special geometric operators. In the forth section some results are presented and compared with other results derived from theoretical studies or from well studied instances of the problem.

Section five presents a different insight of the ESTP, since a subdivision of the prob-

lem domain is proposed where a partial Steiner tree is derived for each subset of points. The final answer of the ESTP is obtained, thanks to a merging process. The article will end with the presentation of some ideas to be used in future works.

2 Preliminaries

2.1 ESTP Characterization

The Euclidean Steiner Tree Problem is the second most studied optimization problem, just following the Traveler Salesman Problem. It is an NP-hard problem that intrinsically depends on the topology of the instance in study, meaning that two different instances with the same number of points may have different Minimum Steiner Trees (MStT) even if one instance is obtained from the other with a small perturbation of some of its points.

Given a set S of N points in the plane, it is very well-known that the minimum spanning tree (MSpT) (the tree with minimum length that has S as its vertex set) can be computed in $O(N \log N)$ time. In fact, a Steiner Tree (StT) of S is obtained by adding to the original set S some special located additional points (the Steiner points) and computing the MSpT of this new collection. Of course, the number and the location of those points are unknown in advance and no rule can be established about it. The effect of this addition is the shortening of the total length of the MSpT.

Obviously, the Minimum Steiner Tree (MStT) is the shortest among the StT's for a given set of points. In fact, it is an extremely difficult geometric combinatorial problem. Figure 1 shows a set of points, its MSpT, and its MStT.

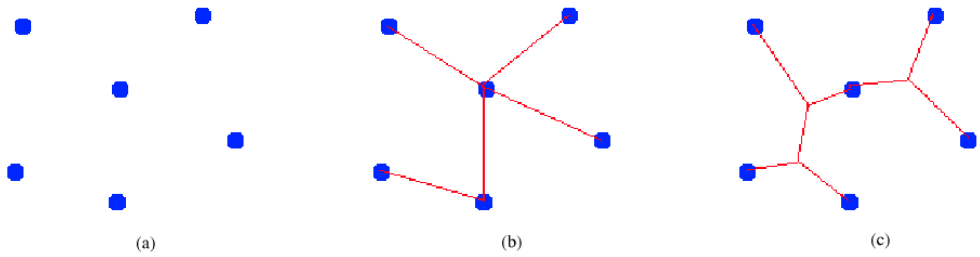

Figure 1: A set of points, its MSpT, and its MStT.

However, studies conducted on the ESTP provided some guidelines that we will use in this work, (Hwang et al., 1992). We must point out the following ones:

1. *Degree requirements* of Steiner points in a MStT:
   (a) No two edges can meet at an angle less than $120°$.
   (b) A StT has no crossing edges.
   (c) Each Steiner point has degree three.
   (d) Each Steiner point has among its adjacent at most another Steiner point.
2. The Steiner hull (the region of the plane where all Steiner vertices lie) of a set of Euclidean points is included in the convex hull of that set.

3. MStT is a global minimum, thus corresponding to the shortest StT for a given set of Euclidean points.
4. A topology is a Steiner topology if it meets the degree requirements of a MStT and can be uniquely partitioned into edge-disjoint subgraphs, being each of them a full Steiner topology.
5. There is an upper bound of $N - 2$ for the maximum number of Steiner points in any ESTP instance (N is the number of points in the instance).

As we will see, the first item will be crucial in order to accelerate the convergence of our algorithms. Item 2 allows us to restrict the domain of our problem bounding the search space for our genetic algorithm to evolve, thus increasing its effectiveness. In item 3 it is said that in each ESTP instance several local minima may exist scattered in the domain, but only one global minimum exists. The item 4 guarantees the existence of a relative minimal StT in each partition of the instance. This observation is crucial since it allows the new approach for the ESTP with a large number of points conducting to a significant reduction in the processing time that we propose in Section 5. The last item establishes an upper bound very useful in the representation of the problem inside the GA.

NP-hard problems only allow approximate solutions, which must be evaluated by some criteria. According to this and when no other alternative remains, we will use the theoretical Steiner ratio as a quality measure to appreciate the solutions provided by our computational model. It is defined as the minimum relation between the lengths of a minimum Steiner Tree and a minimum spanning tree (Hwang et al., 1992; Gao et al., 1994). For a set of Euclidean points N, the following must be observed:

$$\rho(N) = \frac{|MStT|}{|MSpT|} = \frac{\sqrt{3}}{2} \approx .8667, \tag{1}$$

where $|MStT|$ and $|MSpT|$ are the lengths of those trees and ρ is the Steiner ratio.

2.2 Genetic Algorithms Features

Based on some Darwinian ideas of the species' evolution, a genetic algorithm mimics an evolutionary process in a computer. It is developed under the idea of the survival of the fittest and uses identical Nature terms, as the operators commanding the evolutionary process are (Holland, 1992). GAs had been broadly used, both as a stand alone computational tool or in combination with other heuristics, in a wide class of problems as: optimization, scheduling, modeling and so on.

An initial formulated process is submitted to several repeated iterations — evolutions — during some time, expecting to find an answer for a given problem. Vaguely speaking such systems maintain a population of individuals, the potential solutions, and applying some internal processes of selection and recombination of the represented information, they evolve toward a global optimal solution (Goldberg, 1989; Michalewicz, 1996).

However, the no-free-lunch theorems (NFL) (Wolpert and Macready, 1996; Wolpert and Macready, 1997) are addressed to the evolutionary algorithms, ensuring the hopelessness in the existence of a magic algorithm to solve any problem, or in one that outperforms all others. Unless there exists some suitable operators that are correlated to the features of the problem, there is no reason to believe the algorithm will do better than a random search. Putting in other terms, it is necessary to 'tailor' our genetic algorithm to the ESTP, to became useful in this context (Culberson, 1998).

With this ideas in mind and following a careful analysis of the ESTP, we decided to tailor our GA ,mainly in the following three capital areas:

1. Codification and representation of the problem.
2. Performance (time and accuracy) of the algorithm.
3. External independence of the operators.

The modifications introduced in the genetic algorithm, are supported by some recent theoretical studies and by the observation of its performance and behavior, when experiencing with some instances of the ESTP.

Solutions are random generated in the beginning of the process and submitted to a series of internal transformations afterward, favoring the construction of the optimal solution. The choice of a suitable codification of the solution for the ESTP must be found, otherwise a malfunction of the algorithm will happen or worse, incorrect solutions could be thrown out.

Among several dificulties associated to the problem itself, as its *hardness*, the GA must deal (avoid) also with the epistasis, which is a phenomena associated with the codification of the problem and responsible for some drifts of the final solution (Jong et al., 1997; Reeves and Wright, 1995). An outcome suffers from epistasis when some interrelated forces between genes appears, conducting to the formation of blocks inside the genome. These blocks can contribute negatively to the building of the optimum solution, leading to the deception. This misleading effect is being studied and some results points out that a possible cause for the presence of epistasis could be related with the bitwise representation of the problem, (Heckendorn and Whitley, 1999; Naudts et al., 1997). As it will explained later in section 3.1.3, our GA will surpass this problem using the real codification in the genome representation of the solutions.

Item 2 is important due to obvious reasons. It is known that GAs spend a lot of time within the iterative process, most of it resulting from the fitness evaluation of identical genomes. Significant reduction in the overall processing time can be achieved if the evaluation occurs only with the new offsprings.

Another drawback is the external dependence of the operators, meaning that each kind of instance of the ESTP requires the tunning of the operators to get a good behavior from the GA. This is true but it is a timing consuming iterative process — trial and error — that is unacceptable in practical terms. A new approach is reported by the use of the number of the inputed points of each ESTP instance.

All this perspectives are implemented in a computational program — the cGa model — that will be the main concern of the next section.

3 Model description

This section is subdivided in two parts. In the first subsection we begin to describe how we succeed to model our GA to the ESTP and prove that the cGa performs well with different instances of the problem. The next subsection is devoted to report the behavior of the cGa with instances having a large number of points.

3.1 Development and testing of the cGa model

3.1.1 Objective function

A level of performance, known as fitness, must be assigned to each genome (solution). This is ensured by a fitness function, which returns a value for each genome in the pop-

ulation. It is this value that will be used to differentiate the most well-suited genomes, from those who are not.

$$f(StT_i) = \sum_{n=1}^{|StT|_i - 1} (min_{i \neq j}\{dist(x_i, x_j)\}, \forall x_m \in StT_i)_n \quad (2)$$

Formula 2 formalizes the fitness function used in cGa, which is the minimum spanning tree length estimation, for short. It uses the Euclidean distance between two points, denoted by $dist(x_i, x_j)$.

Unfortunately, an estimated value is not always a useful quantity to guide the genetic search because it can reflect unreal situations, thus misleading the evolution of genetic algorithm. To avoid this situation, the cGa model uses a linear scaling, protecting itself against premature convergence. It is an explicit fitness re-mapping (Goldberg, 1989; Michalewicz, 1996).

3.1.2 Representation of the ESTP

Genetic algorithms always departs from an initial population of genomes — a set of solutions — generated randomly. These genomes are submitted to an iterative process of modifications due to the action of some suitable operators, until some criteria is reached. The way used to represent a (solution of the) problem inside the GA is crucial and it will determine the optimum reachability and accuracy (Goldberg, 1989; Holland, 1992; Michalewicz, 1996).

Geometrically, we can devise two kinds of points in the ESTP: the initial set provided by the instance of the problem, and the Steiner set of points. The former set is is known in advance and it is fixed. The later one is unknown in number of points as much as in their location in the Euclidean plane, and they are the changeable part of the problem.

To depict a solution inside our computational model, each genome represent the set of Steiner points, which will be added to the inputed instance, both contributing to the future Steiner tree. The problem can be formalized in the following terms:

$$StT = ESTP \cup StPTS \quad (3)$$

The Formula 3 states that a Steiner tree (StT) is the union of the sets of points from the ESTP instance and the Steiner points (StPTS) coded in each genome (Jesus et al., 1999).

3.1.3 Genome encoding

In our computational model, each genome may have a maximum of $N-2$ Steiner points and each one is expressed as a linear combination of the points belonging to the Steiner hull (Jesus et al., 1999).

The Figure 2 provides a better understanding in how to form a genome. Our Steiner hull (StH) is composed by five points (belonging to the ESTP instance too), $StH = \{a, b, c, d, e\}$, and we will be able to generate three Steiner points, at most. In this case the representation will be achieved by the use of an array with a maximum length of fifteen elements, each one storing a floating point value (α) in the range $[0, 1]$. Each Steiner point is expressed by a linear combination of the points that forms the Steiner hull.

To process the Euclidean coordinates of each Steiner point we will use the coordinates of each point in the Steiner hull, weighted by the respective random value of α, as stated in Formula 4.

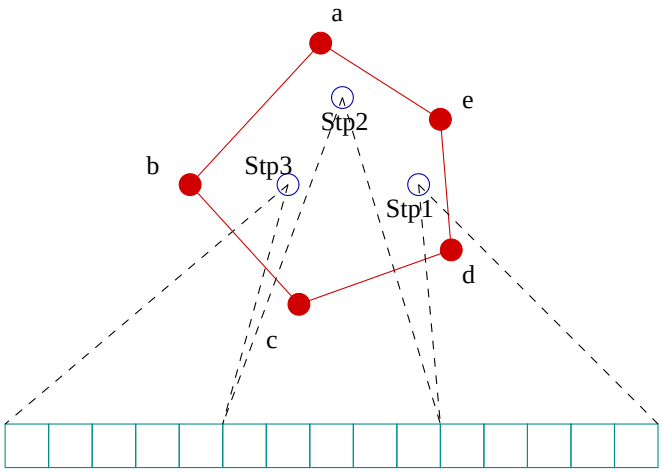

Figure 2: Steiner points' set representation in a genome.

$$(x, y) = \left(\sum_{i=1}^{K} x_i * \alpha_i, \sum_{i=1}^{K} y_i * \alpha_i \right) \tag{4}$$

Expressing each Steiner point this way ensures that in all future generations new points will appear always inside the Steiner hull, thus in the domain of our problem. It directs the processing power of the genetic algorithm to the exploration of a restricted area.

Another important feature of this codification method is the avoidance of binary digits, since each genome only will use real values in its genes. The use of a floating point codification, models the behavior of the GA in a superior way, avoiding some noisy effects (epistasis) and favors the convergence of the process. However, this decision definitely disables the possibility of the exact computation by the cGa model, but as the main concern is to approximate solutions for the ESTP, it seems to be a good exchange.

3.1.4 cGa definition and basic operators

The cGa computational model is developed using a steady state genetic algorithm. It differs from others in the way how populations succeed to each other as iterations are passing by, allowing a percentage of overlapping. It is a way to exercise some pressure in the process, introducing some refinement in each generation thus favoring the exploitation feature of the GA.

To decide whose genomes will be submitted to the next phases of the evolutionary process, a selection method must be used. Among several possible techniques the 'roulette wheel' method was picked up. It ensures a fairy good process of choice, it is easy to implement and it is used in many other situations with success. (Bäck, 1991; Bäck and Hoffmeister, 1991; Michalewicz, 1996).

Afterward, pairs of genomes are submitted to crossover if some probability of occurrence is surpassed. The uniform crossover used in the cGa model, is a dynamic and non-deterministic method since it does not decide on how many or what positions, gene replacements will take place. Replacements are decided by the use of a mask

initialized randomly. Balancing its occurrence and the way it happens is a clue to ensure sufficient diversity in the population, without destroying constructive blocks of genes. This is important, as it ensures a sufficient wide coverage of the search space responding to the combinatorial feature of the ESTP (Bruce and Simpson, 1999).

Mutation is the next operator to be used in the evolutionary process. Upon a predefined probability of occurrence, it is decided that a gene in the offspring will be submitted to mutation. A new floating point value is generated using the actual value and the normal distribution, to substitute the old one.

3.1.5 Specific problem operators

The representation depicted in the previous subsection, always starts with the maximum of $N - 2$ Steiner points, allowing the known full Steiner tree representation. However, optimum results could have a lesser number of Steiner points and the cGa model will use some heuristics to find out other representations toward to the minimal Steiner tree, during the evolutionary process.

The 'survival area' operator is responsible for the diminishing of the number of Steiner points in each genome. It mimics the social organization of living beings in Nature, establishing a survival area where nothing can happen without the consent of the owner. Sometimes some individuals claim for the same space. Depending on the strengths in presence, the overall geographical area could be redistributed by the allocation of those invaders.

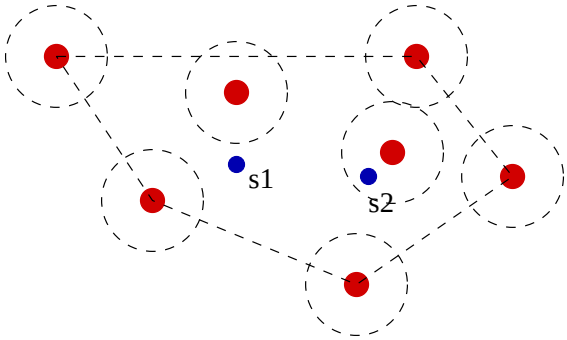

Figure 3: Sketch of the survival area operator action.

The 'survival area' is a geometric operator that will be described using the Figure 3 as a guideline. Input points are spread out occupying some area represented as a broken line joining the extreme points of the instance. To estimate the survival area of each point, represented as a circle, simply average the total area in the convex hull by the number of points inside it (in the drawing, the area will be divided by 7). Points $s1$ and $s2$ are Steiner points processed in the iteration, representing two different situations. The former will be accepted in the population, in spite of the later that will be deleted.

Nevertheless, points can have some fancy distribution inside the convex hull, requiring some adjustments on radius of the circle of each survival area. This correction can be achieved using a coefficient of correlation estimated for the input points, for example.

The Algorithm 1 presents the pseudocode for the survival area operator and has an expected time complexity of $O(N^2)$, in the worst case. The algorithm uses two lists of points, the $all\_pointsLST$ and the $steiner\_ptsLST$. The first one owns all points of the

Algorithm 1 The survival area algorithm

Require: $radius$, $all\_pointsLST$, $steiner\_ptsLST$
Ensure: $all\_pointsLST$ with a survival area
1: **for** $(i = 0; i < all\_pointsLST.size(); i++)$ **do**
2: $xPT \leftarrow all\_pointsLST[i]$
3: Create circle $xCIR$, centered at xPT
4: **for** $(j = i; j < all\_pointsLST.size(); j++)$ **do**
5: $yPT \leftarrow all\_pointsLST[j]$
6: Remove yPT if inside $xCIR$ and belonging to $steiner\_ptsLST$
7: **end for**
8: **end for**

problem and the second one only possesses the estimated Steiner points for the ESTP instance. The algorithm must preserve the input points from remotion.

The 'survival area' operator is applied to each genome after the usual genetic operators, whether they are selected or not for crossover or mutation. It provides a better exploration of the search space, it favors the formation of future trees with a number of Steiner points other than its maximum, increasing the probability of finding a good answer.

The 'optimization tree' algorithm has the main objective to model data to facilitate the final Steiner tree construction. It relies upon some geometric concepts and algorithms, as follows.

Algorithm 2 General tree optimization algorithm

Require: $msptGRAPH$, $Steiner\_pointsLST$
Ensure: $msttGRAPH$
1: Removal of Steiner points with degree 1.
2: Removal of Steiner points with degree 2.
3: Relocation of Steiner points.
4: $msttGRAPH \leftarrow msptGRAPH$

Algorithm 2 introduces, in a superior level of abstraction, the main routine used to generate each solution. As a result a Steiner tree is constructed. In the beginning of the 'optimization tree' algorithm a graph is expected representing a minimum spanning tree with all remaining Steiner points, plus those that belongs to the ESTP instance. The Algorithm 2 relies on the information of this structure and applies the geometric restrictions, concerning the degree and the relative position of each Steiner point. It is a three phase algorithm, developed as follows:

1. This phase consists in traversing the list of points of the graph and to remove all points from the Steiner list with degree one.

2. Points with degree two will be pruned out too. The situation is similar to the previous one, except that the point to be deleted is connected to two others, that must remain connected to each other.

   In the Figure 4 the situation is sketched. On stage (a) the Steiner point s_1 i connected to points a and b. Its elimination implies the replacement of edges $\overline{as_1}$ and $\overline{bs_1}$ by $\overline{ab}$ as shown in (b). This can be accomplished if a and b are connected before deletion of s_1.

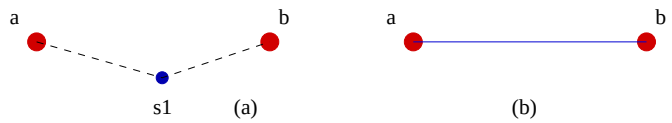

Figure 4: The removal of a Steiner point of degree two.

3. The last step of the 'optimization tree' algorithm is the definitive positioning of the Steiner points processed so far. It is extremely difficult that a point generated by the cGa will be positioned in the right spot, i. e., respecting an $120°$ angle between its edges (Hwang et al., 1992). This objective will be accomplished by the use of the geometric algorithm of Torricelli/Simpson, sketched in Figure 5. It will be responsible to find out the definitive position of our Steiner points.

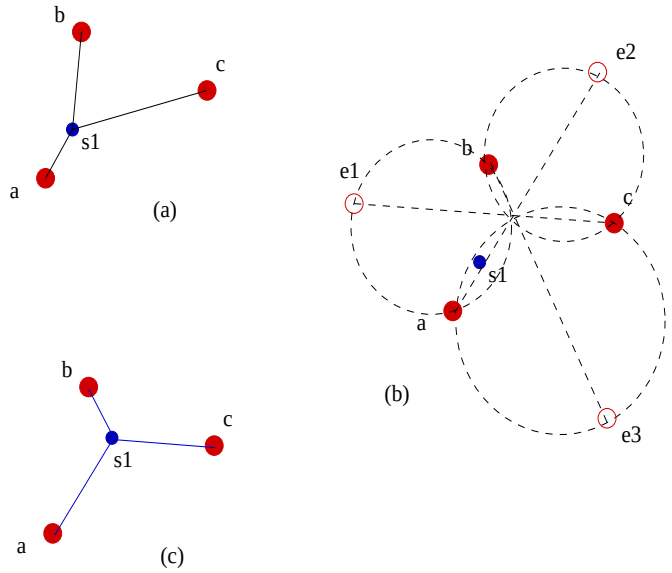

Figure 5: The Torricelli/Simpson algorithm sketch.

In Figure 5 stage (a), an initial position of the point s_1 is presented, contradicting the Steiner point definition. In stage (b) the points e_1, e_2, e_3 are created in a way that they make an equilateral triangle with the nearest two points of a, b or c. The intersection between the Simpson lines [1] is the right position of the true Steiner point.

This phase of the algorithm is particularly important since it:

- Allows a better adaptation to the ESTP problem.
- Favors the shortening of the tree length, since the new point's position minimizes the distance between the involved points.
- Improves the speed of the all process. The point relocation assures that cGa model avoids a lot of generations in finding the optimal position of Steiner points and directs its efforts to the combinatorial process.

[1] Line segment between the equilateral point and the other point not involved in the equilateral triangle.

Algorithms 3 and 4 presents the pseudocode of all these geometric processes. They are retrieved from Algorithm 2, line 3:.

Algorithm 3 Relocation of Steiner points

Require: $msptGRAPH$, $Steiner\_pointsLST$.
Ensure: $msptGRAPH$.
1: Creates $xLST$ with all points with degree 3 using $msptGRAPH$ and $Steiner\_pointsLST$.
2: **if** ($xLST$ is not empty) **then**
3: **for** ($i = 0; i < xLST.size(); i++$) **do**
4: $xPT \leftarrow xLST.point[i]$
5: $yLST \leftarrow$ Connected points to xPT
6: $yPT \leftarrow$ TorricelliPoint($yLST$)
7: Update xPT in $msptGRAPH$ with yPT
8: **end for**
9: **end if**

The Algorithm 3 starts by creating a list of candidates to Steiner points, among all points included in the minimum spanning tree graph. Considering the $xLST$ emptiness, each point is evaluated to collect its connected points in list $yLST$, which will be used in the Algorithm 4 to find out the real position for each Steiner point, (Torricelli point algorithm).

The length of all edges of the triangle that encloses each Steiner point must be estimated, since Algorithm 4 will be called when the triangle is isosceles or scalene. The remaining case is simpler, since the Steiner point is coincident with the center of the circumscribing circumference of the equilateral triangle.

The Algorithm 4 is a modified and simplified version of the Torricelli algorithm. It was developed upon observations made for the location of a Steiner point on a three terminal set of points. Its position results always from the intersection of the circumference circumscribing an equilateral triangle formed by two points and an equilateral point, and the respective Simpson line. This way the computation of remaining circumferences and Simpson lines are avoided.

Report back to Figure 5 and consider that the equilateral triangle is $\triangle(a, b, e_1)$. The intersection between the circumference circumscribing those points and the Simpson line $\overline{e_1 c}$, locates the Steiner point, as well as the other two options if $\triangle(b, c, e_2)$ or $\triangle(a, c, e_3)$ were selected.

With the new point coordinates, estimated by the algorithm, a simple updating operation takes place. No deletions or edge creation are needed, since the Steiner point remains connected to the very same points, as the previous point was.

3.1.6 Improving the evolutionary process

Observing the intermediate results of an execution in any GA and paying a special attention to the fitness value of the genomes, soon some facts catch the eye. Fitness values begin to spread out into the population, until convergence. This fact suggests the existence of equal genomes, meaning that a lot of computation could be avoided if previous fitness values could be 'memorized'.

To improve the evolutionary process a remembering system was developed inside the cGa, memorizing an 'encoded tree' and its fitness value in each entry of the system. This 'memory' was developed using a dictionary tree (Horowitz et al., 1997).

Algorithm 4 The Torricelli point algorithm

Require: $yLST$ (list with three points).
Ensure: $torricelliPT$.
1: Compute distances between points aPt, bPT, cPT from $yLST$.
2: **if** ($\triangle(aPT, bPT, cPT)$ is equilateral) **then**
3: Create circle $xCIR$ from aPt, bPT, cPT
4: $torricelliPT \leftarrow$ Center of $xCIR$
5: **else**
6: Find equilateral point $equiPT$ with aPT, bPT, cPT
7: $xCIR \leftarrow$ Circle from $aPt, bPT, equiPT$
8: $xSEG \leftarrow$ Simpson line from $equiPT$ to cPT
9: $xLST \leftarrow$ Points of intersection from $xCIR$ and $xSEG$
10: **if** ($xLST.first() = equiPT$) **then**
11: $torricelliPT \leftarrow xLST.last()$
12: **else**
13: $torricelliPT \leftarrow xLST.first()$
14: **end if**
15: **end if**

Two $(2, 4)$-trees are used in each run and at the end of each generation an update is made in their contents, preserving last used values. This strategy avoids the excessive growth of the dictionary, considering that the next generation is made upon the genomes of the present one. Each entry in the dictionary is associated with each genome. The value that will be stored in the information field must be related with a key field, upon a guaranteed uniqueness.

Algorithm 5 Key entry algorithm

Require: $genome$
Ensure: key
1: $a, b, key \leftarrow 0$
2: **for** ($i = 0; i < length(genome); i++$) **do**
3: **if** ($(imod2) = 0$) **then**
4: $a \leftarrow a + genome[i]$
5: **else**
6: $b \leftarrow b + genome[i]$
7: **end if**
8: **end for**
9: $key \leftarrow (a^2 + b * a + a/b)$

The Algorithm 5 shows how the key value is generated. It is based on the random values stored in the genome and gives a special heed to where each value is located. The key value is estimated upon values located in odd or even elements of the genome array and stored on a and b variables, which will be used to compute the final key.

This method reduces the processing time of our model in some 30%, depending on the population size and in the diversity of the genomes.

3.1.7 Adapting the cGa model for every ESTP

The combinatorial feature of the ESTP along with some others, bestows a high rank of difficulty making all solutions to be unpredictable, even for those instances that seem to be very similar.

Nevertheless, instances of the ESTP can be divided in two classes: regular grids of points and complete random sets of points. In general, the former class of points, due to its geometric feature presents a much higher level of difficulty, then the latter one. The relative position of the nearest points have an important role in the intrinsic difficulty of each ESTP instance. The immediate reflex of this observation is the different needs (in time) for the cGa model to approximate an useful answer for a specific instance of this type.

At this point a fundamental question must be stated. How much time must be allowed to the model to find out a solution for any ESTP problem? Or, to put in other terms, what is the balance between accuracy and processing time for all ESTP instances? No definitive answers can be given to these questions.

Apart from the time spent in the evolutionary process, GAs also has some drawbacks, related with the tunning of the operators. It is desirable, for practical use, to achieve some simplicity in setting the external parameters, always assuring the performance of the computational model, with a minimum loss of accuracy (Bäck, 1991; N. Schraudolph, 1992; Haynes, 1998; Tucson and Ross, 1998; Harik and Lobo, 1999; Deb and Beyer, 1999).

At this point, we tried to deal with these two barriers acting against the practical use of the cGa model and two different approaches are presented, both presuming that the factor time is relevant:

1. The experiments made with several ESTP instances from both classes of instances, lead to the conclusion that the population size of $2N$ assures sufficient diversity in the model.

   The mutation operator depends on the population size and on the genome length. It acts on each gene at a turn, and it is responsible for the exploitation of the search space. The mutation rate was fixed at

   $$p_m = \frac{0.3}{(N-2)}, \tag{5}$$

   with a direct relation with the problem itself. When a rate is ascribed to the operator it is also said that in each generation it is expected a certain number of mutations, at most. The maximum number of mutations allowed in each generation is given by

   $$nr_m = p_m * l * 2N, \tag{6}$$

   being nr_m the number of mutations, p_m the probability of mutation and l the length of each genome.

   The cGa model performs robustly and achieves very good approximate solutions within the very first part of each run. Remaining generations have a very low fitness improvement. Based on the statistics maintained by the GA, a termination function was developed to detect a stopping point. The termination occurs when one of the following conditions is verified:

- A tolerance for the standard deviation of the population (10^{-6}).
- A predefined number of generations.
- The unattainable theoretical optimum is reached.

2. A second alternative is implemented using the previous one, but assuring the execution of a minimum number of generations. The stop criteria will be applied after that coercive execution.

   It seems obvious that a problem instance with few points needs less generations than another one with a larger number of points. Once again we will use direct dependence from the problem being submitted, using its number of points. Formula 7 expresses that relation:

$$nr_g = 10 * N, \tag{7}$$

   being nr_g the number of generations and N the number of points, as usual.

   In section 4 both versions will be recognized as Va and Vb, respectively.

4 Evaluation

Instances of ESTP in regular grids are particularly difficult to solve. The structured topology imposes restrictions to the position of the Steiner points that can be combined in a superior way to produce different solutions.

The optimum solution for regular grids of the ESTP are Steiner trees resulting from the combination of some well known components, identified as I, Y and X. They generate some excesses length, which are reflected in the final tree. These kind of ESTP instances were well investigated and it is possible to know in advance the optimum Steiner tree length for each instance (Brazil et al., 1996).

Next we will present some results achieved by the cGa model on three regular instances with 3×4, 5×5 and 10×10 points [2]. Their optimum solutions are reported on Table 1.

Table 1: Optimum Steiner trees lengths for the lattice tests (Brazil et al., 1996).

Type	$\|MStT\|$
3×4	10.19611
5×5	22.12428
10×10	90.20629

Figure 6 shows some outcomes proposed by the cGa model for each of the ESTP instance tested. The StTs for the 3×4, 5×5 and the 10×10 grids of points have lengths of 10.2153, 22.2878 and 91.9111 respectively. In average the cGa model performed quite well, since the relative error considering the values of Table 1 are 0.9%, 2.0% and 2.4% for the 12, 25 and 100 points, respectively.

Random sets of points were used also to test the cGa model. They have been subdivided into two different classes, according to the distribution used to spread out each set of points in the Euclidean plane. We used the uniform and the normal distributions to generate different instances of the ESTP with 12, 25 and 100 points.

[2] In fact they are lattices of points, since each pair of points is separated by a unit length.

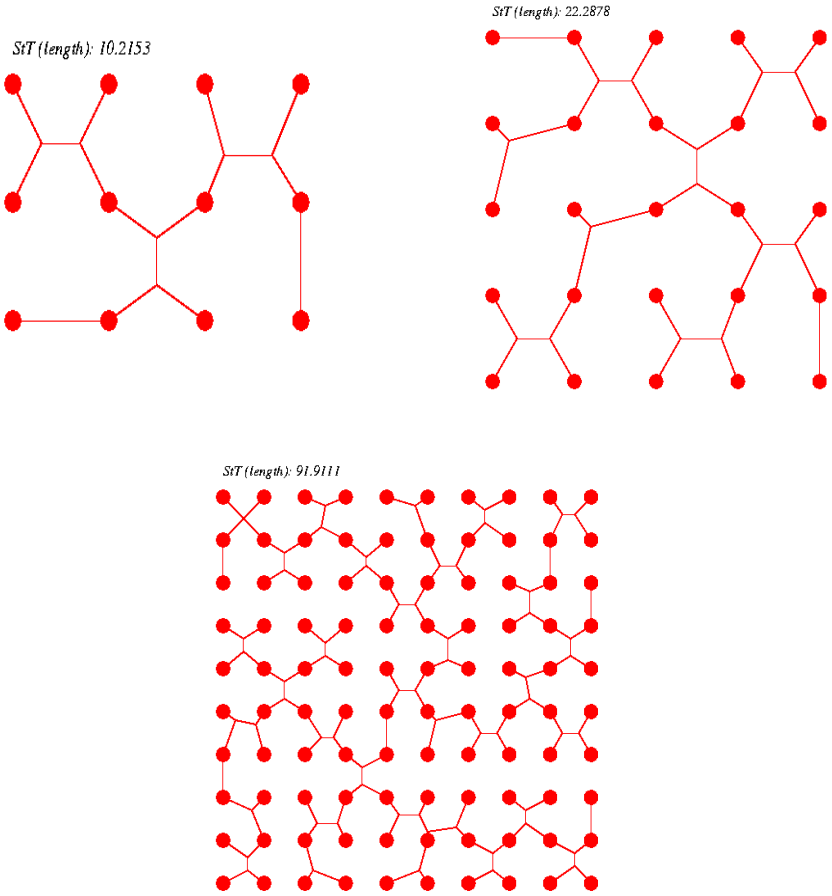

Figure 6: Some StTs on regular grids of 12, 25 and 100 points, achieved by cGa model.

Unlike the described previously, it is not an easy task to find out the optimum Steiner tree length for any random set of points. This time we will use the Euclidean Steiner ratio as a comparison reference as stated in (1). Conclusions should be drawn carefully since there are instances where the gap between their MSpT and MStT lengths are so small that can conduct to erroneous interpretations of results (Hwang et al., 1992).

Figures 7 and 8 presents some Steiner trees found for each set of points distributed uniformly and normally, used to test the cGa model.

The cGa model reveals some particular difficulty with this sets of points, in particular with a small amount of points. This behavior results from an insufficient adaptation of the model to the problem instance. It must be remembered that the same set of parameters was used to run the cGa model in all ESTP instances and it is obvious that the intrinsic difficulty of some problems becomes real when the perfect match, between model and problem, fails.

The Table 2 summarizes, in a compacted form, the results achieved by each version

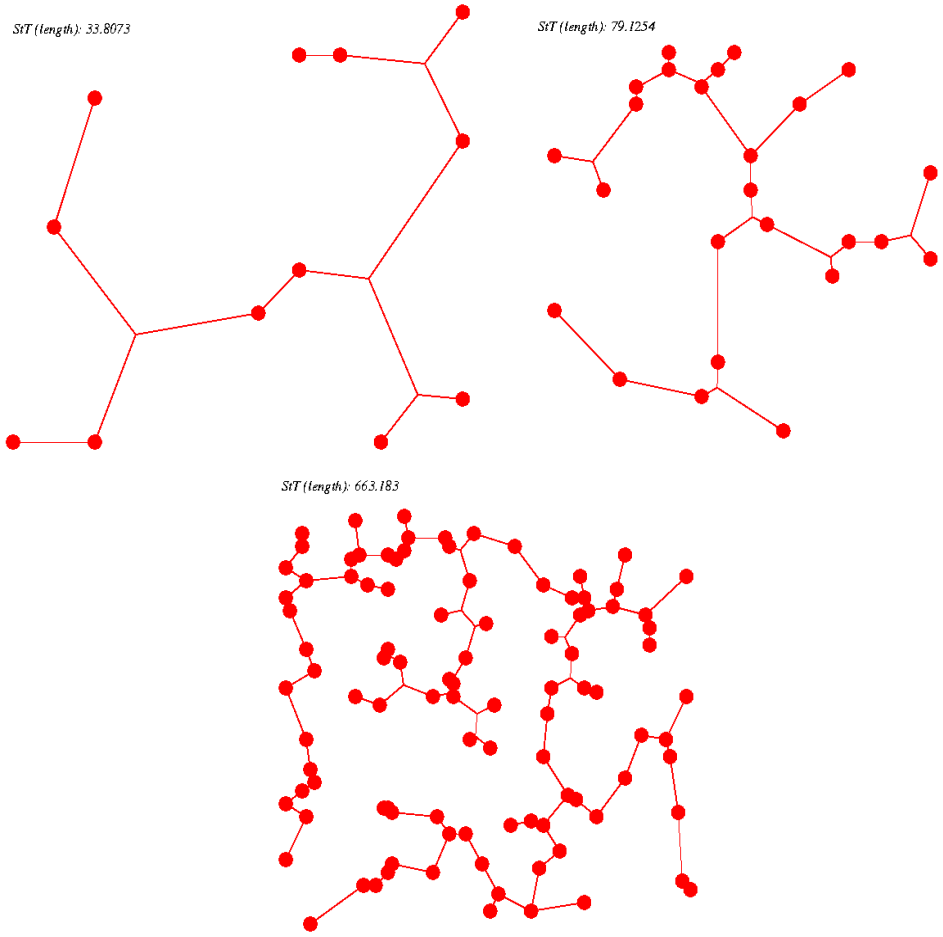

Figure 7: Some StTs on uniform sets of 12, 25 and 100 points, achieved by the cGa model.

Table 2: Compacted results from cGa model (Versions a and b) with random sets of points (ϵ (%) vs. time (sec)).

	Va		Vb	
Type	ϵ	t	ϵ	t
12p	12.54	0.2	12.48	1.8
25p	12.41	2.5	12.28	11.9
100p	13.04	207.2	12.89	493.5

of our computational model, considering an estimated average for the same number of points in both kinds of random instances. Results are reported considering the relative error ϵ and the execution time t, for both versions of the cGa model. As expected Version a take less time but achieves worst results then Version b, which assures a minimum number of iterations in each run of the computational model.

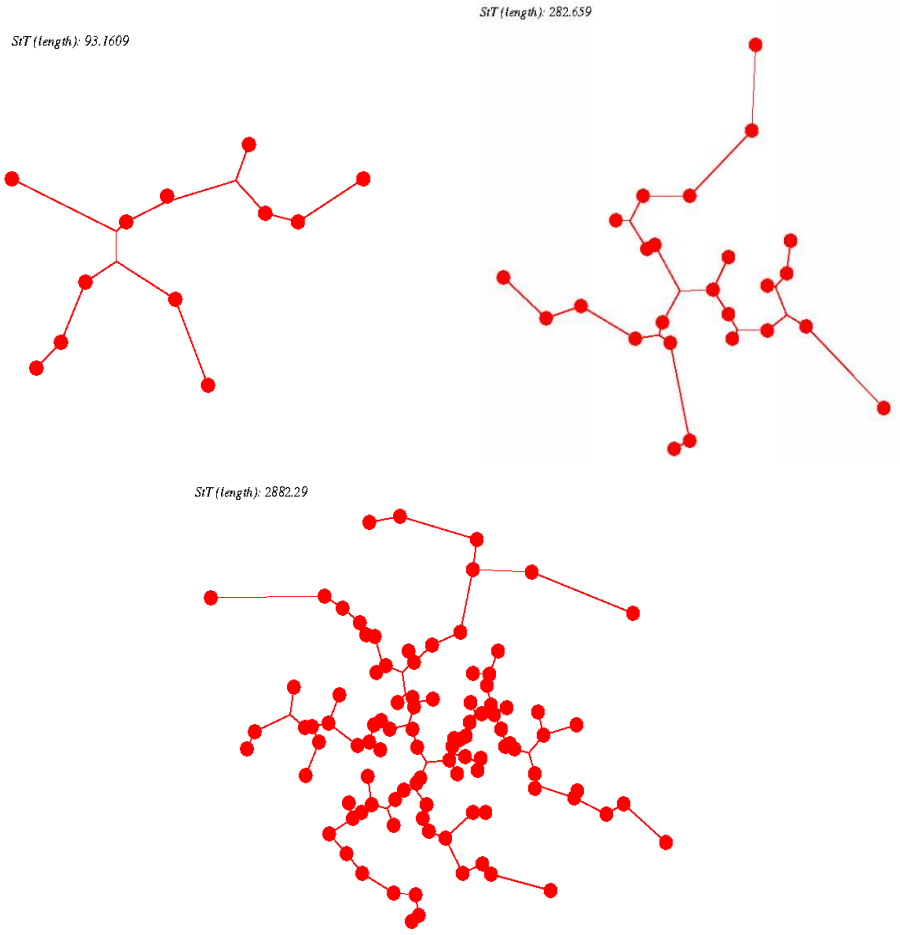

Figure 8: Some StTs on normal sets of 12, 25 and 100 points, achieved by the cGa model.

From this collection of results, it is shown that an higher error bound of 15% in the accuracy for the cGa model is respected, even taking the theoretical Steiner ratio as reference.

From the Optimization Research repository of problems (OR library) we collected a group of Euclidean Steiner Tree Problems with different sizes to test the version b of the cGa model. We will use this version because it presents a suitable balance between accuracy of the solution and the processing time spent.

The Table 3 summarizes the results achieved with the cGa model, on instances with 10 to 100 points. Each row presents a instance of the problem, its minimum spanning and Steiner trees lengths, [3] as well as the StT length found by the cGa model, its relative error ϵ and the time t, in seconds, spent in the optimization process.

[3] These values are available on the OR library for each problem.

Table 3: Results from cGa (Version b) on some ESTP instances from the OR library.

PROBLEM CHARACTERISTIC			cGa RESULTS		
Points	$\|MSpT\|$	$\|MStT\|$	$\|StT\|$	ϵ_{StT}	t
10	2.33009	2.22807	2.22825	0.01%	0.71
20	3.21282	3.07164	3.15226	2.6%	3.32
30	3.74951	3.61384	3.66409	1.4%	7.49
40	4.03362	3.92835	3.9864	1.5%	19.33
50	4.96763	4.83660	4.90235	1.4%	32.36
60	5.40313	5.21425	5.27403	1.2%	71.18
70	5.53935	5.32759	5.48558	2.9%	270.99
80	6.30885	6.03885	6.23166	3.2%	159.49
90	6.17564	6.04563	6.1202	1.2%	223.07
100	6.90325	6.67469	6.84741	2.6%	134.35

5 Clustering

Considering the robustness and accuracy provided by the cGa model, as it was proved in the previous subsection, an approach to the processing of large instances of the Euclidean Steiner Tree Problem can be developed.

It is known that practical problems can easily reach hundreds and even thousands of points in a single ESTP instance, requiring a suitable solution in a short time window having as much accuracy as possible, to be useful. So far the diminishing of the processing time was achieved by acting in the GA part. However, current work signals out that the processing time spent to find a Steiner tree for a ESTP instance, rapidly deteriorates with the input size. Putting in other words, if the number of points stays in a reasonable level, an important improvement could be achieved if we can transform the ESTP instance.

Upon this consideration, it seems reasonable to subdivide the input instance of our problem into smaller groups of points (clusters), in a preprocessing phase. Afterward each cluster will be processed by the cGa model providing a partial Steiner tree. The final answer is created in a merge process, where each partial Steiner is 'bind' to each other, in a construction phase.

5.0.8 ESTP instance division

Instances of the ESTP can vary as long as we want and it is difficult the have a method that optimally 'clusterize' the points for any instance of the problem. If points are distributed forming a regular grid, clusterization based on distance does not take any effect at all and a straight division must be done. In practice, only a small amount of problems falls into this kind of instances.

To produce balanced groups of points from an instance of the ESTP we will closely follow the following rule:

$$|C| = \sqrt{N}, \tag{8}$$

where $|C|$ denotes the number of clusters to be formed. Once a distribution of $\approx \sqrt{N}$ points for each cluster is achieved, an upper limit of N is settled for the overall preprocessing time phase.

Clusters are formed in a two phase process:

1. The nearest neighborhood of second order ($2NN$ method).

2. The hull-seeds method.

All random instances of the ESTP are submitted to the first phase. If the result does not conform with the expected one (resulting from the 8) the hull-seeds method will be applied after.

Starting with the $2NN$ method and considering that no information about the topology is available, it is easy to find the closest points using the nearest neighborhood of second order method.

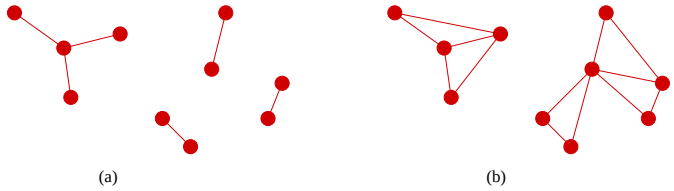

Figure 9: The $2NN$ method applied in a small set of points.

We will use the Figure 9 as a support for our explanation of the method. A small set of points are sketched and in stage (a) it is identified the establishment of links between points that are closer to each other. In phase (b) the subgraphs show the two closest points to each of them. This way we assure that clusters with a minimum of three points exists, meaning that each point must be connected twice (one link to each of the two nearest points). This method is highly dependent on the set topology, and leads to unbalanced clusters of points most of the times, thus contradicting our guideline.

The hull-seeds method must be called then. In the end it assures the existence of $|C|$ clusters resulting of the subdivision of the set of points.

The Figure 10 will be used to introduce this method. It begins by the estimation of the number of seeds for a particular ESTP instance (there will be $|C|$ seeds) and they are distributed along the convex hull of the instance, though the existence of a point is not necessary. In phase (a) seeds are located at p, q, r. Using the input order, the proximity of each point is evaluated upon what a decision is taken to add it to the closest cluster, taking each seed as a reference. Afterward each seed is reallocated in the middle distance from its previous position and the newly inserted point. Stage (b) shows the position of those seeds after inclusion of points, in the indicated order. This process is repeated until all points are assigned to a cluster.

The hull-seeds method sometimes provides clusters with less than three points. Whenever it occurs it is necessary to reallocate those points starting with those clusters which have less points and are nearby.

If an initial instance can be divided successfully by the appliance of the methods just described, an important improvement in the overall processing time is achieved. However, the preprocessing phase is exposed to some vulnerability by the intrinsic complexity of the inputed points topology, requiring a special care.

Clusterization also brings another important benefit: the possibility of parallel or distributed processing of the ESTP. In fact, the subdivision into clusters of points enables to address each cluster of points to a different processor, providing that some policy of control is used.

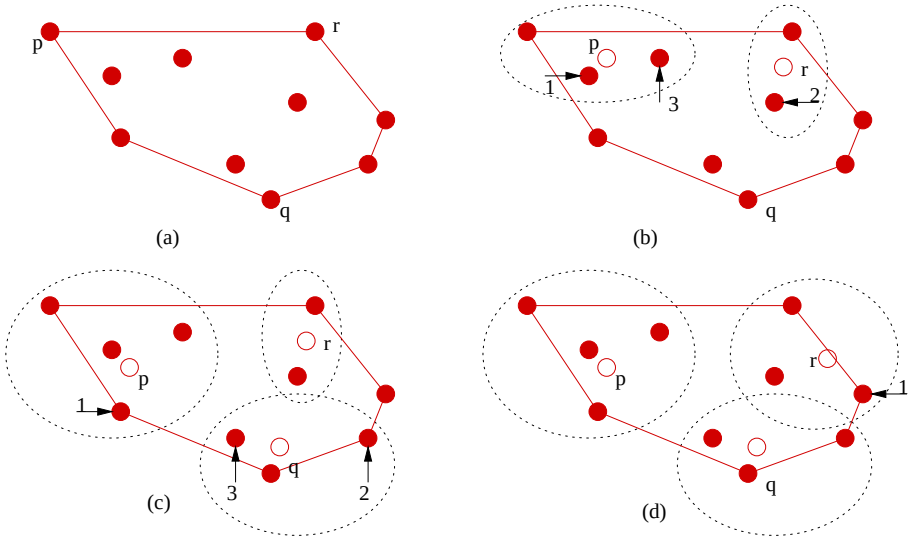

Figure 10: The hull-seeds method.

5.0.9 Reconstruction phase

Having a set of partial Steiner trees, each one produced by the cGa model for each cluster, a final and global answer must be derived for the ESTP instance, i.e., a final Steiner tree must be "constructed" from all the partial Steiner trees. This will be accomplished by a merge phase, where all partial Steiner trees must be assembled. In this merge process some extra Steiner points will be used, if necessary, to 'glue' all parts.

Partial Steiner trees are linked together by the selection of some points, specially located. These points are distinguished themselves by the following two properties:

- They belong to the Steiner hull of each cluster or partial Steiner tree (they are terminal points).
- Only one point per cluster or partial Steiner tree can be selected.

The proximity between Steiner trees was the main criteria used to select points. Sets of points are established according to the number of partial answers in evaluation and direct links between StT can be established, or some extra Steiner points could be detected by an extra run of our computational model.

5.1 Testing the clustering technique

To test the cGa model, we collected some special large instances of the ESTP in the OR repository of problems and they were submitted to our program.

The Table 4 shows some results achieved with some of those instances from which no previous knowledge is available. Each entry in the table represents an instance of the ESTP along with their $MSpT$ length and the results achieved (StT length and time in seconds).

6 Conclusions and open problems

The main goal of this investigation was achieved. It was proved that the Euclidean Steiner tree problem can be approximated in reasonable time within a bounded degra-

Table 4: Results from cGa(v4a) on some ESTP instances.

PROBLEM CHARACTERISTIC		cGa RESULTS	
Points	$\|MSpT\|$	$\|StT\|$	t
250	10.6885	10.6784	1001.3
500	14.6164	14.5821	2979.7
1000	20.7829	20.7811	37154

dation in its accuracy.

This achievement was possible only by the perfect adaptation of a genetic algorithm to the ESTP, with the precious help of some geometric operators. All these ideas are condensed in the cGa computational model, whose detailed description can be found in this paper. However, a lot of improvements can still be introduced, allowing an increase of the overall performance of the computational model (mostly in its accuracy).

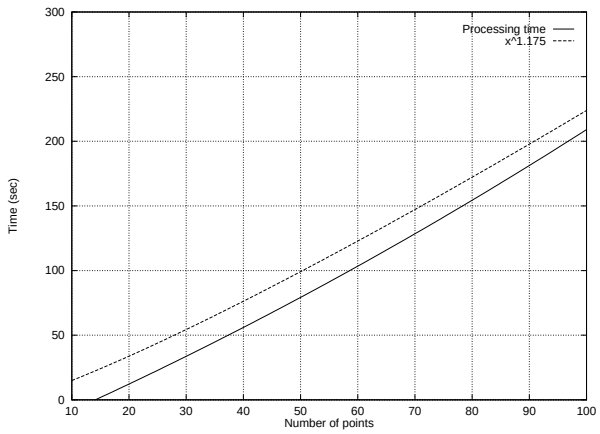

Figure 11: Complexity behavior

Concerning the final performance of the cGa model we present in the Figure 11 a final balance considering the processing time spent and the size of the input. A comparison between the time complexity of the model with the function $x^{1.175}$ is shown, revealing some advantage to our model.

However and as expected, the ESTP investigation using genetic algorithms do not ends here. Still a lot of effort should be done, namely in the following areas:

- Representation of the ESTP considering its complexity;
- Clustering or other techniques able to subdivide the ESTP instance;
- Steiner tree reconstruction phase.

It was proved that the distance between points does not suffices to represent the topology and the related geometric complexity of an ESTP instance. To improve the adaptation of the cGa model to the problem, it will be necessary to quantify how complex an instance is and select the set of parameters in accordance with.

As demonstrated, one way to improve the processing time of large instances of the ESTP, is to make its subdivision in smaller sets of points. Investigation should be directed to this item, to promote better heuristics to achieve regular clusters (with identical number of points) and preserving the overall geometric complexity of the problem instance. The symbiosis between clustering and adaptive learning could provide some new interesting insights.

Our strategy in the cGa model development for large instance of the problem, was based in a reconstruction phase, where a final Steiner tree is constructed. It is possible to have some improvement in this phase too, by the use of new heuristics that favors the global perspective of the problem.

Since the instance subdivision seems to be the way to take practical advantage from the Euclidean Steiner Tree Problem, it is natural the suggestion of the parallelization or process distribution on a cluster of computers.

References

Bäck, T. (1991). Optimization by means of genetic algorithms. Technical report, Department of Computer Science, University of Dortmund.

Bäck, T. and Hoffmeister, F. (1991). Extended selection mechanisms in genetic algorithms. Technical report, Department of Computer Science XI, University of Dortmund.

Beasley, J. E. (1989). An sst-based algorithm for the Steiner problem in graphs. *Networks*, 19:1–16.

Brazil, M., Rubinstein, J. H., Thomas, D. A., Weng, J. F., and Wormald, N. C. (1996). Minimal Steiner trees for rectangular arrays of lattice points. *Journal of Combinatorial Theory*, A(79):181–208.

Bruce, I. D. and Simpson, R. J. (1999). Evolution determined by trajectory of expected populations: Sufficient conditions, with application to crossover. *Evolutionary Computation*, 7(2):151–171.

Culberson, J. C. (1998). On the futility of blind search: An algoritmhic view of "no free lunch". *Evolutionary Computation*, 6(2):109–127.

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

Deb, K. and Beyer, H.-G. (1999). Self-adaptation in real-parameter genetic algorithms with simulated binary crossover. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *GECCO'99 - Proceedings of the Genetic and Evolutionary Computation Conference*, volume Vol. I, pages 172–179. Morgan Kaufmann Publishers, Inc.

Dreyer, D. R. and Overton, M. L. (1996). Two heuristics for the Steiner tree problem. Technical report, NYU Computer Science Dept.

Esbensen, H. (1995). Finding (near-) optimal Steiner trees in large graphs. In Eshelman, L. J., editor, *6th International Conference on Genetic Algorithms*, pages 485–491.

Gao, B., Du, D.-Z., and Graham, R. L. (1994). The tight lower bound for the Steiner ratio in Minkowski planes. In *10th Computational Geometry*, pages 183–191. ACM.

Garey, M. R., Graham, R. L., and Johnson, D. S. (1977). The complexity of computing Steiner minimal trees. *SIAM Journal Applied Mathematics*, 32:835–859.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc.

Harik, G. R. and Lobo, F. G. (1999). A parameter-less genetic algorithm. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *GECCO'99 - Proceedings of the Genetic and Evolutionary Computation Conference*, volume Vol. I, pages 258–265. Morgan Kaufmann Publishers, Inc.

Haynes, T. (1998). Collective adaptation: The exchange of coding segments. *Evolutionary Computation*, 6(4):311–338.

Heckendorn, R. B. and Whitley, D. (1999). Predicting epistasis from mathematical models. *Evolutionary Computation*, 7(1):69–101.

Hesser, J., Maner, R., and Stuckey, O. (1989). Optmization of Steiner trees using genetic algorithms. In *3rd International Conference on Genetic Algorithms*, pages 231–236. Morgan Kaufmann Publishers, Inc.

Holland, J. (1992). *Adaptation in Natural and Artificial Systems*. MIT Press.

Horowitz, E., Sahni, S., and Rajasekaran, S. (1997). *Computer Algorithms / C++*. Computer Science Press.

Hwang, F., Richards, D., and Winter, P. (1992). *The Steiner Tree Problem*. Elsevier Science Publishers B.V.

Hwang, F. K. (1986). A linear time algorithm for full Steiner trees. *Operation Research Letters*, 5:235–237.

Jesus, M., Jesus, S., and Márquez, A. (1999). Optimización de algoritmos genéticos en el procesamiento de árboles de Steiner. In Badenas, J., a, P. G., López, A., Pla, F., Sánchez, J. S., and Sanchiz, J. M., editors, *VIII Encuentros de Geometría Computacional*, pages 33–41.

Jong, K. A. D., Potter, M. A., and Spears, W. M. (1997). Using problem generators to explore the effects of epistasis. In Bäck, T., editor, *7th International Conference on Genetic Algorithms*, pages 338–345. Morgan Kaufmann Publishers, Inc.

Kapsalis, A., Rayward-Smith, V. J., and Smith, G. D. (1993). Solving the graphical Steiner tree problem using genetic algorithms. *Journal Operations Research*, 44(4):397–406.

Kou, L., Markowsky, G., and Berman, L. (1981). A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145.

Melzak, Z. A. (1961). On the problem of Steiner. *Canadian Math. Bulletin*, 4:143–148.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springler-Verlag, 2nd edition.

N. Schraudolph, R. B. (1992). Dynamic parameter encoding for genetic algorithms. Technical report, Computer Science & Engineering Department, University of California.

Naudts, B., Dominique, and Verschoren, A. (1997). Epistasis as a basic concept in formal landscape analysis. In Bäck, T., editor, *7th International Conference on Genetic Algorithms*, pages 65–72. Morgan Kaufmann Publishers, Inc.

O'Rourke, J. (1998). *Computational Geometry in C*. Cambridge University Press, 2nd edition.

Preparata, F. P. and Shamos, M. I. (1985). *Computational Geometry, An Introduction*. Springler-Verlag.

Rayward-Smith, V. J. and Clare, A. (1986). On finding Steiner vertices. *Networks*, 16:283–294.

Reeves, C. R. and Wright, C. C. (1995). Epistasis in genetic algorithms: An experimental design perspective. In Eshelman, L. J., editor, *6th International Conference on Genetic Algorithms*, pages 217–224. Morgan Kaufmann Publishers, Inc.

Smith, W. D. (1992). How to find Steiner minimal tree in Euclidean d-space? *Algorithmica*, 7:137–177.

Tucker Jr., A. B., editor (1997). *The Computer Science and Engineering Handbook*. CRC Press, Inc.

Tucson, A. and Ross, P. (1998). Adapting operator settings in genetic algorithms. *Evolutionary Computation*, 6(2):161–184.

Warme, D. M., Winter, P., and Zachariasen, M. (1998). *Advances in Steiner Trees (Exact Algorithms for Plane Steiner Tree Problems: A Computational Study)*. Kluwer Academic Publishers.

Winter, P. (1985). An algorithm for the Steiner problem in the Euclidean plane. *Networks*, 15:323–345.

Wolpert, D. H. and Macready, W. G. (1996). No free lunch theorems for search. Technical report, Santa Fe Institute.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Tansactions on Evolutionary Computation*, 1(1):67–82.