Hybrid Greedy Genetic Algorithm for the Euclidean Steiner Tree Problem

Andrey V. R. Oliveira[1], **Bruna A. Osti**[1], **Danilo S. Sanches**[1]

[1]Department of Computing
Federal University of Technology - Paraná (UTFPR)
Avenida Alberto Carazzai, 1640 – Cornélio Procópio – PR – Brazil

andreyvro@hotmail.com, brunaosti@alunos.utfpr.edu.br,

danilosanches@utfpr.edu.br

Abstract. *This paper presents a genetic algorithm for the Euclidean Steiner tree problem. This is an optimization problem whose objective is to obtain a minimum length tree to interconnect a set of fixed points, and for this purpose to be achieved, new auxiliary points, called Steiner points, can be added. The proposed heuristic uses a genetic algorithm to manipulate spanning trees, which are then transformed into Steiner trees by inserting and repositioning the Steiner points. Greedy genetic operators and evolutionary strategies are tested. Results of numerical experiments for benchmark library problem (OR-Library) are presented and discussed.*

1. Introduction

The Steiner minimal tree problem (SMTP) can be mathematically described as follows:

Find a geometric network $T = (V, E)$ (a connected graph with a set of vertices V and a set of edges E) so that $N \subseteq V$ and $S = V \backslash N$ is a (possibly empty) set of points known as *Steiner points*, so that $\sum_{e \in E} l(e)$ (sum of the lengths of edges $e \in E$) , is minimized [Brazil et al. 2013].

The SMTP aims to find a network to interconnect a set of fixed points (terminals) in a metric space, and can add new points (Steiner points) in order to minimize the total distance of the network [Hwang et al. 1992]. This problem has several variants; such as *Euclidean Steiner tree problem* (ESTP) when the metric space is Euclidean; *rectilinear Steiner tree problem* (RSTP) that uses the taxicab geometry [Hanan 1966]; *Steiner tree problem in graphs* (or STPG), which aims to optimize already established networks [Koch and Martin 1998]; and the *bottleneck Steiner tree problem* (BSTP), whose objective is to minimize the length of the largest edges of a Steiner tree [Wang and Du 2002].

Given v fixed points in a metric space, in order to obtain a solution for SMTP it is necessary to determine: the number of Steiner points, the coordinates of the Steiner points in the metric space; and the topology obtained from the points. It is assumed that a solution for SMTP is a tree T (otherwise, it will not minimize the length), and that any Steiner point in T has at least three incident edges. Edges at a given vertex have at least 120° angle, which results in the fact that every Steiner point has three incident edges forming angles of 120°, which is known as *angle property*. A minimum Steiner tree with fixed n points has a maximum of $n - 2$ Steiner points, and if it has the maximum number of Steiner points, the tree is called a *full Steiner tree* (FST) [Gilbert and Pollak 1968].

In order to obtain a good approximate solution of feasible computational time for ESTP in $\mathbb{R}^2$ (two-dimensional space), this work presents a heuristic based on genetic algorithm (GA). GAs are stochastic methods that look for the best solution to a problem among a large number of possible solutions. They were proposed and developed in the 1960s by John Holland, his students and his colleagues, at the University of Michigan. These computational paradigms were inspired by the mechanics of natural evolution, including survival of the fittest, crossover, and mutation [Goldberg 1989].

In this work, a steady-state GA is used to investigate the effect of generation overlap, the behavior of crossover operators and mutation for trees, and the impact of applying evolutionary strategies. The algorithm uses a population of spanning trees that are transformed into Steiner trees because a spanning tree usually works with less memory resources than a Steiner tree. Genetic operators are greedy because they tend to support lower cost edges, which results in the creation of topology close to minimum spanning trees (MSTs); therefore, these operators rely on the hypothesis that spanning trees close to the minimum tend to be transformed into good Steiner trees. Also, we have used two algorithms to generate the Steiner trees, resulting in a hybrid GA approach that allows fast convergence and high quality solutions.

2. Related Work

In 1992, Smith [Smith 1992] developed the first exact algorithm to solve the ESTP in $\mathbb{R}$ with dimension greater than 2. The algorithm uses a "branch-and-bound" method to construct FSTs and stores them in a structure called "vector-topology". For an SMT to be obtained from a vector-topology, it is necessary that the Steiner points are in optimal positions. In 2008, Fampa and Anstreicher [Fampa and Anstreicher 2008] proposed improvements in the Smith algorithm, creating a new algorithm that was named "Smith+".

Some heuristics use MST as the basis for generating SMT. The Thompson algorithm [Thompson 1973], generates the MST, searches for sets of three vertices connected with an angle smaller than 120° at the intersection of the edges, and then inserts a Steiner point into the Torricelli point of the triangle formed, disconnects the vertices, and reconnects them to the Steiner point. The process of inserting Steiner points is repeated iteratively until forming an FST, which then has its Steiner points repositioned in optimal positions in order to obtain the SMT. Dreyer and Overton [Dreyer and Overton 1998] developed a heuristic based on Thompson's heuristic, which repositions the Steiner points after insertion of all Steiner points.

Barreiros [Barreiros 2003] uses a two-phase GA; in the first phase it decides how to partition a set of vertices and in the second phase it finds the SMT of each partition. The GA of Jesus et al.[Jesus et al. 2004] codes the solution through a vector representing the position and the adjacent points to the Steiner points. Yang [Yang 2006] presents a GA whose approach is based on moving the Steiner points, constructing a tree joining nearby points, and at the end eliminating Steiner points that do not have degree three and invalidate the angle property. Bereta [Bereta 2018] proposes a memetic algorithm, which encodes the solution through a vector that stores the coordinates of the Steiner points, and a bit that informs whether the gene (Steiner point) is active or not.

The state-of-art for ESTP in R^d, with $d = 2$, is the exact algorithm "Geosteiner" [Winter and Zachariasen 1997], and for $d > 2$, it is the exact algorithm "Smith+"

[Fampa and Anstreicher 2008]. As for non-deterministic algorithms, Laarhoven and Ohlmannn [Laarhoven and Ohlmannn 2010] present good results in viable computational time, already Yang [Yang 2006] shows a better quality in the solutions, but with high computational time when compared to the other heuristics.

3. A Genetic Algorithm for Euclidean Steiner Tree Problem

This study proposes a heuristic that uses a genetic algorithm (GA) in order to find good solutions for the ESTP, in viable computational time. GAs are stochastic search procedures inspired by the Darwinian principles of natural selection and the ideas of Mendelian genetics. An GA begins with a population of randomly created candidate solutions and implements probabilistic and parallel exploration in the search space using selection, crossover, and mutation genetic operators [Goldberg 1989], [Mitchell 1998].

In a *steady-state* GA, only a few individuals are updated with each generation, unlike *generational* GA, in which the entire population is updated. Such an approach was popularized by the "G Enitor" system of Darrell Whitley and Joan Kauth's [Whitley 1988]. When compared to a generational GA, the steady-state GA uses less memory and processing per generation because there is only one population, and only a small portion of the population is selected to be processed by genetic operators. As individuals parents can remain in the population for a long time, fit individuals can dominate the population rapidly, resulting in a premature convergence [Luke 2013].

3.1. Methodology

The proposed heuristic represents the solutions through spanning trees, transforms them into full Steiner trees (FSTs) by the Thompson method [Thompson 1973], reposition the Steiner points to their ideal positions through the algorithm of Smith [Smith 1992], and in the end, removes unfeasible Steiner points from the best FST. Genetic operators act under the topology of spanning trees, and since these represent only fixed points, there is no need to worry about point constraints. When working with FSTs, the search for the number of points during the execution of the algorithm is avoided; and when using a deterministic algorithm to reposition the Steiner points, it is avoided the search for the ideal position of the Steiner points, so in this approach the GA is in charge of searching only the topology.

The process of inserting Steiner points into a spanning tree in order to connect sets of three vertices and then removing the cycle-forming edges is called *local steinerizations* [Hwang et al. 1992]. To add Steiner points to a tree through local steinerizations, a vertex with a degree greater than one is chosen, such as $v1$ connected to two other vertices $v2$ and $v3$, inserts a Steiner point in the Torriceli point of the triangle formed by $v1, v2, v3$, disconnects $v1$ from $v2$ and $v3$, and reconnects them to the Steiner point. To transform a spanning tree into a Steiner tree, it is necessary to apply local steinerization over all fixed vertices with degree greater one, until all vertices of the tree have degree one. If a vertex $v1$ has a degree greater than two, one must start the steiner process by the pair of vertices adjacent to $v1$ that results in the smallest angle in $v1$. This process allows, at the end of the $v1$ steinerizations, the pair of vertices adjacent to $v1$ with greater angle in $v1$ will be connected to this.

After all the fixed points of a spanning tree go through the process of local steinerization, it is necessary to reposition the Steiner points so that its angle property is met,

and so the tree has a smaller overall length. For this purpose the iterative method of Smith [Smith 1992] is used, in which at each step, the coordinates of the Steiner points tend to the optimal position. The algorithm considers the edges as a set of ideal springs, and at each iteration solves a system of linear equations, in order to minimize the potential energy of the springs system. Smith shows that for any initial position defined for the Steiner points, this method converges to the only optimal coordinates of the Steiner points. The studies by Fampa and Anstreicher [Fampa and Anstreicher 2008] and Forte et al. [Forte et al. 2015] also use this method to reposition Steiner points. For more details on this iterative procedure, see Smith [Smith 1992].

Once GA completes the iterations, an FST is returned as the best solution, and if Steiner points are placed on other points, they can be removed as they do not contribute to length minimization. To eliminate a Steiner point S that is positioned above another point called $V1$, simply connect the incident vertices in $S1$ to $V1$, and then delete $S1$.

3.2. Operators

In order for the proposed GA to represent trees, the solutions are represented by *direct encoding* by adjacency matrix, and the genetic operators and the other algorithms are applied directly in this structure.

To calculate the fitness of an individual (solution) I_1, its chromosome is copied to a temporary individual I_{temp}, which has its topology transformed Steiner tree, its length calculated and defined as the fitness of I_1. The length of the tree is the sum of the distances between the connected vertices, so as the problem is approached in $\mathbb{R}^2$, each vertex has a value x and y which represents its position in the plane, then the Euclidean distance between two points $P = (p_x, p_y)$ e $Q = (q_x, q_y)$ is given by $d(P,Q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$ [Deza and Deza 2009]. Since V is the set of vertices of a tree, and A is an adjacency matrix that determines the topology of the tree, such that if V_i and V_j are adjacent, then $A_{i,j} = 1$, if not $A_{i,j} = 0$, the total Euclidean length of the tree $l(V)$ is given by $l = \sum_{i<j} A_{i,j} d(V_i, V_j)$ [Gilbert and Pollak 1968].

The Prim algorithm [Prim 1957] creates a spanning tree in a greedy fashion, so that from an initial node, the lower cost edge joining a new vertex to the growing tree is added repeatedly. To create the initial population, our GA uses the algorithm *PrimRST* [Raidl and Julstrom 2003], where the choice of new edges is performed in a random way.

In this work a tournament selection operator defines the individuals that will be sent to the crossover and mutation operators. At the end of the process, the worst individual in the population is replaced by the generated child.

Crossover This paper proposes two crossover operators, inspired by the crossovers for trees shown by Raidl and Julstrom [Raidl and Julstrom 2003], which unites two trees and applies an algorithm such as Prim [Prim 1957] or Kruskal [Kruskal 1956] to select edges at random, and thus create a new solution. In this study, both operators are greedy, that is, tend to preserve lower cost edges, rather than randomly choosing them. They are applied on the $G_{cr} = (V, T_1 \cup T_2)$ graph, where T_1 and T_2 is the set of edges of parent trees, that is, after joining two candidate solutions, the crossover operator chooses which edges will make up the child solution. This operation is shown in Figure 1.

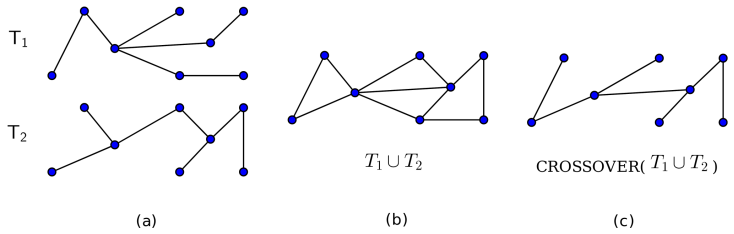

Figure 1. Crossover operator to trees

In Figure 1 (a) two trees T_1 and T_2 are presented. The Figure 1(b) shows a tree obtained by joining the set of edges T_1 and T_2. And in the Figure 1(c) the resulting tree is shown after the application of one of the crossover algorithms in the tree of Figure 1(b).

The first crossover operator, based on the Prim's algorithm [Prim 1957], constructs a solution by adding new vertex to a forming tree, in a relatively greedy way. After inserting a random vertex to start the solution, the other vertex are inserted iteratively, and at each iteration only the first or second lower cost edge can be added. The operator is shown in the Algorithm 1.

$G(V, E) = G_1 \cup G_2$; // parents
$T \leftarrow \emptyset$;
choose a random starting node $s \in V$;
$C \leftarrow \{s\}$; // set of connected nodes
$A \leftarrow \{e \in E | e = (s, v), v \in V\}$; // eligible edges
while $C \neq V$ **do**
  $rndG =$ random number $\in \{1, 2\}$;
  choose the $rndG^{th}$ edge $(u, v) \in A, u \in C$ at min cost;
  $A \leftarrow A - \{(u, v)\}$;
  if $v \notin C$ **then** // connect v to the partial tree
    $T \leftarrow T \cup (u, v)$;
    $C \leftarrow C \cup \{v\}$;
    $A \leftarrow A \cup \{e \in E | e = (v, w) \wedge w \notin C\}$;
  end
end
return T;

Algorithm 1: Relatively greedy Prim crossover

The second operator, creates the individual child using a heuristic similar to the Kruskal's algorithm [Kruskal 1956]. Initially the child receives all the edges that are common in both parents, then the other edges are added iteratively based on a tournament between two random edges, one from each parent. The tournament selects the lowest cost edge, and if its addition does not generate cycles, it is embedded in the tree. The Algorithm 2 shows this operator.

Both operators are considered as exploitation, since only the edges that make up the parents are used to create the child, and it is not possible to generate "new" edges.

$G(V, E) = G_1 \cup G_2$; // parents
$T \leftarrow \emptyset$;
for *all edge* $e(u, v) \in E$ **do**
  if $e \in E_1$ *and* $e \in E_2$ **then** // common edges
    $T \leftarrow T \cup \{(u, v)\}$;
    $E \leftarrow A - \{(u, v)\}$;
  end
end
while $|T| < |E| - 1$ **do** // remaining edges
  choose an edge $(u, v) \in (E \cap E_1)$;
  choose an edge $(x, z) \in (E \cap E_2)$;
  if *length*(u, v) $<$ *length*(x, z) **then**
    $(a, b) \leftarrow \{(u, v)\}$;
  else
    $(a, b) \leftarrow \{(x, z)\}$;
  end
  $E \leftarrow E - \{(a, b)\}$;
  if (a, b) *are not yet connected in* T **then**
    $T \leftarrow T \cup \{(a, b)\}$;
  end
end
return T;

Algorithm 2: Relatively greedy Kruskal crossover

Mutation The mutation operator aims to (re) introduce genetic information into the population. To provide high locale, a mutation operator usually must make a small change in a parent. For tree-based problems, the least possible change is the replacement of an edge with another viable edge [Raidl and Julstrom 2003]. Two greedy mutation operators are presented, the first one is able to generate changes of greater impact in the solution, supporting the exploration, since the second one, as it only changes nearby edges, creates a descendant that is a close neighbor.

The first operator chooses a random edge and removes it, which results in two disconnected sub-trees, then two vertices are chosen, one in each part to form a new edge, which reconnects the sub-trees. Among the candidate edges for reconnection, it is chosen randomly between the first or the second edge of lower cost.

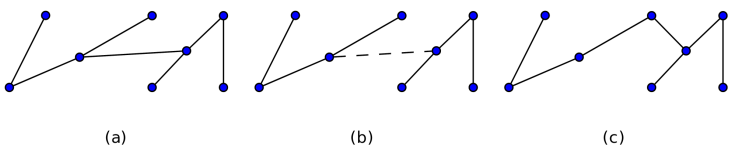

Figure 2. First mutation operator (Rem)

Figure 2(a) shows an example tree to which the first mutation operator will be applied. In the tree of Figure 2(b), a random edge, shown by the dashed line, is removed. Figure 2(c) shows the resulting tree after adding a new edge that reconnects the subtrees.

The second operator randomly chooses a vertex $v1$, and one of its adjacent vertices $v2$, disconnects $v1$ from $v2$, and reconnects it to the vertex adjacent to $v2$ that is closer to $v1$, called $v3$. This operator creates small modifications in the topology, generating offspring close to the parent.

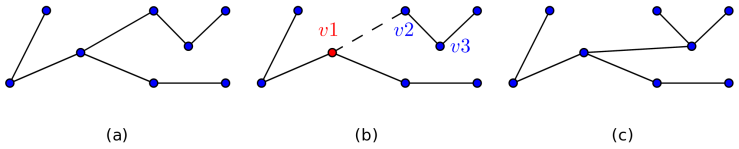

Figure 3. Second mutation operator (Adj)

Figure 3(a) shows the tree to which the second mutation operator will be applied. In the tree of Figure 3(b), a vertex ($v1$) and one of its incident vertices ($v2$) are randomly chosen. $v1$ is disconnected from $v2$ and reconnected to the vertex $v3$, which is adjacent to $v2$. Figure 3(c) shows the resulting tree after mutation.

4. Computational Experiments

To evaluate GA's ability to provide good quality solutions, the proposed algorithm was implemented in the C++ language, and ESTPs are used from the OR-Library benchmark library [Beasley 1990]. Each problem has 15 instances, and for comparisons, optimal problem solutions are used with 10 to 100 points in $\mathbb{R}^2$. The quality of the solutions is calculated by the reduction percentage of the solution over the minimum spanning tree (MST), so that T is a solution obtained for a set of points X, the quality of the solution is given by $Q = ((l(MST) - l(T)) * 100)/l(MST)$. As the algorithm is stochastic in nature, to obtain the best solution and the mean solution, in each experiment each instance is executed 30 times with a population of 50 individuals.

4.1. Operator Analysis

The first test investigate the performance of genetic operators of crossover and mutation in a steady-state GA. The use of this type of GA, instead of the populational GA tends to support computational time and memory consumption, since populational GA creates a new population with each generation, to which the steady-state creates only a new individual. For the experiments, the crossover operators were called "Prim" and "Kruskal", because they are based on these algorithms, the mutation operator that removes an edge and divides the tree into sub-trees and then regroups it called a "Rem", and the mutation operator that creates edges based on adjacent vertices is called "Adj".

The experiment was performed with all instances until an optimal result is reached or there is no improvement in the solution for 100 generations. The results show that the Prim crossover operator converges faster than Kruskal, which is due to the fact that Prim is more greedy because in a solution under construction in choosing a new edge, Prim compares the cost of several, while Kruskal compares only two. As Kruskal preserves the common edges of the two parents, at first many bad edges can be maintained in the solutions. As for the mutation operators, Rem converges faster than Adj, because it generates solutions that are more distant from the parents, supporting the exploration, and consequently, the maintenance of the population diversity.

4.2. Generation Gap Algorithm

In a steady-state GA, only one individual is replaced in a fixed-size population with each generation. By increasing the amount of substitutions per generation in a steady-state GA, new individuals mingle with individuals from older generations, i.e, there is a generation overlap, and then the algorithm is called a "generation gap algorithm" [Jong and Sarma 1992]. The first study attempting to evaluate the effects of overlapping generations is presented in [De Jong 1975]; in it a parameter G called a *generation gap* is used to control the fraction of the population to be replaced with each generation.

Since g is the percentage of individuals replaced in a population of size n in a generation, this experiment shows the effect of the overlap of generations on the genetic operators, through the variation of g on a population with $n = 50$. In this experiment, the crossover operators (Prim and Kruskal) conjugated with the best mutation operator of the initial experiment (Rem) were tested. The g parameter was tested with $10, 25, 50, 75$ and 90% and the individual child replaces the worst individual in the population. Figure 4 shows the average result of this experiment, applied under the 1st instance of the of 100 points problem for 30 generations.

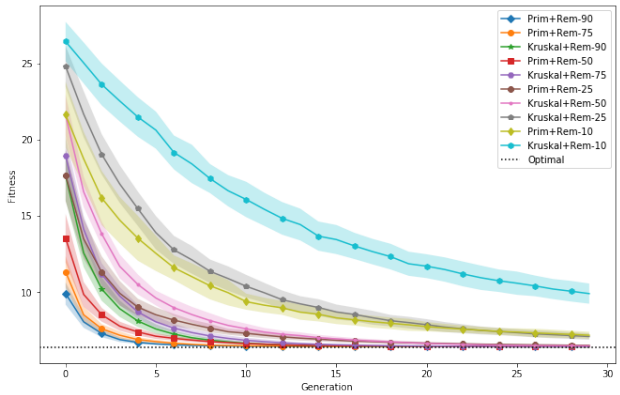

Figure 4. Operators on generation gap algorithm. Instance: 100.1

In Figure 4, tests with $g > 50$ show a fast convergence with a high rate of overlapping generations. The "Prim+Rem90" and "Kruskal+Rem90" tests have the best results to run in a few generations, but they also have a longer average run time.

In order to diversify search space exploration methods, one approach in which the two mutation operators ("Adj+Rem") are applied together was tested with $g = 90\%$. The maximum number of generations is defined as 5 times the number of points of the problem, establishing a proportionality with the size of the problem. Algorithms are applied under all instances until an optimal result is reached or stagnant for 50 generations. The average results are shown in Figure 5.

Also in Figure 5, we can see that the approaches using the two mutation operators together have achieved better results. In addition, the Prim crossover operator found better results on most problems when compared to Kruskal. it is important to note the worst individual in the population is removed at each application of genetic operators, as a consequently, population diversity also decreases very fast, and lead the algorithm stuck in local optima.

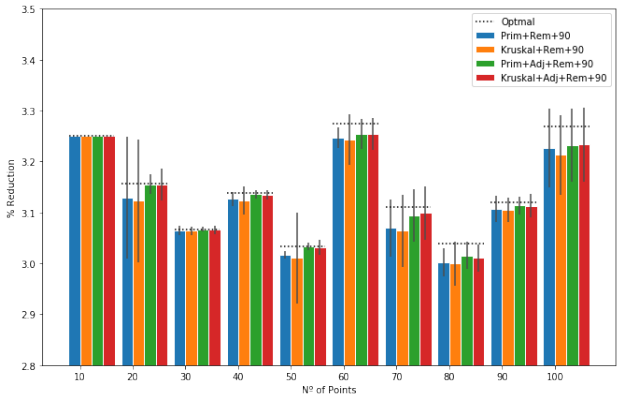

Figure 5. Operators on generation gap algorithm

4.3. Populational Reset

To increase population diversity, an experiment was conducted to investigate the effect of population redefinition in case of stagnation. In this experiment, the best approach of the previous experiment ("Prim+Adj+Rem+90") is compared to the same approach with the addition of two types of population reset. Execution ends if fitness does not improve for 25 followed generations, and population redefinition whenever the population standard deviation is less than 0.03. The first test shows a reset that holds the best individual and replaces the rest with new solutions and is called "Prim+Adj+Rem+RstElitist", and the second replaces the best individual in the population with a new one, and is called of "Prim+Adj+Rem+RstBest". The results of this experiment are shown in Figure 6.

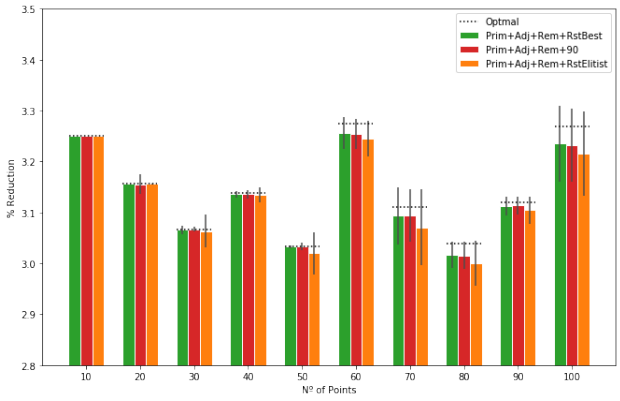

Figure 6. Operators on generation gap algorithm with reset

In Figure 6, the "Prim+Adj+Rem+RstBest" approach yields a better result than "Prim+Adj+Rem+RstElitist", showing an improvement of 0.01% in the mean reduction percentage, and 30% in the average execution time, when compared to the non-reset approach. As selection is by tournament, withdrawal of the best individual allows other individuals with potential to develop. The removal of many individuals is not beneficial as much genetic material is lost, which makes the approach "Prim+Adj+Rem+RstElitist" approach inferior to the approach that does not use reset.

4.4. Performance Analysis

The last experiment uses the "Prim+Adj+Rem+RstBest" configuration, in which 90% of the population is replaced with each generation. If n is the number of points, the execution is terminated when it reaches the optimum, run for $n*10$ generations, or if the fitness does not improve for 50 consecutive generations. If the population reaches a standard deviation lower than 0.03, a population reset is applied that replaces the best individual.

The results are compared to the stochastic algorithms of Beasley and Goffinet [Beasley and Goffnet 1994] and Laarhoven and Ohlmannn [Laarhoven and Ohlmannn 2010], because through the first it is possible to observe the evolution in the quality of the approaches over time, and the second presents the latest comparable stochastic approach. Both heuristics use the Delaunay triangulation to generate candidate Steiner points. Table 1 present the comparison of the results.

n	Beasley and Goffinet (1994)	Laarhoven and Ohlmann (2010)		Hybrid Genetic Algorithm		Opt.
	$\overline{\sigma}\pm$Std.	σ^n_{best}	$\overline{\sigma}\pm$Std.	σ^n_{best}	$\overline{\sigma}\pm$Std.	
10	3.22±1.875	**3.25**	3.22±0.151	**3.25**	**3.25±0.000**	3.25
20	3.12±0.972	3.15	3.13±0.058	**3.16**	**3.16±0.000**	3.16
30	2.95±0.754	**3.07**	3.03±0.068	**3.07**	**3.07±0.004**	3.07
40	2.97±0.633	**3.14**	3.11±0.051	**3.14**	**3.14±0.000**	3.14
50	2.92±0.423	**3.03**	3.00±0.049	**3.03**	**3.03±0.014**	3.03
60	3.18±0.371	**3.27**	3.23±0.058	**3.27**	**3.26±0.031**	3.27
70	2.95±0.374	**3.11**	3.08±0.050	**3.11**	**3.09±0.055**	3.11
80	2.92±0.686	**3.03**	2.98±0.048	**3.03**	**3.02±0.025**	3.04
90	2.95±0.502	3.11	3.04±0.062	**3.12**	**3.12±0.014**	3.12
100	3.07±0.337	3.26	3.19±0.061	**3.27**	**3.23±0.065**	3.27

Table 1. Performance Analisys

In Table 1, "n"represents the number of fixed points of the problem, "σ^n_{best}" the average of the best percent reduction per point, "$\overline{\sigma}\pm$Std." the percentage of average reduction per point and its standard deviation, and "Opt." the average reduction of the optimal per point available in the Or -Library.

Analyzing the results it is possible to see that the Genetic Algorithm presents better results than the heuristics that use the Delaunay triangulation. With respect to GA, the average percentage of reduction ($\overline{\sigma}$) presents optimal or very close results for the tests performed with $10, 20, 30, 40, 50$ and 90 points; and for the others, the reduction is from 0.01% to 0.04% below the optimum. The best percent reduction (σ^n_{best}) is not optimal for 80 points, where GA does not find the best solution in any of the 30 runs of 2nd, 3rd, and 4th instance, however, the result is 0.01% less than optimal.

To compare the means of the two best algorithms, a paired *t-test* was used, since the values are in a normal distribution. The test resulted in $p_{value} \approx 0$, showing that there is a significant statistical difference between the algorithms.

5. Conclusions

ESTP is a difficult problem to be solved, and since exact methods can be computationally demanding, this study presents a heuristic for ESTP in $\mathbb{R}^2$ whose base is a genetic algorithm. The heuristic has greedy genetic operators proper to the tree data structure, coupled with a method that allows the transformation of spanning trees into Steiner trees.

The initial experiments investigate the overlap of generations through a gerational gap algorithm in which a new individual replaces the worst individual in the population. Larger overlays were more efficient, although they consumed more computational time. The use of two mutation operators was also efficient because it allows different ways of exploring the search space. Tests with 90% substitution per generation showed that the operator of crossover Prim with the operators of mutation Ajd and Rem obtained the best result.

With the gerational gap, the heuristic showed an improvement, however, fit individuals can dominate the population quickly, which led to population reset investigation. No reset resulted in an increase in expressive quality, however, the technique of removing the best individual in case of stagnation resulted in a 30% improvement in runtime.

In the end, an GA prepared on the basis of the experiments was compared with other relevant heuristics in the literature. The GA showed good performance, generating results close to that of optimal solutions in viable computational time.

Acknowledgments

The authors would like to thank Federal University of Technology - Parana and National Council for Scientific and Technological Development (CNPq), for supporting this study.

References

Barreiros, J. (2003). An hierarchic genetic algorithm for computing (near) optimal Euclidean Steiner trees. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*. Springer-Verlag.

Beasley, J. E. (1990). OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*.

Beasley, J. E. and Goffnet, F. (1994). A Delaunay triangulation-based heuristic for the Euclidean Steiner problem. *Networks*.

Bereta, M. (2018). Baldwin effect and lamarckian evolution in a memetic algorithm for Euclidean Steiner tree problem. *Memetic Computing*.

Brazil, M., Graham, R. L., Thomas, D. A., and Zachariasen, M. (2013). *On the Story of Euclidian Steiner Tree Problem*. Archive for History of Exact Sciences.

De Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan. AAI7609381.

Deza, M. M. and Deza, E. (2009). *Encyclopedia of Distances*. Springer Berlin Heidelberg.

Dreyer, D. and Overton, M. (1998). Two heuristics for the Euclidean Steiner tree problem. *Journal of Global Optimization*.

Fampa, M. and Anstreicher, K. M. (2008). An improved algorithm for computing Steiner minimal trees in Euclidean d-space. *Discrete Optimization*.

Forte, V. L. d., Montenegro, F. M. T., Brito, J. A. d. M., and Maculan, N. (2015). Iterated local search algorithms for the Euclidean Steiner tree problem in n dimensions. *International Transactions in Operational Research*.

Gilbert, E. N. and Pollak, H. O. (1968). Steiner minimal trees. *SIAM Journal on Applied Mathematics*.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison Wesley.

Hanan, M. (1966). On Steiner's problem with rectilinear distance. *SIAM Journal on Applied Mathematics*.

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

Jesus, M., Jesus, S., and Márquez, A. (2004). Steiner trees optimization using genetic algorithms. In *Technical report*. Centro de Simulação e Cálculo.

Jong, K. A. D. and Sarma, J. (1992). Generation gaps revisited. In *FOGA*.

Koch, T. and Martin, A. (1998). Solving Steiner tree problems in graphs to optimality. *Networks*.

Kruskal, Jr., J. B. (1956). On the shortest spanning tree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc.*

Laarhoven, J. W. and Ohlmannn, J. W. (2010). A randomized Delaunay triangulation heuristic for the Euclidean Steiner tree problem in $\mathbb{R}^d$. *Journal of Heuristics*.

Luke, S. (2013). *Essentials of Metaheuristics*. Lulu.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press.

Prim, R. (1957). Shortest connection networks and some generalizations. *Bell System Tech. J.*

Raidl, G. R. and Julstrom, B. A. (2003). Edge sets: an effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*.

Smith, W. D. (1992). How to find Steiner minimal trees in Euclidean d-space. *Algorithmica*.

Thompson, E. A. (1973). The method of minimum evolution. *Annals of human genetics*.

Wang, L. and Du, D.-Z. (2002). Approximations for a bottleneck Steiner tree problem. *Algorithmica*.

Whitley, D. (1988). Genitor : a different genetic algorithm. *Proceedings of the Rocky Mountain conference on artificial intelligence, 1988*.

Winter, P. and Zachariasen, M. (1997). Euclidean Steiner minimum trees: An improved exact algorithm. *Networks*.

Yang, B. (2006). A hybrid evolutionary algorithm for the Euclidean Steiner tree problem using local searches. In *Knowledge-Based Intelligent Information and Engineering Systems*.