Evolutionary Intelligence
https://doi.org/10.1007/s12065-023-00893-5

RESEARCH PAPER

On the computation of Delaunay triangulations via genetic algorithms

Paraskevas Dimitriou[1] · Vasileios Karyotis[1]

Received: 18 January 2023 / Revised: 17 November 2023 / Accepted: 25 November 2023

Abstract
In this work, we introduce a new approach for computing Delaunay triangulations. Delaunay triangulations have numerous applications in geolocation, communications and other ICT systems or practical applications. Having available various types of approaches for computing such structures is rather desired from an implementation and computational point of view. We adopt Genetic Algorithms for computing Delaunay triangulations and present the design and evaluation of our novel approach. We consider a set of points in the plane as vertices and connect them with edges, creating the point graph. We have developed in C++ an application framework based on genetic algorithms, called `Delaunay_Genetic`, which produces the Delaunay triangulation structure of a given set of points in the plane. `Delaunay_Genetic` considers a novel graph-based chromosome representation of desired solutions, creates an initial population of individuals (chromosomes), an initial generation, and produces from the original population (generation) new generations of individuals in each repetition of the genetic process of Reproduction. Each new generation emerges more robust than the previous one. Our evaluations have revealed that the Delaunay triangulation yielded by `Delaunay_Genetic`, achieves an accuracy of 98–100% of the optimal Delaunay triangulation, while maintaining good convergence speed. Despite its limitations in computational time and space, the proposed novel approach exhibits several complementary benefits to computational geometry based approaches, such as allowing the insertion of new points in the triangulation dynamically, leading to seamless adaptation to new conditions, parallelization of the computational process and tolerance to noise regarding the coordinates of the points. Therefore, this work provides a useful alternative approach for computing Delaunay triangulations.

Keywords Genetic algorithms · Delaunay triangulation · Evolutionary computing · Computational geometry · Performance evaluation

1 Introduction

Various ICT applications employ Delaunay triangulation (DT) in their computations as part of their broader operations. Relevant examples span many different and diverse fields of interest, such as Geographical Information Systems (G.I.S.) [1], data visualization and interpolation, optimization, pattern recognition applications [2], mobile ad hoc networks [3], etc. In general, a triangulation is a subdivision of a planar object into triangles. Delaunay triangulation aims to maximize the minimum of all the angles of the triangles formed in a Triangulation. We consider the typical case of given points on a plane, which may correspond to locations, data, people, etc. Therefore, in such case, DT refers to maximizing the minimum of all angles formed by the given discrete points on a plane.

Delaunay Triangulation Definition, [4]: Suppose T_1 is a Triangulation of a set of points P belonging to the same plane, from which m triangles arise. This means that there will be $3m$ interior angles of triangles in total, denoted as $a_1, a_2, ..., a_{3m}$. Assume these angles are sorted in ascending order of their size, i.e., $a_i \leq a_j$, where $i < j$. One may define $\mathbf{a}(T_1) = [a_1, a_2, ..., a_{3m}]^T$ the vector containing these angles. Let T_2 be another Triangulation of the same set of points P, and let $\mathbf{a}(T_2) = [a'_1, a'_2, ..., a'_{3m}]^T$ be its vector angle. T_1 angle vector will be greater than T_2 angle vector if $\mathbf{a}(T_1)$ is lexicographically greater than $\mathbf{a}(T_2)$, i.e., if there is an index $1 \leq i \leq 3m$ such that:

✉ Vasileios Karyotis
karyotis@ionio.gr

Paraskevas Dimitriou
c20dimi@ionio.gr

[1] Department of Informatics, Ionian University, Tsirigoti Sq. 7, 49100 Corfu, Greece

Published online: 28 December 2023

$a_j = a'_j$ for every $j < i$, and $a_i > a'_i$.

In this way we define that $\mathbf{a}(T_1) > \mathbf{a}(T_2)$. According to this, a triangulation T is called optimal Triangulation or Delaunay Triangulation, if $\mathbf{a}(T) \geq \mathbf{a}(T')$ for all T' triangulations of P point set. □

From the above definition of optimal triangulation, one may conclude that the optimal triangulations will include triangles with the largest smallest angles relative to a non-optimal triangulations. Thus, if the angles of two triangulations are sorted in ascending order, the one that is optimal will always be lexicographically larger than the non-optimal.

Delaunay triangulation is popular among many different ICT applications because the triangles it produces are of very high quality. Every triangle in Delaunay triangulation is as close to the equilateral triangle as possible and extreme triangles (triangles with very small angles-long sides) typically need to be avoided in different applications. For example, when creating contour curves in maps, it is typically better to avoid equilateral triangles [4]. Similarly, in wireless ad hoc networks, a transmitter strives to avoid forming extreme triangles with its neighboring nodes, since the latter lead to long distances between the transmitter-receiver pair, and thus increased power consumption and interference [3].

Due to the widespread use of Delaunay triangulation in many applications, several algorithms have been developed attempting to triangulate large sets of points as efficiently as possible. There are various relevant families of algorithms, according to the employed operation from each algorithm, such as Local Improvement Algorithms, Incremental Instruction or Construction, Sweep-line approach, Incremental Insertion Algorithms, Incremental Search or Gift Wrapping Algorithms, Divide and Conquer Algorithms, and Higher Dimensional Embedding Algorithms. We provide a brief overview of each family in the following section.

It is possible to increase the efficiency of each algorithm by using special data structures, e.g., temporary storage, etc. Algorithmic complexity varies in terms of speed and storage and usually it is between $O(n^2)$ and $O(nlogn)$, depending on the family the algorithm belongs to and its implementation. In exceptional cases a complexity of $O(n)$ has been achieved.

Our work takes on a different approach. We introduce the use of evolutionary computing, and more specifically genetic programming for computing Delaunay triangulations. We present the design and evaluation of this novel approach, which for the first time it employes Genetic Algorithms for the solution of DT. We consider a set of points in the plane connected with edges creating the point graph. We have developed an application framework based on Genetic Algorithms, called `Delaunay_Genetic`, in C++, which produces the Delaunay Triangulation structure of a given set of points in the plane. `Delaunay_Genetic` considers a novel chromosome representation of desired solutions, creates an initial population of individuals (chromosomes), an initial generation, and reproduces from the original population (generation) new generations of individuals in each repetition of the genetic process of Reproduction. Each new generation is more robust than the previous one. Our evaluations have revealed that the Delaunay triangulation yielded by `Delaunay_Genetic`, achieves an accuracy of 98-100% of the optimal Delaunay Triangulation, while maintaining good convergence speed. The proposed novel approach exhibits several complementary benefits, such as allowing the insertion of new points in the triangulation dynamically, thus adapting to new conditions, parallelization of the computation capability and tolerance to noise regarding the coordinates of the points. At the same time it exhibits lower convergence time and higher memory requirements than the state-of-the-art computational geometry based approaches. Therefore, one may consider the proposed approach as a useful alternative for computing Delaunay Triangulations in scenarios where the traditional computational geometry algorithms cannot perform seamlessly.

The rest of this paper is organized as follows. In Sect. 2, we present relevant works and distinguish our contribution. Section 3 describes the considered model and develops the genetic algorithm approach, while Sect. 4 presents the developed algorithm. Section 5 provides evaluation results, and finally, Sect. 6 concludes the paper.

2 Related work

In this section, we present the relevant state-of-the-art and distinguish our contribution. We first review approaches related to Delaunay triangulations, and then review approaches related to genetic algorithms.

2.1 Computing delaunay triangulations

Local Improvement Algorithms: These algorithms [6, 7], are greedy and are mainly used in two-dimensional spaces. First they classify the set of points with respect to an axis and then calculate by some method the resulting triangle (e.g., the sweep line method), and then add a new point from the remaining ones. After each point entry, the algorithm connects it with the rest in an evolving convex hull. The initial Triangulation is then optimized to Delaunay Triangulation as follows: flip if necessary the diagonal of the convex tetrahedron formed by the addition of the new point (flip edge method) so that the criteria of either the empty outline circle or the maximization of the smallest angle are met (max-min angle). All other edges are added to a queue to check them in turn.

The success of the local improvement algorithm is guaranteed and leads to Delaunay Triangulation. However, in 3D space it does not always succeed because the flip edge method does not work effectively. The complexity of this type of algorithm is $\Omega(n^2)$, since it is necessary several times after an edge reversal to check backwards and the neighboring curved quadrilaterals if they contain right common edges or they also need reversal.

Incremental Instruction/Construction: Algorithms of this type [7–14], start by connecting two adjacent points from the set of P points, thus creating the first edge. Then they find the circumscribed circles with the smallest radius to add new points from the given set to the already existing Triangulation. These circles refer to the edges of the convex hull formed in each Triangulation. The convex hull is constantly increasing in size until the points of the set that do not participate in the Triangulation are exhausted. Such algorithms need to know in advance all the points of the given set in order to be able to select each time the new points that will participate in the Triangulation. Usually for greater speed it is selected from the beginning to classify the set of points, in relation to an axis, or to insert them in a suitable structure (e.g., sparse arrays). These algorithms (with proper preparation of the given set of points) have complexity usually $O(n\ logn)$, which however increases when the distribution of points is random, and can reach in the worst case $O(n^2)$.

Sweep Line Method This method is used in several incremental construction algorithms [7–9], and guarantees a complexity of $O(n\ logn)$ in the worst case. It is commonly used in two-dimensional spaces and uses the Delaunay circumscribed circle criterion. According to the Sweep line method, we first classify the set of points with respect to a component of the coordinates of the points. Then we define a horizontal line, which divides the plane into two parts, above and below the line. In the general step of the algorithm, below the line (or above if one starts scrolling from top) are the points of the set that are already triangulated and above the line the non-triangulated ones. The horizontal line moves each step upwards if:

- it meets the vertex of a circumscribed circle and adds the new triangle formed in the Triangulation as legal.
- it meets within a circumscribed circle of a triangle a new point, then it produces new circumscribed circles for each side of the triangle except its external side.

Incremental Insertion Algorithms: Algorithms of this type [14] are quite similar to Incremental Instruction algorithms in that they also add new points to an existing Triangulation. However, there are two important differences. First, the new points that are added are within the existing Triangulation. Secondly, the order in which the new points will be added is not important, which does not require any special distribution of points. The Triangulation is constructed so that all points of the set are inside a large triangle (the tetrahedron for three-dimensional space), so only the width of the space of the points is needed as input.

The most typical algorithm of this category is the **Bowyer-Watson** algorithm. According to it, the algorithm creates a a super triangle, which inscribes all the points of the set P that we take as input. A point of P to this super triangle is added each time and all edges for which the circles of the points that make them up contain the new point (that is, we use the Delaunay criterion) are deleted. The convex polygon created by the erasure of the edges is triangulated relative to the new point. By deleting the edges and re-triangulating, one always manages to have optimal Triangulation after each insertion of a new point. The complexity of the algorithm depends on the distribution of points it takes as input. If point distribution is normal or if the points it receives as input are sorted by an axis, then there is no need to make many Triangulations and asymptotically the execution time becomes $O(n\ logn)$. However, in the case of random point distribution, this time in the worst case can become $O(n^2)$.

Incremental Search or Gift Wrapping Algorithms: Algorithms of this kind [15] are usually applied in three-dimensional spaces, but are not widely used due to their high complexity. The algorithm starts by selecting a triangle serving as the initial base (seed), which has two sides incomplete. An unfinished face is one that is not yet part of a tetrahedron consisting of Delaunay triangles (constrained Delaunay tetrahedron). For a face to be complete it must either be a boundary face of the convex hull to be formed, or a new tetrahedron containing it must have been formed. This algorithm uses a suitable structure, for instance a hash table, in which the incomplete faces, initially the two sides of the first triangle, are stored. The algorithm steps are as follows:

1) Randomly select from the abstract structure an incomplete face let f and find a vertex (point) υ that completes f.
2) If there is no such vertex then f is the boundary (outer) face of the convex hull that has been created.
3) If f is not a boundary face and has no special symbolism that it is indeed a boundary face, then take the vertex υ and form a new tetrahedron with this face.
4) Check the faces of the new tetrahedron formed outside of f if they already exist in the abstract structure.
5) Remove those that already exist in the abstract structure from them because they are now complete, otherwise add each new aspect to the abstract structure.
6) If there are no faces in the structure then end.

The complexity of the algorithm is high because each step requires $O(n_v n_f)$ time. Therefore, the complexity is $O(n_v n_f n_s)$, where s is the number of points. The complexity can be improved by using the Sweep line algorithm for better results.

Divide and Conquer Algorithms: Algorithms of this type [16–18] are optimal in two-dimensional spaces and have complexity $O(n\ logn)$ in the average and in the worst case. In the beginning, the set of points P is divided retroactively into subsets until each subset has a few points and its Triangulation takes time $O(1)$. Then begins the phase of merging the triangulated subsets from level to level until the final Triangulation occurs. The phase with the highest cost is that of merging, because that is where the points of each sub-problem are initially triangulated, but also the correction of the existing Triangulations, a correction which in the worst case can be spread to all existing triangles. The implementation of these algorithms is quite difficult due to the merging phase and becomes even more difficult when the dimension of the point space is greater than 2, which also increases the complexity (for dimension 3 it is $O(n^3)$). However, due to the nature of Divide and Conquer algorithms they can be used and adapted relatively easily to parallel processing programs [18], which makes them attractive. In case they are used in parallel processing programs, some knowledge of the set of points they receive as input is usually needed in order for the division and consequently the paralleling of the sub-problems to be as balanced as possible.

Higher Dimensional Embedding Algorithms: Algorithms in this category [19] project the points of a space E^d into a space E^{d+1} thus creating a convex case. From the faces of the triangles that constitute the convex case that has been formed and their projection in space E^d, the Delaunay Triangulation emerges. The complexity of this type of algorithm is usually $O(n\ logn)$ in the average case and $O(n^2)$ in the worst case.

2.2 Genetic algorithms

Genetic Algorithms have been used as a tool in several problems of Computational Geometry [4]. The high complexity that characterizes many of the problems of Computational Geometry, especially when the volume of data to be calculated is large enough, has led in many cases to Genetic Algorithms. This is due to their random nature, as an algorithmic method of problem solving, as well as the low complexity of implementation that they have exhibit.

Below we present some research works solving problems of Computational Geometry using Genetic Algorithms. In these works, we can see that the use of Voronoi diagrams as well as the solution of its dual problem, Delaunay Triangulation, can be used to solve many problems of Computational Geometry.

2.2.1 An algorithm for constrained coverage problem

The work in [21] tries to find an efficient solution to solve problems related to sensor networks (or antennas for wireless communication) and in particular locations where sensors must be installed in order to have the best possible network coverage. The problem consists of a given set of points in a space as well as a radius, namely the range of the sensors. The objective is to find the least possible number of points to place the sensors in order to have the best possible coverage of the area defined by the points of the given set. Also, there are some restrictions, i.e., some sub-areas within the large area that define the points of the given set, in which sensors cannot be placed. These areas are usually represented by polygons and represent natural obstacles such as lakes, highways, etc. Due to such constraints, the complexity increases considerably and becomes $O(mh^2)$, where m is the number of vertices of the polygons given as constraints and h the number of points in the area defining its convex hull. Also, in addition to the area restrictions there is a coverage radius limit of each sensor that is set for all sensors. The authors proposed a Genetic Algorithm to solve the problem, with the following key points:

Population size and initialization: To initialize the population, they first calculate the convex hull of the area and take the points that make it up. Random combinations of these points will be the initial population of each individual (chromosome) may have a different size (different number of points-genes). At chromosome points they randomly add some sensors. Sensors number must be minimized so that they can cover the entire area.

Selection and Fitness Function: Algorithm selects from the total population the most robust individuals whose descendants will be passed on to the next generation. Each individual in the population can potentially become a parent in a pseudo-random but also objective way as follows: At first, algorithm randomly selects a position of an individual from 1 to the total number N. Let r be the range of each sensor to be placed. Then using the objective (fitness) function algorithm choose the best individual who are in positions k to $k + r$ (if the sum exceeds the length of the population then the algorithm continues from the beginning). After the first parent is found, the previous steps are repeated so that the second parent can be found, and process stops. The objective function applied to the prospective parents of individuals of the population examines at every step the number of sensors that each individual of the population has, in order to select individuals with the minimum sensors number, even if overlapping, who can cover the requested area.

Crossover operator: For each pair of individuals (parents) have been selected with the selection operator, the crossover operator is applied. For each point contained by the first parent algorithm find the first and second nearest points to it, of the second parent. Next calculates Manhattan distance differences between these three points. First parent points are then sorted according to the value found in descending order. Then, for each point of the first parent, the point closest to the second parent is selected for all points of the first parent, creating pairs of points that are multiplied linearly by a random value between 0 and 1, thus producing two new prospective offspring. This process is repeated for the second parent just as it was done for the first taking two other prospective offspring. Among them, the two best offsprings are chosen to pass on to the next generation.

Mutation Operator: Mutation operator is used to avoid unwanted convergences to local optimums by randomly changing one or more points of some individuals of the generated population at a time. In this step algorithm selects with some probability (usually small) an individual of the population and then finds the point which encloses most of the points in the circle centered on this point, i.e. they actually find a Voronoi cell with the most points inside it. If the radius of this circle is greater than the range of a sensor, then algorithm adds a sensor to that individual in the population otherwise removes one.

According to the authors, their algorithm can provide a satisfactory solution in time $O(Nmn^2)$, where N is the size of the population, m is the number of polygons which constitute the constraints and n is the number of points given as input to the algorithm and which constitute the covered area. Without restrictions the algorithm runs in time $O(Nnlogp)$, where p is the sensors number.

2.2.2 A genetic algorithm for generating alternative mobile agent tracks to avoid moving obstacles

The work [22] proposes a combination of Genetic Algorithm and Computational Geometry (Delaunay Triangulation and Linear Interpolation) to calculate the trajectory a moving agent must follow, in order to avoid another moving agent to reach his goal. The application comes from the online game Soccer where the first agent represents a player with the ball in his possession, while the second agent a defender.

To make the algorithm faster they divide it into two parts or phases, the offline phase where the player with the ball in his possession is not threatened, and the online phase where this player will have to choose a trajectory in order to avoid a defender who closes the passage. In the offline phase of the algorithm all the calculations are done with the help of the Genetic Algorithm they propose, i.e., all the trajectories are produced approximately according to the data that exist at the moment, while in the online phase, where a trajectory must be selected immediately, they choose the best with the help of Delaunay Triangulation and linear interpolation applied to the point where the player (agent) is with the ball in his possession but also to the point where the defender is. We can see here the use of Delaunay Triangulation as a tool for encoding a space in order for an algorithm to have criteria to make important decisions about the continuation of its processing.

The following are elements and steps of the proposed algorithm used in the offline stage, where the genetic algorithm is applied:

Trajectory Plans: They are the people (chromosomes) of the population at all times and contain all the information needed in order to be able algorithm to make critical decisions during the game. Each trajectory plan consists of many nodes, each of which contains the (x, y) coordinates of the point where the node is located, as well as the acceleration of the agent, the direction of his body, and the direction of the ball.

Population size and initialization: Initially an original trajectory plan is created through the DEdit tool by inserting all the nodes with some arbitrary initialization and then based on this prototype the individuals of the initial population are created.

Selection and Fitness Function: The selection of an individual to pass on to the next generation is done in a deterministic way using the objective function which evaluates this person. The objective function for a trajectory plan calls the node objective function to evaluate each node it contains and the total sum is the result that returns. Each node of a trajectory plan is evaluated based on specific values of various variables it contains. Some of these variables are the node variable that shows the direction of the agent's body and the ball in his possession, the node variable that contains the angle formed by the direction of the ball with the direction of the defender and the node variable that contains the angle that forms the direction of the attacking agent with the direction of the defender.

At the online stage of the game and after the various trajectory plans have been produced, the agent in possession of the ball will have to decide which trajectory to choose in order to avoid another agent. Depending on the position of the agent, the appropriate trajectory plan is selected and based on this, the trajectory that will be followed each time is selected. Because the point where the agent is located does not necessarily belong to a node of the trajectory plan, the Delaunay Triangulation is used in all nodes of the trajectory plan as well as the linear interpolation to find the nodes of the triangle in which the agent is in possession of ball.

Crossover operator: It is applied between two parents and produces two offsprings to be passed on to the next generation. The process of crossover is repeated continuously until the number of offsprings reaches the desired number and is applied to two parents again based on the probability of crossover or not. According to the crossover process proposed by the authors, the parameter k of each offspring acquires a weighted content of the corresponding parameters of its parents.

Mutation Operator: The mutation uses a number from the Gaussian distribution that has a value between -1.8 and 1.8 and which is added to all the variables of the chromosome selected for mutation. Also taking part in the mutation is another variable called mutation-coefficient, which also contains a small value, but which is constant and has been applied to the mutations of the original population.

2.2.3 A genetic algorithm for selecting the most appropriate compressed image

The work [23] propose a genetic algorithm in order to be able to select the best compressed image from a number of alternative compressed representatives, based on a special criterion that we will describe below. Their work extends compression methods based on the Delaunay Triangulation, which treats an original image file with scales of gray as an illustration of the 2.5 dimensional space. In this space each pixel of the image consists of the parameters x for the column, y for the row and z for the scale of gray which has values from 0 to 255 (i.e information range 8 bits).

According to compression methods using Delaunay Triangulation, the goal is to select as few of the pixels of an image defined by (x, y, z) as possible from these pixels, but to obtain a satisfactory approximation of the original image. To do this compressed methods consider each pixel as the vertex of a graph whose edges correspond to sides of triangles. In these triangles the Delaunay criterion is applied, i.e that of the circumscribed circle in which only the vertices that make up the triangle are located. At each vertex the x, y coordinates are not affected, while the value z is approached with some tolerance error. This approach is done with the help of linear interpolation. The effort is the difference of the real value of z with its approximation value always being less than an error and this applies to all points - pixels that have been selected for Triangulation and consequently for its approximate (compressed) representation original image.

Population size and initialization: Atoms of the population consist of pixels with information (x, y, z) the number of which is predetermined from the beginning on each chromosome and does not change during processing. Initialization involves the creation of a specific set of chromosomes which, however, come from two sources. In the first case the chromosomes are created by completely random selection of pixels from those of the image, while in the second case the chromosomes have been created by known compression algorithms that use Triangulation. This combination, according to the authors, creates better results.

Selection and Fitness Function: The selection process aims to select a pixel that gives better results than the rest. To do this it uses a criterion called ***vertex selection criterion***. The vertex selection criterion evaluates all vertices (pixels) that have not been added to the Triangulation, in order to select the best of them. In order not to perform complex computational operations and to burden the algorithm with great complexity, a local error metric is usually chosen, which is the vertical error. The evaluation of a pixel located in the $x - column$ and $y - row$ of the image file results from the absolute value of the difference between the actual value of z and its approximation value resulting from the linear interpolation. Having set the maximum vertical error we can now define the metric quality of an Image as PSNR (Peak Signal to Noise Ratio) approach which is measured in db (decibels). The previously mentioned PSNR calculation is used as an objective function. The choice is due not only to the rapid calculations of the vertical error it contains but also to the fact that it gives higher values for the most suitable individuals than the less suitable ones, as stated by the theory of genetic algorithms that it must make an objective function.

Crossover operator: The crossover process followed here is uniform, i.e. each gene (pixel) of one parent corresponds to the gene located in the same position in the other parent. In this process, the genes of the first parent go through and with a 50% probability they alternate the two genes that are in the same positions in each parent.

Mutation Operator: With mutation process, as we know from the theory of genetic algorithms, there is a precaution against converging the algorithm to a local minimum or maximum. Here the probability of mutation of a gene is 2%. During the mutation of a gene, the coordinates of the gene (pixel) change to a small degree, as a result of which it moves in space in a certain direction. Special care has been taken by the authors so that during its movement it does not fall on another part of the chromosome (conflict).

3 Genetic algorithm for delaunay triangulation

The proposed algorithm adopts the Genetic Algorithm paradigm. One of the important issues is the representation of the solution search space. We represent each individual of the population as a graph whose vertices correspond to each point of a given set P that was given as input to the Delaunay Triangulation.

Our approach uses the criterion of legal edge to eliminate illegal edges when they occur. In particular, when two edges are found to intersect, then one of them is legal and the other one is illegal. Both of edges are contained in the quadrilateral which is defined by their four end-points.

The legal edge criterion applies to individuals in the algorithm population, which have been coded as graphs with vertices and edges. Vertices are the points of the plane given as input to the algorithm and edges are connections between vertices of the graph (Fig. 1). Each generation according to the Genetic Algorithms consists of the individuals of the population, i.e., the graphs, on which genetic operators of selection, crossover and mutation are applied. After creating a number of generations our algorithm converges to a solution which is the desired Delaunay Triangulation.

Population Encoding: As we mentioned before, each member of the population is represented in our algorithm as an undirected graph whose vertices correspond to each point in the given set. If two points are joined together then an edge is created with endpoints these two points.

Population Initialization: According to the population encoding individuals as undirected graphs with vertices and edges, each individual in the population initially adds edges between its vertices randomly, but these edges must not intersect. Edges are considered as line segments. Each line segment can intersect at some point (or points) by another one (or more than one) line segment. After every individual initialization, the algorithm checks for crossed edges and removes the illegal ones. Below we describe how to remove the illegal edges.

Population Selection and Fitness Function: In the selection step we use an objective function that returns the ratio of the number of edges of an individual to the total length of these edges. The algorithm first takes the maximum fitness value from the individuals of the population and then for each individual takes its fitness value, divides it by the maximum and produces a pseudo-random number between [0,1] to decide whether this individual will go through the process of crossover. If the result is positive, then selects it, otherwise not. This fact results in the presence of one or more individuals in the population (the most robust) more than once.

Crossover: During the crossover genetic process (Fig. 2), two selected individuals of the population are crossed and from this cross, the algorithm creates two offspring. The offspring will pass on to the next generation replacing their parents.

The proposed algorithm creates these offspring with two different ways:

The first offspring retains the best characteristics of their two ancestors. Specifically, it contains all the edges of the two ancestors which do not intersect. It also contains from the edges of its two ancestors that intersect, those which give the best result. These edges lead to quadrilateral triangles with the largest angles according to Delaunay criterion. It should be mentioned here that this offspring comes mainly from the parent with the best fitness. The algorithm adds to best parent the edges from the second parent that did not exist in the best parent, plus the best of the edges (legal) that intersect between the two parents.

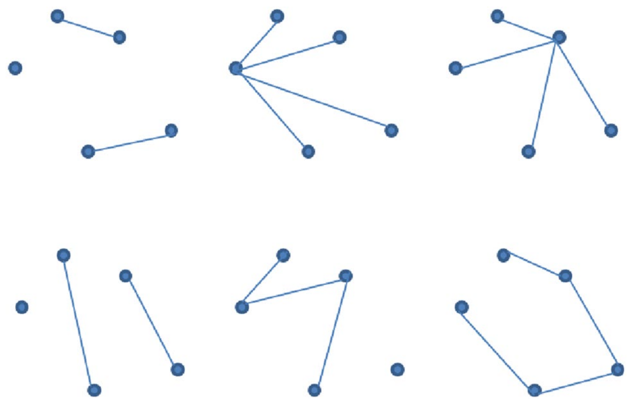
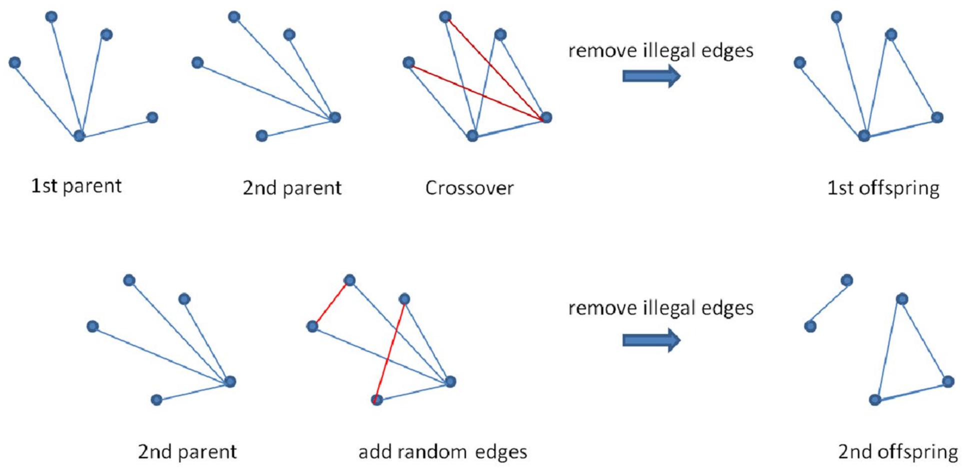


Fig. 1 Sample population with six individuals

Fig. 2 Crossover process



The second offspring arises from the parent with the worst fitness. The algorithm adds to this parent extra edges arbitrarily and randomly. Then, the algorithm removes all illegal edges so that a second offspring is created. This process leads to more remarkable individuals in the population from generation to generation, since edges are selected through the intersections, leading to better results, so to edges that fit better to Delaunay Triangulation.

Mutation: During mutation, the algorithm removes randomly from an individual from one to all its edges. Mutation probability is small, usually 0.1 or less.

Legal edge selection: As mentioned earlier, we consider each edge of the graph as a line segment beginning and ending at the two vertices it joins. These vertices belong to the given set of points in the plane. The algorithm first checks if two line segments (two edges contained in a graph - individual) intersect based on known formulas we have from geometry (use of determinants). If there is a cross, then algorithm checks which one of the two edges is legal and which one is illegal. The algorithm therefore considers these two edges as diagonals of the quadrilateral formed by the points of their ends (Fig. 3). The algorithm calculates the angles of the triangles formed by each diagonal (edge) on the quadrilateral that form their points. For these calculations we use the Cosine Law. Then, the algorithm creates two angle vectors and fills them with the angles of every quadrilateral, sorts them and compares them in lexicographical order. The legal edge corresponds to bigger vector of these.

Figure 3 shows two edges that intersect. The shorter one has p_l and p_k as ends while the other has p_i and p_j. We consider the quadrilateral defined by the points (p_l, p_k, p_i, p_j) once with the diagonal edge defined by the points (p_l, p_k) and once with the diagonal edge defined by the points (p_i, p_j). In the quadrilateral with diagonal (p_l, p_k) we have the angles vector $A(T_1) = (a_1, a_2, a_3, a_4, a_5, a_6)$, while in the quadrilateral with diagonal (p_i, p_j) we have the angles vector $A(T_2) = (b_1, b_2, b_3, b_4, b_5, b_6)$. After sorting them we compare them in lexicographical order. If $A(T_1) > A(T_2)$ then we chose as legal the edge which defined from (p_l, p_k) points, else the edge which is defined from (p_i, p_j) points.

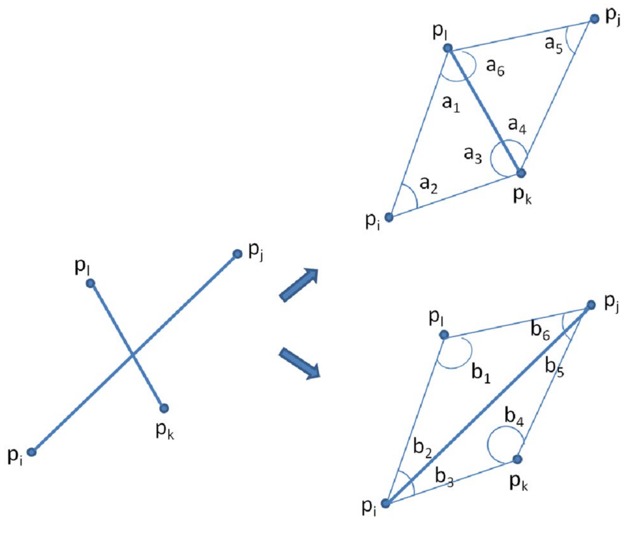


Fig. 3 Legal edge selection process

4 Representation and implementation of proposed algorithm

The designed algorithm, receives as input a file with the point coordinates in the plane and creates a Delaunay Triangulation using a Genetic Algorithm, where the vertices of the Triangulation's triangles are the given points. The algorithm is denoted as `Delaunay_Genetic`. Based on the points of the set, `Delaunay_Genetic` creates graphs, as explained analytically in subsection 4.2, where each graph is an individual of the population. Each graph has as vertices the given points and as edges line segments that connect these graph vertices. The flowchart of `Delaunay_Genetic` is shown in Fig. 4. In the following, we detail the various parts of the `Delaunay_Genetic`.

4.1 Input and parameters

The data entry in `Delaunay_Genetic`, i.e. the points of the given set that we want to triangulate, is done through a text file which has the following format: In the first line of the file there is the total number of points of the set, while in the lines that follow and in each of them there are the coordinates of each point of the set. The coordinates of each line are separated by a space. A sample of a input file is show in Fig. 5. Here we must point out that when `Delaunay_Genetic` reading the file, stores the data in a table structure. `Delaunay_Genetic` then sorts points by x coordinate. This is needed to make a better population initialization later.

`Delaunay_Genetic` can get additional arguments to configure its execution. Specifically, we can enter the number of iterations of the genetic algorithm L, i.e., after how many generations we will get the result. We can also enter the mutation probability M as well as the number of individuals P that will participate in each generation - repetition.

4.2 Chromosome representation

Each Chromosome (individual) in the population is represented in `Delaunay_Genetic` as a graph whose vertices correspond to each points in the given set. If two points are connected then an edge is created between the corresponding vertices. Formally, we consider a Chromosome (points set) as a graph $G = (V, E)$, where V is the vertex set, i.e., labeled

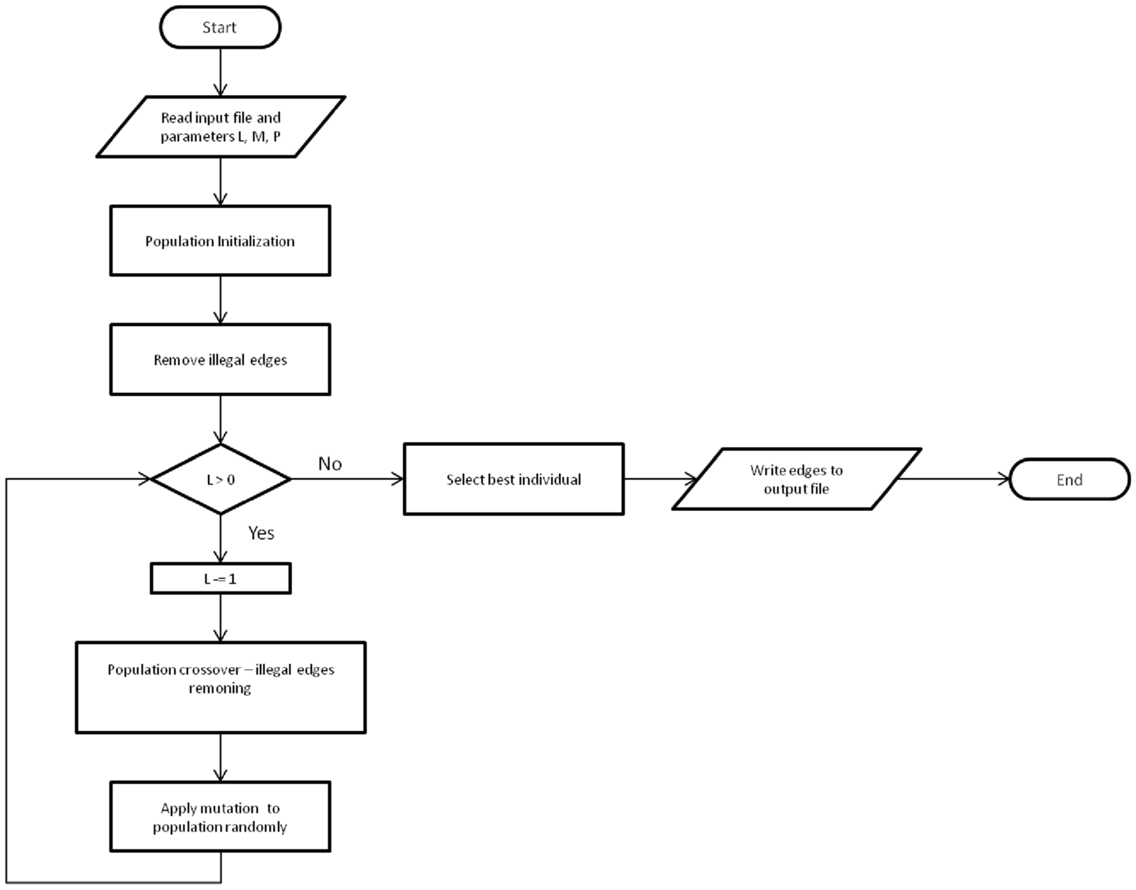
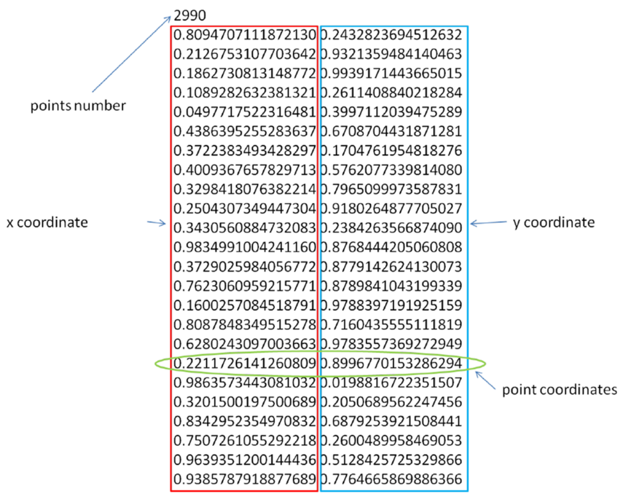

Start
Read input file and parameters L, M, P
Population Initialization
Remove illegal edges
L > 0
No
Yes
Select best individual
Write edges to output file
End
L -= 1
Population crossover – illegal edges remoning
Apply mutation to population randomly

Fig. 4 Delaunay_Genetic Flowchart


points number
x coordinate
y coordinate
point coordinates

```
2990
0.8094707111872130 0.2432823694512632
0.2126753107703642 0.9321359484140463
0.1862730813148772 0.9939171443665015
0.1089282632381321 0.2611408840218284
0.0497717522316481 0.3997112039475289
0.4386395255283637 0.6708704431871281
0.3722383493428297 0.1704761954818276
0.4009367657829713 0.5762077339814080
0.3298418076382214 0.7965099973587831
0.2504307349447304 0.9180264877705027
0.3430560884732083 0.2384263566874090
0.9834991004241160 0.8768444205060808
0.3729025984056772 0.8779142624130073
0.7623060959215771 0.8789841043199339
0.1600257084518791 0.9788397191925159
0.8087848349515278 0.7160435555111819
0.6280243097003663 0.9783557369272949
0.2211726141260809 0.8996770153286294
0.9863573443081032 0.0198816722351507
0.3201500197500689 0.2050689562247456
0.8342952354970832 0.6879253921508441
0.7507261055292218 0.2600489958469053
0.9639351200144436 0.5128425725329866
0.9385787918877689 0.7764665869886366
```

Fig. 5 Input File format sample

points from the point set, and E is the edge set. A vertex u corresponds to a tuple (x, y) where $x, y \in \mathbb{R}$ and represent coordinates of a point in the two dimensional space. For two vertices $u, v \in V$, edge $e(u, v) \in E$ represents a link between vertices u and v. In order to implement such an encoding of a Chromosome `Delaunay_Genetic` uses objects of the classes described below:

`Edge Class`**:** Each object of this class that we have developed implements an edge of the graph and contains two references to objects of the Vertex class (described below), which represent two vertices of a graph, i.e., two points of the input set of points. It also contains a variable, which is the length of the edge and which is calculated when the edge is created. In addition it contains a Boolean variable which is true when this edge intersects with another, otherwise its value is false. Because the graph is undirected, two edges are identical if they contain the same points in any order. Therefore, in this class there is a function that compares two edges of two graphs are identical or not.

`Vertex Class`**:** Each object of this class that we have developed implements a vertex of the graph. It contains a pointer to a point in the input set as well as a map object, which contains all the edges that have this point at one end. For search purposes, each edge has as key in the map an alphanumeric that represents the identifier of the point of the other end of the edge from the vertex where we are. For example, if we connect the point with identifier "10" to the point with identifier "130" with an edge then the key of the map for that particular edge will be "130", that is the other end of the edge. Finally, this class contains functions for inserting an edge into a vertex (vertices connecting), deleting an edge from a vertex (vertices disconnecting), and a function computing the similarity of a vertex with some vertex of another graph.

`Graph Class`: Each object of this class that we have developed implements a graph that is a Chromosome (individual) of the population. This class contains, in principle, a map with the vertices of the graph, whose key is the identifier of a point corresponding to a vertex. It contains another map with the edges of the graph, which has an alphanumeric key as its key, which consists of the two identifiers of the points that make up the edge, separated by an underscore. For example, if an edge connects the points with identifiers "135" and "25" respectively, then the key in the map has the form "135_25" or "25_135" depending on the order in which the ends of the edge have been given. These two edges are identical since the graph is undirected. Other members of this class are a variable that stores the count of the graph's vertices, a variable that stores the count of its edges, and a variable that stores the total length of all the edges of the graph. The latter variable is dynamically updated whenever an edge is added or removed from the graph. In conclusion, each chromosome consists of all points contained in input set and its length is the number of these points.

4.3 Population initialization

We define as population P_n, a finite set of chromosomes (graphs or individuals) $G_i = (V, E_i)$ for $0 \leq i < n$. Set V is always the same for all chromosomes, because all chromosomes have the same vertices but potentially different edges. Hence, the edge sets E_i. Each chromosome in the population is initialized by adding edges between the vertices of its graph. The edges are added methodically and not completely randomly. Specifically, `Delaunay_Genetic` takes advantage of the fact that the vertices of the graph are sorted by the x-coordinate. Thus `Delaunay_Genetic` starts from a random starting point (vertex) and joins the vertices together to form triangles. Next, `Delaunay_Genetic` continues to create triangles until it reaches the last point. Then it randomly adds some extra edges. Finally it checks if there are edges that intersect between the edges added to the graph. If there are, then it removes the illegal edges from them. As a result, there are legal edges in the population, that is, we have more robust individuals.

`Delaunay_Genetic` generally does not allow for intersecting edges in the graphs - individuals of the population. To achieve this it applies a special algorithm which it calls at every step where edges are added to the individuals of the population. The operation of the algorithm is described in next subsection.

4.4 Objective function

With our Genetic Algorithm we try to maximize a function $f(x), x \in P_n$. The problem here is to find chromosome x_{opt} such that:

$$f(x_{opt}) \geq f(x), \forall x \in P_n. \tag{1}$$

Let us assume, without loss of generality, that:

$$f(x) > 0, \forall x \in P_n. \tag{2}$$

Then this becomes a constrained optimization problem. For this problem, we define the objective function or fitness function obj for a chromosome x that is equivalent to the function f:

$$obj(c) = f(x), \tag{3}$$

with c corresponding to x.

In order to select the most suitable individuals of the population, we have implemented the objective (fitness) function as to return for an individual (chromosome) in the population the number of its edges. This is a nice criterion for indicating the value of an individual, because the edges it contains may have been preferred to others that intersected with them and are therefore legal. Concluding, the more legal edges an individual has, the better that individual is. Delaunay Triangulation consists of only legal edges. The number of edges of an individual already exists within, so the objective function just returns this value. Hence its complexity is $O(1)$.

4.5 Selection

The individuals of the population that will participate in the reproduction are represented by a table that contains their positions in the table of the population. We have created two ways to have this table each time.

4.5.1 Stochastic selection

Stochastic Selection follows the following steps for selection:

- Calculate the objective value $obj(c_i)$ for each chromosome $c_i \in P_n$, where $0 \leq i < n$.

- Find the maximum fitness of the population $Obj = max\{obj(c_0), obj(c_1), \cdots, obj(c_{n-1})\}$.
- Calculate the probability g_i of selection, where $g_i = obj(c_i)/Obj$.
- Generate a random number rnd_i from [0, 1] and compare g_i and rnd_i to decide whether the specific individual will go through the process of crossing. If the result is positive, then it inserts its place in the population table in the vector that will return.

Note that, the population table is cyclically accessed each time until the table that returns this function is filled. This results in the presence of one or more individuals in the population (the most robust) more than once, possibly. Also mention that each person selected, occupies the first or last place in the table with a probability of 50%. However, if the population is small, e.g., we only have two individuals, then returns the table with both individuals.

4.5.2 Roulette selection

Roulette Selection follows the next steps for selection:

- Calculate the objective value $obj(c_i)$ for each chromosome $c_i \in P_n$ where $0 \leq i < n$.
- Find the total fitness of the population $Obj = \sum_{i=0}^{n-1} obj(c_i)$.
- Calculate the probability g_i of selection, where $g_i = obj(c_i)/Obj$.
- Calculate cumulative probability $G_i = \sum_{j=0}^{i} g_j$ for $c_i (i = 0, 1, \cdots, n-1)$.
- For $j = 0, 1, \cdots, n-1$,
  - generate a random number rnd_j from [0, 1],
  - if $rnd_j \leq G_0$ then select c_0; otherwise select c_i where $0 \leq i < n$ if $G_{i-1} < rnd_j \leq G_i$.

As we mention just before, Roulette Selection first fills a table, each position of which corresponds to one individual in the population, with decimal numbers that result from the quotient of the fitness value of each individual with the total sum of the fitness values of all individuals in the population. In each position of the table, in addition to the quotient we mentioned, the value of the previous cell of the table is added, i.e., we have progressive sums each time. This table according to the above contains values in the interval [0, 1]. After this table is filled then we start and produce pseudo-random numbers in [0, 1] and for each of them we find the position corresponding to the table by entering this position in the table to be returned. For example, if position 3 contains the value 0.45 and position 4 the value 0.55 and the pseudo-random number is 0.53 then in the table of results will enter 4. However position 4 can be presented more times as a result. Therefore, there may be individuals in the table who will be returned more than once in the population, since there may correspond to the same place in the table that represents an individual more than once. Also here, each individual selected enters the first or last position of the table with a probability of 50%. However, if the population is small, e.g., we only have two individuals, then returns the table with both individuals, as stochastic selection.

4.6 Population crossover

During the genetic process of Crossover, two individuals of the population are crossed and from this cross, two new individuals emerge who will pass on to the next generation in the place of those who were crossed. The two new offspring emerge in two different ways.

Assume from Selection function two chromosomes $parent_1 = (V, E^{parent1}), parent_2 = (V, E^{parent2})$. We can create from these chromosomes two offsprings as follow:

$$offspring_1 = (V, E^{offspring1}), \tag{4}$$

where

$$\begin{aligned} E^{offspring1} =& E^{parent1} \cup E^{parent2} \\ & - not\_legal\_edges(E^{parent1} \cup E^{parent2}). \end{aligned} \tag{5}$$

The function *not_legal_edges* applies the legal edge criterion, as we define it in section III, to every pair cross edges of the edge set that takes as input and returns a set with all not legal edges from an edges set.

$$offspring_2 = (V, E^{offspring2}), \tag{6}$$

where

$$\begin{aligned} E^{offspring2} =& create\_rnd\_edges(V) \\ & - not\_legal\_edges(create\_rnd\_edges(V)). \end{aligned} \tag{7}$$

The function *create_rnd_edges* takes as input a vertices set, create and returns a random edges set. Inside this set probably exists many cross edges.

Let us explain more analytically the above notations. The first offspring retains the best characteristics of their two ancestors. Specifically, it contains all the edges of the two ancestors which do not intersect. It also contains from the edges of its two ancestors that intersect those that give the best result, i.e., lead to quadrilateral triangles with the largest angles. It should be mentioned here that this offspring comes mainly from the parent with the best fitness to which we add the material that is not the same, i.e., the edges, from the second parent plus the best of the edges (legal edges) that intersect between the two parents.

The second offspring comes from the parent with the worst fitness from which we remove all its edges, since the best of them now exist in the first offspring. Then we add new edges to the new offspring in a random but organized way. Specifically, we join each vertex of the graph with an arbitrary number of vertices that are located after it in terms of x coordinates. Then we remove the edges that may intersect and keep the best of them. In this way, offspring emerge with new genetic material that will help converge the algorithm during the next crosses.

4.7 Mutation

The genetic process of Mutation follows the Crossover genetic process and is applied to an individual in the population at random. During the Mutation, some successors are selected in which we remove from one to more edges. This step exists so that according to the theory of Genetic Algorithms to avoid any conversion to a local maximum. In our implementation, mutation is an occasional random removal of an edge from E_i set of selected chromosome $c_i \in P_n$. It is performed on each edge of each chromosome in P_n with the probability $q(> 0)$. Mutation is performed in the following way: For every edge $e_j \in E_i$ of chromosome $c_i = (V, E_i)$:

- generate a random number *rnd* from [0, 1]
- if $rnd \leq q$ mutate the edge e_j by removing it from the E_i so $E_i^{'} = E_i - \{e_j\}$. Otherwise the e_j remains.

4.8 Reproduction

In order to calculate the optimal solution, it is necessary to repeat the steps of Reproduction in the individuals of the population. After a number of repetitions (stopping criterion), there should be an optimal solution and in our case the Delaunay Triangulation of the points of the set that `Delaunay_Genetic` has accepted as input. The basic steps in our Genetic Algorithm are described below.

(1) Generate an initial population P_n of size n and calculate fitness value of each chromosome $c_i \in P_n$.
(2) Perform Selection operation on P_n to result in P_n^1.
(3) Perform Crossover on P_n^1 to result in P_n^2.
(4) Perform Mutation operation on P_n^2 to result in P_n^3.
(5) Write P_n^3 as P_n.
(6) If stopping criterion is not satisfied go to step 2.

The stopping criterion in our genetic algorithm is a finite number of repetitions. More detailed according to our implementation in each loop `Delaunay_Genetic`:

(1) Takes from some selection function, from those we have previously described, the table, each position of which contains the position of a selected individual in the population table.
(2) It creates a new temporary population table, the newpopulation, where we place the individuals of the original population table, which we remove in order to accept the new individuals that will emerge after the application to the individuals of the previous generation of genetic processes.
(3) In individuals of the present generation it applies the process of crossing in pairs as found in the newpopulation table. In the function we call (crossover) we give each pair in the order resulting from their robustness index, as we have mentioned in the Crossover subsection.
(4) Individuals of the new generation that emerge are given to the population table and we empty the newpopulation.
(5) In the individuals of the new generation we apply (in some of them depending on the probability of mutation) the Mutation process.

When looping is over, then we have the final generation from which with a loop we can find the most robust individual, that is, the one with the highest value that fitness returns. For this individual, we call a function that gives us a formatted output file that we describe in next subsection.

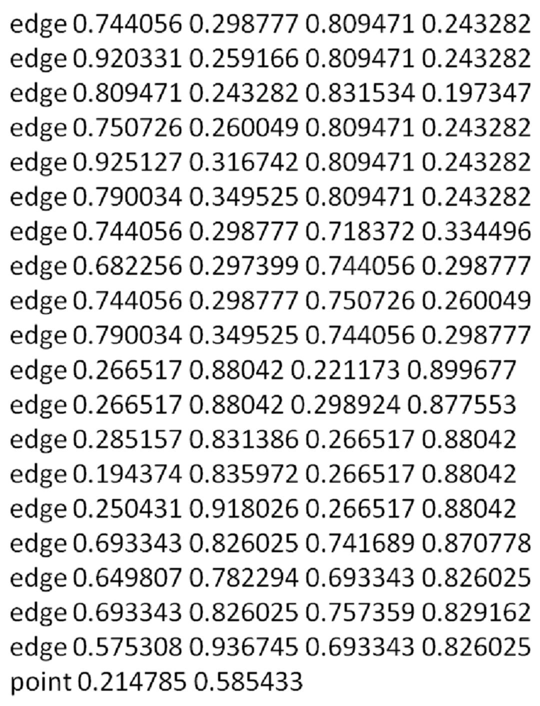
4.9 Output file

Output File is a text file that contains the edges of the best individual of last population. The file name is always `edges_X.txt`, where X is the number of edges contained in this individual. For example output file named `edges_835.txt` means that the file contains 835 edges. Figure 6 shows a file segment of this format.

As shown in this Figure, each line of this file starts with the word edge and is followed by the coordinates of the points that make up the edges of the edge. If `Delaunay_Genetic` fails to connect one point to another to create an edge that starts or ends at that point, then it displays it with the word point and its coordinates, as shown in the last line of the Fig. 6.

4.10 Remove illegal edges

`Delaunay_Genetic` always keeps the individuals of the population consistent. By consistency we mean that in every individual of the population there are no intersecting edges. If there were intersecting edges then the application would never be able to provide optimal solutions. `Delaunay_Genetic` checks for intersecting edges in two cases: Each time edges are added to an individual in the population and each time an intersection is made between two individuals. In both cases `Delaunay_Genetic` calls a check function which it gives as argument a table containing



```
edge 0.744056 0.298777 0.809471 0.243282
edge 0.920331 0.259166 0.809471 0.243282
edge 0.809471 0.243282 0.831534 0.197347
edge 0.750726 0.260049 0.809471 0.243282
edge 0.925127 0.316742 0.809471 0.243282
edge 0.790034 0.349525 0.809471 0.243282
edge 0.744056 0.298777 0.718372 0.334496
edge 0.682256 0.297399 0.744056 0.298777
edge 0.744056 0.298777 0.750726 0.260049
edge 0.790034 0.349525 0.744056 0.298777
edge 0.266517 0.88042 0.221173 0.899677
edge 0.266517 0.88042 0.298924 0.877553
edge 0.285157 0.831386 0.266517 0.88042
edge 0.194374 0.835972 0.266517 0.88042
edge 0.250431 0.918026 0.266517 0.88042
edge 0.693343 0.826025 0.741689 0.870778
edge 0.649807 0.782294 0.693343 0.826025
edge 0.693343 0.826025 0.757359 0.829162
edge 0.575308 0.936745 0.693343 0.826025
point 0.214785 0.585433
```

Fig. 6 Output File sample

edges. This table in the first case is the table of the edges of an individual's graph, while in the second case it contains edges of two individuals to give the best of them to one of these individuals.

The check function firstly orients the line segments (edges) with respect to their *x* coordinate, so that the initial point of the segment is before its end point. Then we enter in a table that we call events, pairs which have as key the start point of each segment and as data the position of the segment in the segments table, so that we can access it at any time. In the events table are all the segments which we have to check if they intersect with each other. Every segment contains a Boolean variable. We set this variable true if a segment is intersected with another segment and is illegal. We have described how to select a legal edge from two edges in section 3.

We also use another table called sweep. This table contain pairs in which the key will be a point and data the position of the segment in the segments table where this point belongs to. In essence, table sweep plays the role of the vertical scan line of the plane and in which we enter each time each point that this line meets. Each time we encounter an initial point of a segment, i.e., it exists in the events table, then we take it out of there and check if these segment intersects with any of those already in the sweep table.

To avoid to check the entire contents of this table, we exclude from the intersection checking the segments that are above the level, or below or even before the part that we check. In fact, those whose end is before the beginning of the segment we are checking are removed from the sweep table. After checking an intersection, if the segment we are examining has set its intersect variable to true, then checking for the specific segment stops and we continue with the next point (beginning of a segment) that contains the event. If not, then we look at the intersection variable of the other segment and if it is true, we remove this segment from the sweep table and we continue checking for the rest segments of the sweep table. When events table become empty then we stop intersection checking.

4.11 Specifications

`Delaunay_Genetic` is written in `C++`. The choice of this language is due to the speed it offers in executing `Delaunay_Genetic` as well as the object-oriented way that a `C++` user can write an application. `Delaunay_Genetic` has been developed with the principles of object oriented programming that lead to more organized and easier to understand code. We have also used some of the structures offered by the `C++, STL (Standard Library)` which are vector, map and multimap. In particular, the choice of the last two `STL` structures was made because they organize their content as balanced binary search trees, so that data can be found within them in a more efficient way. Complexity of finding data in such structures is logarithmic $O(logn)$. Some of the functions we have used are `insert` which inserts a new node in the tree of the structure, `find` which finds a node in the tree in logarithmic time and `erase` which removes a node from the tree of the structure in logarithm time too.

5 Evaluation

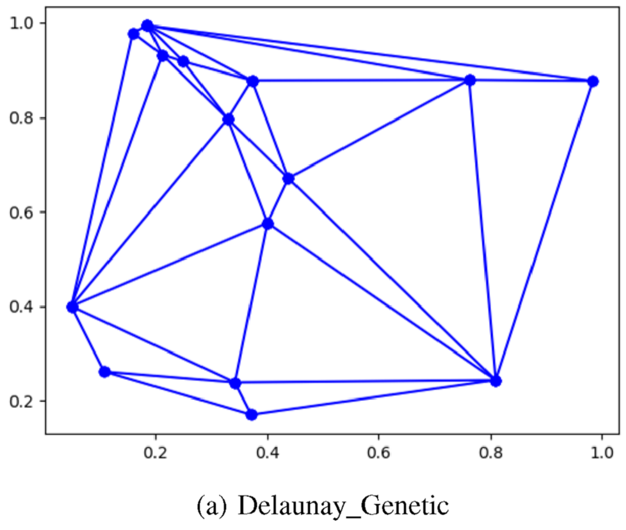
In order to show the correctness of the results of our algorithm we have implemented comparative tests with a well established algorithm, namely `Bowyer - Watson` algorithm. We have used it as benchmark algorithm because it finds the correct Delaunay Triangulation results with low time complexity $O(nlogn)$. Regarding such comparison, there are two major aspects one needs to investigate, namely correctness and complexity. In the comparison of our algorithm with the `Bowyer - Watson` algorithm we evaluated the first on the basis of identifying the edges created by the `Bowyer - Watson` algorithm over a set of points, and checking how many edges are matched by the execution of our algorithm over the same point set. With respect to computation time, we employed the average execution time, computed over a number of different topologies of the same size, as a rather indicative parameter of the complexity involved in each approach.

Regarding the algorithm itself, each execution test shows us the parameters with which `Delaunay_Genetic` ran. Specifically, it shows which selection function was used (roulette or stochastic), the individual number of each generation, how many generations were created (i.e., how many repetitions were made), the probabilities of crossing and mutation, as well as the number of edges created in the various `Delaunay_Genetic` executions. These edges are evaluated for their correctness, by the corresponding edges created by the `Bowyer - Watson` algorithm for the same set of points, and by how many edges are matched by the execution of the two algorithms. There is also execution time for both algorithms (average time for `Delaunay_Genetic`). Finally, in each comparative test we provide figures depicting the two triangulations (in the case of `Delaunay_Genetic` the best it has achieved).

In order to be able to evaluate the results of the benchmarks more effectively, we also present charts that show the evolution of the algorithm from generation to generation and how much each generation contributes to the final result, via the number of edges that implement the Triangulation. Each comparative test corresponds to its chart.

Test_15: The information for this test is shown in Fig. 7 and Table 1. As we observe `Delaunay_Genetic` achieves correct Triangulation, exactly the same as `Bowyer - Watson`. In the chart of Test_15 shown in Fig. 8, we observe that `Delaunay_Genetic` has converged on the solution almost from generation 8 onwards. However due to the initialization of the individuals of the previous generation population, it has already reached an acceptable result from the first generation, having created 30 edges. The majority of these edges are legal, this is due to the nature of the `Delaunay_Genetic` intersection, which eliminates the illegal edges when intersected with legal ones.

Test_55: Similarly, in this Test (Fig. 9 and Table 2) we notice that our Algorithm finds the right Triangulation, but



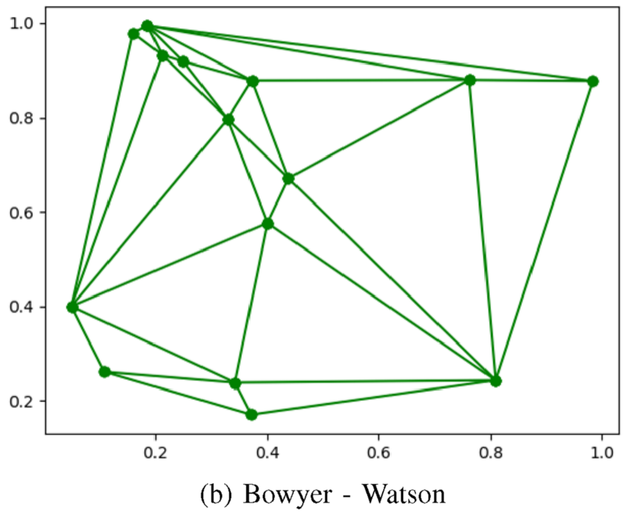


Fig. 7 Comparative Test with 15 points

Table 1 Test_15 Information data

Test Identity: Test_15	
Executions number	10
Selective Function	Stochastic Selection
Population	4
Generations number	10
Crossover Probability	0.99
Mutation Probability	0.14
Delaunay_Genetic average edge number	35
Bowyer-Watson edge number	35
Similarity	100%
Delaunay_Genetic average execution time (seconds)	0.000001
Bowyer-Watson execution time	0.000001

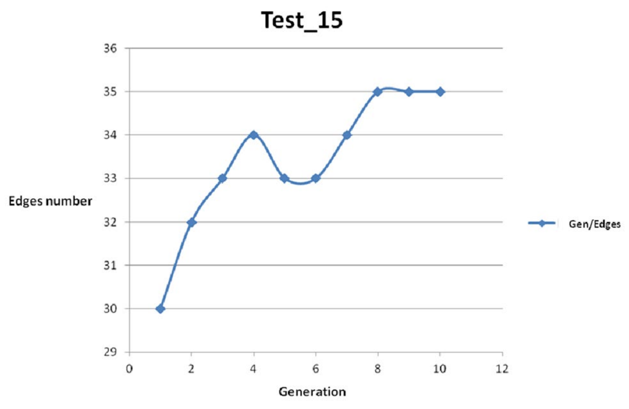


Fig. 8 Test_15 chart

Table 2 Test_55 Information data

Test Identity: Test_55	
Executions number	10
Selective Function	Stochastic Selection
Population	8
Generations number	10
Crossover Probability	0.99
Mutation Probability	0.14
Delaunay_Genetic average edge number	150
Bowyer-Watson edge number	150
Similarity	100%
Delaunay_Genetic average execution time (seconds)	0.000023
Bowyer-Watson execution time	0.000001

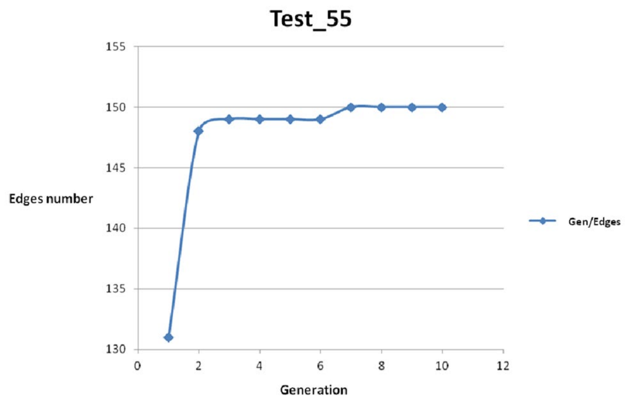


Fig. 10 Test_55 chart

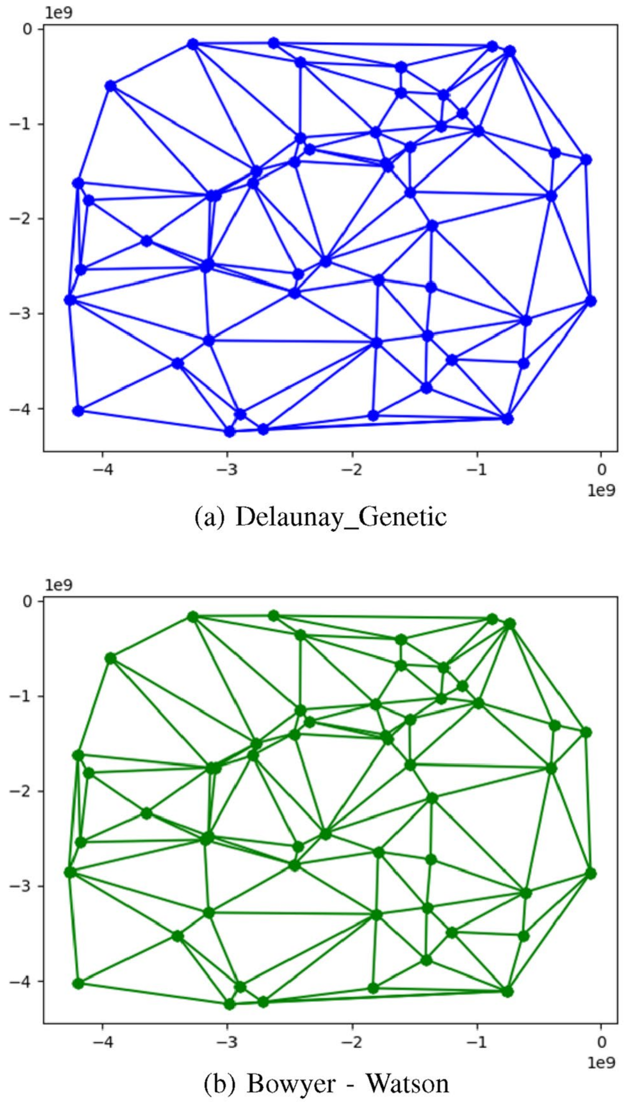


Fig. 9 Comparative Test with 55 points

Table 3 Test_299 Information data

Test Identity: Test_299	
Executions number	10
Selective Function	Stochastic Selection
Population	8
Generations number	10
Crossover Probability	0.99
Mutation Probability	0.14
Delaunay_Genetic average edge number	880
Bowyer-Watson edge number	880
Similarity	99.80%(878*similaredges*)
Delaunay_Genetic average execution time (seconds)	3.625704
Bowyer-Watson execution time	0.000001

it takes longer execution time. In general, `Delaunay_Genetic` in its current version is much slower than `Bowyer - Watson`, but its results are correct. This is very important because of the random nature of the algorithm. In the chart of Test_55 shown in Fig. 10, we observe similar behavior to the previous Test. `Delaunay_Genetic` from the first generation and due to the initialization has already found 130 of the 150 edges, while from the seventh generation onwards has already found the required Triangulation.

Test_299: In this Test (Fig. 11 and Table 3) we observe that our Algorithm almost found the correct Triangulation, i.e., all the edges that make up the triangles except two. In the second or third execution, the algorithm finds 100% of the correct edges. Execution time is quite high compared to the `Bowyer - Watson` algorithm. In the chart of Test_299 shown in Fig. 12, we observe that from the first generation we start to have good results, i.e., it has found about 750 edges of Triangulation out of the 880 required. Then in the next four generations the algorithm manages to

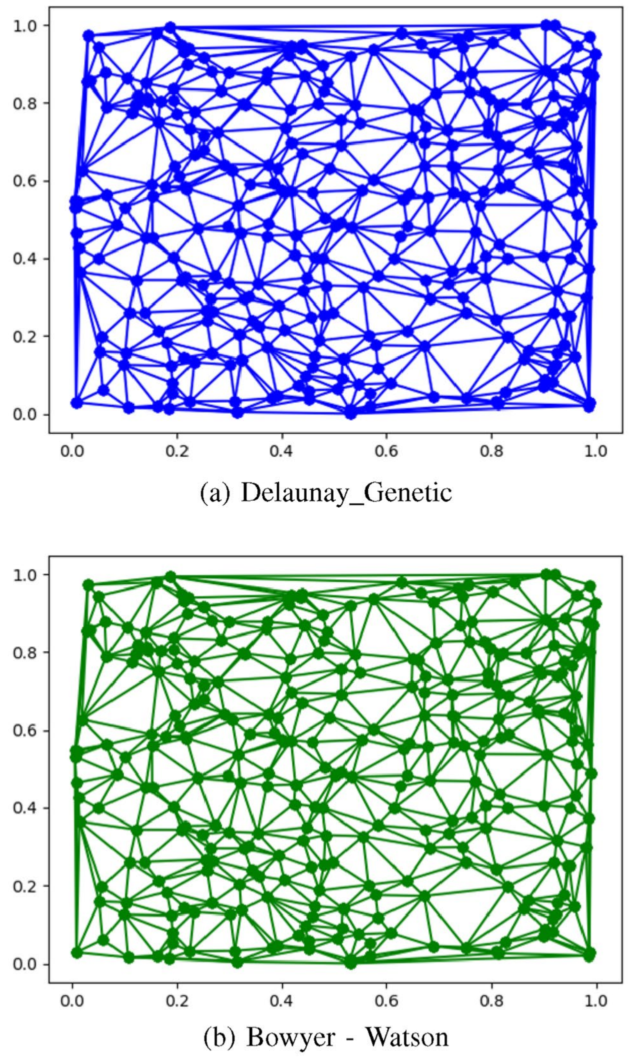


Fig. 11 Comparative Test with 299 points

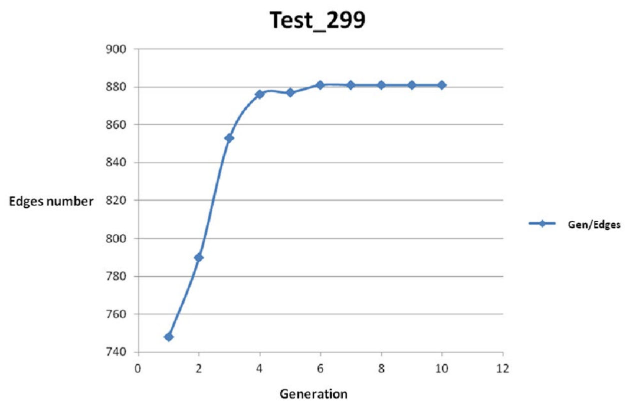


Fig. 12 Test_299 chart

Table 4 Test_999 Information data

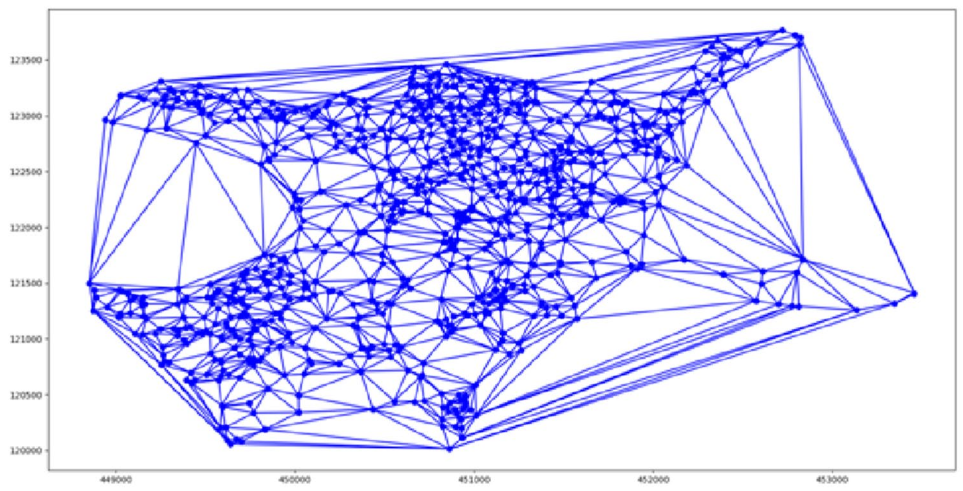
Test Identity: Test_999	
Executions number	10
Selective Function	Stochastic Selection
Population	8
Generations number	10
Crossover Probability	0.99
Mutation Probability	0.14
Delaunay_Genetic average edge number	2951
Bowyer-Watson edge number	2981
Similarity	98.70%$(2941 similaredges)$
Delaunay_Genetic average execution time (seconds)	28.826808
Bowyer-Watson execution time	0.109394

converge towards the solution. During the analysis of the results we noticed that the selection method which is the stochastic one helps a lot. If we replace the stochastic method with the roulette, then performance degrades.

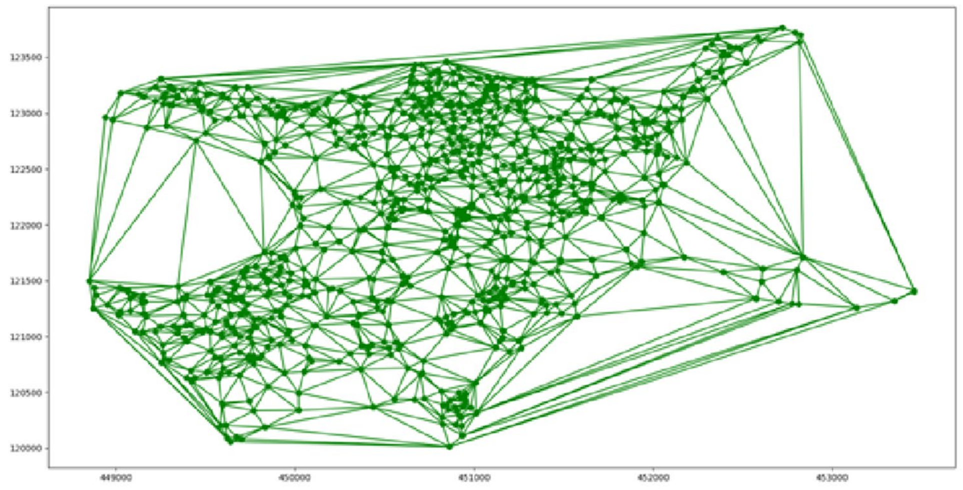
Test_999: In this Test also (Fig. 13 and Table 4), we can observe that although similarity is achieved to a large extent, the execution time of `Delaunay_Genetic` differs significantly from the corresponding execution time of our benchmark algorithm. In the chart of Test_999 shown in Fig. 14, we can see that the convergence of the results with those of benchmark algorithm, is done smoothly from the first generations. And here we notice that since the start of the algorithm we have found most of the edges of the Triangulation (although some of them are not legal), while then the algorithm converges more slowly to an acceptable result.

Sensitivity Analysis Test: To examine the effect of various parameters on our algorithm, specifically crossover and mutation probability and how much their values affect the results of the `Delaunay_Genetic`, we ran the algorithm on the 299-point data set. We present the results from these executions in Table 5 and Fig. 15. From the results of these executions, we infer the important role of crossover in `Delaunay_Genetic`. When the crossover is performed on two chromosomes, the resulting offspring are usually more valuable because they have crossed their edges so that the legal ones have remained from those that make up the optimal triangulation. What matters, of course, is that the parents differ enough from each other to get even better cross-breeding results. On the contrary, during mutation we remove edges hoping that they will be replaced by different ones at some later time, possibly avoiding local minima. However, when we remove edges the chromosomes become poorer from edges so that their crossover does not produce optimal results. Therefore, when the probability of mutation is high then the algorithm cannot produce appreciable results. This can also be seen in Fig. 16. In this Figure we

Fig. 13 Comparative Test with 999 points



(a) Delaunay_Genetic



(b) Bowyer - Watson

show how many edges of the Delaunay Triangulation the three experiment cases find with the different parameters of crossover, mutation that we have performed in any case of them. The case 3 executions experiment is very low in performance than the other two due to the increased mutation probability.

In the same set of points we also examined the effect on `Delaunay_Genetic` of changing population chromosome number as well as the number of generations. We present the results from these executions in Table 6 and Fig. 16. From the results of these executions we conclude that there should be a tradeoff between the number of chromosomes in the population and the generations that we will execute the `Delaunay_Genetic`. For example, in case 2 where we reduced the population by half, we gain in time but the triangulation results are not as good as in case 1. Correspondingly in case 3, where we have kept a low number of chromosomes but have increased the generations from ten to twenty, we notice that we still gain in time (about half of case 1). The results are better than case 2 but still lesser than the results of case 1.

From all the above Tests and charts we can conclude the following:

- The initialization method is very important because it creates several edges between the vertices (points) in each individual (graph) of the population, so that at the intersection the legal ones are selected from them to the individuals of the next generation that will emerge. Generations from the beginning on wards contribute less in terms of the new edges that will be added to the Triangulation. Of course, their role remains important in order to have an optimal result.

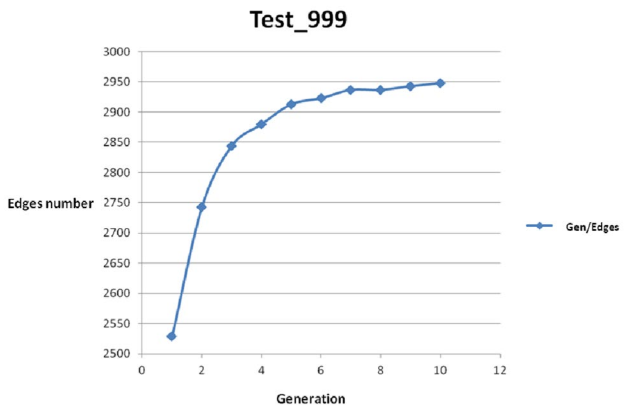


Fig. 14 Test_999 chart

- There are generations that jump in terms of the number of edges they contribute. This means that a suitable condition could be found that leads to generations that can offer greater leaps or that additional research will be useful to lead to more efficient, legal edge-producing generations.
- In all the Tests that have been done, the stochastic selection method yielded better results than the roulette method.
- The running time of the algorithm is quite high compared to the conventional computational geometry based Delaunay Triangulation finding algorithms. This is due to the operation of the intersection which has a polynomial complexity approaching the square. Of course, conventional algorithms have been optimized over time by various researchers, and there is very little literature on finding Delaunay Triangulation using Genetic Algorithms.

6 Conclusion

In this work, we proposed a Genetic Algorithm for finding Delaunay Triangulation given a set of points of the plane. For this purpose we implemented `Delaunay_Genetic`, which manages to find Triangulations very close to the normal Delaunay Triangulation and in most cases with an accuracy that exceeds 98% when it does not find the exact Triangulation. The present paper proposes the first approach for solving the Delaunay Triangulation problem with Genetic Algorithms. Even though Genetic Algorithms have been proposed in a couple of works for solving other geometric problems, no previous attempt on solving the Delaunay Triangulation problem has been made. Therefore, the paper presents the first such approach, investigating the positives and negatives of such an attempt.

The implementation of `Delaunay_Genetic` was based on the coding of population individuals

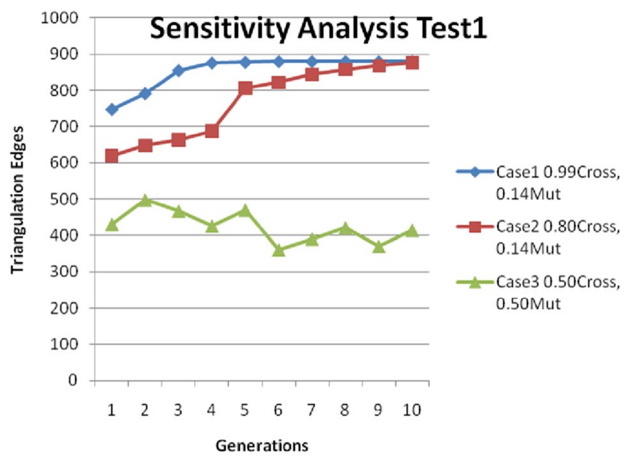


Fig. 15 Sensitivity Analysis Test1_299 chart

Table 5 Test_299 Sensitivity Analysis

Test Identity: Sensitivity Parameters Test1_299			
Executions number	10	10	10
Selective Function	Stochastic Selection	Stochastic Selection	Stochastic Selection
Population	8	8	8
Generations number	10	10	10
Crossover Probability	0.99	0.80	0.50
Mutation Probability	0.14	0.10	0.14
Delaunay_Genetic average edge number	880	876	414
Bowyer-Watson edge number	880	880	880
Similarity	99.80%(878*similaredges*)	99.00%(872*similaredges*)	44.00%(387*similaredges*)
Delaunay_Genetic average execution time (seconds)	3.625704	3.230000	1.558077
Bowyer-Watson execution time	0.000001	0.000001	0.000001

Table 6 Test_299 Sensitivity Analysis

Test Identity: Sensitivity Parameters Test2_299			
Executions number	10	10	10
Selective Function	Stochastic Selection	Stochastic Selection	Stochastic Selection
Population	8	4	4
Generations number	10	10	20
Crossover Probability	0.99	0.99	0.99
Mutation Probability	0.14	0.14	0.14
Delaunay_Genetic average edge number	880	688	847
Bowyer-Watson edge number	880	880	880
Similarity	99.80%(878*similaredges*)	65.00%(575*similaredges*)	90.34%(795*similaredges*)
Delaunay_Genetic average execution time (seconds)	3.625704	0.916522	1.860036
Bowyer-Watson execution time	0.000001	0.000001	0.000001

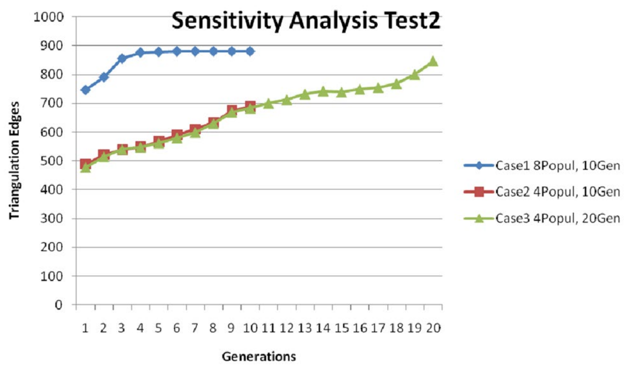


Fig. 16 Sensitivity Analysis Test2_299 chart

(chromosomes) in graphs with vertices the points of the given set of points and edges the straight segments connecting these points. During its operation, `Delaunay_Genetic` tries to find the sides of the triangles, which are the edges of the final graph, using the Delaunay criterion for lexical comparison of the angles of the triangles. These edges result from the operation of the Crossover between individuals of the population from generation to generation. In the final outcome there is no illegal edge in the resulting Triangulation.

The implementation of Delaunay Triangulation with Genetic Algorithms offers several benefits over other attempts to implement it. As Genetic Algorithm `Delaunay_Genetic` can work in dynamic environments, i.e., in environments where the data changes during the operation of the algorithm, for example, when dynamic new points to be triangulated are added or when some points are removed. In these environments, the known algorithms have to be restarted, while `Delaunay_Genetic` can continue its execution adapting to the new conditions that have arisen.

As a consequence of the previous benefit, we can manage and process more easily data that is given to us and has sources of uncertainty, i.e. it has noise. The dynamic nature of offsprings generation also allows us during the processing and search for optimal triangulation to be able to change the context of the search space by possibly incorporating new constraints or even changing or modifying the objective function we use. Therefore we can triangulate a space that initially has noise at the given points and then adjust the initial triangulation to more reliable data.

An additional benefit we have is that `Delaunay_Genetic` can be parallelized due to its nature thus increasing its calculation speed. We can therefore divide the initial population into subsets and compute the Delaunay Triangulation in less time by exploiting the multicore CPUs or GPUs of our system. Similar to the modeling of the problem we propose, we can also work with Distributed systems by making the necessary changes to our `Delaunay_Genetic` algorithm.

In addition to the benefits resulting from the novel modeling proposed in this paper, there are drawbacks and limitations. A drawback is that there are cases where `Delaunay_Genetic` does not always find the optimal solution even though the solution it finds is quite close to it (close to 99%). Another drawback of `Delaunay_Genetic` is that it requires more computation time than the faster traditional Delaunay Triangulation algorithms. Also a limitation that exists and is inherent in the Genetic Algorithms methodology is the memory requirements that exist in order to maintain and evolve the population of possible solutions. As a limitation, we could also mention that there is no clear termination condition, as a result of which we have to predetermine the number of generations that must pass in order to stop the execution of the algorithm.

In the future, we will focus towards improving several aspects of our approach, especially the execution time, utilizing various software and hardware improvement capabilities as we mention previously.

Acknowledgements Not applicable.

Author Contributions VK conceived the original idea, while PD developed the proposed algorithm and prepared its implementation. PD performed all evaluations. PD and VK prepared the text of the manuscript. All authors reviewed the manuscript.

Funding Open access funding provided by HEAL-Link Greece.

Declarations

Ethics statement All of the material is owned by the authors and/or no permissions are required.

References

1. Roy P, Mandal JK (2012) A novel spatial fuzzy clustering using delaunay triangulation for large scale GIS data (NSFCDT). Procedia Technol 6:452–459
2. Ogawa H (1986) Labeled point pattern matching by delaunay triangulation and maximal cliques. Patt Recogn 19(1):35–40
3. Li X-Y, Calinescu G, Wan P-J, Wang Y (2003) Localized delaunay triangulation with application in Ad Hoc wireless networks. IEEE Trans Parall Distrib Syst 14(10):1035–1047. https://doi.org/10.1109/TPDS.2003.1239871
4. Berg M, Cheong O, Kreveld M, Overmars M (2008) Computational Geometry Algorithms and Applications, 3rd edn. Springer-Verlag, Berlin, Heidelberg
5. Maur P (2002) Delaunay Triangulation in 3D. State of the Art and Concept of Doctoral Thesis. Technical Report No. DCSE/TR-2002-02. http://www.kiv.zcu.cz/publications/
6. Sloan SW (1987) A fast algorithm for constructing Delaunay Triangulations in the plane. The University of Newcastle, Australia. https://www.newcastle.edu.au/__data/assets/pdf_file/0017/22508/13_A-fast-algorithm-for-constructing-Delaunay-Triangulations-in-the-plane.pdf
7. Yonghe L, Jinming F, Yuehong S (2013) A simple sweep-line delaunay triangulation algorithm. http://papersub.academicpub.org/Global/DownloadService.aspx?ID=15630
8. Shewchuk JR (2000) Sweep algorithms for constructing higher-dimensional constrained Delaunay Triangulations. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.40.8648 &rep=rep1 &type=pdf
9. Biniaz A, Dastghaibyfard G (2011) A faster circle-sweep Delaunay Triangulation algorithm. http://cglab.ca/~biniaz/papers/Sweep%20Circle.pdf
10. Boissonnat J-D, Devillers O, Hornus S (2009) Incremental construction of the Delaunay triangulation and the delaunay graph in medium dimension. https://hal.archives-ouvertes.fr/inria-00412437
11. Guibas L, Knuth D, Sharir M (1992) Randomized incremental construction of Delaunay and Voronoi Diagrams. http://www.wias-berlin.de/people/si/course/files/Guibas92-RandomizeIncr.pdf
12. Lischinski D (1993) Incremental Delaunay Triangulation. http://www.karlchenofhell.org/cppswp/lischinski.pdf
13. Anglada MV (1998) An improved incremental algorithm for constructing restricted Delaunay Triangulations. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.61.3862 &rep=rep1 &type=pdf
14. Liu J-F, Yan J-H, Lo SH (2013) A new insertion sequence for incremental Delaunay Triangulation. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.470.9175 &rep=rep1 &type=pdf
15. Tianyun S, Wang W, Zhihan L, Wei W, Xinfang L (2015) Rapid Delaunay Triangulation for randomly distributed point cloud data using adaptive Hilbert curve. https://www.sciencedirect.com/science/article/abs/pii/S0097849315001223
16. Tanemura M, Ogawa T, Ogita N (1983) A new algorithm for three-dimensional voronoi tessellation. https://www.sciencedirect.com/science/article/pii/0021999183900876
17. Cignonit P, Montanit C, Scopigno R (1998) DeWall: A fast divide and conquer Delaunay Triangulation algorithm in E^d. http://www.personal.psu.edu/faculty/c/x/cxc11/AERSP560/DELAUNEY/8_Divide_and_Conquer_DeWall.pdf
18. Lemaire C, Moreau M (2000) A probabilistic result on multi-dimensional Delaunay Triangulations, and its application to the 2D case. https://core.ac.uk/download/pdf/82499702.pdf
19. Wenzhou W, Yikang R, Fenzhen S, Liang C, Jiechen W (2014) Novel parallel algorithm for constructing Delaunay Triangulation based on a twofold-divide-and-conquer scheme. https://www.tandfonline.com/doi/abs/10.1080/15481603.2014.946666
20. Tereshcenko V, Taran D (2012) Optimal algorithm for constructing the Delaunay Triangulation in E^d. https://publications.waset.org/11000/pdf
21. Davoodi M, Mohades A, Rezaei J (2009) A Genetic Algorithm for the Constrained Coverage Problem. https://www.researchgate.net/publication/225722346
22. Amoozgar M, Khashabi D, Heydarian M, Nokhbeh M, Shouraki SB (2012) Generating Motion Patterns Using Evolutionary Computation in Digital Soccer. https://www.researchgate.net/publication/233982312_Generating_Motion_Patterns_Using_Evolutionary_Computation_in_DigitalSoccer
23. Sakhi AOB, Naimi BHM (2015) New Search Method for Approximating Images with Delaunay Triangulation. https://www.researchgate.net/publication/275885483_New_Search_Method_for_Approximating_Images_with_Delaunay_Triangulation