International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:1, 2015

A Genetic Based Algorithm to Generate Random Simple Polygons Using a New Polygon Merge Algorithm

Ali Nourollah, Mohsen Movahedinejad

***Abstract*—**In this paper a new algorithm to generate random simple polygons from a given set of points in a two dimensional plane is designed. The proposed algorithm uses a genetic algorithm to generate polygons with few vertices. A new merge algorithm is presented which converts any two polygons into a simple polygon. This algorithm at first changes two polygons into a polygonal chain and then the polygonal chain is converted into a simple polygon. The process of converting a polygonal chain into a simple polygon is based on the removal of intersecting edges. The experiments results show that the proposed algorithm has the ability to generate a great number of different simple polygons and has better performance in comparison to celebrated algorithms such as space partitioning and steady growth.

***Keywords*—**Divide and conquer, genetic algorithm, merge polygons, Random simple polygon generation.

I. Introduction

THE presented algorithm in this paper generates random simple polygons from a given set of points called S, where $S = \{p_1, p_2, \dots, p_n\}$ and the points lie in a two dimensional plane. The problem of generating simple polygons interests researchers because of its wide area of applications. The most important application of generating random simple polygons is to evaluate the correctness of other computational geometry algorithms which their input is simple polygons. Generating sufficient test cases from user world to test the computational geometry algorithms is a challenging task. To ease testing procedure, an algorithm can be designed to generate polygons uniformly at random.

Unfortunately, there is no polynomial-time algorithm to generate all possible simple polygons with a given vertex set uniformly at random. All previous works are based on applying some heuristic or generating a specific class of polygons. Some of these heuristics has caused the generation of specific classes of polygons such as monotone polygons [1] and star shaped polygons [2]. Although there are a lot of heuristics to generate random polygons from n points, by now it is an open problem to generate all the polygons uniformly at random. An algorithm can generate simple polygons uniformly at random if and only if the probability of generating each polygon is $1\ j$; where the total number of possible polygons to generate is j.

Ali Nourollah is with the Shahid Rajaee Teacher Training University, Tehran.
Mohsen Movahedinejad is with the Shahid Rajaee Teacher Training University, Tehran (e-mail: m.movahedinejad@srttu.edu).

Each heuristic omits one part of the problem space and then generates the polygons, which this causes some polygons not to be generated by the heuristic. Since genetic algorithm searches the whole problem space, it can generate all the possible polygons, but it is too slow to solve the problems with great n. To overcome this problem a divide and conquer approach is used. So the proposed algorithm can solve problems with more than 5000 points.

Since 1992, the generation of geometric objects has become an interesting research topic to the researchers for its different applications. Epstein [3] studied random generation of triangulation. Zhu designed an algorithm to generate an x-monotone polygon uniformly at random on a given set of vertices [1]. A heuristic [4] for generating simple polygons was investigated in 1991. The “2-Opt Move” heuristic was first proposed to solve the traveling salesman problem by J.van Leeuwen et al. [5]. In 1996, Thomas Auer et al. [2] presented a study of all heuristics present at that time and reported a variety of comparisons among them.

The organization of paper is as follows. In the second section the needed preliminaries are mentioned. In the third section the genetic algorithm is discussed and in the fourth section the divide and conquer process is covered. The fifth section shows the experimental results and the comparisons among the existing algorithms.

II. Preliminaries

The notations used in this paper are as follows. The input points set which lies in a two-dimensional plane are shown with $S = \{p_1, p_2, \dots, p_n\}$, where n is the number of input points. The input points are supposed to lay in general positions for simplicity. A polygonal chain is specified by a sequence of points $(q_1, q_2, \dots, q_n)$ called its vertices [9] and a simple polygonal chain is one in which only consecutive segments intersect and only at their endpoints [9]. The vertices with degree one is called the heads of the chain. The words chain or polygonal chain would refer to a simple polygonal chain if seen anywhere in the paper.

The terms points and vertices are used interchangeably throughout the paper. A polygon P is said to be simple [6] if it consists of straight, non-intersecting line segments called edges that are joined pair-wise to form a closed path. The adjacent edges of polygon meet only at their common end point known as vertices. An edge connecting two points p and q are denoted by pq. Here and throughout the paper, unless qualified otherwise, we take polygon to mean simple polygon on the plane. A subset S of the plane is called convex [6] if and only if for any pair of points $(p,q) \in S$ the line segment pq is completely contained in S. The Convex Hull $CH(S)$ of a point set S is the smallest convex set that contains S. Convex hull of a point set S is represented by set of vertices that defines hull edges. Many algorithms have been presented [6]-[10] for finding convex hull. A point p is said to be visible [11] from a point q if the line segment pq does not intersect any other line segment nor does not pass through a third point r.

III. Genetic Algorithm

Genetic algorithm is a special kind of evolutionary algorithms that use Biology techniques such as inheritance and mutation. This algorithm has been applied to a wide variety of combinatorial optimization problems and is the subject of numerous recent books [12]-[15], [20] and conference proceedings [16]-[19]. Some traditional heuristics and some Meta heuristics like Tabu search generate a simple solution to the problem and try to improve it. Sometimes these algorithms are trapped in local minimums. On the opposite side, genetic algorithms generate a large number of solutions and make changes in each solution. The number of solutions generated is called the population. Any member in the population is called an individual or a chromosome. Each chromosome is an encoded version of a solution. The answer to the question of how to code the solution may be different for different problems and a given problem may be coded in more than one way. The purpose of coding the solutions is to translate them into strings or sequences of numbers, so the genetic operators can change and improve these sequences.

In each step, several operators make changes to the generation chromosomes and construct a new generation which has better chromosomes. This makes the chromosomes closer to the solution. Many GA operators have been proposed. The four most common are reproduction, crossover, mutation and fitness function. These four operators are used the proposed algorithm.

A. Reproduction Operator

Reproduction consists of simply copying some chromosomes from the previous generation into the next. This operator preserves very high-quality chromosomes in the population. In this algorithm the elitist solutions in the population are copied to the next generation. This guarantees a constant supply of good individuals for mating.

B. Random Key Crossover Operator

Crossover chooses two "parents" randomly and produces one or more "offspring" that contain some characteristics of genes from the parents. Crossover can be performed in a deterministic manner like "one point" crossover or "two point" crossover, where one or two cutoffs are selected at random. A new chromosome is made of the combination of two chromosomes such that genes which appear before a certain cutoff comes from parent 1 and genes after the cutoff comes from parent 2. This process can happen in a random manner, with each gene taken from a given parent with a certain probability. In most of the optimization genetic algorithms one- or two-point crossover operators are used. These operators are not applicable for some problems such as traveling sales man problem or generating random simple polygons, because they produce infeasible chromosomes.

The GA presented in this paper uses random keys to encode solutions. The use of random keys is described in [21] and is useful for problems that require permutations of the integers and for which traditional one- or two-point crossover presents feasibility problems. Consider a 5-node instance of the random polygon generation problem. The best way to encode the problem is to make a permutation of nodes which shows the orders in which nodes are connected to each other. (e.g., the solution 4 2 1 5 3 represents the order of nodes connected to each other, which is: 4 → 2 → 1 → 5 → 3→ 4.). One- or two-point crossovers may cause some sequence where some vertices are not in them. These chromosomes represent infeasible polygons. For example, the parents 1 4 3 | 2 5 and 2 4 5 | 3 1 produce the children 1 4 3 3 1 and 2 4 5 2 5, where none of them are feasible.

In the random key method, a two dimensional array is used which the first row is filled with random numbers uniformly selected from [0,1]. The second row shows the vertices of the polygon. A chromosome with 5 vertices is shown below.

Random key	0.42	0.06	0.38	0.48	0.81
Decodes as	3	1	2	4	5

To decode a polygon from a chromosome, the array should be sorted with respect to the first row. In the sorted array the elements of the second row shows the order of the vertices connecting to each other. In other words, the nodes which the difference of their random values is smaller are connected to each other. The above chromosome shows the polygon where 1 → 2 → 3 → 4 → 5→ 1 shows the order of vertices connected to each other. Now, Standard one- or two-point crossover techniques over the first row of the array will generate children that are guaranteed to be feasible.

C. Mutation Operator

The mutation operator selects some genes of a chromosome at random and changes their values, this is the same concept as biology, any small changes in one chromosome may cause in better offspring, and absolutely this event may occur conversely. In this algorithm the mutation operator picks up two genes and reassigns them with random numbers between 0 and 1.

D. Fitness Function

Fitness function acts on each chromosomes and calculate how much a chromosome is near to a simple polygon. In this paper the number of intersections between edges is the main criterion to calculate the fitness function. The number of intersections for each edge is counted, then the values for all the edges are summed up together and the produced number is returned. If the returned value is equal to zero the termination criteria is met and the algorithm finishes, else the fitness for other chromosomes is evaluated. If in a generation no chromosome meets the termination criteria the other iteration occurs. Fig. 1 shows a chromosome with fitness value equal to 5 and the chromosome shown in Fig. 2 has the fitness value of zero.

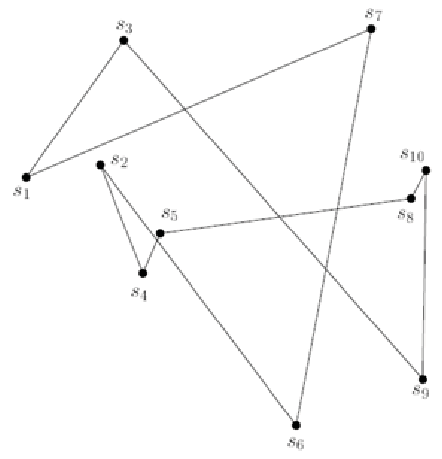


Fig. 1 The fitness functions gets 5

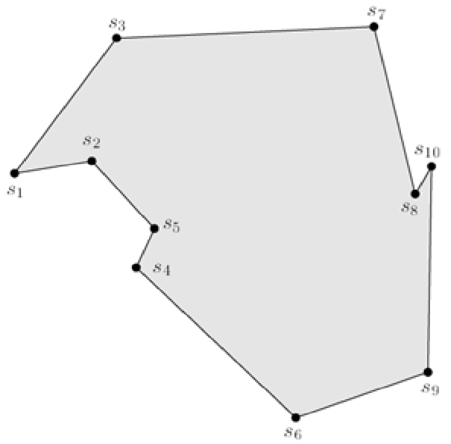


Fig. 2 The fitness functions gets zero

At each generation, 20 percent of the population comes directly from the previous population via reproduction. 50 percent is generated via crossover, and 10 percent are new chromosomes generated randomly. The remaining 20 percent is generated via mutation.

Genetic algorithm is a complete search algorithm which this is an advantage of the algorithm, but it has a drawback. It makes the search process slow. The execution of the algorithm for input points up to 20 returns the simple polygon correctly, but for more than that the algorithm is unable to find the simple polygon due to the constraints imposed on the CPU-time consumption and the available main memory. This is an important drawback which makes the algorithm useless. To overcome this drawback, the divide and conquer approach is used.

IV. Divide and Conquer Process

High CPU consumption is the most important disadvantage of genetic algorithm. Since CPU consumption increase exponentially with the increase of number of points, it's impossible to reform this algorithm to find polygons for big ns. To overcome this drawback we have used a divide and conquer approach. The main idea is to divide the problem space into small number of points, so the genetic algorithm can find the sub polygons easily and then we propose a new merge algorithm which merges all the polygons and produces a unit simple polygon.

A recursive procedure is needed to divide the points. First a random number between 3 and $n-3$ is chosen and is named Z, then a real random number between zero and 180 is generated and is named $\propto$. The number $\propto$ shows a direction and all the points are sorted in this direction. The sorted points can be easily divided into two groups by Z such that the first Z points lie in one group and the other $n-Z$ points lie in the other group. If two new groups have less than twenty points, the genetic algorithm is called for each group and a simple polygon is generated, else each group is divided into smaller groups, recursively. This preprocesses helps to overcome the CPU consumption problem.

In this section a new algorithm which merges the polygons and produces a unit polygon is proposed. It's clear that the convex hull of each group of points does not intersect, because they are divided by a line. To merge two polygons which their convex hulls do not intersect, the closest pairs of points are chosen, each from one polygon. These two points are connected to each other. Then just one of the edges in each polygon which has intersection with these points is omitted. This process changes two polygons into a polygonal chain. In Fig. 3, points b and a are the closest pairs, so they are selected and connected to each other. Then one edge from each polygon is omitted. This edge should have intersection with selected points. bc is omitted from the right polygon and ad is omitted from the left polygon. The produced polygonal chain is shown in the right side of Fig. 3.

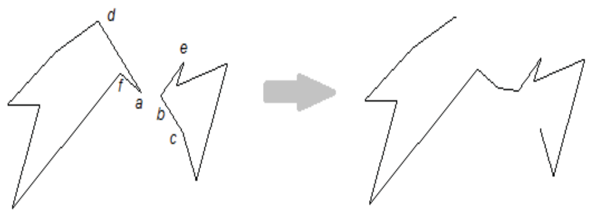


Fig. 3 Change of two polygons into a polygonal chain

Now, the algorithm which converts any kind of polygonal chain into simple polygon is presented. In a polygonal chain the degree of the first and the last nodes is one and the other nodes have the degree of two. The nodes whose degree is one are selected. We call them P_{first} and P_{last} and imagine a hypothetical segment between P_{first} and P_{last}. If the segment $P_{first}\,P_{last}$ does not intersect any edges of the polygonal chain,

P_{first} would be connected to P_{last} and the algorithm finishes. If there are some edges which have intersection with the segment $P_{first}\,P_{last}$, the intersection point for each edge and the distance between the intersection points and P_{first} and P_{last} are calculated. Then the closest edge to P_{first} and P_{last} are selected.

Next, one of the heads P_{first} or P_{last} is selected at random. The selected head is called X and the other is called Y. The polygonal chain is traversed from X until the closest segment of the polygonal chain to X which intersected with the segment $P_{first}\,P_{last}$ is reached. The vertices of this segment are called M and N where M is the vertex seen first on the traverse. Now the segment MN is omitted from the chain, and the segment XN is added instead. So, a new polygonal chain is produced and Y and M are the heads of this chain. Then, if M sees Y, these two heads are connected to each other and the algorithm finishes, or else the program calls another recursion with the newly produced chain.

In some cases, it is impossible to add a segment to the chain because the vertices of that segment do not see each other. In Fig. 4, according to the algorithm, we have to connect X to N and remove MN, but X and N are not visible. In this case, we call another recursion of the algorithm with the same chain, but the heads of chain change to X and N. In this case, the traverse begins from X. mn intersects with XN, so mn is removed, and Xn is added to the chain. Now, a new polygonal chain is produced. X and Y are the heads of the new chain (Fig. 4 (b)). Starting the traverse from X, MN is removed, XN is added to the chain, and the final chain is produced (Fig. 4 (c)).

The proposed algorithm is continued for the newly generated chain recursively until we get a chain whose heads can see each other, and thus the algorithm finishes. It could be demonstrated that the number of edges which are omitted from the chain (the number of recursions) is lower than n, so this proves that the algorithm is terminable.

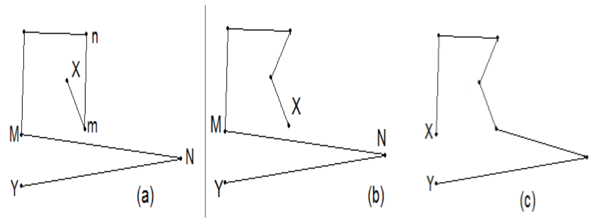


Fig. 4 In the case which adding to the chain is impossible; a new recursion is called again

Figs. 5, 6 show the steps of the algorithm over a twenty-point set. Fig. 5 (a) shows the input chain. Points a and b are the heads of the chain and a is selected as X at random. The closest intersection to a is the segment MN, so the segment MN is omitted and the segment aN is added. Fig. 5 (b) shows the newly produced chain after one step.

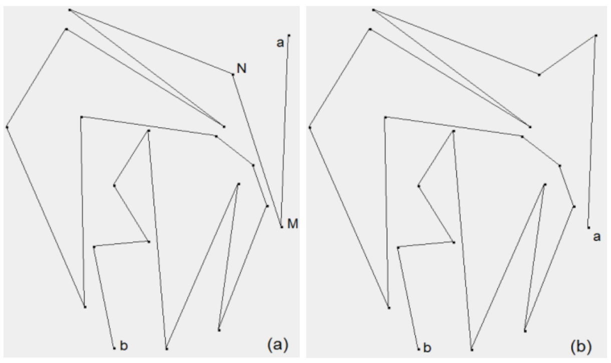


Fig. 5 (a) The input chain, (b) the chain produced after one recursion

Fig. 6 (a) shows the polygonal chain produced in the previous step and that the point b is selected as X and MN is the closest segment to b. According to the algorithm, MN is removed and bN is added to the chain. Fig. 6 (b) shows the newly generated chain.

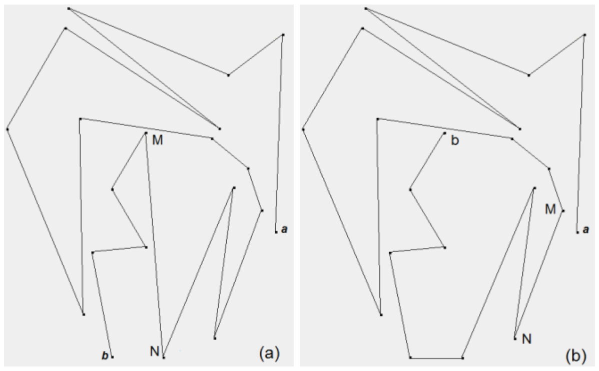


Fig. 6 (a) The chain produced in the first step (b) the new chain generated from 6 (a)

Figs. 7 (a)-(c) show the last three recursions of the algorithm. In each figure one intersection between a and b is removed, and a new chain is generated. Each newly generated chain is the input of a new recursion until a and b are visible to each other, so a and b connect to each other and the polygon is generated. Fig. 8 shows the pseudo code written for this algorithm.

The time complexity of converting a chain into a simple polygon is $O(n*l)$, where $0 < l \leq n$. Clearly, n is the number of points, and l is the number of intersections which cause the change to the chain. In other words, l is equal to the number of recursions happened in the algorithm. In the mentioned algorithm, the time complexity has a direct correlation with the input chain. If the number of edges of the chain which intersect with the segment $P_{first}\,P_{last}$ increases, the time complexity of the algorithm increases, too. In the best case, the time complexity of the algorithm is $O(n)$, where the heads of the chain (P_{first} and P_{last}) are visible to each other. So no recursion occurs. In the worst case, $n-3$ edges of the chain intersect with the segment $P_{first}\,P_{last}$, so the time complexity would be $O(n^2)$.

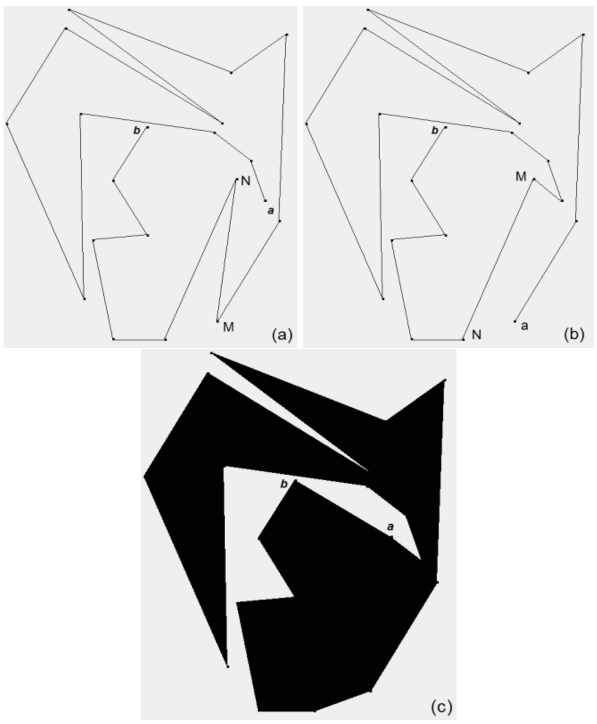


Fig. 7 Three last recursions which cause the generation of polygon

```
Algorithm: $ConvertChainToPolygon(C, count, f, l)$
Input:
    $C$ is a chain, count is the number of points, $f$ and $l$ are the heads of the chain.
Output:
    a simple polygon.
1.  $\beta \longleftarrow \phi$.
2.  Traverse the chain to find the segments that intersect with segment $fl$ and add them into $\beta$.
3.  if $\beta = \phi$ do
4.      connect $f$ to $l$ and return the generated polygon.
5.  else
6.      Choose one of the points $f$ and $l$ at random, and call it $X$, and call the other one $Y$.
7.  Traverse the chain from $X$, and find the closest
8.  segment to $X$, and call this segment $MN$.
9.  $M$ is the first point seen on the traverse.
10. If visible $X, N$ do
11. $C \longleftarrow C - MN$
12. $C \longleftarrow C + XN$
13. return $ConvertChainToPolygon(C, count, M, Y)$.
14. else
15. $C = ConvertChainToPolygon(C, count, X, N)$.
16. return $ConvertChainToPolygon\ C, count, M, Y$ .
17. endif
18. endif
```

Fig. 8 The algorithm to convert a chain into a simple polygon

V. Experimental Results

There are two main criteria to compare the performance of algorithms which generate random simple polygons. The main criterion is the number of different polygons generated by the algorithms and the other is the amount of uniformity of algorithm in generating random polygons. Each algorithm capable of generating more polygons with a high degree of uniformity is better than other algorithms.

In this paper the methods like 2-opt Moves, Steady Growth 1 and 2, Space Partitioning, Incremental Construction and Backtracking and Permute and Reject are compared. Number of different polygons generated is the criterion of the comparison. Ten different 10 points sets are used as samples and the performance of existing algorithms are compared using these samples. The whole number of possible simple polygons for each point sets are calculated by using a modified version of Incremental Construction and Backtracking algorithm. Table I shows the total number of possible simple polygons for each sample.

TABLE I
Number of Simple Polygons for 10 Points Set Problems

Polygon number	Simple polygon size
	10
1	320
2	349
3	596
4	179
5	237
6	840
7	324
8	378
9	549
10	560
$n-1$! 2	181440

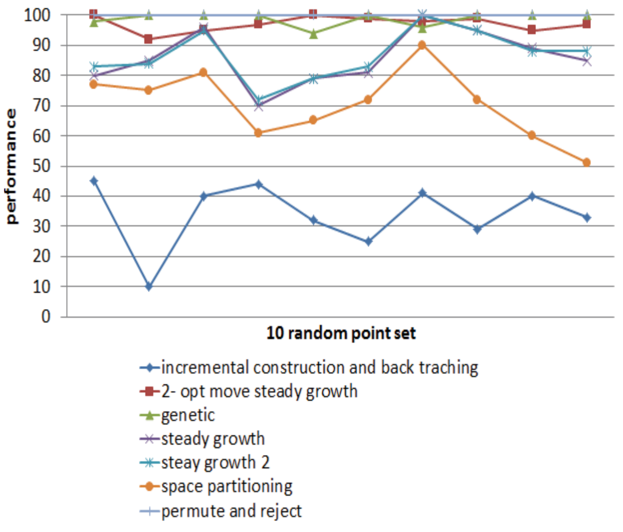


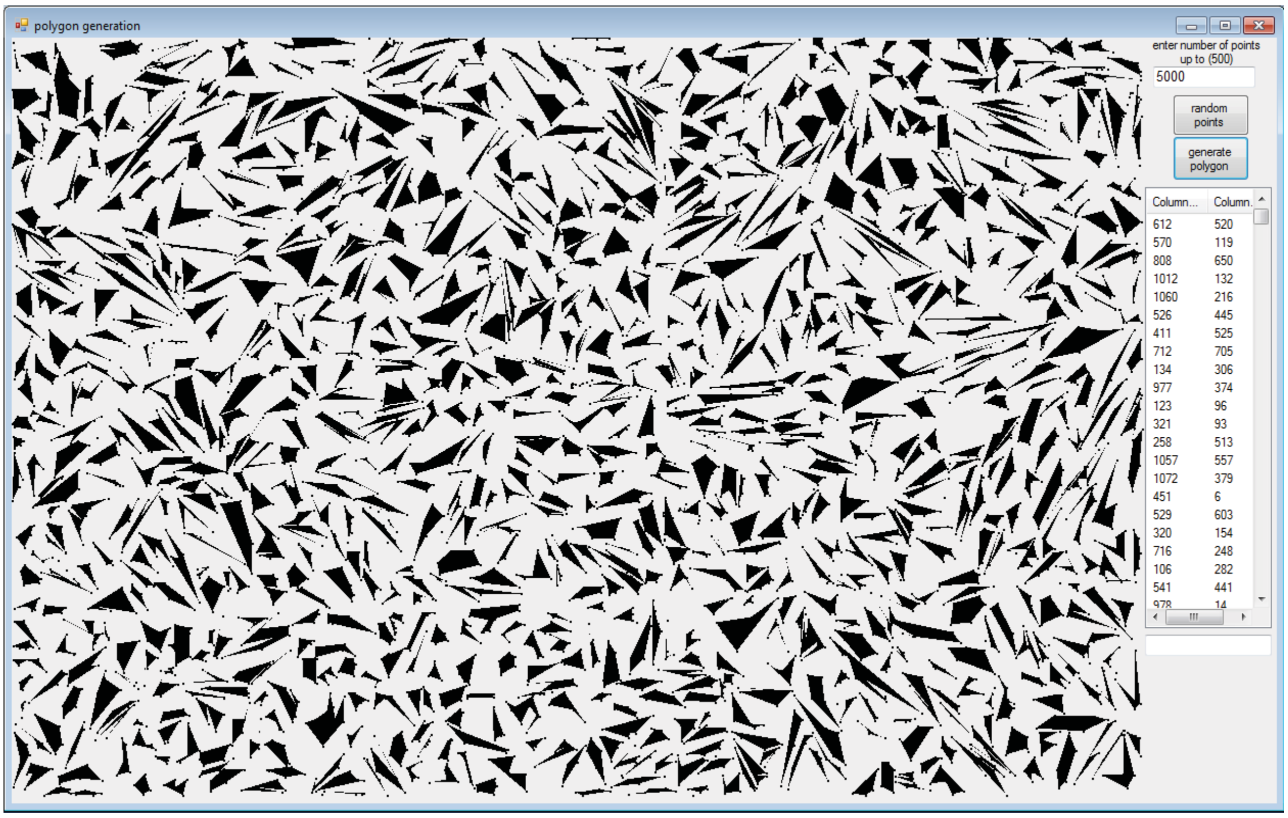
Fig. 9 The performance of 7 methods where genetic algorithm is better than steady growth and space partitioning and can be compared with 2-opt moves

The number of different polygons generated by each algorithm is calculated from the division of number of different polygons generated by the algorithm over the whole number of existing polygons (the values written in Table I).

Let t denote the number of polygons generated by the algorithms and k denote the number of simple polygons that exist on the set. If m is the number of different simple polygons generated by the algorithms, so the value of $m\ \min(t,k)$ is always between zero and one and describes the performance of the algorithms. This value is calculated for all the algorithms and all the samples and the results are depicted in Fig. 9.

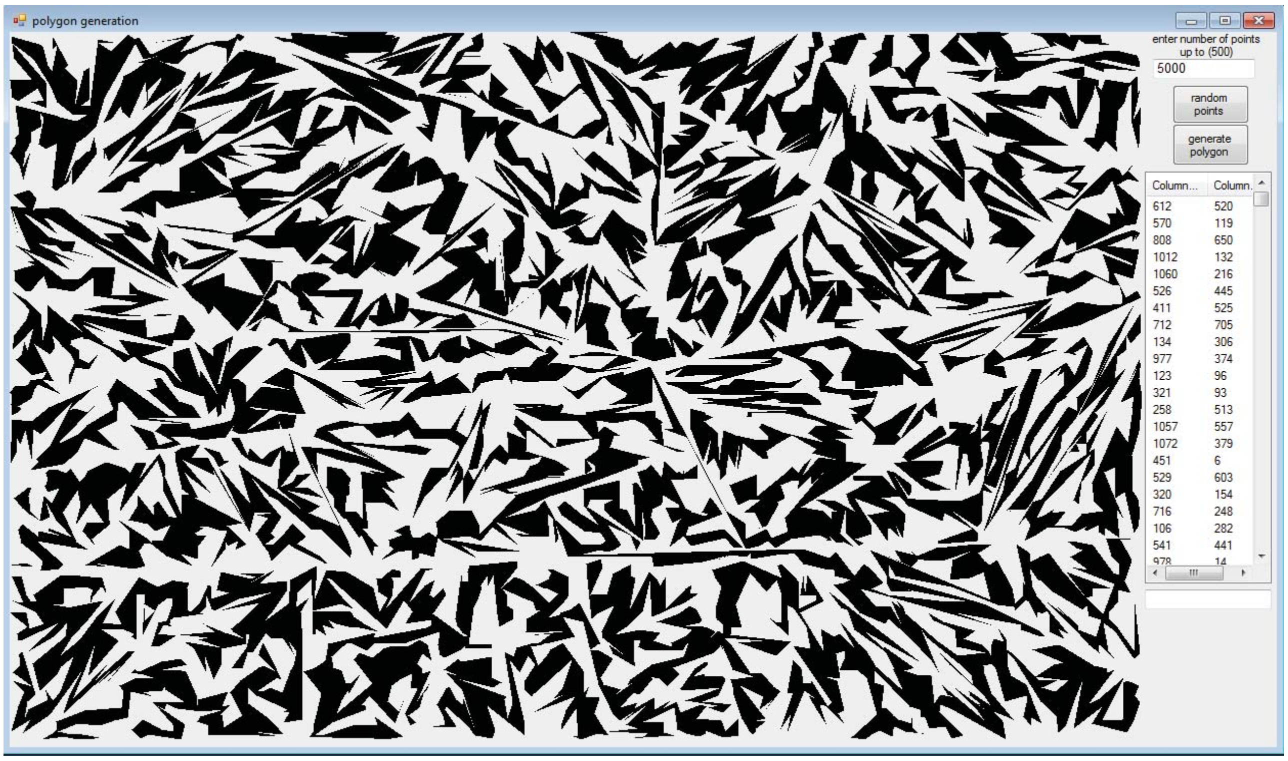
Auer in [2] compared the algorithms in Fig. 8. In this paper the genetic algorithm based on divide and conquers is compared to the existing algorithms. In each sample 10000 simple polygons is generated for each algorithm, and the performance of the algorithms is calculated according to $m\ \min(t,k)$. The proposed algorithm has the performance of 100 percent for seven samples of ten samples. The results showed that this algorithm is always better than Space Partitioning and in 9 samples is better than Steady Growth. In some samples the proposed algorithm won over the 2-opt Move algorithm and in some samples lost. Fig. 10 shows the generated random simple polygons for 5000 points, and Fig. 11 shows the unit simple polygon generated after merging the small simple polygons.

High time complexity is the nature of genetic algorithm. But in the case of random polygons parallel algorithm can be used. Since all the divided parts of the problem act independently, genetic algorithm can be run for every single sub problems simultaneously. This reduces the time needed to solve the genetic algorithm for all the sub problems to the time needed to solve a sub problem.



Column...	Column.
612	520
570	119
808	650
1012	132
1060	216
526	445
411	525
712	705
134	306
977	374
123	96
321	93
258	513
1057	557
1072	379
451	6
529	603
320	154
716	248
106	282
541	441
978	14

Fig. 10 The polygons generated for each group of points using genetic algorithm



5000

Column...	Column.
612	520
570	119
808	650
1012	132
1060	216
526	445
411	525
712	705
134	306
977	374
123	96
321	93
258	513
1057	557
1072	379
451	6
529	603
320	154
716	248
106	282
541	441
978	14

Fig. 11 The unit polygon generated after merging process

References

[1] C. Zhu, G. Sundaram, J. Snoeyink, J. S. B. Mitchel, Generating random polygons with given vertices, Computational Geometry: Theory and Application (1996) 277–290.
[2] T. Auer, M. Held, RPG: Heuristics for the generation of random polygons, in: Proc. 8th Canadian Conference Computational Geometry, 1996, pp. 38–44.
[3] P. Epstein, J. Sack, Generating triangulation at random, ACM Transaction on Modeling and Computer Simulation VOL 4 NO 3 (1994) 267–278.
[4] J. O'Rourke, M. Virmani, Generating random polygons, in: Technical Report 011,CS Dept., Smith College, Northampton, MA 01063, 1991,pp. 38–44.
[5] J. V. Leeuwen, A. A. Schoone, Untangling a travelling salesman tour in the plane, in: 7th Conference Graph-theoretic Concepts in Computer Science, 1982, pp. 87–98.
[6] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications 3rd ed, Springer-Verlag TELOS Santa Clara, CA, USA, 2008.
[7] R. L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, Information Processing Letter (1972) 132–133.
[8] F. P. Preparata, S. J. Hong, Convex hulls of finite sets of points in two and three dimensions, Commun. ACM (1977) 87–93.
[9] R. A. Jarvis, On the identification of the convex hull of a finite set of points in the plane, Information Processing Letter (1973) 18–21.
[10] A. C. Yao, A lower bound to finding convex hulls., J. ACM (1981) 780–787.
[11] S. K. Ghosh, Visibility Algorithm in the Plane, Cambridge University press, 2007.
[12] K.F. Man, K.S. Tang, S. Kwong, Genetic Algorithms: Concepts and Designs, Springer, New York, 1999.
[13] J.E. Rawlins, Gregory (Eds.), Foundations of Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1991.
[14] L.D. Whitley (Ed.), Foundations of Genetic Algorithms 2, Morgan Kaufmann, San Mateo, CA, 1993.
[15] A.M.S. Zalzala, P.J. Fleming (Eds.), Genetic Algorithms in Engineering Systems, The Institution of Electrical Engineers, London, 1997.
[16] J.T. Alander (Ed.), Proceedings of the First Nordic Workshop on Genetic Algorithms and their Applications (1NWGA), January 9–12, Vaasa Yliopiston Julkaisuja, Vaasa, 1995.
[17] W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, R.E. Smith (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, FL, July 13–17, Morgan Kaufmann, San Mateo, CA, 1999.
[18] R.K. Belew, L.B. Booker (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms, University of California, San Diego, July 13–16, Morgan Kaufmann, San Mateo, CA, 1991.
[19] J.R. Koza, D.E. Goldberg, D.B. Fogel, R.L. Riolo (Eds.), Genetic Programming: Proceedings of the First Annual Conference, Stanford University, July 28–31, MIT Press, Cambridge, 1996.
[20] G. Winter, J. Pe´riaux, M. Gala´n, P. Cuesta (Eds.), Genetic Algorithms in Engineering and Computer Science, Wiley, New York, 1995.
[21] J.C. Bean, Genetic algorithms and random keys for sequencing and optimization ORSA, Journal on Computing 6 (1994) 154–160.