Proceedings of the 7th Oklahoma Symposium on Artificial Intelligence, November 18-19, 1993, Stillwater, OK, pp. 122-131.

Near-Optimal Triangulation of a Point Set using Genetic Algorithms*

Yu Wu
Roger L. Wainwright

Department of Mathematical and Computer Sciences
The University of Tulsa
600 South College Avenue
Tulsa, Oklahoma 74104-3189
wuyu@euler.mcs.utulsa.edu
rogerw@penguin.mcs.utulsa.edu

ABSTRACT

This paper explores the triangulation of a point set using genetic algorithms (GA). We implemented several GAs including a parallel genetic algorithm for solving the triangulation problem. We developed a crossover operator specifically for the triangulation problem. The various data structures needed to implement the crossover are presented. Since our crossover operator is also the mutation operator, we investigated the effect of our genetic algorithms with and without mutation. We compared our genetic algorithms against the best known heuristic algorithm and a simulated annealing implementation for the triangulation problem. We tested all of the algorithms using randomly generated datasets of various sizes and contrived datasets. Our results showed the parallel GA obtained the best overall performance, while the simulated annealing algorithm was the worst algorithm.

1. INTRODUCTION

Given a set *S* of *n* distinct points in the Euclidian plane, join them by nonintersecting straight line segments so that every region internal to the convex hull is a triangle. This results in a triangulation of the set *S*. A triangulation of *n* points is not unique. However, being a planar graph, a triangulation of *n* points has at most 3*n*-6 edges. Define *T(S)* as a triangulation of *S*. Also define the weight (cost) of a triangulation, *W(T(S))*, as the sum of the Euclidian length of all of the edges in *T(S)*. The optimal triangulation problem finds a triangulation *T(S)* such that *W(T(S))* is minimum among all possible triangulations of *S*. The optimal triangulation problem has applications in finite element analysis. It also has applications in numerical interpolation of bivariate data when the function values are available at *n* irregular-spaced data points (x_i,y_i) in a plane and an approximation to the function at a new point (x,y) is desired [19].

The two best known algorithms for the triangulation problem are the *greedy algorithm*, and the *Delaunay* triangulation algorithm. The Delaunay triangulation algorithm for a set *S* is the straight-line dual of the *Voronoi* diagram of the set *S*. The greedy algorithm executes in $O(n^2\log n)$ time, and the Delaunay triangulation algorithm executes in $O(n\log n)$ time. Pseudo code for the greedy algorithm is described by Lloyd [14]. It has been conjectured that both the greedy algorithm and the Delaunay triangulation algorithm always produced the optimal (minimum weight) triangulation. This was disproved by Lloyd [14] in

* Research supported by OCAST Grants AR2-004 and ARO-038 and Sun Microsystems Inc.

1977. The triangulation problem is classified as an *NP*-hard problem, where the existence of a polynomial-time algorithm remains open. However, given a set of points that form a convex hull, the optimal triangulation can be determined in polynomial time. For example, the triangulation algorithm for points on a convex hull can be easily implemented using dynamic programming in $O(n^3)$ time using $O(n^2)$ space [22]. Corman [5] gives an excellent description of this problem for the interested reader.

Given the significance of the triangulation in a variety of numerical, scientific and engineering applications, developing optimal or near-optimal triangulation algorithms is extremely important. In this paper, we present a genetic algorithm (GA) implementation for the triangulation problem. We compare our results with the greedy algorithm and the simulated annealing (SA) technique for the triangulation problem. Randomly generated datasets and contrived datasets are used to make comparisons. The rest of the paper is presented as follows. In Section 2, the fundamentals of simulated annealing are reviewed. In Section 3, the fundamentals of genetic algorithms are presented along with our GA implementation for the triangulation problem. We developed a sequential GA and also a parallel GA implementation for the triangulation problem. Results and conclusions are presented in Section 4.

2. SIMULATED ANNEALING TRIANGULATION

Simulated annealing is a stochastic computational technique for finding near optimal solutions to large optimization problems. The method of simulated annealing is an analogy with thermodynamics, specifically in the manner that metals cool (anneal), or in the way liquids freeze and crystallize. If liquid metal cools too rapidly, atoms do not have time to line themselves up to form a pure crystal. Instead a rather high energy state is reached. However, if the metal is cooled slowly atoms have time to line themselves up to form a minimum energy system [17]. Metropolis [16] in 1953 was the first to incorporate these concepts into numerical calculations. Kirkpatrick [11] in 1983 was the first to use this concept to solve combinatorial optimization problems. It is assumed the reader is familiar with the fundamentals of simulated annealing.

The simulated annealing triangulation algorithm that we used was developed by Sen and Zheng [21]. Their algorithm is summarized below. In the triangulation algorithm, simulated annealing starts with an arbitrary initial triangulation. The objective function, which is the sum of the Euclidian length of all of the edges in the triangulation, is analogous to the current energy state of the system. A convergence condition of simulated annealing is that any state can be reached from any other state. A simple perturbation function is used to move from one state to another.

In a triangulation of a set, *S*, an edge that is not on the convex hull is called an *internal edge*. Each internal edge in a triangulation is shared by two triangular faces and the combination of the two faces forms a quadrilateral. The perturbation function randomly selects one of the internal edges in the current triangulation, identifies the associated quadrilateral and replaces the internal edge by the alternate diagonal edge. This is illustrated in Figure 1(a). The edge connecting *A* and *C* can be replaced by the edge connecting *B* and *D*. Care must be taken because it is not possible to exchange the edges when the quadrilateral is not convex. See Figure 1(b). In this case another internal edge is randomly selected until a convex quadrilateral is found.

Changes from one state to another (ie., a new triangulation from a previous triangulation) that result in a reduced objective function are always accepted. This is a analogous to slow cooling. However, changes that increase the objective function are only accepted with probability p(d,T) = exp(-d/T), where d is the change in the objective function from one triangulation to another, and T is the "temperature" of the system. The temperature parameter controls the annealing process. By occasionally allowing the energy of the system to rise slightly before cooling again is analogous to occasionally allowing for a worse triangulation. This may allow the system to avoid falling into a local minima where it cannot get out.

The annealing schedule, controlled by the parameter T, requires some experimentation. Sen and Zheng used an initial temperature of 50 times the average length of all of the internal edges of the initial triangulation. The number of iterations at each temperature was 0.9 times the number of internal edges. The temperature decrement is defined as $T_{next} = 0.9 * T_{current}$. Typically the system freezes at a certain temperature. In this case the annealing stops when the cost of the triangulation resulting from three consecutive temperature changes is within a small tolerance. The interested reader is referred to Sen and Zheng [21] for more details on this particular algorithm. Nahar *et al.* [18] give an excellent overview of simulated annealing.

3. GENETIC ALGORITHM TRIANGULATION

Genetic Algorithms are applicable to a wide variety of problems. In particular, genetic algorithms have been very successful in obtaining near-optimal solutions to many different combinatorial optimization problems [1-3,6-8,17,24]. Genetic algorithms use probabilistic rules to evolve a population from one generation to the next. The transition rules going from one generation to the next are called genetic recombination operators. These include Reproduction (of the more "fit" chromosomes), Crossover, where portions of two chromosomes are exchanged in some manner, and Mutation. Crossover combines the "fittest" chromosomes and passes superior genes to the next generation thus providing new points in the solution space.

In a generational GA the offspring are saved in a separate pool until the pool size is reached. Then the children's pool replaces the parent's pool for the next generation. In a steady-state GA the offspring and parents occupy the same pool. Each time an offspring is generated it is placed into the pool, and the weakest chromosome is "dropped off" the pool. These two cases represent two extremes in pool management for genetic algorithms. We tested a steady-state GA and a generational GA for comparison on each of the datasets. It is assumed the reader is familiar with the fundamentals of genetic algorithms.

Genetic algorithm packages for a single processor have been available for several years. A steady-state GA such as GENITOR [23] and a generational GA such as GENESIS [10] are the two example packages that have been available for several years. The research reported here made use of LibGA [4], a GA package developed in house. LibGA offers the best of GENESIS and GENITOR including the ability to use a steady-state or a generational approach or a combination of both. Davis, Goldberg and Rawling provide an excellent in depth study of genetic algorithms [6,7,9,20].

We considered several chromosome representations for a triangulation of a set, S. Unfortunately, the triangulation problem is not an order-based problem, where a chromosome is a permutation of the set of points. Hence the traditional crossover techniques for order-based problems such as Cycle, PMX, Edge Recombination, Order1, Order2, etc. do not work well for this problem. It is possible to enumerate all possible edges among a set of n points and represent a triangulation as a fixed length bit string, where a "1" bit indicates the presence of a particular edge, and a "0" bit represents the absence of the edge. However, for large problems this results in long relatively sparse bit strings. Furthermore, traditional bit string crossover operators produce infeasible triangulations in nearly every instance.

Consider the set of six points in Figure 2, with an example triangulation. We elected to represent a chromosome as a list of edges in the triangulation. The chromosome representation for the triangulation in Figure 2 is given below.

((1,2), (1,3), (1,6), (2,3), (2,4), (3,4), (3,6), (4,5), (4,6), (5,6)).

The edges are shown in sorted order. In practice, the order is irrelevant, and the edges are not placed in sorted order unless the chromosome is actually selected for reproduction. In our representation a chromosome length is the number of edges, which is fixed for a given set of points, as long as no three points are colinear. We represent a chromosome in an edge array data structure shown in Figure 3(a). Once a chromosome is selected for reproduction, the edge array is sorted in order to easily generate the associated triangle array, as shown in Figure 3(b). We used the edge-exchange crossover operator; the same technique used in the simulated annealing algorithm. An edge is randomly selected from the chromosome, for example edge (4,6) in Figure 3(a). Next the triangle array is search for triangles with this common edge, for example (3,4,6) and (4,5,6) in Figure 3(b). This yields two triangles with a common edge. If the resulting quadrilateral is not convex, then another random edge is selected and the process is repeated. If the quadrilateral is convex, then the selected edge is replaced with the alternate edge. Our crossover operator is asexual; that is one parent produces one child. However, the LibGA package requires two parents to generate two children. To remain compatible with the other operations in LibGA, we used two independent crossover operations on two parents each time to produce two children. The authors are not aware of any previous research using genetic algorithms to solve the triangulation problem.

Another possible asexual crossover operator uses the triangle array to detect a 5-point convex polygon. Once located there are five possible triangulations for the polygon, and the crossover operator selects the best one. We did not implement this operator. This concept could be extended to large size polygons, however, the computation is very expensive. We used a mutation rate of 0.05. The mutation operator is the same as the crossover operator.

We used a population size between 35 and 50 for the generational GA, and a population size of 75 for the steady-state GA. We found the smaller sized populations (35 to 50) generally did better than larger sizes for the generational GA. In general, the steady-state GA requires a larger population pool than the generational GA to avoid premature convergence.

Lee and Schacter [13] proved that given a set S, of n points, any triangulation, $T(S)$, has the same number of triangles, $N_t = 2(n-1) - N_h$, and the same number of edges, $N_e = 3(n-1) - N_h$, where N_h is the number of the points on the convex hull of S. We used the above observations to help generate the initial population. For a given set of points, we determined the number of points on the convex hull, and hence the number of required edges for a valid triangulation. We randomly placed edges in the set of points such that no two edges crossed until the proper number of edges were placed. This did not always produce a valid triangulation due to the random placement of the edges. If an invalid triangulation resulted, it was discarded and another attempt was made until the proper number of valid triangulations were generated to fill the pool. This turns out to be a fairly effective procedure for generating an initial population of valid triangulations. For example, for $n = 40$ using the generational GA, 48 attempts were required to generate a pool size of 36 valid triangulations. The steady-state GA only required 98 attempts to generate 75 valid triangulations. However, for $n = 190$ using the generational GA, 114 attempts were required to generate a pool size of 36 valid triangulations. The steady-state GA required 254 attempts to generate 75 valid triangulations.

Before running the test cases we anticipated the GA approach would produce superior results than the SA approach for several reasons. Simulated Annealing uses one feasible solution and a perturbation function. The GA has a population of feasible solutions to choose from, while performing the same perturbation function. While the SA approach accepts superior perturbations and occasionally a worse one, the GA uses a selection bias to manipulate superior chromosomes. Finally, the mutation operator in the GA allows (in theory) for all parts of the state space to be searched. We also anticipated the GA would require longer execution times to complete compared to the SA algorithm. Results shown in the next section verified our initial assumptions.

4. RESULTS AND CONCLUSIONS

We compared the greedy algorithm, simulated annealing algorithm and various genetic algorithms for solving the triangulation problem using two types of datasets. A Type I dataset consists of a collection of n randomly generated points in a plane. The (x,y) coordinate of each point was generated randomly in the range of 150 to 550. A Type II dataset consists of a set of points that reside entirely on an arc of a circle, such that the arc subtends less than 180 degrees. An example is shown in Figure 4. This type of dataset was suggested by Manacher and Zobrist [15], and is known to be difficult for the greedy algorithm to yield a good solution.

Since our crossover operator is the same as the mutation operator, one might initially conclude that mutation may be of little use in our GA implementations of the triangulation problem. To study the effect of mutation, we tested the generational GA and the steady-state GA with and without mutation. We also developed a parallel triangulation genetic algorithm by modifying HYPERGEN for our specific problem. HYPERGEN [12] is a parallel genetic algorithm package implemented specifically for the hypercube architecture. HYPERGEN was developed as a research tool for investigating parallel genetic algorithms applied to combinatorial optimization problems. HYPERGEN distributes the initial population evenly among the processors. Each processor (island) executes a sequential GA on its subpopulation performing crossover and mutation. HYPERGEN is a modular collection of routines for generating the initial population, evaluation function, selection (based on a bias function), reproduction, mutation, migration interval, migration rate and summary statistics. The sequential GA on each processor is performed automatically for the user. The periodic migration of genetic material between processors is also performed automatically for the user. One of the design requirements for HYPERGEN is that the user can use the package without concerning himself with the hypercube topology, message passing, or other details of parallel processing. In all cases our parallel GA was implemented using four processors.

Table I depicts the results of seven triangulation algorithms for Type I datasets: greedy, generational GA without mutation, generational GA with mutation, steady-state GA without mutation, steady-state GA with mutation, simulated annealing, and a parallel GA. We tested Type I datasets for n = 10, 20, 30, 40, ..., 200. Results from Table I indicate the generational and steady-state genetic algorithms consistently performed better on larger problems when implemented with mutation. Specifically, Table I shows the generational GA with mutation obtained better results than the generational GA without mutation for $n >= 150$. Similarly the steady-state GA with mutation obtained better results than the steady-state GA without mutation for $n >= 90$. This signifies that mutation (even though it is the same as the crossover operator) is significant, especially as the state space size increases.

In 16 of the 20 test cases shown in Table I, the generational GA obtain the same or better results than the steady-state GA (considering with and without mutation). Furthermore, In every instance (except one case) the generational GA obtain the same or better results than the simulated annealing algorithm. In exactly half of the 20 data sets tested, the generational GA obtain the same or better results than the greedy algorithm. In general, the genetic algorithm triangulation outperformed the greedy algorithm on datasets of size 130 or less. The greedy algorithm consistently did better than the generational genetic algorithm for datasets size 140 to 200. However, the best overall algorithm was the parallel genetic algorithm, obtaining the best result of any algorithm in approximately 80% of the cases. The parallel GA, in general, performed very well for larger datasets (dataset size 110 and larger). Considering overall performance, the ranking of the algorithms from our results in Table I for Type I datasets are: parallel GA (best), greedy (second), and generational GA with mutation (third). The SA algorithm was the worst algorithm. Sen [21] concluded in his research that the greedy algorithm for Type I datasets performed about the same as his simulated annealing algorithm. We did not obtain the same results.

We attribute the superior performance of the parallel GA over the sequential GAs to the separately maintained population pools allowing for the independent breeding of chromosomes in separate islands. The parallel GA allows for an excellent mix of the chromosomes insuring a more rigorous search through the state space. We did not spend much time fine tuning the GA parameters. We believe with the proper fine tuning of the GA parameters that the sequential and parallel GAs could easily obtain even better results than those reported in Table I. To illustrate an example performance of the GA for triangulation, Figure 5 depicts a random triangulation from the initial population pool for the random set of 150 points, and Figure 6 shows the final result obtained by the parallel GA.

Table II depicts the results of the greedy, generational GA, and simulated annealing triangulation algorithms for Type II datasets. The steady-state GA was not included since its performance was consistently inferior to the generational GA. We generated Type II datasets of size n, for $n = 10, 20, 30, 40, 50, 100$, and 200. Results show in five of the seven datasets tested the generational GA outperformed the simulated annealing algorithm. Furthermore, the generational GA in every instance obtained better results than the greedy algorithm, except for $n = 10$ where the same result was obtained. These results serve to illustrate the genetic algorithm is again superior to the simulated annealing algorithm, and whether random or specially contrived data is used, the genetic algorithm should be the algorithm of choice for the triangulation problem.

ACKNOWLEDGEMENTS

This research has been supported by OCAST Grants AR2-004 and ARO-038. The authors wish to acknowledge the support of Sun Microsystems Inc. The authors also wish to thank Subhajit Sen and Si-Qing Zheng [21] for supplying the authors with their simulated annealing software.

REFERENCES

[1] J.L. Blanton and R.L. Wainwright, "Vehicle Routing with Time Windows using Genetic Algorithms", *Proceedings of the Sixth Oklahoma Symposium on Artificial Intelligence*, November, 1992. pp242-251.

[2] D.E. Brown, C.L. Huntley and A.R. Spillane, "A Parallel Genetic Heuristic for the Quadratic Assignment Problem", *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989.

[3] A.L. Corcoran and R.L. Wainwright, "A Genetic Algorithm for Packing in Three Dimensions", *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, 1992, pp. 1021-1030, ACM Press.

[4] A.L. Corcoran and R.L. Wainwright, "LibGA: A User-friendly Workbench for Order-based Genetic Algorithm Research", *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, pp. 111-118, 1993, ACM Press.

[5] T.H. Corman, C.E. Leiserson, and R.L. Rivet, *Introduction to Algorithms*, The MIT Press, Cambridge, Mass., 1990, pp. 301-336.

[6] L. Davis, ed., *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publisher, 1987.

[7] L. Davis, ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[8] K.A. De Jong and W.M. Spears, "Using Genetic Algorithms to Solve NP- Complete Problems", *Proceedings of the Third International Conference on Genetic Algorithms*, June, 1989, pp. 124-132.

[9] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[10] J. Grefenstette, GENESIS, Navy Center for Applied Research in Artificial Intelligence, Navy research Lab., Wash. D.C. 20375-5000.

[11] S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi, "Optimization by Simulated Annealing", *Science*, May, 1983, vol. 220, pp 671-680.

[12] L.R. Knight and R.L. Wainwright, "HYPERGEN: A Distributed Genetic Algorithm on a Hypercube", *Proceedings of the 1992 Scalable High Performance Computing Conference, SHPCC'92*, Williamsburg, Va., April 26-29, 1992.

[13] D.T. Lee and B.J. Schacter, "Two Algorithms for Constructing a Delaunay Triangulation", *International Journal of Computer and Information Sciences*, vol. 9, no. 3, 1980, pp. 219-243.

[14] E.L. Lloyd, "On Triangulations of Points in the Plane", *Proceedings of the 18th Annual IEEE Conference on the Foundations of Computer Science*, pp. 228-240, 1977.

[15] G.K. Manacher and A.L. Zorbist, "Neither the Greedy nor the Delaunay Triangulation of a Planar Point Set Approximates the Optimal Triangulation", *Information Processing Letters*, vol. 9 no. 1, July, 1979, pp. 31-34.

[16] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, "Equation of State Calculation by Fast Computing Machines", *J. Chem. Phys.* vol. 21, 1953, p. 1087.

[17] P.P. Mutalik, L.R. Knight, J.L. Blanton and R.L. Wainwright, "Solving Combinatorial Optimization Problems Using Parallel Simulated Annealing and Parallel Genetic Algorithms", *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, pp. 1031-1038, 1992, ACM Press.

[18] S. Nahar, S.S. Sahni, and E. Shragowitz, "Simulated Annealing and Computational Optimization", *International Journal of Computer Aided VLSI Design*, vol. 1, pp. 1-23, 1989.

[19] F.P. Prepara and M.I. Shamos, *Computational Geometry - An Introduction*, Springer Verlag, 1985.

[20] G. Rawling, ed., *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, 1991.

[21] S. Sen and S. Zheng, "Near-Optimal Triangulation of a Point Set by Simulated Annealing", *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, pp. 1000-1008, 1992, ACM Press.

[22] S.A. Strate and R.L. Wainwright, "Load Balancing Techniques for Dynamic Programming Algorithms on Hypercube Multiprocessors", *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, pp. 562-569, 1993, ACM Press.

[23] D. Whitley and J. Kauth, GENITOR: A Different Genetic Algorithm, *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, Denver, Co., 1988, pp. 118-130.

[24] D. Whitley, T. Starkweather, and D. Fuquat, "Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator", *Proceedings of the Third International Conference on Genetic Algorithms*, June, 1989.

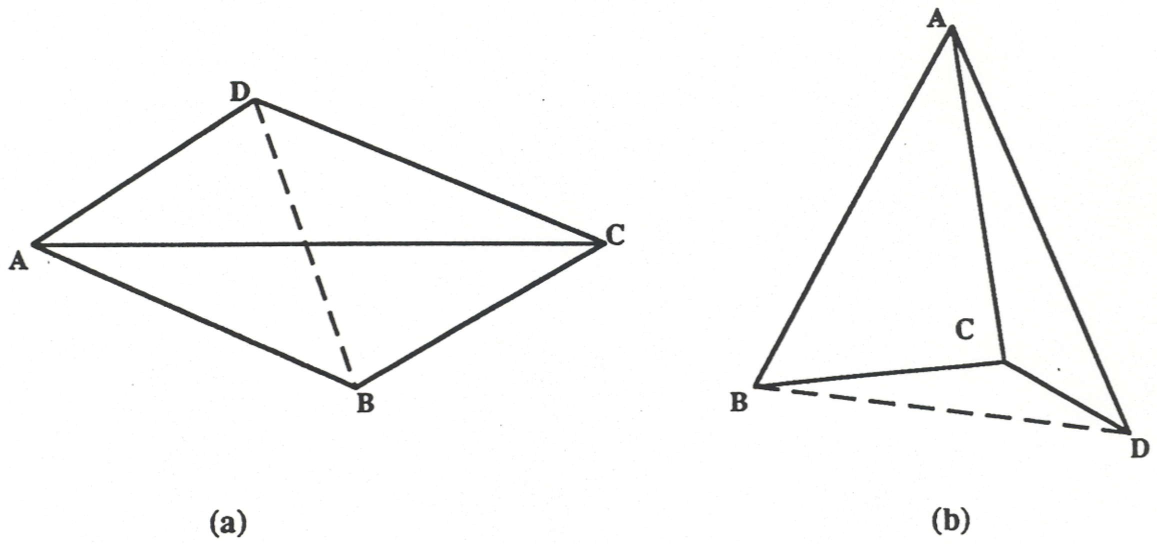

Figure 1: Definition of a move.
(a) Replacing edge AC by BD.
(b) Edge AC is not replaceable by BD.

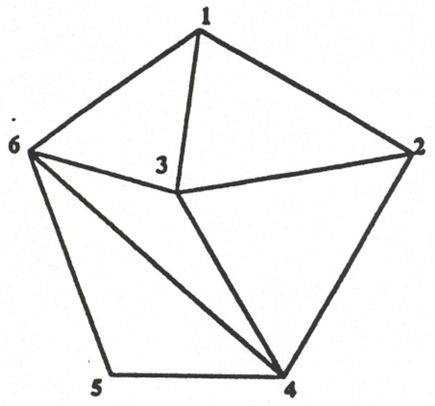

Figure 2: An example of triangulation with 6 points.

Edge index	1	2	3	4	5	6	7	8	9	10
endpt1	1	1	1	2	2	3	3	4	4	5
endpt2	2	3	6	3	4	4	6	5	6	6

Figure 3 (a): The edge array data structure for Figure 2.

Triangle index	1	2	3	4	5
endpt1	1	1	2	3	4
endpt2	2	3	3	4	5
endpt3	3	6	4	6	6

Figure 3 (b): The associated triangle array data structure for Figure 2.

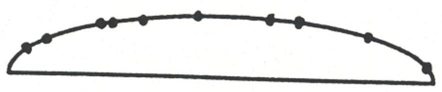

Figure 4: A set of points that reside entirely on an arc of a circle, such that the arc subtends less than 180 degrees.

Number of Points	Algorithm						
	Greedy	Gen. GA without/Mut.	Gen. GA with Mut.	S-S GA without/Mut.	S-S GA with Mut.	SA	Parallel GA
10	2705.2*	2705.2*	2705.2*	2705.2*	2705.2*	2705.2*	2705.2*
20	4795.9*	4795.9*	4795.9*	4795.9*	4795.9*	5016.6	4795.9*
30	7177.6	7176.9*	7176.9*	7176.9*	7176.9*	7513.5	7176.9*
40	8668.1	8659.9	8668.1	8659.9	8680.4	8657.2*	8660.9
50	9737.0	9737.0	9728.9*	9728.9*	9736.8	9815.5	9728.9*
60	10938.2	10938.2	10930.0*	10969.2	11018.0	11267.0	10930.0*
70	11729.3*	11762.1	11738.0	11773.1	12125.9	12194.9	11730.4
80	12440.6	12427.6	12496.9	12497.0	12507.5	12902.8	12426.8*
90	13161.9*	13180.6	13258.3	13246.1	13230.7	13645.0	13165.9
100	13960.1	13910.5*	13910.5*	14011.3	13943.7	14844.0	13946.8
110	14535.1	14573.3	14604.1	14747.3	14621.2	15004.8	14518.7*
120	15130.3	15124.9	15135.5	15382.9	15200.5	15770.0	15116.4*
130	15635.6	15624.3	15643.2	15788.2	15667.5	16503.3	15562.8*
140	15947.8	15964.7	15992.5	16069.6	15964.4	16548.2	15933.6*
150	17003.2	17021.1	17020.6	17512.4	17016.9	17962.7	16981.8*
160	17692.8	17733.5	17707.7	17925.6	17736.8	18826.0	17687.1*
170	18157.6	18200.9	18191.3	18605.5	18180.6	20460.8	18154.4*
180	18722.5	18801.0	18795.2	19048.4	18719.9*	20798.8	18728.9
190	19281.6	19514.4	19338.2	19802.4	19493.9	20659.2	19276.5*
200	19711.2*	20223.0	20131.8	20591.7	20234.4	21882.9	20031.7

Table I: Triangulation Results for Random Point Sets (Type I Datasets) Using Various Algorithms. (* Indicates the Best Result)

Number of Points	Algorithm		
	Greedy	Generational GA	SA
10	2265.8	2265.8	2300.9
20	2857.6	2844.9	2869.0
30	3279.3	3247.7	3212.3
40	3544.1	3524.9	3527.3
50	3814.1	3763.2	3752.2
100	4393.5	4374.2	4380.9
200	5005.4	4989.6	5060.3

Table II: Triangulation Results for Contrived Point Sets (Type II Datasets) Using Various Algorithms

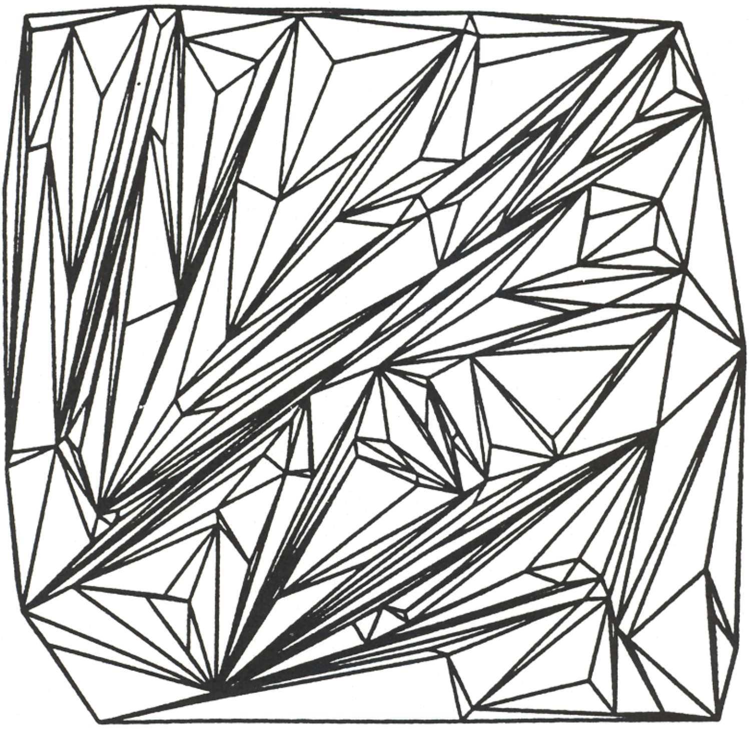

Figure 5: A Random Triangulation from the Initial Pool
Random Set of 150 Points

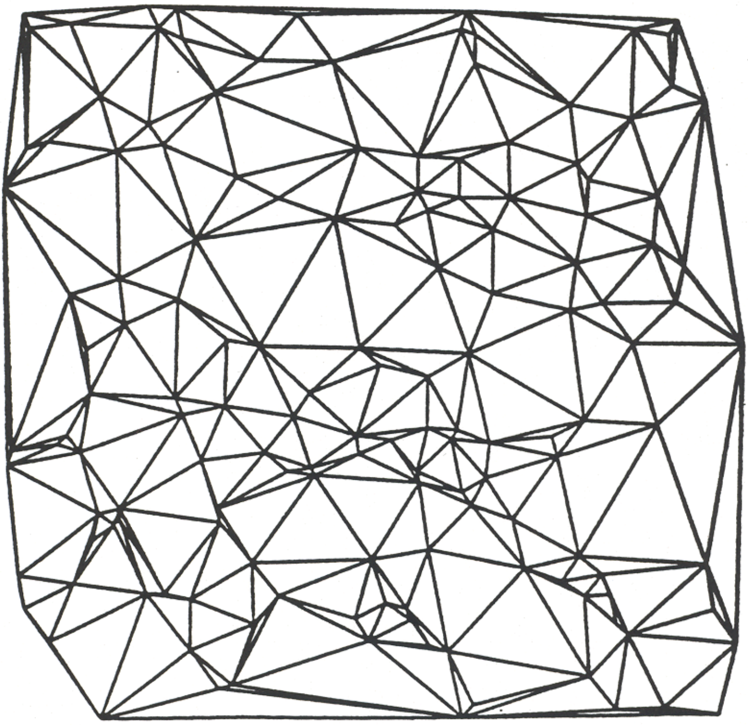

Figure 6: The Best Triangulation from Parallel GA
Random Set of 150 Points