The "Divide and Conquer" technique to solve the Minimum Area Polygonalization problem

Maksym Osiponok
Faculty of Computer Sciences and Cybernetics
Taras Shevchenko national university of Kyiv
Kyiv, Ukraine
osipyonok@ukr.net

Vasyl Tereshchenko
Faculty of Computer Sciences and Cybernetics
Taras Shevchenko national university of Kyiv
Kyiv, Ukraine
vtereshch@gmail.com

***Abstract* — We consider an application of the divide and conquer technique to the Minimum Area Polygonalization (MAP) problem. It is known that MAP problem belongs to NP-Hard set of problems. In this paper we propose a heuristic algorithm for solving the Minimum Area Polygonalization problem that is based on recursive subdivision of the set of points into two smaller subsets, constructing approximated solutions for the subsets and merging them with respect to minimizing the total area using the minimum area quadrilateral between two polygons. The algorithm of finding such a quadrilateral is described at this paper as well. Time complexity of the entire algorithm is $O(n^2)$ using $O(n)$ memory.**

Keywords*— *simple polygonalization, heuristic algorithm, minimum area polygon, divide and conquer strategy

I. Introduction

An object shape approximation by more simpler objects is one of the important problems in computational geometry. In most cases, an object shape approximation is sought, with respect to some geometric measure, for example, with the smallest area, smallest perimeter, etc.

In the paper we considered the problem of finding the minimal area simple polygon (MAP), for a given set of points, all vertices of which belong to a given set. In practice, this problem arise in the context of geo-sensor networks [1, 2], and in GIS systems [3, 4]. In [5], an NP-hardness of a given problem is proved, that is, in order to find an exact solutin, we must overcome all exponentially large number of possible simple polygons that can be constructed on a given set of points [6]. Therefore, it is expedient to consider heuristic approaches to the solution of the MAP problem.

Analysis of recent research. To date, a heuristics in combination with simple polygon generation algorithms, are used to solve the MAP problem [7]. For example, for the SteadyGrowth algorithm, the detailed description of which can be found in [7], the following heuristic criteria are used:

1. Building an initial triangle:
   (a) Random triangle
   (b) Greedy triangle
2. Feasible point selection:
   (a) Random feasible point
   (b) Greedy feasible point
3. Edge selection:
   (a) Greedy edge

In [8, 9], this approach to the construction of an approximated solution is also described in detail. A modified version of this approach is proposed in [9], which has the $O(n^3)$ time complexity at the expanse of memory $O(n^2)$. Also, there is a randomized algorithm for solving the MAP problem with time complexity $O(n^2 \log n)$ using $O(n)$ memory [11]. In [12, 13] authors formulate MAP problem as one of generalized problems of nonlinear programming models, and present a genetic algorithm to solve it. The only algorithm that gives an exact solution is "Permute and Reject" [7,9]. It generates all possible permutations of the set of points, for each permutation this algorithm checks whether it determines a simple polygon, and among of the generated simple polygons, the one that has the smallest area is chosen.

The novelty and idea. In the paper we propose an approach that allows to find an approximate solution for the MAP problem in polynomial time.

Paper's aim. Develop a greedy algorithm for approximating the solution of the MAP problem using the "Divide and conquer" technique.

II. Problem and algorithms

Problem MAP. A set S of N points is given on a plane. It is necessary to construct a simple polygon with the smallest possible area, which would cover a given set of points, and which vertices are all points of the set S.

A. The algorithm idea

If number of elements in the set S is small enough (e.g. 5), we can find an exact solution using any brute-force algorithm, like "Permute and Reject" [7, 9]. Otherwise we divide the set S into two subsets S_1 and S_2, so that the number of elements in the subsets differs by no more than one. Find the solutions of the MAP problem for these two subsets recursively. Now we have approximations for sets S_1 and S_2, and we have to merge them into one simple polygon, using a quadrilateral of the minimum area, so that the resulting polygon will not contain self-intersections.

Now we will describe necessary procedures used by the proposed algorithm for solving the MAP problem.

B. An algorithm for finding a minimal area quadrilateral between two non-intersecting simple polygons

Without loss of generality, we assume that all vertices of the first polygon are to the left of the vertices of the second one. Also, for definiteness, we assume that the vertices of polygons are given in the counterclockwise order. For those polygons construct the upper and lower tangent. Consider the subchain of the first polygon, which is from the point of intersection of the lower tangent to the upper, and the subchain of the second polygon, which is from the point of the intersection of the upper tangent to the lower one (Fig.1). Consider all possible quadrilaterals without self-intersections, which contain one edge from each of subchains and do not cross the subchains in other points, and choose among of them the one with the smallest area.

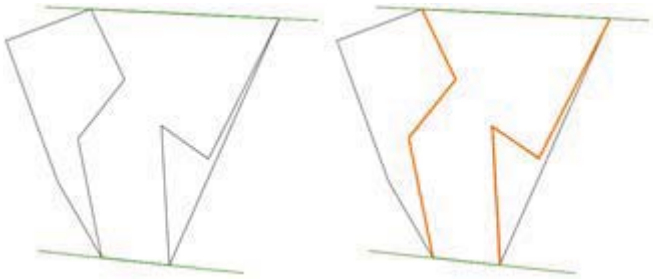

Fig. 1. Two subchains between which a minimum area quadrilateral is sought.

Consider the polygon formed by the subchains, and segments of the upper and lower tangents. For each vertex of one of the subchains (for definiteness, we consider that the first one was chosen) construct a visibility polygon from this vertex (Fig. 2).

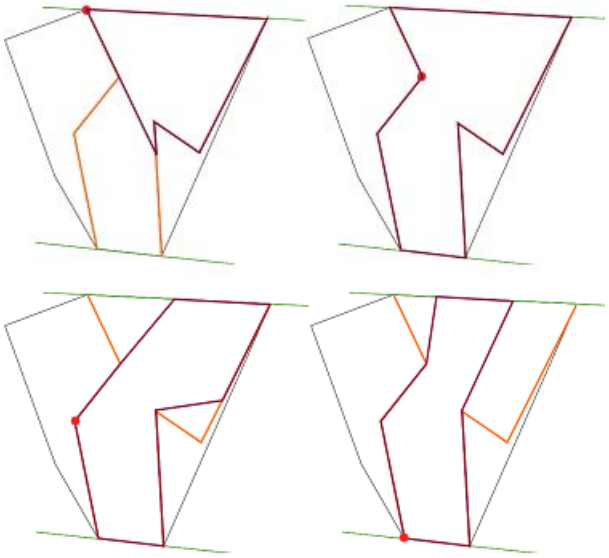

Fig. 2. Visibility polygons for each vertex of the left subchain.

Then, if for two edges (u_1, u_2) from the first subchain, and (v_1, v_2) from the second subchain, the vertex v_1 is visible from the vertex u_1, and the vertex v_2 is visible from the vertex u_2, and the segments (u_1, v_1), (u_2, v_2) do not intersect, then these edges define a correct quadrilateral (Fig.3) among of which we choose the having the smallest area (Fig. 4).

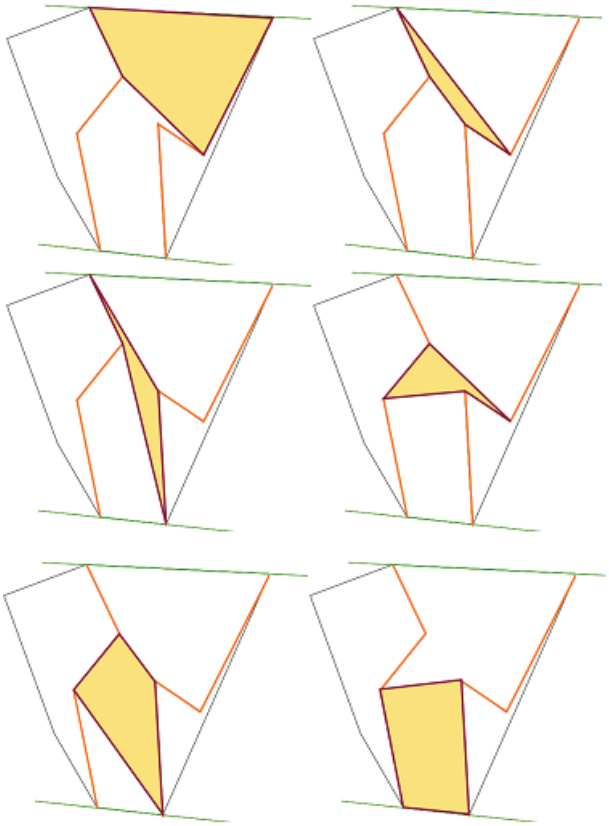

Fig. 3. All possible quadrilaterals.

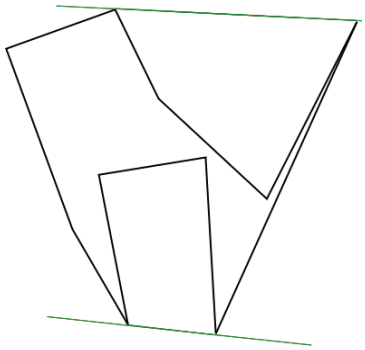

Fig. 4. The resulting polygon.

C. Algorithm

Input: Set S of n points on a plane. Output: A simple polygon that approximates minimum area polygonalization of the given set S.

Pre-processing: Sort out all points by the x-coordinate from left to right.

1. If $|S| \leq 5$, then return the exact solution found by brute-force algorithm.

Otherwise, divide the set S into two parts S_1 and S_2. Find a solution for these subsets recursively.

2. Merge the solutions for the subsets S_1 and S_2 using *MergeSimplePolygon* procedure, this gives a solution for the entire set S.

Procedure *MergeSimplePolygons*: *Input:* Two simple polygons P_1 and P_2.

Output: A simple polygon, obtained by merging polygons P_1 and P_2.

1. Find convex hulls of polygons P_1 and P_2: $CH(P_1)$ and $CH(P_2)$.

2. Find upper and lower tangent for $CH(P_1)$ and $CH(P_2)$.

3. Find two subchains between two polygons: l_1 and l_2.

4. For each edge from l_1: Find a set of visible vertices from the end of the segment. For each edge from l_2: Check the *IntersectionWithSubchainsTest* condition, and in the case of a successful passage, if area of the current quadrilateral is smaller than all considered before it, then remember the current one.

5. Merge the polygons and remove two extra edges.

IntersectionWithSubchainsTest: Input: Edges (u_1, u_2) and (v_1, v_2). Output:

True - if the edges define the correct quadrilateral, *False* - otherwise.

1. If segments (u_1, v_1) and (u_2, v_2) intersect - return False.

2. If vertex v_1 is visible from vertex u_1, and vertex v_2 is visible from vertex u_2, return *True*, otherwise return *False*.

D. Complexity

Theorem 1. *The approximate solution of MAP problem can be found in $O(n^2)$ time with $O(n\log n)$ pre-processing time.*

Proof. Pre-processing - sorting out points of the set S takes $O(n\log n)$ time [15]. Since the algorithm uses the "Divide and Conquer" strategy, if the complexity of the merge function is $O(f(n))$, then the overall complexity of an algorithm is determined by the solution of the recurrence $T(n) = 2T(n/2) + f(n)$. It makes sense to construct convex hull using the "Divide and Conquer" strategy as well. As we have two convex hulls from previous step, we can join them in $O(n)$ time [16]. During merging two convex polygons we also find two tangents between those polygons that are needed for *MergeSimplePolygons* procedure. Construction of two subchains l_1 and l_2 requires traversing along the contour of the corresponding polygons, therefore, it takes $O(n)$ time. In paragraph 2.1, an algorithm that finds a quadrilateral of minimal area, which is used for merging two simple polygons, is described. It is possible to find the set of visible vertices of a polygon from a vertex in $O(n)$ time [17], this procedure have to be performed for all vertices of one of the subchains, therefore the total complexity of finding the minimum area quadrilateral is $O(n^2)$. Thus, the time complexity of the merge procedure is $O(n^2 + n) = O(n^2)$. That is, we have the recurrence $T(n) = 2T(n/2) + n^2$. This recurrence can be solved using Master Theorem [18], and the solution is $O(n^2)$.

Theorem 2. *The approximate solution of MAP problem can be found using $O(n)$ memory.*

Proof. Sorting requires no more than $O(n)$ memory. Maintaining the convex hull of subsets also requires $O(n)$ memory [15]. Now we need to show that the minimal area quadrilateral could be found using $O(n)$ memory. In fact, to implement the given algorithm it is enough to build to lists l_1 and l_2 with length $O(n)$. Visibility polygon from a vertex could be found using $O(n)$ memory with an algorithm, provided in [16], as for each edge of one of the subchains we need only two visibility polygons (one for each vertex of an edge), this part of an algorithm requires $O(n)$ memory. Thus, the total memory complexity of an algorithm is $O(n)$.

E. Implementation

To test the efficiency of the described algorithm we have implemented it on C++ using framework Qt 5.10. There is two ways of specifying an input data for an algorithm - manual input and automatic generation of a given number of random points. We compared effectiveness of the algorithm with the two algorithms described in [7, 9]: "Permute and Reject" and "Greedy triangles". Since the first one can't be applied to large amount of points, we had limited the number of randomly generated permutations to 500 000 different simple polygons. The average smallest area of the polygons generated by "Permute and Reject" algorithm on a set of 100 points was 222 506, and on average it took 37 seconds per one test case, while the average area of polygons generated by the algorithm described in the paper was 190 242, and one test case took 4 milliseconds on average. Examples of polygons, generated by the implemented algorithm are shown in the figures below (Fig. 5, Fig. 6).

Fig. 5. Output of the algorithm for 30 input points.

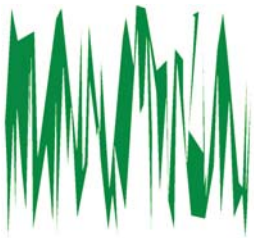

Fig. 6. Output of the algorithm for 100 input points.

Running time of the algorithm, depending on the number of input points is shown below (Fig. 7).

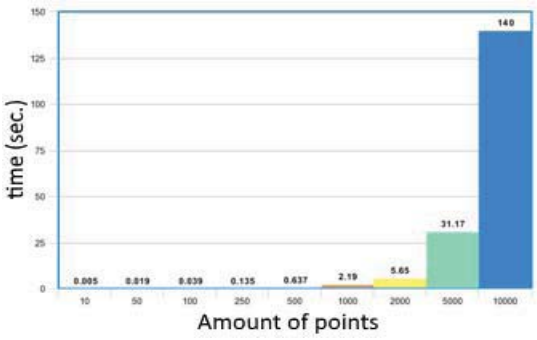

Fig. 7. Running time histogram for different amount of input points.

Also we compared the running time of our approach with the "Permute and Reject" algorithm [9] on different

amount of points. As it can be understood from the runtime comparison chart (Fig. 8) out approach is much more efficient (in sense of running time) on large data sets. In addition, we compared the average ratio of the area of the polygon produced by our algorithm with the one produced by "Greedy triangles" on the same data set. It turns out that on smaller data sets "Greedy triangles" algorithm produces more precise results (for data sets of 50 points the ratio is 1.139775) while on a larger ones difference in accuracy decreases (for data sets of 100 points the ratio is 1.088246 and for data sets of 200 points the ratio is 0.997589). This ratio is also represented at Fig. 9.

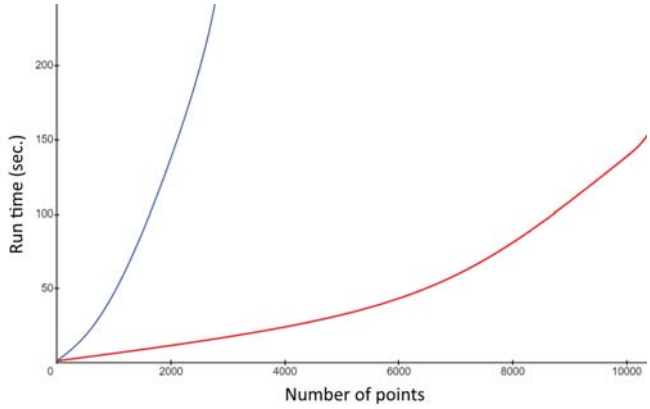

Fig. 8. Running time comparison: our approach (red curve) and "Greedy triangle" (blue curve).

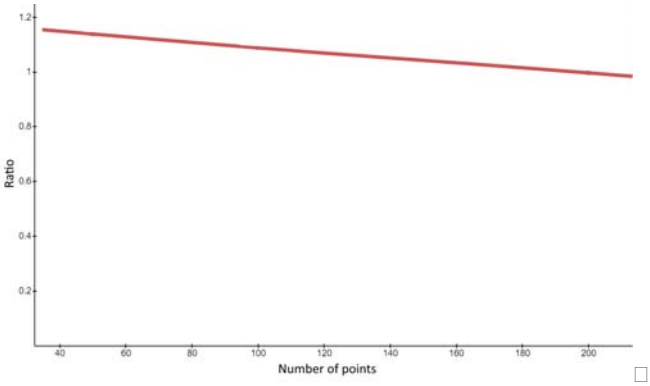

Fig. 9. Ratio of the average area of the polygons produced by our algorithm to the average area of the polygons produced by "Greedy triangle".

CONCLUSIONS

In the paper, a heuristic algorithm for constructing an approximate solution of the Minimum Area Polygonalization problem is proposed. The algorithm is based on the Divide and Conquer strategy. The described algorithm can be used in various application areas where the MAP problem arises. For an instance minimum area can be used as one of the restrictions on polygonalization in pattern recognition and image reconstruction problems [19]. We have proved that the time complexity of the algorithm is $O(n^2)$, pre-processing phase complexity is $O(n\log n)$ and memory usage $O(n)$. This follows that the given algorithm is one of the fastest algorithms for constructing the approximate solution of the MAP problem. Improvement of the accuracy of the constructed solution is possible by running the algorithm several times on a transformed set of points, for example, we can rotate all the points at a certain angle (which is equivalent to dividing the set of points with a line that is not parallel to the X axis). Determining the optimal number of iterations requires further investigation.

REFERENCES

[1] M. F. Worboys, M. Duckham, "Monitoring qualitative spatiotemporal change for geosensor networks", International Journal of Geographic Information Science 20(10), 2006, pp. 1087-1108.

[2] A. Galton, " Dynamic collectives and their collective dynamics", COSIT 2005, LNCS, vol. 3693, pp. 300-315, 2005.

[3] H. J. Miller, J. Han J., Eds, Geographic Data Mining and Knowledge Discovery, CRC Press, 2001.

[4] A.Galton, M. Duckham, "What is the region occupied by a set of points?", GIScience 2006, LNCS, vol. 4197, pp. 81-98, 2006.

[5] S. P. Fekete, On Simple Polygonalizations with Optimal Area, Discrete and Computational Geometry. Springer, 2000.

[6] Generating random simple polygons, http://jeffe.cs.illinois.edu/open/randompoly.html. Last accessed 09 Apr 1999.

[7] T. Auer, M. Held, "Heuristics for the Generation of Random Polygons", in: 8th Canad. Conf. Comput. Geometry, pp. 38-44. Carleton University Press, 1996.

[8] M. T. Taranilla, E.O. Gagliardi, G. H. Penalver, "Approaching Minimum Area Polygonization", in: XIV Workshop de Investigadores en Ciencias de la Computacion, pp. 271-281, 2012.

[9] V. Tereshchenko, V. Muravitskiy, Constructing a simple polygonalizations. Journal of World Academy of Science, Engineering and Technology (77), pp. 668-671, 2011.

[10] Tangents to and between 2D Polygon, http://geomalgorithms.com/ -_tangents.html. Last accessed 12 Oct 2001.

[11] P. Jiju, P. Amal, M. Ramanathan, Empirical Study on Randomized Optimal Area Polygonization of Planar Point Sets. Journal of Experimental Algorithmics 21, (2016), 10.1145/2896849.

[12] P. Jiju, P. Amal, M. Ramanathan, A Randomized Approach to Volume Constrained Polyhedronization Problem. J. Comput. Inf. Sci. Eng 15(1), 2015, DOI: http://dx.doi.org/10.1115/1.4029559.

[13] S. Asaeedi, F. Didehvar, A. Mohades, NLP Formulation for Polygon Optimiza tion Problems. Mathematics 7(1), 2019.

[14] E. Donald, On Simple Polygonalizations with Optimal Area. Discrete and Computational Geometry. Springer, 2000.

[15] D. E. Knuth, The Art of Computer Programming - Sorting and Searching, 3 (2nd ed.), Boston: Addison-Wesley, ISBN 0201896850, 1998.

[16] F. P. Preparata, S.J.Hong, "Convex Hulls of Finite Sets of Points in Two and Three Dimensions", Communications of the ACM 20, pp. 87-93, 1977.

[17] B.Joe, R.B. Simpson, Corrections to lee's visibility polygon algorithm. BIT Numerical Mathematics 27(4), pp. 458-473, 1987.

[18] The Master Theorem, http:math.dartmouth.edu/archive/m19w03=publichtml=Section5¡2:pdf. Last accessed 4 Oct 2017.

[19] L. Deneen, G. Shute, "Polygonizations of point sets in the plane. Discrete and Computational Geometry" 3, pp.7-87, 1988.