On Existing Min Area Polygonalization Algorithms And Possible Improvements

Maksym Kovalchuk

Faculty of Computer Sciences and Cybernetics

Taras Shevchenko national university of Kyiv

Kyiv, Ukraine

max.koval4uk@ukr.net

Abstract. It was proven that Minimum Area Polygonalization (MAP) problem belongs to the NP-Hard set of problems. Therefore, different approximation algorithms were developed. In this paper, we suggest a modification to the recently proposed application of the "Divide And Conquer" technique to the MAP problem. Our algorithm computes slightly more cases and often outperforms other algorithms in terms of polygon area. The complexity of the modified algorithm is $O(n^2 log(n))$ using $O(n)$ memory. We also introduce postprocessing technique. It takes solutions from other algorithms and makes local points rearrangement if it minimizes polygon area. The complexity of postprocessing is $O(n^2)$ using $O(n)$ memory. Experimental results indicate that it is expedient to use our algorithm in combination with postprocessing under time pressure. If no time pressure it is also useful to try randomized MAP algorithms in combination with postprocessing.

Keywords: computational geometry, minimum area polygonalization, divide and conquer, postprocessing

I. Introducing

Computational Geometry plays important role in nowadays problemsolving. It offers various solutions for both deeply theoretical research and applied software. One type of problem is constructing different shapes optimizing parameters like perimeter and area.

In this paper, we consider the problem of finding the minimal area polygon (MAP), vertices of which are all points from the given set. In practice, this problem arises in the context of geo-sensor networks [1,2], and GIS systems [3, 4]. Like Traveling Salesman Problem, MAP is NP-hard, which was proven in [5]. It makes it almost impossible to solve the problem provably optimal for big enough data sets. Most related works focus on approximation of MAP, rather than finding the best possible polygonalization.

Related works. The simplest algorithm for finding optimal MAP is MAP_PermuteAndReject with complexity $O(n!)$. MAP_PermuteAndReject brute-force all permutations, each corresponding unique polygon, checks if the polygon is simple and takes one with minimal area. In [9, 10] MAP was formulated as the IP problem. Such an approach

works faster than MAP_PermuteAndReject without losing optimality. MAP for 24 points was computed in about 90000 seconds [10].

Algorithm MAP_Greedy proposed in [8]. The idea is to start with the convex hull of points and iteratively add feasible points that form maximal area triangle with edges of the previously computed polygon. The complexity of the algorithm is $O(n^4)$ with $O(n)$ memory usage or $O(n^3)$ with $O(n^2)$ memory usage. Algorithm MAP_DAC with complexity $O(n^2)$ described in [13]. Authors on each step divide point set on 2 subsets, recursively solve the problem for smaller sets and merge found polygons with minimal area quadrilateral. If the subset of the points has less than 6 elements MAP_PermuteAndReject is used.

Randomized algorithms were proposed in [11, 12]. MAP_RS [12] consist of 6 strategies how to choose the initial triangle, next point, and next edge. We believe that complexity of MAP_RS is $O(q\,n^3)$, where q is the number of trials and n is the number of points. MAP_RAND [11] starts with the triangle. On each step, it chooses a random point and connects it with the edge that minimizes the polygon area. MAP_RAND is the only algorithm that processes both inside the polygon and outside the polygon types of points. In [11] MAP_RAND described with $O(q\,n^2\,log(n))$ complexity, but we believe that complexity can be improved to $O(q\,n^2)$ using linear time polygon visibility algorithm [14]. In [12] algorithm based on Ant Colony Optimization was proposed.

The rest of this article is organized as follows. In Section II, we formulate the MAP problem, describe a modification to the MAP_DAC algorithm, and propose a simple postprocessing technique. In section III, we analyze the complexity of described algorithms and their correctness. In section IV, we present implementation details, a comparison of MAP_DAC2 with the existing approximation MAP algorithms, and results of postprocessing output of other algorithms. We present our conclusion in Section V.

II Proposed 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. Modification of MAP_DAC

Proposed in [13] algorithm performs next steps.

1. If the number of points is less than 6, return MAP using MAP_PermuteAndReject;
2. Sort out all given points by X coordinate;
3. Split sorted points into 2 subsets S_1 and S_2;
4. Solve recursively (from step 1) for S_1 and S_2, getting polygons P_1 and P_2;
5. Find minimal area quadrilateral Q connecting P_1 and P_1;
6. Merge P_1 and P_2 based on Q and return result.

In such a way we always split the point set by the vertical line. We can consider lines with different slope, but while it can be optimal for some point subsets it would not for other.

Therefore we propose to maintain 2 solutions for each point subset, one for vertical separation and other for horizontal separation. In such a way we bring more local optimality without much loss of execution time efficiency.

Our algorithm MAP_DAC2 performs the next steps.

1. If the number of points is less than 6, return MAP using MAP_PermuteAndReject;
2. Sort out all given points by X coordinate and split sorted points into 2 subsets S_x^1 and S_x^2;
3. Solve recursively (from step 1) for S_x^1 and S_x^2 getting polygons $P_x^{1,1}$, $P_x^{1,2}$, $P_x^{2,1}$ and $P_x^{2,2}$;
4. Consider polygons PM_x^1, PM_x^2, PM_x^3, PM_x^4 received by merging pairs $\{P_x^{1,1}, P_x^{2,1}\}$, $\{P_x^{1,1}, P_x^{2,2}\}$, $\{P_x^{1,2}, P_x^{2,1}\}$, $\{P_x^{1,2}, P_x^{2,2}\}$ respectively;
5. Let P_x be one of the PM_x^1, PM_x^2, PM_x^3, PM_x^4 with the minimal area;
6. Sort out all given points by Y coordinate and split sorted points into 2 subsets S_y^1 and S_y^2;
7. Solve recursively (from step 1) for S_y^1 and S_y^2 getting polygons $P_y^{1,1}$, $P_y^{1,2}$, $P_y^{2,1}$ and $P_y^{2,2}$;
8. Consider polygons PM_y^1, PM_y^2, PM_y^3, PM_y^4 received by merging pairs $\{P_y^{1,1}, P_y^{2,1}\}$, $\{P_y^{1,1}, P_y^{2,2}\}$, $\{P_y^{1,2}, P_y^{2,1}\}$, $\{P_y^{1,2}, P_y^{2,2}\}$ respectively;
9. Let P_y be one of the PM_y^1, PM_y^2, PM_y^3, PM_y^4 with the minimal area;
10. Return pair $\{P_x, P_y\}$.

All steps of MAP_DAC2 except 4 and 8 are straightforward. Step 4 is described in the original paper for MAP_DAC [13]. Step 8 is almost the same as 4, but with horizontal separation. In case implementation is hardcoded for vertical separation, it is possible to interchange X and Y coordinates of points to bring step 8 to 4.

B. Postprocessing of MAP solution

It is typical for MAP approximation algorithms to use heuristics that greedily choose point, edge, quadrilateral, etc. Such heuristics may be optimal on the current step but can be nonoptimal in the future. We propose a simple postprocessing technique that removes 1 point at a time and tries to add it in the more optimal position. Postprocessing lies in the next steps.

1. Consider all points P_i of polygon P = P_1, P_2, ..., P_n one by one;
2. If we can remove P_i such that P^* = P_1, ..., P_{i-1}, P_{i+1}, ..., P_n is simple polygon(*) and segment (P_{i-1}, P_{i+1}) of P^* is visible from P_i (**) then remove P_i else goto step 1 and consider next point;
3. If P_i is inside P^* then find edge (P_{j-1}, P_j) of P^* visible from P_i that maximize area of triangle $\{P_{j-1}, P_i, P_j\}$. if P_i is outside P^* then find edge (P_{j-1}, P_j) of P^* visible from P_i that minimize area of triangle $\{P_{j-1}, P_i, P_j\}$;
4. Insert P_i such that new value of polygon P = P_1, ..., P_{j-1}, P_i, P_j, ..., P_n and goto step 1 considering next point.

To check (*) we can simply check whether edge (P_{i-1}, P_{i+1}) intersect other edges of P^*. To check (**) we can use the visibility polygon algorithm proposed in [14]. To check whether

a point is inside or outside of the polygon we can use the windmill algorithm. To find edge (P_{j-1}, P_j) we can use visibility polygon computed when checking (**).

In practice, we ran several trials of postprocessing until no more area minimizing points rearrangements are possible.

III Complexity and Correctness

A. Complexity

Theorem 1. Algorithm MAP_DAC2 can be implemented with complexity $O(n^2\ log(n))$.

Proof. MAP_DAC2 is recursive algorithm. During the one iteration, it sorts out points twice, calls merging function 8 times, and calls itself recursively 4 times with 2 times smaller point set. Complexity of sorting is $O(n\ log(n))$. Complexity of merging function is $O(n^2)$ [13]. That is we have recurrence $T(n) = 4 * T(n/2) + 8 * O(n^2) + 2 * O(n\ log(n))$. After simplification we get $T(n) = 4 * T(n/2) + O(n^2)$. Last recurrence can be solved using Master Theorem and the solution is $O(n^2\ log(n))$.

Theorem 2. Postprocessing can be implemented with complexity $O(n^2)$.

Proof. We can check (*) and (**) as described above in $O(n)$ time. The complexity of the windmill algorithm is also $O(n)$. If while checking (**) visibility polygon already constructed then we can find edge (P_{j-1}, P_j) of step 4 with complexity $O(n)$. Therefore, we can process one point in $O(n)$ time. We need to process n points which lead to overall complexity $O(n^2)$.

In practice, we ran postprocessing several times until the polygon area converge. During the experimental study, maximal number of postprocessing trials was 10. We believe that running postprocessing until area converges also has $O(n^2)$ complexity.

B. Correctness of MAP_DAC2

Correctness of MAP_DAC2 consists in the existence of quadrilateral merging 2 polygons on steps 4 and 9. MAP_DAC2 tries to merge 4 pairs of polygons on both step 4 and step 9. During the experimental study, we found pairs that cannot be merged as such a quadrilateral do not exist. Despite it, at least one of 4 pairs always could be combined when running MAP_DAC2 on our data set. We don`t know whether it holds in the general case. In the implementation we assign infinity area to the incorrect polygonalization, leading the algorithm to choose one with a smaller area. In case the algorithm approved to be incorrect in general we suggest running several trials with randomly rotated input points.

C. Correctness of postprocessing

Trap region of polygon P is the area of the plane from which no edges of P are completely visible (Fig. 1). Concept of trap region introduced in [11]. Therefore, for the algorithm to be correct it is necessary to not lead input points into the trap region or somehow deal with the trap region otherwise. [11] propose to ran MAP_RAND with different random

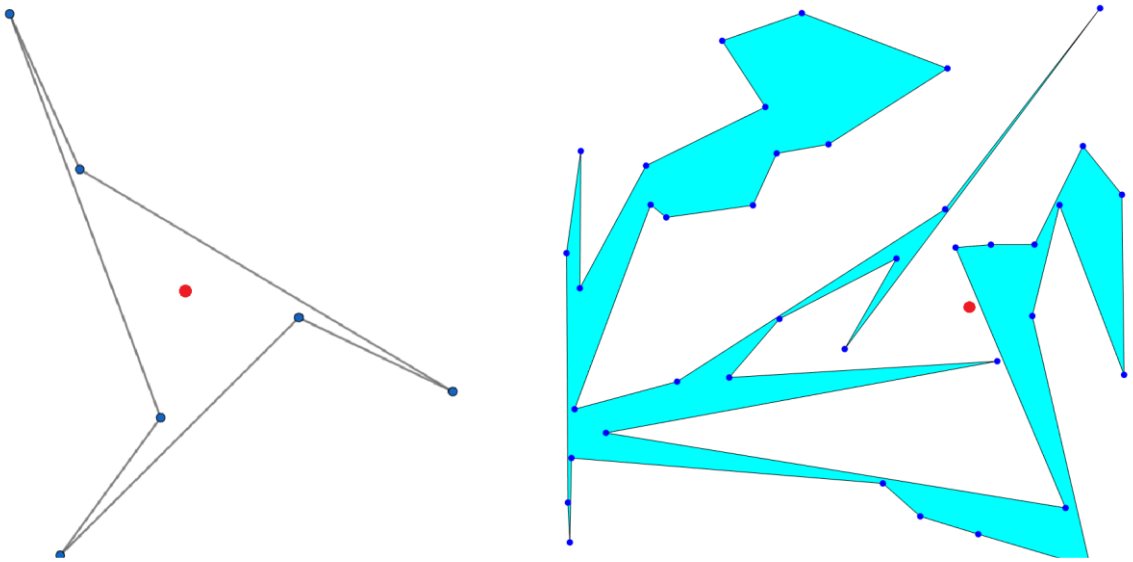

Fig. 1. Red point is inside the trap region. (a) point inside the polygon. (b) point outside the polygon.

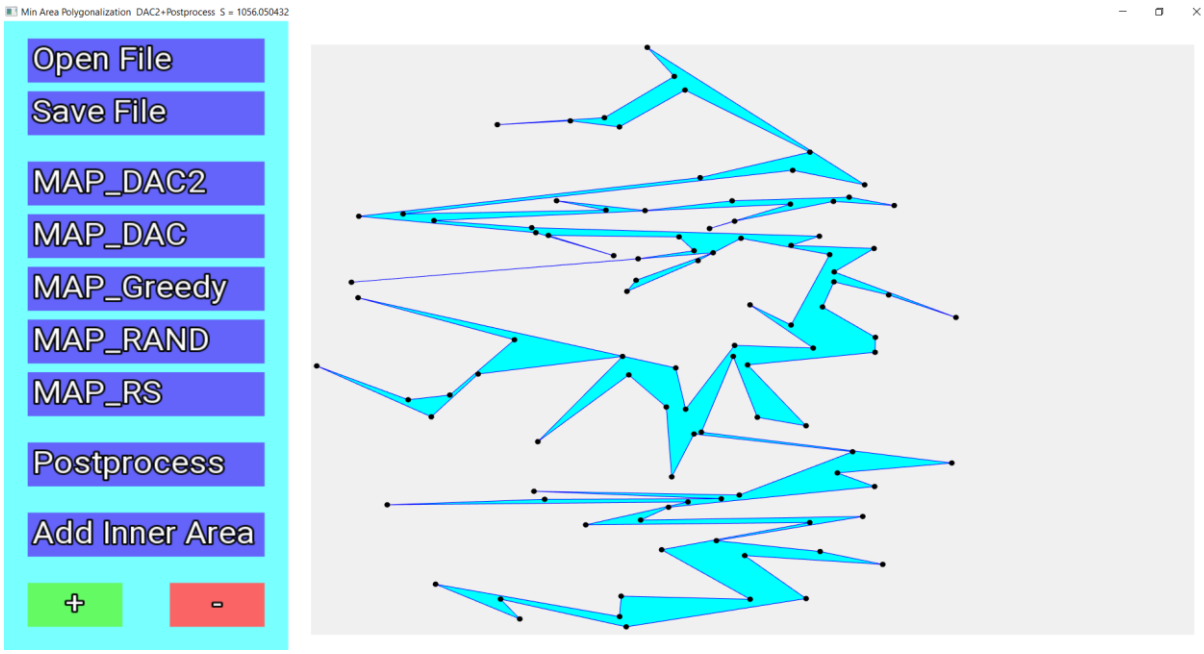

Fig. 2. The interface of the developed program.

seed in case of some point in the trap region. The same can be done in MAP_RS. So, we can consider randomized algorithms to be correct. During the experimental study, MAP_Greedy fell into the trap region several times. Because MAP_Greedy described in [8] is a deterministic algorithm, we claim it to be incorrect in the general case.

Theorem 3. The postprocessing algorithm never throws removed points into the trap region of P^*. The postprocessing algorithm either improves the area of the given polygon or does not change it.

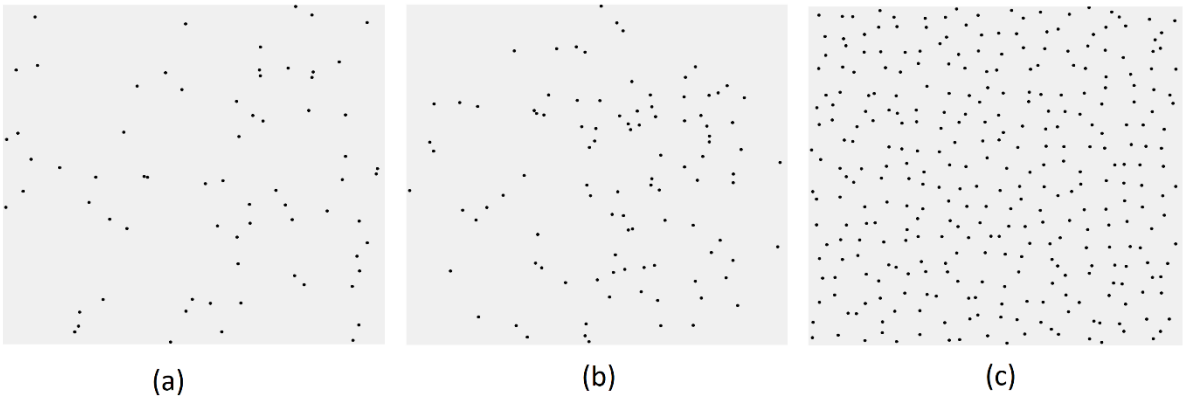

Fig. 3. (a) 1_square.txt (b) 1_circle.txt (c) 1_grid.txt

Proof. For every point there possible 3 distinct cases.

1. (+) At least one of (*) or (**) does not hold and the point is not removed;
2. (++) Point is removed and added such that old and new values of P are the same;
3. (+++) Point is removed and added such that old and new values of P are not the same.

In case (+), we do not remove points, therefore correctness is held. Otherwise (cases (++), (+++)) point is removed. In case (+++) not only correctness is held but also area is improved. If both (+) and (+++) are not met then from (+) we have guarantee of (**). That is, edge (P_{i-1}, P_{i+1}) of P^* is visible from P_i So, removed point P_i is not in the trap region but overall area remains the same as before (case (++)).

IV Experimental Results

To test described algorithms and compare them with previously proposed algorithms we have implemented MAP_DAC2, MAP_DAC, MAP_Greedy, MAP_RAND, MAP_RS, and postprocessing. We write in C++ using the SFML library for a graphical interface (Fig. 2). There are 2 ways of specifying the data – reading points from the file and adding or deleting points via clicking on the drawing panel.

To compare different algorithms 30 random point sets are generated. There are 3 types of point sets, each type consist of 10 sets of different sizes. The First 2 types are random points in rectangle and circle, named *_square.txt and *_circle.txt respectively. Points from the third type form a grid with some random shift from the exact position, named *_grid.txt. Different types are shown in Fig. 3.

We ran all 5 algorithms on described point sets (Fig. 4 (a)-(e)). For randomized algorithms MAP_RAND and MAP_RS we set q=200 trials and take the best produced polygonalization. Table 1 shows the results. As MAP_Greedy was discovered to be incorrect, our implementation of this algorithm ignores points that are in the trap region. Cases with incorrect output polygonalization are noted with the asterisk. In 17 out of 30 cases the best result was computed by MAP_DAC2. In the other 13 cases, the best polygonalization was computed by MAP_RS. Regardless of MAP_RS can outperform MAP_DAC2 in terms of

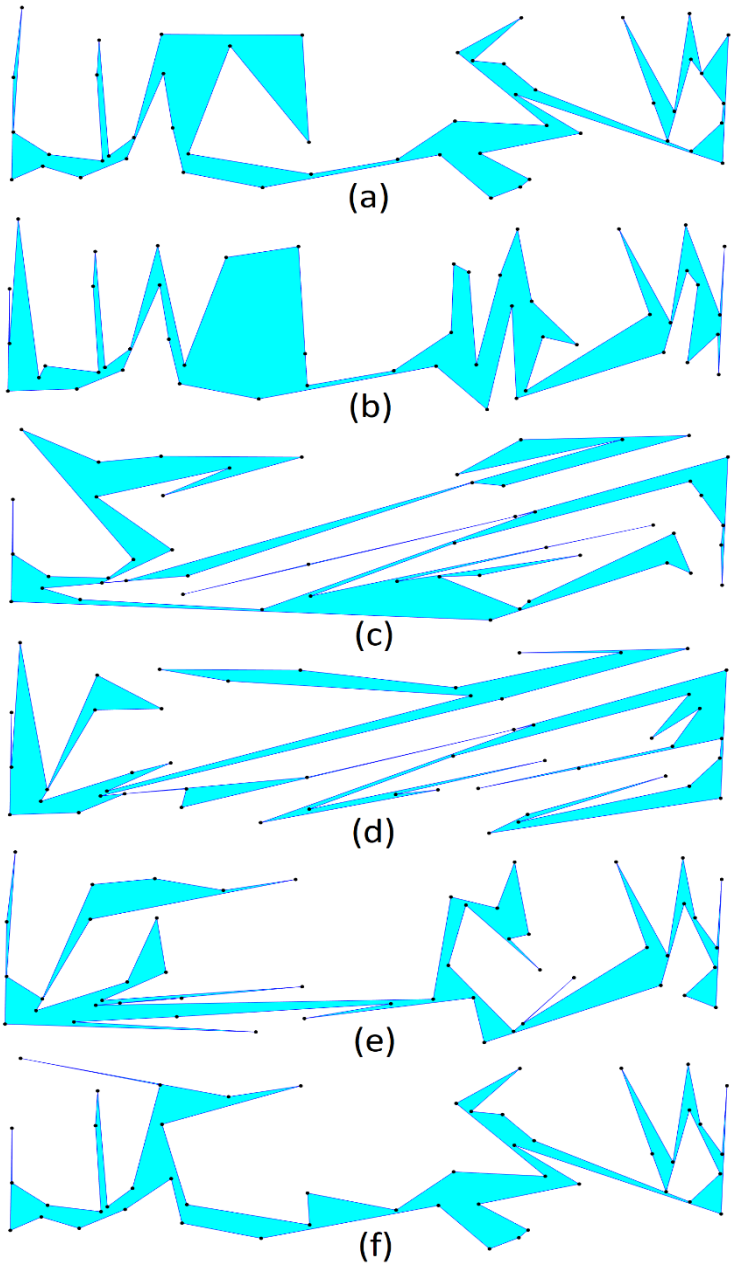

Fig. 4. Polygonalization of 5_square.txt (a) MAP_DAC2 (b) MAP_DAC (c) MAP_Greedy (d) MAP_RAND (e) MAP_RS (f) MAP_DAC2_P

polygon area, MAP_RS is quite slow relative to MAP_DAC2 as their complexities are $O(q\ n^3)$ and $O(n^2\ log(n))$ respectively.

We also ran postprocessing on output from all algorithms on all point sets (Fig, 4 (f)). Table 2 shows areas of polygons after postprocessing in percentages relative to the area before postprocessing. We emphasize that postprocessing allows decreasing the area of solutions up to 10-25%. Table 3 shows results after postprocessing. The result is improved by 18.7% on average. MAP_DAC2_P outputs the best result in 17 cases, MAP_RS_P in 11 cases, and MAP_RAND_P in 2 cases. We apply postprocessing only to the best output of MAP_RS and MAP_RAND. It is expected to get even better results if postprocessing is applied after every trial of randomized algorithms.

File name	Number of points	**MAP_DAC2**	MAP_DAC	MAP_Greedy	MAP_RAND	**MAP_RS**
0_circle.txt	50	1546.95	2205.22	2163.97	1421.74	**1337.32**
0_grid.txt	225	**5073.6**	7552.84	5893.04	6450.58	5136.01
0_square.txt	20	1649.18	2569.88	1783.53	1347.51	**1275.03**
1_circle.txt	100	1407.81	2478.42	1613.12	1285.63	**1245.65**
1_grid.txt	324	**7509.8**	10717.5	9285.85	9478.32	7556.45
1_square.txt	70	2029.32	2206.15	2844.91	1834.54	**1613.95**
2_circle.txt	150	1459.24	2173.04	2132.49	1606.34	**1402.67**
2_grid.txt	441	**10196.1**	14775.8	12771.9	12862.2	10229.6
2_square.txt	220	2022.32	3182.49	2937.25*****	2242.83	**1869.77**
3_circle.txt	200	1565.39	2702.54	1597.23	1674.86	**1398.37**
3_grid.txt	576	**13397.4**	19157.1	18039.6	16774.6	13640.4
3_square.txt	470	**1851.76**	2916.31	2669.09	2294.82	1903.7
4_circle.txt	250	1534.87	2405.29	1873.09	1704.48	**1466.64**
4_grid.txt	729	**16209.4**	22362.3	20766.8*****	21420.7	17387.4
4_square.txt	820	**1881.42**	2910.44	2708.03	2452.73	2018.38
5_circle.txt	300	1533.45	2192.7	2027.85	1805.44	**1479.12**
5_grid.txt	900	**19876**	28519.2	26701.5	26726.3	21612.8
5_square.txt	50	589.507	797.318	681.784	594.001	**497.325**
6_circle.txt	350	**1441.19**	2430.5	1901.1	1792.82	1475.15
6_grid.txt	330	**6667.67**	10850.5	9933.22*****	8969.63	7247.62
6_square.txt	140	560.736	875.54	680.984	597.184	**530.764**
7_circle.txt	400	**1397.47**	2552.57	1923.42	1809.85	1492.82
7_grid.txt	360	**7818.05**	12676.4	10598.4*****	9783.51	8104.49
7_square.txt	290	564.116	933.543	741.509	675.447	**561.196**
8_circle.txt	450	**1485.48**	2288.47	1966.1	1695.7	1515.07
8_grid.txt	390	**7870.21**	13452.7	11478.4	10362.4	8182.76
8_square.txt	500	**548.995**	973.862	772.515	701.326	573.593
9_circle.txt	500	**1536.36**	2544.76	2084.82	1819.55	1591.75
9_grid.txt	420	9318.83	13578.1	12283.1	11080.6	**8967.92**
9_square.txt	770	**561.227**	945.21	826.275*****	718.896	567.384

Table 1. Comparison of results obtained from MAP_DAC2, MAP_DAC, MAP_Greedy, MAP_RAND, MAP_RS.

V Conclusion

In this paper, we present modification (MAP_DAC2) to the Divide and Conquer based algorithm for constructing an approximate solution of the MAP problem. The algorithm computes slightly more cases compared to the unmodified version but shows significant improvement in the polygon area. The complexity of the algorithm is $O(n^2\ log(n))$ using $O(n)$ memory. We also propose the algorithm for postprocessing output of other algorithms that improves local optimality via local rearrangements point by point. The complexity of postprocessing is $O(n^2)$ using $O(n)$ memory.

We conduct experimental study to compare existing algorithms. MAP_DAC2 turns out to be competitive in terms of polygonalization area while working much faster than the best of existing algorithms. We also compare results before and after postprocessing. Postprocessing improved best polygonalizations by 18% on average which is an outstanding result.

File name	MAP_DAC2_P	MAP_DAC_P	MAP_Greedy_P	MAP_RAND_P	MAP_RS_P
0_circle.txt	83.2882	68.8231	74.8971	95.2749	86.6654
0_grid.txt	78.4822	60.7262	93.39	83.7179	85.1955
0_square.txt	86.2036	56.9215	91.4348	100	98.5454
1_circle.txt	75.0139	53.6432	92.613	74.8542	92.1643
1_grid.txt	77.4988	60.4907	89.2939	79.3326	75.3776
1_square.txt	84.4191	81.2336	88.9499	82.773	95.9746
2_circle.txt	84.6908	68.3225	82.913	83.7697	82.2322
2_grid.txt	76.7843	59.1727	91.3193	81.1598	83.3493
2_square.txt	77.272	60.4619	89.8424*	71.8307	91.8665
3_circle.txt	82.1769	55.8258	94.2239	77.7306	85.2177
3_grid.txt	74.7865	55.3249	88.3407	80.2147	83.075
3_square.txt	79.7649	59.1584	85.9197	74.7312	87.1332
4_circle.txt	77.9657	62.6643	92.1431	82.6544	83.4875
4_grid.txt	81.1176	62.3924	83.0463*	76.909	81.0698
4_square.txt	79.6647	59.2644	83.1866	76.7587	77.6908
5_circle.txt	81.8064	63.7366	84.6136	84.7903	78.679
5_grid.txt	78.9024	59.4176	85.437	80.5203	79.2287
5_square.txt	82.5412	70.5668	93.033	85.7604	82.9386
6_circle.txt	85.8389	60.589	85.0262	77.9972	83.1042
6_grid.txt	77.2097	66.6462	87.6776*	78.2857	85.3425
6_square.txt	83.7059	64.268	78.2077	85.5597	86.3094
7_circle.txt	80.6722	62.6201	82.9018	77.7518	80.1766
7_grid.txt	76.3475	65.5676	83.5795*	80.737	81.5681
7_square.txt	82.0708	57.3291	86.3043	83.8413	83.6514
8_circle.txt	81.367	63.3459	79.5047	72.7502	79.4567
8_grid.txt	75.0734	66.7353	86.8286	77.9401	80.2878
8_square.txt	80.2742	56.8465	82.3947	80.5178	85.0199
9_circle.txt	77.5403	57.7584	89.2339	74.1333	82.8734
9_grid.txt	81.2925	64.3985	80.572	79.5746	81.9839
9_square.txt	77.0793	61.8968	81.2972*	74.8826	82.044

Table 2. Area of polygons after postprocessing relative to Table 1.

During the research of existing and new MAP algorithms, we considered 3 tactics – producing new polygonalization, postprocessing of polygonalization, and constructing new better polygonalizations via combining other polygonalizations. Producing new polygonalization is relatively well studied in the literature. We propose the first MAP postprocessing technique in this paper. Polygon combining is left for future research. We believe that all 3 tactics deserve 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.

File name	DAC2_P	DAC_P	Greedy_P	RAND_P	RS_P	Result before postprocessing	Result after postprocessing
0_circle.txt	1288.43	1517.7	1620.75	1354.56	**1158.99**	1337.32	**1158.99**
0_grid.txt	**3981.88**	4586.55	5503.51	5400.29	4375.65	5073.6	**3981.88**
0_square.txt	1421.65	1462.81	1630.76	1347.51	**1256.49**	1275.03	**1256.49**
1_circle.txt	1056.05	1329.5	1493.96	**962.349**	1148.04	1245.65	**962.349**
1_grid.txt	5820	6483.09	8291.7	7519.4	**5695.87**	7509.8	**5695.87**
1_square.txt	1713.14	1792.14	2530.54	**1518.5**	1548.98	1613.95	**1518.5**
2_circle.txt	1235.84	1484.68	1768.12	1345.62	**1153.44**	1402.67	**1153.44**
2_grid.txt	**7829.01**	8743.24	11663.3	10438.9	8526.33	10196.1	**7829.01**
2_square.txt	**1562.69**	1924.19	2638.89*	1611.04	1717.69	1869.77	**1562.69**
3_circle.txt	1286.39	1508.71	1504.97	1301.88	**1191.66**	1398.37	**1191.66**
3_grid.txt	**10019.5**	10598.7	15936.3	13455.7	11331.8	13397.4	**10019.5**
3_square.txt	**1477.05**	1725.24	2293.27	1714.95	1658.75	1851.76	**1477.05**
4_circle.txt	**1196.67**	1507.26	1725.93	1408.83	1224.46	1466.64	**1196.67**
4_grid.txt	**13148.7**	13952.4	17246.1*	16474.5	14095.9	16209.4	**13148.7**
4_square.txt	**1498.83**	1724.86	2252.72	1882.68	1568.09	1881.42	**1498.83**
5_circle.txt	1254.46	1397.55	1715.84	1530.84	**1163.76**	1479.12	**1163.76**
5_grid.txt	**15682.7**	16945.4	22812.9	21520.1	17123.5	19876	**15682.7**
5_square.txt	486.586	562.642	634.284	509.418	**412.474**	497.325	**412.474**
6_circle.txt	1237.1	1472.61	1616.43	1398.35	**1225.92**	1441.19	**1225.92**
6_grid.txt	**5148.09**	7231.47	8709.21*	7021.94	6185.3	6667.67	**5148.09**
6_square.txt	469.369	562.691	532.582	510.949	**458.1**	530.764	**458.1**
7_circle.txt	**1127.37**	1598.42	1594.55	1407.19	1196.89	1397.47	**1127.37**
7_grid.txt	**5968.88**	8311.64	8858.07*	7898.91	6610.68	7818.05	**5968.88**
7_square.txt	**462.974**	535.192	639.955	566.304	469.448	561.196	**462.974**
8_circle.txt	1208.69	1449.65	1563.14	1233.62	**1203.83**	1485.48	**1203.83**
8_grid.txt	**5908.43**	8977.68	9966.53	8076.5	6569.76	7870.21	**5908.43**
8_square.txt	**440.701**	553.607	636.511	564.692	487.669	548.995	**440.701**
9_circle.txt	**1191.3**	1469.81	1860.37	1348.89	1319.14	1536.36	**1191.3**
9_grid.txt	7575.51	8744.08	9896.72	8817.38	**7352.25**	8967.92	**7352.25**
9_square.txt	**432.59**	585.055	671.739*	538.328	465.505	561.227	**432.59**

Table 3. Postprocessing. Comparison of results obtained from MAP_DAC2_P, MAP_DAC_P, MAP_Greedy_P, MAP_RAND_P, MAP_RS_P.

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. Subodh Kumar "Surface Triangulation: A Survey", July 3, 1996

7. David Eppstein, Mark Overmars, Giinter Rote, and Gerhard Woeginger "Finding Minimum Area k-gons", Discrete Comput Geom 7:45-58 (1992)

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

9. Sandor Fekete, Stephan Friedrichs, Michael Hemmer, Melanie Papenberg, Arne Schmidt, Julian Troegel "Area- and Boundary-Optimal Polygonalization of Planar Point Sets", EuroCG 2015, Ljubljana, Slovenia, March 16–18, 2015

10. Sandor Fekete, Stephan Friedrichs, Michael Hemmer, Melanie Papenberg, Arne Schmidt, Julian Troegel, "Computing Area-Optimal Simple Polygonalizations", 36th European Workshop on Computational Geometry, Würzburg, Germany, March 16–18, 2020

11.Jiju Peethambaran, Amal Dev Parakkat, and Ramanathan Muthuganapathy. 2016. An empirical study on randomized optimal area polygonization of planar point sets. J. Exp. Algorithmics 21, 1, Article 1.10 (April 2016).

12. Maria Teresa Taranilla, Edilma Olinda Gagliardi, and Gregorio Hern´andez Pe˜nalver, "Approaching Minimum Area Polygonization"

13. Maksym Osiponok, Vasyl Tereshchenko, "The "Divide and Conquer" technique to solve the Minimum Area Polygonalization problem", IEEE ATIT 2019 ISBN 978-1-72-81-61-44-0/19/$31.00 2019 IEEE

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