# A PARALLEL APPROXIMATION ALGORITHM FOR MINIMUM AREA POLYGONIZATION BASED ON CLUSTERING

Saeed ASAEEDI[1], Mahsa Soheil SHAMAEE[2]

*Minimum area polygonization is a well-known NP-complete problem in computational geometry. It is the problem of finding a simple polygon with minimum area for a given set of points in the plane. We present a parallel approximation algorithm based on clustering to solve this problem in polynomial time. The algorithm has four phases: clustering the points into the meaningful parts, reclustering the big clusters to the smaller ones, finding minimum area polygon for each cluster and, finally merging the polygons. We implement the algorithm and present the results of experiments by comparing the previous works. We compare the average score obtained by our algorithm and that of the previous methods. The score obtained by an algorithm is the ratio given by the area of the computed polygon using that algorithm divided by the area of the convex hull.*

**Keywords**: Minimum area polygonization, Approximation algorithm, Hierarchical clustering, Partitional clustering, Computational geometry

## 1. Introduction

Minimum Area Polygonization (MAP) was shown to be NP-complete by Fekete [1, 2], who also proved that no polynomial time approximation algorithm exists for MAP [3]. It is proved in [4] that computing α-Concave hull, as a generalization of MAP, is still NP-complete. The α-Concave hull on a set of points is the minimum area simple polygon containing those points with angular constraint.

The most related problem to MAP is Traveling Salesman Problem (TSP). Although there exist many algorithms to approximate and randomize TSP [5, 6, 7, 8, 9, 10], there are few studies on MAP. Taranilla et al. [11] presented three heuristic algorithms to obtain approximate solutions for MAP. Crombez et al. [12] proposed two algorithms, greedy method and local search, to find the maximum and minimum area polygons on a set of points. Maximum area polygonization (MAXP) is NP-hard same as MAP [1, 2]. Fekete presented a 1/2-approximation

[1] Department of Computer Science, Faculty of Mathematical Sciences, University of Kashan, Kashan 87317-53153, I. R. Iran, e-mail: asaeedi@kashanu.ac.ir

[2] Department of Computer Science, Faculty of Mathematical Sciences, University of Kashan, Kashan 87317-53153, I. R. Iran.

algorithm for MAXP [3]. Using the technique of randomized incremental construction, Peethambaran et al. [13] presented a greedy heuristic for MAP and MAXP. Muravitskiy and Tereshchenko [14] gave a greedy algorithm to solve MAP. The idea of the algorithm is simple: Compute the convex hull of the points and remove the largest possible triangle constructed by the inner points decrementally. Osiponok and Tereshchenko proposed a divide and conquer algorithm [15]: Divide the set of points into two subsets, construct approximated minimum area polygon recursively, and finally merge them.

In [16] a randomized approximation algorithm is presented for minimal and maximal volume polyhedronization of three-dimensional point sets. As the recent study, Fekete et al. [17] developed exact methods for MAP and MAXP based on integer programming. In recent years, a workshop was held at the 2019 Computational Geometry Week (CG Week) in Portland that focused on optimum area polygonization [18, 19].

In [20, 21] cluster polygonization had investigated by Lee and Estivill-Castro. They presented a linear time algorithm to transform point clusters into polygons. In this paper, we use clustering algorithms to split the points into small enough subsets, then transform each cluster into minimum area polygon locally and finally merge the computed polygons trying to keep minimality.

Two main categories of clustering are partitioning and hierarchical. Agglomerative or divisive hierarchical algorithms try to build a hierarchy of clusters and they are generally parameter-less. In this paper, we use the agglomerative hierarchical clustering algorithm presented in [22] to cluster the points into the meaningful parts. In this step, the number of clusters is not determined exactly, and it depends on the position of the points. The k-means algorithm [23] is a well-known partitional clustering method. Here, we use the k-means algorithm to recluster the big clusters to the fixed size smaller ones. We find the minimum area polygon on each cluster and then merge them to construct the approximated minimum area polygon on the points. In [24, 15] some polygon merging algorithms are shown. We present a new merging algorithm to keep minimality as much as possible. The rest of the paper is as follows: In the section 2, the parallel approximation algorithm is presented to solve MAP. In section 3, the numerical results are presented and discussed. Finally in section 4, we conclude the paper highlighting its achievements.

## 2. Approximation algorithm for MAP

Let $S$ be a set of points in the plane and $P_m(S)$ be the simple polygon on $S$ with the smallest possible area. In this section, we present a parallel approximation algorithm to compute $P_m(S)$. The algorithm has four phases: (1) classify $S$ into clusters of points close to each other by using agglomerative

hierarchical clustering algorithm, (2) recluster the big clusters containing more than 6 points into fixed size partitions, (3) parallelly, compute $P_m(s_i)$ for each final cluster $s_i$ of points, and (4) merge all polygons $P_m(s_i)$ by adding connecting rectangles with smallest areas.

In [22], an agglomerative hierarchical clustering algorithm using a sweeping approach was presented. The algorithm uses two horizontal sweep-lines moving through $S$ from bottom to top, and the clusters were constructed during this process. The time complexity of the algorithm is $O(n \log n)$. We use this clustering technique for the first phase of our algorithm. Fig. 1.b, Fig. 2.b and Fig. 3.b show how clustering algorithm works for random points, the dataset taken from SPAETH cluster analysis database [25] and sample points from TSPLIB benchmark data [26], respectively.

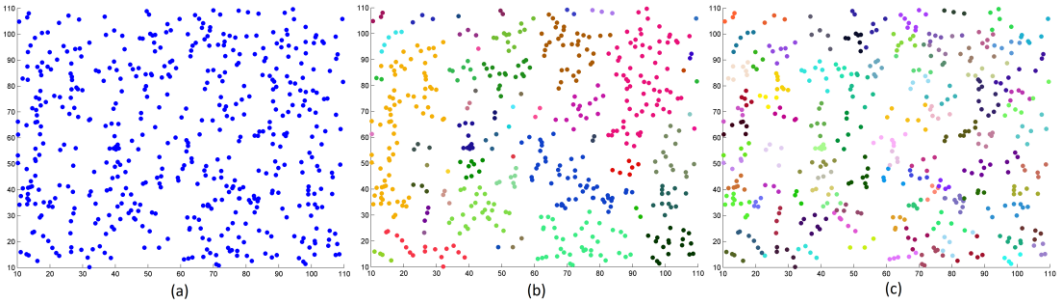


Fig. 1. (a) Set of 500 random points, (b) Output of first phase, (c) Output of second phase.

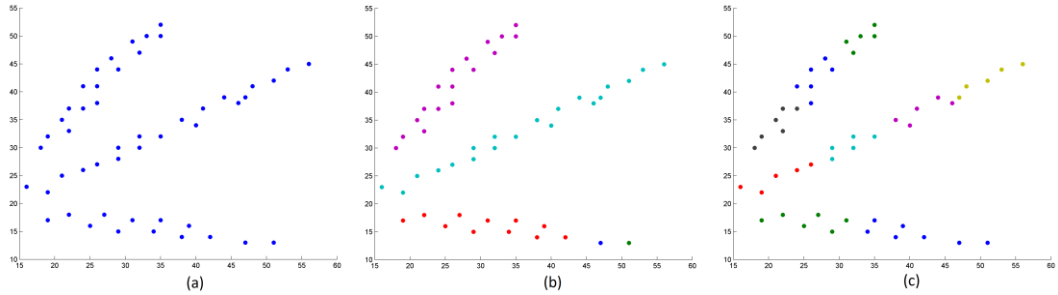


Fig. 2. (a) Set spaeth06 of SPAETH, (b) Output of first phase, (c) Output of second phase.

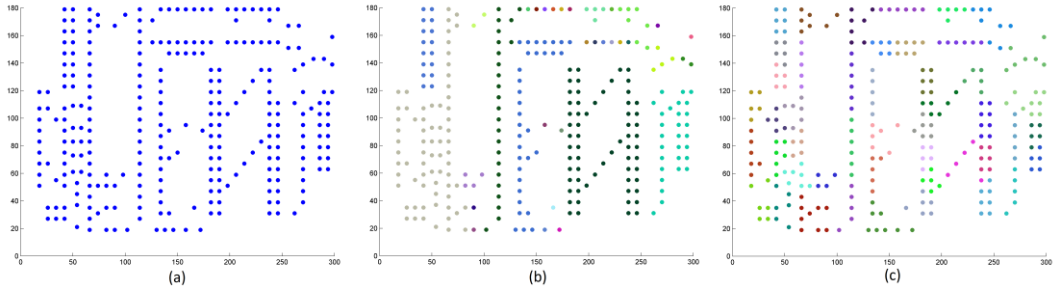


Fig. 3. (a) Instance a280 of TSPLIB, (b) Output of first phase, (c) Output of second phase.

The presented algorithm in this paper is a divide and conquer algorithm. The algorithm first split the points into the small enough subsets, then compute minimum area polygon on each subset using the exact algorithm, and finally merge the computed polygons. Some clusters obtained from the first phase may be too large and therefore the exact algorithm is not efficient for these clusters. As the second phase, we use a partitional clustering algorithm to recluster the big clusters for obtaining small subsets.

We use the k-means algorithm to split the big clusters into fixed size partitions. Since any exact algorithm is fast enough on the set of 6 points, we consider this fixed size to be 6. We investigate the effect of different values for this cluster size on the efficiency of our algorithm in section 3. Let $S_i$ be the set of $n_i$ points of $i$th cluster such that $n_i>6$. We run the k-means algorithm on $S_i$ with $k=n_i/6$ centroids. Fig. 1.c, Fig. 2.c and Fig. 3.c depict the subclusters on random points, SPAETH and TSPLIB, respectively.

The outputs of the second phase of the algorithm are clusters each of which contains at most 6 points. In the next phase, we parallelly compute the minimum area polygon on each cluster using an exact algorithm. The exact algorithm is a full search of all simple polygons to find the minimum one. The time complexity of this algorithm is constant, $O(6!)$. Fig. 4.b, Fig. 5.b and Fig. 6.b show the output of this phase for the sets of 100, 200 and 500 random points, respectively.

As the final phase, we merge the polygons computed from the previous phase. We connect the polygons with minimum area rectangles. To address this issue, we use a greedy method: find the minimum area empty simple rectangle $R$ that connects two polygons without intersecting with others, and merge them using $R$. Fig. 4.c, Fig. 5.c and Fig. 6.c illustrate how this greedy method works on the sets of 100, 200 and 500 random points, respectively.

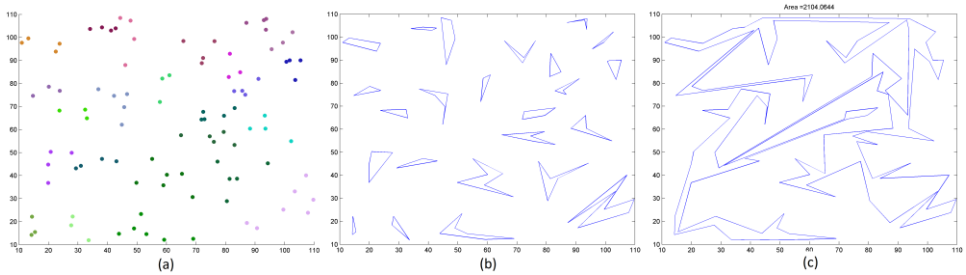


Fig. 4. (a) Final clusters on a set of 100 points, (b) Minimum area polygon on each cluster, (c) Merge the polygons to compute the approximated minimum area polygon.

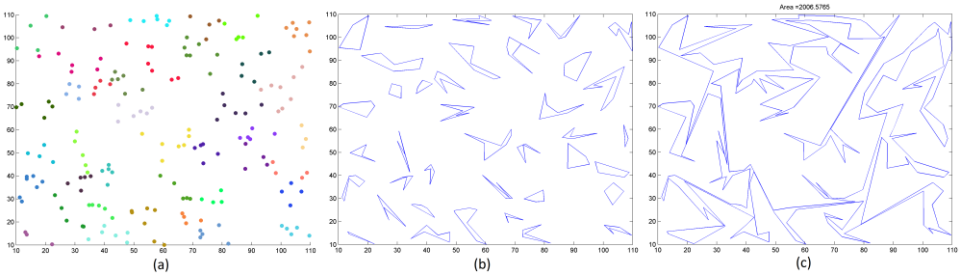


Fig. 5. (a) Final clusters on a set of 200 points, (b) Minimum area polygon on each cluster, (c) Merge the polygons to compute the approximated minimum area polygon.

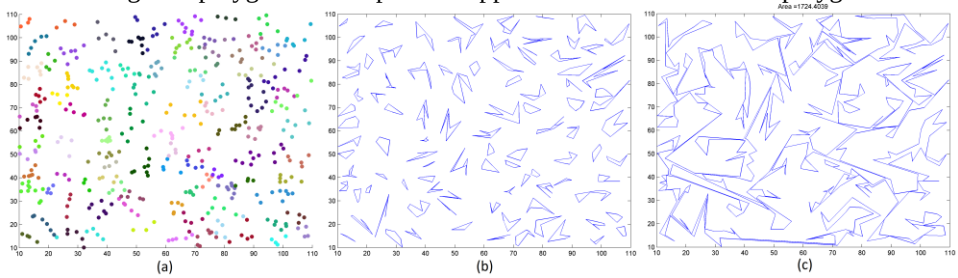


Fig. 6. (a) Final clusters on a set of 500 points, (b) Minimum area polygon on each cluster, (c) Merge the polygons to compute the approximated minimum area polygon.

Consequently, the pseudo code of our algorithm is shown in algorithm 2.1. The function *agglomerativeClustering* partitions the input parameter, *points*, to the clusters based on the presented algorithm in [22]. *kMeans* is a function with parameters *s* (the input points) and *n/6* (*n*=the size of *s*) that returns *n/6* clusters of *s* based on k-means method. The function *parallelExactAlgorithm* runs the exact algorithm of MAP on each cluster of "*secondClusters*", parallelly. The *polygons* returned by *parallelExactAlgorithm* are merged together by *mergePolygons*.

**Algorithm 2.1** Parallel Approximation Algorithm for MAP

**Require:** $points \in \mathbb{R}^2$
**Ensure:** $approximatedPolygon$
  $firstClusters \leftarrow agglomerativeClustering(points)$
  $secondClusters \leftarrow \emptyset$
  **for each** cluster $s$ in firstClusters **do**
    $n \leftarrow size(s)$
    **if** $n > 6$ **then**
      $newClusters \leftarrow kMeans(s, \frac{n}{6})$
      $secondClusters \leftarrow secondClusters + newClusters$
    **else**
      $secondClusters \leftarrow secondClusters + s$
    **end if**
  **end for**
  $polygons \leftarrow parallelExactAlgorithm(secondClusters)$
  $approximatedPolygon \leftarrow mergePolygons(polygons)$

The function *mergePolygons* is described in algorithm 2.2:

**Algorithm 2.2** mergePolygons

**Require:** $polygons$
**Ensure:** $approximatedPolygon$
  **while** $size(polygons) > 1$ **do**
    $rec_{i',j'} \leftarrow$ Rectangle with the minimum area connecting two polygons $P_{i'}$ and $P_{j'}$ in $polygons$
    $rec \leftarrow \min_{i',j'} rec_{i',j'}$ and $i, j \leftarrow \arg\, min_{i',j'} rec_{i',j'}$
    $newPolygon \leftarrow$ Merge two polygons $P_i$ and $P_j$ using $rec$
    Remove $P_i$ and $P_j$ from $polygons$
    Add $newPolygon$ to $polygons$
  **end while**
  $approximatedPolygon \leftarrow polygons$

Since the time complexity of other phases are less than that of phase 4, algorithm 2.2 and therefore algorithm 2.1 runs in $O(n^4)$ time where $n$ is the number of points.

## 3. Numerical results

In this section, we first compare our results with those of obtained from the exact algorithm on the small datasets. Then we compare our algorithm with previous studies such as [13], [15] and [27] on the uniform datasets of 2019 CG Challenge [18, 19]. We implement the divide and conquer [15] and randomized incremental [13] algorithms to compare with our approach. Fig. 7 shows the results of these algorithms on the same set of 100 points. We also compare the algorithms on the non-uniform datasets which are collections of the separate point sets. Finally, we run our algorithm on existing datasets such as TSPLIB, SPAETH and instances of 2019 CG Challenge and obtain approximated minimum area polygon on these datasets.

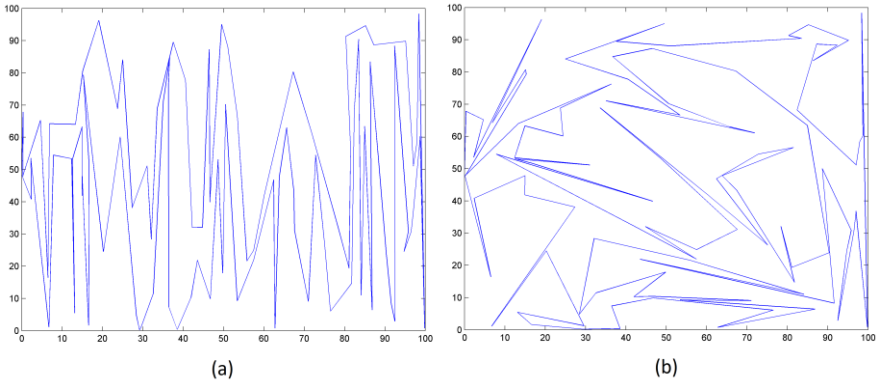


Fig. 7. (a) The wavy shape polygon constructed by divide and conquer algorithm, (b) The constructed polygon by randomized incremental algorithm.

Table 1 shows the results of analyses of datasets of 6-11 points. On each dataset, our algorithm is compared with the exact full search algorithm. In table 1, the area values are the average over 100 instances for each size. The average of execution time for each size is compared as shown in Fig. 8.

*Table 1*

**Comparison of our algorithm with the exact algorithm**

| Dataset size | Minimum area | Approximated area | Percentage |
|---|---|---|---|
| 6 | 1375.66 | 1421.23 | 96.80 |
| 7 | 1568.535 | 1762.48 | 89.00 |
| 8 | 1485.43 | 1887.93 | 78.68 |
| 9 | 1504.080808 | 2020.707071 | 74.43 |
| 10 | 1496.782828 | 2033.838384 | 73.59 |
| 11 | 1396.35 | 1945.9 | 71.75 |

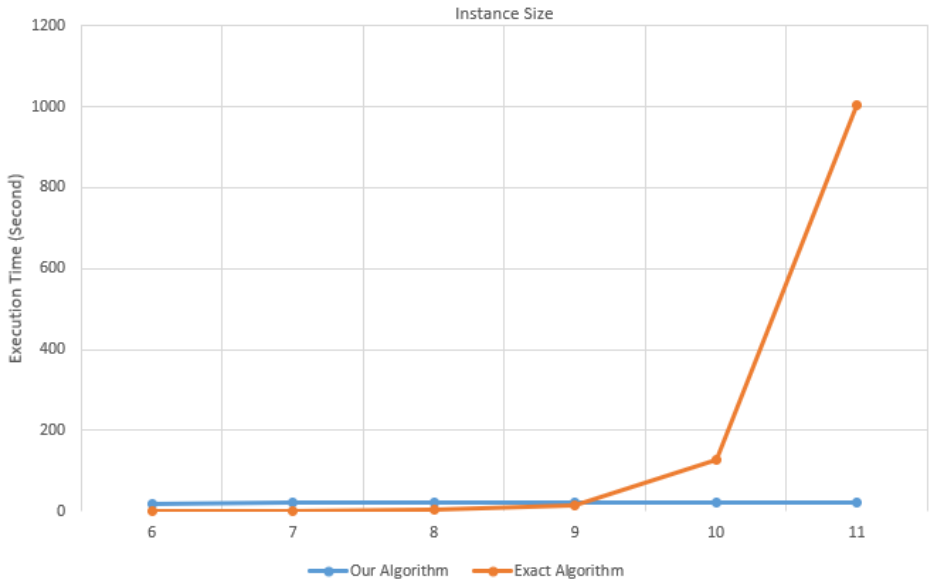


Fig. 8. Comparison of the execution time of our algorithm and the exact algorithm.

We run our algorithm on a collection of benchmark instances of 2019 CG Challenge [18]. We compare our result with that of obtained by the approximation algorithm, APX, presented in [15], the randomized algorithm, RAND, presented in [13] and the greedy algorithm, Greedy, presented in [27]. The Greedy algorithm is one of the best approximation algorithms presented in 2019 CG Challenge (See [19]). We compare the scores obtained by the algorithms in Fig. 9. The score obtained by an algorithm for each instance is the ratio given by the area of the computed polygon using that algorithm divided by the area of the convex hull.

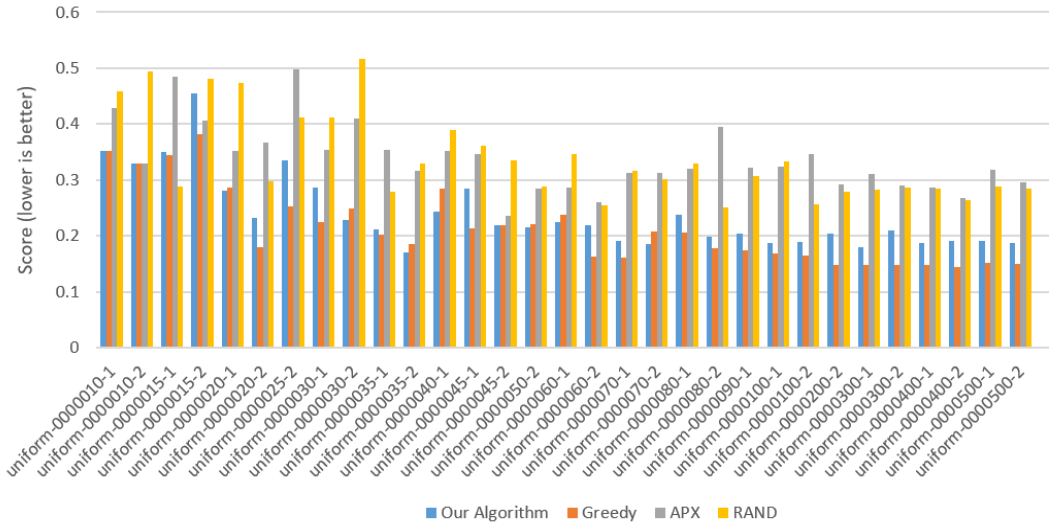


Fig. 9. The comparison results of our algorithm, APX, RAND and Greedy on uniform datasets.

Since our algorithm is based on clustering, it works better than other algorithms on datasets with the collections of the separate parts, i.e., the points are not located uniformly over the plane. Fig. 10 shows an example of these datasets and the results of our algorithm and Greedy algorithm on it. Also, Fig. 11 illustrates the efficiency of our algorithm compared with APX, RAND and Greedy on these datasets. In Fig. 11, the average scores are computed over 100 instances of non-uniform datasets of 10, 15, ..., 500 points.

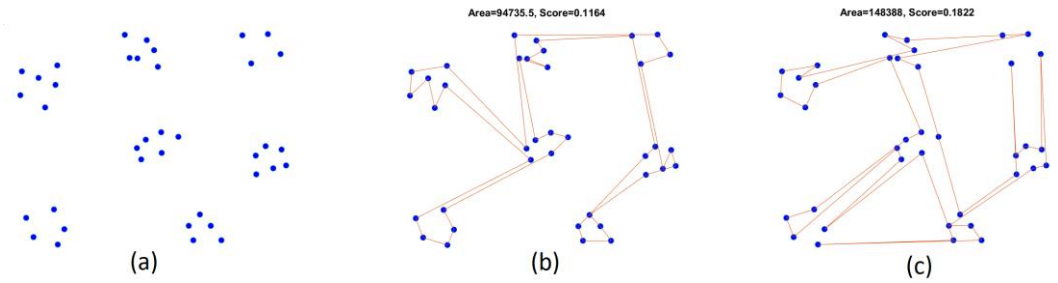


Fig. 10. (a) A non-uniform dataset of points, (b) The polygon computed by our algorithm, (c) The polygon computed by Greedy.

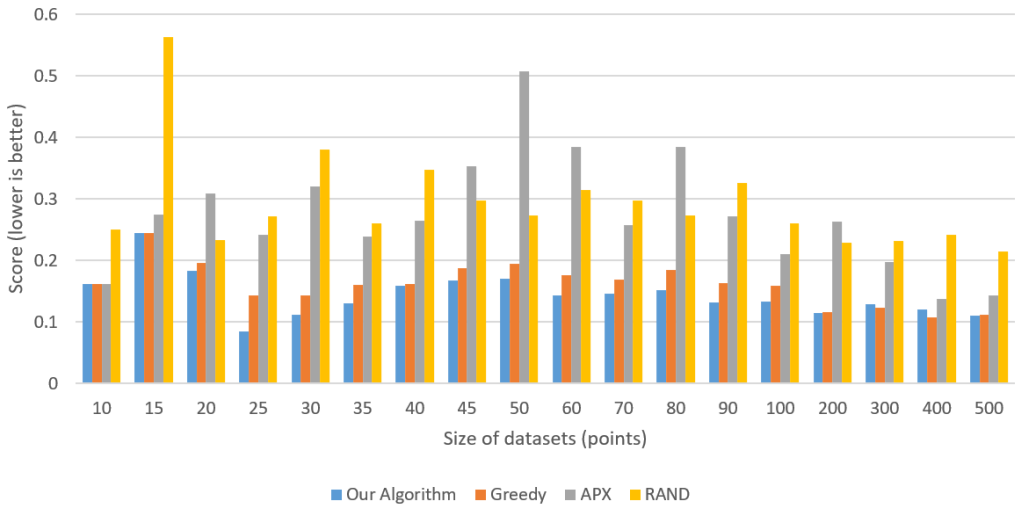


Fig. 11. The comparison results of our algorithm, APX, RAND and Greedy on non-uniform datasets.

Table 2 shows the results of our algorithm on datasets of TSPLIB, SPAETH and CG Challenge. In this table, U100, U200 and U500 are uniform-0000100-1, uniform-0000200-1 and uniform-0000500-1 instances of 2019 CG Challenge, respectively. Also, bier127, pr439 and ali535 are instances of TSPLIB and spaeth04, spaeth06 and spaeth08 are instances of SPAETH. The outputs of our algorithm on U500, spaeth06 and bier127 are depicted in Fig. 12.

*Table 2*

**Approximated minimum area polygon on instances of TSPLIB, SPAETH and 2019 CG Challenge**

| Dataset | Approximated minimum area | Execution time (Seconds) |
|---|---|---|
| U100 | 6246074 | 351.1284912 |
| U200 | 28521846 | 2880.690073 |
| U500 | 168670866 | 33502.14948 |
| d198 | 23297.26 | 2760.576258 |
| ch150 | 96407.25203 | 957.4282641 |
| bier127 | 31123728 | 444.6934746 |
| spaeth04 | 387.5 | 261.4493785 |
| spaeth06 | 148 | 111.4929528 |
| spaeth08 | 379 | 227.0705052 |

**Remark 3.1.** We run our algorithm for the different values of the fixed cluster size. Fig. 13 and Fig. 14 show the average score obtained by our algorithm and the execution time of our algorithm over the datasets of 10, 20, 50, 80 and 100 points for the cluster size of 4, 5, 6, 7 and 8 points, respectively.

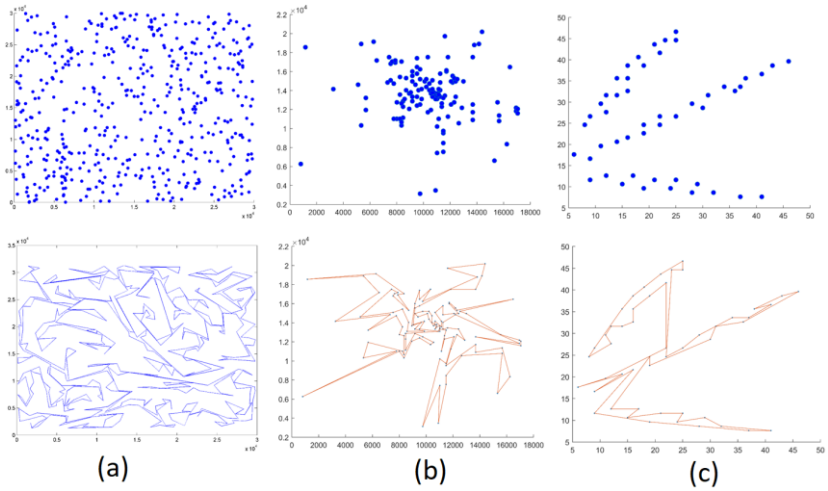


Fig. 12. The computed polygon by our algorithm on (a) instance uniform-0000500-1 of 2019 CG Challenge, (b) spaeth06 and (c) bier127.

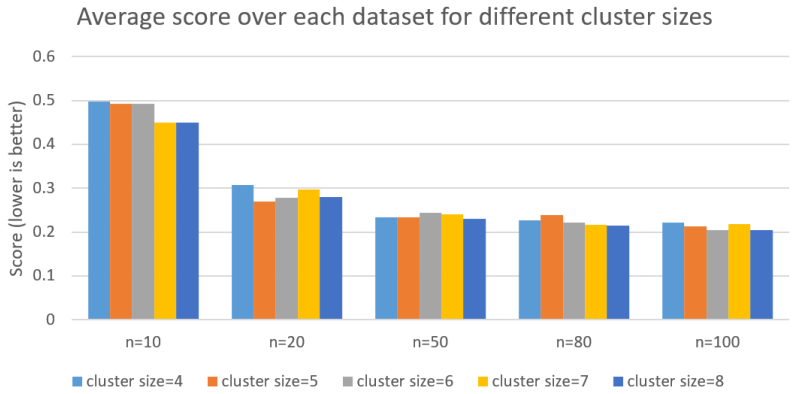


Fig. 13. The average score obtained by our algorithm over the datasets of 10, 20, 50, 80 and 100 points for the different cluster sizes.

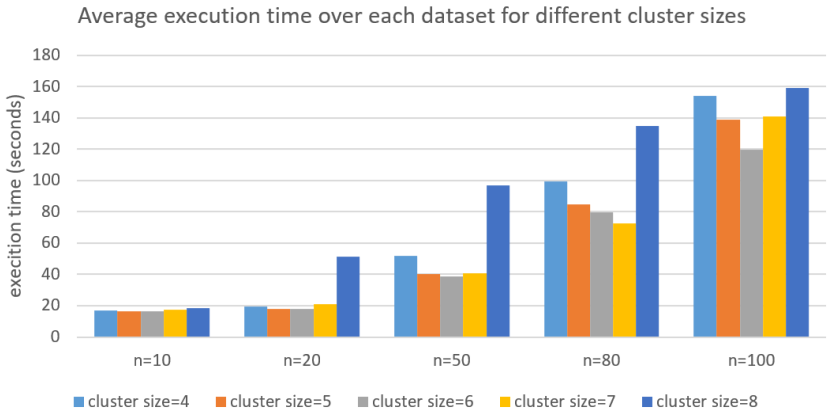


Fig. 14. The average execution time of our algorithm over the datasets of 10, 20, 50, 80 and 100 points for the different cluster sizes.

## 4. Conclusions and future work

In this paper, we presented a parallel approximation algorithm for minimum area polygonization based on clustering. On a set *S* of points, we clustered *S* to meaningful parts and split the large parts to small enough subsets. Parallelly, minimum area polygon is computed on each final subsets by exact algorithm. Finally, the minimum area polygons are merged together to construct an approximated polygon with minimum area on *S*. Based on our experimental results, our algorithm is shown to be more efficient than previous studies on non-uniform datasets. As a future work, we are going to use this algorithm to solve maximum area polygonization and minimum and maximum perimeter polygonization. Also, the constraints on the angles, area and perimeter can be added to the considered problem. As another future work, the algorithm can be extended to work with the points on the higher dimensions.121

**Acknowledgements**

The research of the first author is partially supported by the University of Kashan under grant number 991449/2.

## R E F E R E N C E S

[1]. S. P. Fekete, On simple polygonalizations with optimal area, Discrete & Computational Geometry 23 (1) (2000) 73–110.

[2]. S. P. Fekete, W. R. Pulleyblank, Area optimization of simple polygons, in: Proceedings of the ninth annual symposium on Computational geometry, ACM, 1993, pp. 173–182.

[3]. S. P. Fekete, Geometry and the travelling salesman problem., University of Waterloo, 1992.

[4]. S. Asaeedi, F. Didehvar, A. Mohades, α-concave hull, a generalization of convex hull, Theoretical Computer Science 702 (2017) 48–59.

[5]. Y. Bartal, L.-A. Gottlieb, R. Krauthgamer, The traveling salesman problem: low dimensionality implies a polynomial time approximation scheme, SIAM Journal on Computing 45 (4) (2016) 1563–1581.

[6]. Y. Bartal, L.-A. Gottlieb, A linear time approximation scheme for euclidean tsp, in: 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, IEEE, 2013, pp. 698–706.

[7]. V. Traub, Approximation algorithms for traveling salesman problems.

[8]. V. Traub, J. Vygen, An improved approximation algorithm for atsp, in: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, 2020, pp. 1–13.

[9]. W. Kongkaew, J. Pichitlamken, A survey of approximate methods for the traveling salesman problem, Kasetsart Engineering Journal 27 (89) (2014) 79–87.

[10]. C. Qi, An ant colony system hybridized with randomized algorithm for tsp, in: Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007), Vol. 3, IEEE, 2007, pp. 461–465.

[11]. M. T. Taranilla, E. O. Gagliardi, G. Hern´andez Pe˜nalver, Approaching minimum area polygonization.

[12]. L. Crombez, G. D. da Fonseca, Y. Gerard, Greedy and local search heuristics to build area-optimal polygons, arXiv preprint arXiv:2106.14728.

[13]. J. Peethambaran, A. D. Parakkat, R. Muthuganapathy, An empirical study on randomized optimal area polygonization of planar point sets, Journal of Experimental Algorithmics (JEA) 21 (1) (2016) 1–10.
[14]. V. Muravitskiy, V. Tereshchenko, Generating a simple polygonalizations, in: 2011 15th International Conference on Information Visualisation, IEEE, 2011, pp. 502–506.
[15]. M. Osiponok, V. Tereshchenko, The" divide and conquer" technique to solve the minimum area polygonalization problem, in: 2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT), IEEE, 2019, pp. 336–339.
[16]. J. Peethambaran, A. Dev Parakkat, R. Muthuganapathy, A randomized approach to volume constrained polyhedronization problem, Journal of Computing and Information Science in Engineering 15 (1) (2015) 011009.
[17]. S. P. Fekete, A. Haas, P. Keldenich, M. Perk, A. Schmidt, Computing area-optimal simple polygonalizations, Journal of Experimental Algorithmics.
[18]. Cg:shop 2019, https://cgshop.ibr.cs.tu-bs.de/competition/cg-shop-2019, accessed: 2021-08-18.
[19]. Demaine, E.D.; Fekete, S.P.; Keldenich, P.; Krupke, D.; Mitchell, J.S. Area-optimal simple polygonalizations: The CG challenge 2019. Journal of Experimental Algorithmics (JEA) 2022, 27, 1–12..
[20]. I. Lee, V. Estivill-Castro, Polygonization of point clusters through cluster boundary extraction for geographical data mining, in: Advances in Spatial Data Handling, Springer, 2002, pp. 27–40.
[21]. I. Lee, V. Estivill-Castro, Fast cluster polygonization and its applications in data-rich environments, Geoinformatica 10 (4) (2006) 399–422.
[22]. K. R. Zalik, B. ˇ Zalik, A sweep-line algorithm for spatial clustering, Advances in Engineering Software 40 (6) (2009) 445–451.
[23]. J. MacQueen, et al., Some methods for classification and analysis of multivariate observations, in: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, Vol. 1, Oakland, CA, USA, 1967, pp. 281–297.
[24]. A. Nourollah, M. Movahedinejad, A genetic based algorithm to generate random simple polygons using a new polygon merge algorithm, International Journal of Computer and Information Engineering 9 (1) (2015) 230–236.
[25]. Spaeth cluster analysis datasets, https://people.sc.fsu.edu/~jburkardt/datasets/spaeth/spaeth.html, accessed: 2021-08-18.
[26]. Tsplib, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95, accessed: 2021-08-18.
[27]. Crombez, L.; da Fonseca, G.D.; Gerard, Y. Greedy and Local Search Heuristics to Build Area-Optimal Polygons. ACM Journal of Experimental Algorithmics (JEA) 2022, 27, 1–11.