An Empirical Study on Randomized Optimal Area Polygonization of Planar Point Sets

Jiju Peethambaran, Amal Dev Parakkat, Ramanathan Muthuganapathy
Advanced Geometric Computing Lab, Department of Engineering Design, Indian Institute of Technology, Madras, India

While random polygon generation from a set of planar points has been widely investigated in the literature, very few work address the construction of simple polygon with minimum area (MINAP) or maximum area (MAXAP) from a set of fixed planar points. Currently, no deterministic algorithms are available to compute MINAP/MAXAP as the problems have been shown to be NP-complete. In this paper, we present a greedy heuristic for computing approximate MINAP of any given planar point set using the technique of randomized incremental construction. For a given point set of n points, the proposed algorithm takes $O(n^2 \log n)$ time and $O(n)$ space. It is rather simplistic in nature and hence very easy for implementation and maintenance. We report on various experimental study on the behavior of randomized heuristic on different point set instances. Test data have been taken from SPAETH cluster data base and TSPLIB library. Experimental results indicate that the proposed algorithm outperforms its counterparts for generating a tighter upper bound on the optimal minimum area polygon for large sized point sets.

Categories and Subject Descriptors: C.2.2 [**Computational Geometry**]: Randomized Algorithms

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Randomized algorithm, Minimal area polygon, Maximal area polygon, Incremental algorithm, Convex n-gons

1. INTRODUCTION

Polygonization of a planar point set S, is a way in which all the points in S can be connected to form a polygon [Denee 1988]. Combinatorial optimization problems such as area and perimeter optimizations of polygonizations are of particular interest to researchers due to its applications in pattern recognition and image processing [Fekete 2000; Denee 1988]. In [Peethambaran et al. 2015], the authors point out that the three dimensional versions of area optimization, i.e. minimum/maximum volume simple polyhedronization have the potentials to play a significant role in futuristic applications such as 4D printing and surface lofting. The area optimization problem referred to as minimum area polygonization (MINAP) asks for simple polygon with a given set of points for which the enclosed area attains the minimum [Fekete 1992]. Perimeter optimization of polygonization, commonly known as geometric travelling salesman problem (TSP) focus on computing the minimum circumference polygon that passes through all the points (or vertices) in the given set. In combinatorial geometry, TSP problems have been extensively studied and several fast approximation algorithms have been proposed [Goyal 2010].

Even though, both TSP and MINAP problems are computationally hard, optimizing the enclosed area of a polygon is analyzed to be more difficult than the minimization of its perimeter. In [Boyce et al. 1985], the authors point out that "while a shorter perimeter implies that the vertices are well localized, small area does not indicate the proximity of the vertices". Fekete [Fekete 1992] substantiates this argument by considering a tour as a set of line segments and a polygonal region as a set of triangles. Vertices of a short edge are necessarily at a close distance but a smaller area triangle can have its vertices far apart as in the case of very thin triangle. Further difficulty arise from self intersections of polygons. While self intersections do not occur automatically in the case of polygon with minimum perimeter, it is algorithmically difficult to build up a simple polygon from triangles [Eppstein et al. 1992; Fekete 1992]. Depending on the spatial distributions, the number of polygonizations [Sharir et al. 2011],

vary drastically among different point sets of same size. For instance, it is found to be exponential in n for some point sets [Denee 1988], where as for convex point sets it is found to be $2n$. Finding the number of polygonizations is shown to be NP-hard [Muravitskiy and Tereshchenko 2011] and hence, for point sets of appreciable sizes, it is extremely difficult to find the optimal solution from the corresponding set of all feasible simple polygonizations.

Little seems to be studied about minimum area polygonizations of fixed points in the plane, although polygonizations, especially randomized polygonizations raised some interest [Auer and Held 1998; Zhu et al. 1996]. A prominent work in the area of randomized polygon generation can be found in [Auer and Held 1998]. A similar work on random generation of x-monotone polygons from a set of points is presented in [Zhu et al. 1996]. In the area optimization domain, a pioneering work by Fekete [Fekete 2000], establishes the NP-completeness of minimum weight polygon or maximum weight polygon of a vertex set, a result which leads to the NP-completeness proof of corresponding area optimization problems. Tereshchenko et al. [Muravitskiy and Tereshchenko 2011] propose a polynomial greedy approximation algorithm for computing the minimal area simple polygon of a planar point set. The proposed greedy algorithm takes $O(n^4)$ time and $O(n)$ space. They show that a preliminary preprocessing of the set of points can further improve the time complexity of the approximation algorithm at the expense of increased memory usage. Their optimized greedy approximation algorithm takes $O(n^3)$ time and $O(n^2)$ space. An impossibility of a constant factor approximation algorithm for MINAP is also stated in [Muravitskiy and Tereshchenko 2011].

In [Taranilla et al. 2011], the authors suggest three different strategies (greedy-MAP, RS-MAP and ACO-MAP) to approximate the minimum area polygonization of planar point sets. The described heuristics employ either a locally optimal choice on the point selection (greedy-MAP) or a random search over the space of feasible solutions (RS-MAP) or ant colony optimization (ACO-MAP) strategy. Recently, the authors proposed randomized heuristics for polyhedronization of three dimensional point sets with minimum and maximum volumes [Peethambaran et al. 2015]. In this paper, we present simple, randomized and greedy algorithms for constructing optimal area (both minimum and maximum area) polygonizations of planar point sets of any size. Besides the intrinsic interest in obtaining the minimum area polygonization of a given set of points, such randomized heuristics for polygon generation can be used to generate large sized random geometric structures (polygonizations with possibly the optimum area) to test and evaluate other geometric algorithms working on polygonal inputs.

The rest of this paper is organized as follows. In Section 2, we explain the incremental construction of minimal area simple polygon and maximal area simple polygon with running time analysis. Section 3 analysis the correctness of the proposed algorithm and put forwards a few hypotheses about the optimal area polygonization. Computational results and implementation details are presented in Section 4. We compare our randomized MINAP algorithm with the existing MINAP algorithms in Section 5. We conclude the paper in Section 6.

2. INCREMENTAL CONSTRUCTION

We start with an overall idea of MINAP construction. Let S=$\{p^0,p^1,p^2,...,p^{n-1}\}$ denotes a set of n points and each point p^i is represented by its x and y coordinates. P_i is an ordered set of points on the polygon in counter clockwise (CCW) direction after the ith execution phase (notionally, points in P_i are used similar to array indices) and E_i is the edge set after the i^{th} execution phase. We use **RAND_MAXAP** and **RAND_MINAP** to refer to randomized incremental algorithm for MAXAP and MINAP, respectively.

Simple polygon generated using RAND_MINAP and RAND_MAXAP are referred to as **randomized MINAP** and **randomized MAXAP** respectively.

2.1. MINAP Construction

The first step of our approach is to select three random points from the point set S and construct a triangle, $\triangle_2$=P_2=$\{p_0,p_1,p_2\}$ (please note that the subscripts are used to denote the random generation of points from the set S), which represents the initial polygon, with the edge set E_2=$\{e_0,e_1,e_2\}$. In the i^{th} iteration, *hull_size* is defined as the number of points on the previous polygon (P_{i-1}). An edge e_i is represented by $\overline{p_i p_{i+1 mod(hull\_size)}}$. Once the initial triangle has been constructed, the algorithm runs for $n-3$ iterations, where each iteration i, consists of the following steps.

(1) Select a point p_i uniformly at random from the set **S-P**$_{i-1}$.
(2) Check whether the point lies interior/exterior to the previous polygon $\mathbf{P}_{i-1}$.
(3) If the point lies interior to $\mathbf{P}_{i-1}$, then find the largest area non-intersecting triangle $\triangle_i$ that the point p_i makes with the edges of $\mathbf{P}_{i-1}$. Let us denote the triangle as $\triangle_i$=$\{p_q,p_i,p_r\}$ where p_q, p_r∈$\mathbf{P}_{i-1}$. Remove $\triangle_i$ from the polygon $\mathbf{P}_{i-1}$ by adding the edges and the point p_i at the appropriate position. Update edge set and vertex set as $\mathbf{E}_i$=$\mathbf{E}_{i-1}$∪$\{p_q,p_i\}$∪$\{p_i,p_r\}$-$\{p_q,p_r\}$ and $\mathbf{P}_i$=$\mathbf{P}_{i-1}$∪p_i.
(4) If the point lies in the exterior/on of the $\mathbf{P}_{i-1}$, then find the smallest area non-intersecting triangle $\triangle_i$ that the point p_i makes with the edges of $\mathbf{P}_{i-1}$. Let us denote the triangle as $\triangle_i$=$\{p_q,p_i,p_r\}$ p_q, p_r∈$\mathbf{P}_{i-1}$. Add $\triangle_i$ to the polygon $\mathbf{P}_{i-1}$ by adding the edges and the point p_i at the appropriate position. Update edge set and vertex set as $\mathbf{E}_i$=$\mathbf{E}_{i-1}$∪$\{p_q,p_i\}$∪$\{p_i,p_r\}$-$\{p_q,p_r\}$ and $\mathbf{P}_i$=$\mathbf{P}_{i-1}$∪p_i.

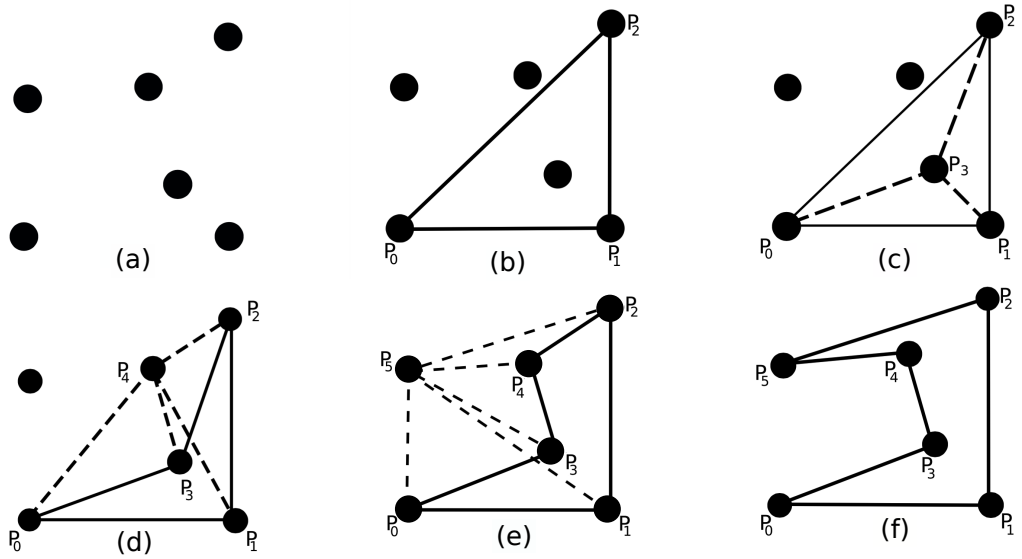


Fig. 1. Incremental construction of minimal area simple polygon.(a)Point set (b) The initial random triangle (c)-(d) Incremental construction (e) Final polygon

The various steps in generating MINAP of a point set are depicted in the Figure 1. A set of six points is given in the Figure 1(a). An initial triangle P_2=$\{p_0,p_1,p_2\}$ is formed by selecting 3 points uniformly at random from the given point set as shown in Figure 1(b). This triangle becomes the initial polygon. In the next iteration, the

selected point (p_3) lies to the interior of the previous polygon P_2. There exist three non-intersecting triangles formed by p_3 with the edges of P_2 as shown by the dashed and dark lines in the Figure 1(c). The largest area triangle, $\triangle p_0,p_3,p_2$ is excluded from P_2 to form the next polygon P_3. The chosen point(p_4) in the next iteration, lies to the exterior of P_3. The point p_4 produces four triangles with the previous polygonal edges, which comprises of two non-intersecting and two intersecting triangles as shown in the Figure 1(d). The smallest area triangle from the non-intersecting triangles, $\triangle p_4,p_3,p_2$ is added to form the current polygon P_4. A similar procedure is followed in Figure 1(e) which lead to the MINAP generation of the given point set in Figure 1(f).

In each iteration, one of the remaining points in S is added to the current polygon which leads to the construction of an approximate minimal area polygon at the end. A simple polygon is generated because in each iteration, a triangle which does not have its edges intersected with the edges of the previous polygon (referred to as **valid triangle**) is either added to the polygon or removed from the polygon. A valid triangle is found out by employing a classical line-sweeping algorithm [Berg et al. 2008] described in Section 2.2.

2.2. Intersection Checking through Line-sweep Technique

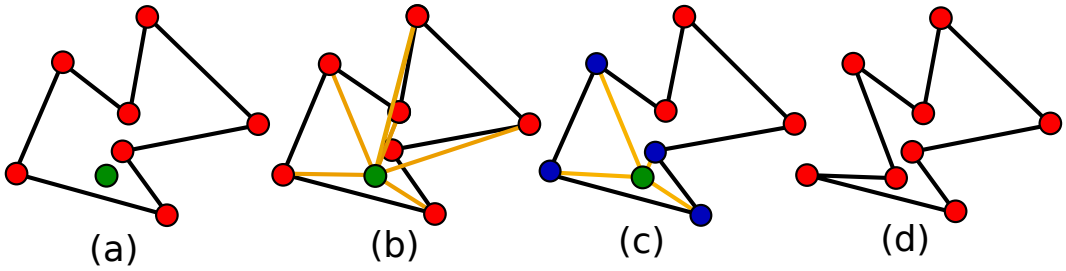


Fig. 2. Valid triangle removal through Line sweeping and walking techniques.

Candidature of a triangle to be added or removed from an intermediate polygon P_{i-1} can be determined in $O(n \log n)$ time by using the classical line sweeping algorithm for line-line intersection checking [Berg et al. 2008]. The idea can be described as follows: Assume that new point under consideration, p_i is interior to P_{i-1} as shown in Figure 2 (a). Let E_{i-1} be the set of all edges of the polygon P_{i-1} (black edges in Figure 2 (b)) and L be the set of all line segments formed between p_i and vertices of P_{i-1} (orange colored edges in Figure 2 (b)). The fact that P_{i-1} is a simple polygon ensures that E_{i-1} does not have any intersecting edges except the intersections at the end points which is considered as a degenerate case. Similarly, all the line segments in L share a common end vertex p_i and therefore, no mutual line segment intersections occur in L (assuming general position). Hence, the algorithm uses a line-sweep method to find out the edges from L, that intersect with the edges in E_{i-1} (refer to Figure 2 (b)). All the intersecting edges are discarded to get the set of edges (Figure 2 (c)) which possibly form **valid** triangles with the previous polygonal edges.

The vertices of P_{i-1} which induce non-intersecting edges with p_i are marked with a label (represented using blue colored vertices in Figure 2 (c)). An $O(n)$ walk over P_{i-1} is performed to identify the largest area (or smallest are) valid triangle which can be constructed with p_i as one of its vertices. While walking, if the adjacent vertices $p_q \& p_r$ of P_{i-1} are marked, then it implies the existence of a valid triangle, $\triangle p_i p_q p_r$. Further, to find out the largest (or smallest) area triangle, the triangle area information is also updated during this walk. Once the required triangle is identified, it is removed (or

ALGORITHM 1: RAND_MINAP(S,n)

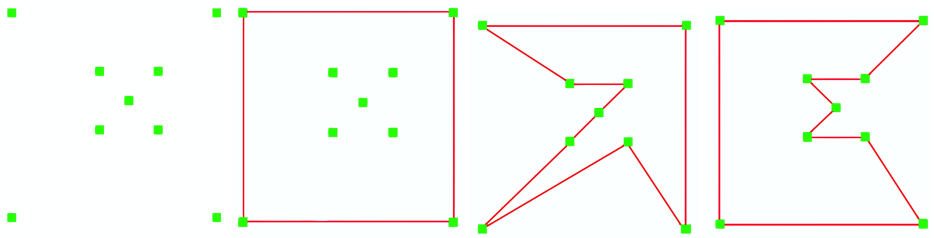
Input: A set of n planar points, S={p^0,p^1..p^{n-1}}.
Output: Minimal Area Simple Polygon of S.
Randomly pick three points from the set S. Label it as p_0,p_1 and p_2;
Initialize P_2={p_0,p_1,p_2}, edge set E_2={(p_0,p_1),(p_1,p_2),(p_2,p_0)} and hull_size=3;
for *i:=3 to n-1 do* **do**
  select a point p_i uniformly at random from $S \setminus P$;
  Construct L consisting of all the edges formed by p_i with the vertices of P_{i-1};
  Apply line sweeping on $E_{i-1} \cup L$ and update L with non-intersecting edges;
  if $p_i \in$ *interior of* P_{i-1} **then**
    Walk over P_{i-1} and remove the largest area $\triangle p_q p_i p_r$ from P_{i-1};
  end
  else
    Walk over P_{i-1} and add the smallest area $\triangle p_q p_i p_r$ to P_{i-1};
  end
  Empty the list L;
  set hull_size=hull_size+1;
  update the set P_i={p_0,p_1,p_2..,p_q,p_i,p_r...$p_{(hull\_size-1)}$} and edge set
  E_i={(p_0,p_1),(p_1,p_2),(p_2,p_3),..,(p_q,p_i),(p_i,p_r)};
end
return P_{n-1} and E_{n-1};

added) to P_{i-1} to construct the current polygon, P_i (Figure 2 (d) shows the output of MINAP). A similar procedure can be applied for p_i, if it lies exterior to P_{i-1}.

The pseudo code of the overall approach is presented in Algorithm 1.

2.3. MAXAP Construction

The maximal area simple polygon construction uses an approach similar to the method described for MINAP construction in the Section 2.1. The only difference occurs when excluding or including the triangles in each iteration. Instead of excluding the largest area non-intersecting triangle if the point lies inside the current polygon, MAXAP construction excludes the smallest area non-intersecting triangle. If the point is exterior to the current polygon, the algorithm includes the largest area non-intersecting triangle to the current polygon. A modified version of the pseudo code (RAND_MINAP) given in the Algorithm 1 can be used for the MAXAP construction as well.



(a) Input (b) 48 sq. units (c) 32 sq. units (d) 37 sq. units

Fig. 3. Maximum area polygonizations generated by different algorithms. (a) Point set (b) Convex hull (c) Result by [Fekete 2000] and (d) Our result. Area of each polygon is mentioned along with the sub figure.

In [Fekete 2000], Fekete put forward an $O(n \log n)$ algorithm (APPROX_MAXAP) to obtain maximum area polygonization. His construction employs a slope-based sorting of input points with respect to a point on the convex hull and then joining the points in the sorted order. A simple strategy based on Euclidean distance is also employed to break the ties between points with the same slope. If the area of the resulting polygon P is bigger than half the area of convex hull, the algorithm returns P, otherwise a complementary simple polygon with respect to the corresponding convex hull is returned. This simple construction guarantees to generate a simple polygon, whose area is larger than half the area of the corresponding convex hull. The area upper bound for any solution to MAXAP is the area of the corresponding convex hull and therefore, the APPROX_MAXAP, in a way, represents a $\frac{1}{2}$-approximation algorithm yielding fast approximation to MAXAP. However, for certain point sets shown in Figure 3, our algorithm returned better solution as compared to APPROX_MAXAP. Our method has the flexibility of choosing the best candidate from the feasible solution space, of course, at the expense of several executions and running time. On the contrary, one of the main advantage as well as the disadvantage of APPROX_MAXAP is its adherence to a fixed sequence of points for computing the required polygon. While the fixed sequence helps in generating the solution quickly, it also limits the algorithms ability to further explore and find out the best candidate from the solution space.

2.4. Complexity

Lemma 2.1 establishes the time complexity of the RAND_MAXAP and RAND_MINAP algorithms.

LEMMA 2.1. *RAND_MINAP or RAND_MAXAP runs in O($n^2 \log n$).*

PROOF. This is an obvious and straight forward statement. The loop for randomly picking the points from the point set runs from 3 to $n-1$, making a total of $n-3$ times. In each iteration of this loop, an $O(n \log n)$ line sweeping is performed to create the list of non-intersecting edges that the selected point forms with the vertices of P_{i-1}. Walking over the polygon and picking the largest (or smallest) area triangle is achieved in $O(n)$. So the overall asymptotical running time is $O(n^2 \log n + n^2) \approx O(n^2 \log n)$. □

One can argue that $O(n^2 \log n)$ is the time incurred due to a single run of $RAND\_MINAP$ and therefore, the worst case time required for the overall strategy, considering all the possible sequences of a point set of size n is $O((n-1)!(n^2 \log n))$. This complexity is astronomically huge. However, the attractive feature of our algorithm is that it chooses the sequences randomly and hence it is very likely that the sequence leading to the optimal solution may be chosen in the very first run of $RAND\_MINAP$. This implies that we get a solution of even large sized point set in $O(n^2 \log n)$, which is a clear advantage especially when considering that MINAP problem has only a few approximate algorithms and brute force technique. On the space complexity, the algorithm maintains three lists P_i, E_i and L, each having a size of $O(n)$ and hence takes no more than $O(n)$ space, overall.

3. CORRECTNESS ANALYSIS

3.1. Existence of Polygonization

The authors [Peethambaran et al. 2015], have employed concepts such as trap regions, minimal set of blocking points and local rearrangements to theoretically analyze the existence of polyhedronization in 3D. We use the term ***valid sequence*** to refer to an ordering of input points which when used by $RAND\_MINAP$ leads to a simple polygonization. In two dimensions, a ***trap region,*** $T(P)$ of a polygon P, is a region in the Euclidean plane from where no edges of P are completely visible (please refer to the

grey-colored region in Figure 4(a))). In an iteration of $RAND\_MINAP()$, if the selected point belongs to any of the trap regions of previous polygon (as shown in Figure 4(a)), it forms only intersecting triangles with the previous polygon and therefore, the algorithm gets stuck. Current implementation deals with such points (lying in trap regions) by allowing the re-execution of the algorithm, right from the start, possibly with a different ordering of the input points. However, this implementation relies on the hypothesis that at least one valid sequence exists for a planar point set and therefore, one of its executions may encounter this sequence, thus producing a simple polygonization. This clearly calls for an analysis on the existence of valid sequences for planar point sets.

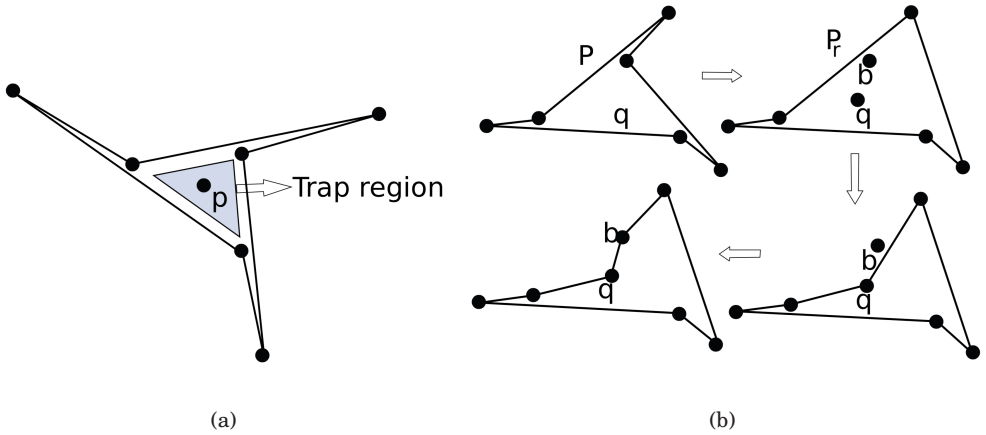


Fig. 4. (a) An example of trap region (grey colored region) (b) Illustration of local rearrangement. In Figure 4(b), P is the previous polygon, q is the trapped point and $\{b\}$ represents the minimal set of blocking points.

To resolve a trap region that may arise in any of the iterations of $RAND\_MINAP$, we employ a local re-arrangement technique on blocking vertices of previous polygon, P. Vertices and edges of P that blocks the visibility of a point $q \subseteq T(P)$ is referred to as blocking vertices and blocking edges, respectively. A minimal set of blocking vertices is the set of minimum number of vertices whose removal causes at least one edge of P to be completely visible from q. Intuitively, a trap region can be resolved by a local rearrangement of current polygonal edges. The process first, detaches the minimal set of blocking points from P and then attaches the trapped input point, followed by the detached points keeping the area constraint invariant. The process is illustrated in Figure 4(b). Using the concepts of local rearrangement and trap regions, we outline our arguments in favor of $RAND\_MINAP$ in polygonizing a planar point set (refer to Proposition 3.1).

PROPOSITION 3.1. *For any finite set of planar points S where $\mid S \mid$=n, there exists an ordering of points, which when subjected to the rules of RAND_MINAP algorithm generates a polygonization of S.*

PROOF. Using induction on $\mid S \mid = n$, we try to show at least one valid sequence for each of the cases. For $n = 3$, it is trivial to find that any permutation of points leads to a polygonization. For $n = 4$, the valid sequence consists of points on the convex hull of S followed by the interior point.

We hypothesise that claim is true for all sets of size $k < n$. Assume that a point $q \in S$ lies in $T(P_{n-1})$ where P_{n-1} is a polygon of $n-1$ points of S. Let P_{n-1} be decomposed into 2 sub sets of points, $\{S_r, B\}$ where $B = \{b_1, b_2, ...b_m\}$, is the minimal set of blocking points and $S_r = P_{n-1} \setminus B$. Using local rearrangements, the construction proceeds by first polygonizing S_r to get P_r. Due to hypothesis, the points in S_r where $\mid S_r \mid < n$, has a valid ordering which leads to polygonization. Then, q is attached to P_r followed by attaching points in B keeping the volume constraint as invariant. Hence, for such a configuration, an ordering which leads to the polygonization of S_r, followed by q, followed by $b_1, b_2, ...b_m$ leads to a simple polygonization. During the construction if trap regions arise again due to any point from B, a similar local rearrangement can be done. Since, the 2D local rearrangements are possible as illustrated in Figure 4(b), we conclude that there exists at least one valid sequence for any point set. □

3.2. Optimal Area Polygonization of Convex Point Sets

A point set S is said to be *convex point set*, if all the points in S are co-located on the convex hull of S. In this section, we show that RAND_MINAP algorithm always generates the optimal result for convex point sets. The optimal result is the convex hull as it is the only simple polygon induced by a convex point set. Let P_{i-1}, be the current polygon and p_i be the point selected in the i^{th} iteration of RAND_MINAP algorithm, then a *potential triangle* can be either of the following.

— a non-intersecting triangle Δ_i, which is a potential candidate for removal from the current polygon if the point p_i lies in the interior of P_{i-1} or
— a non-intersecting triangle Δ_i, which is a potential candidate for addition to the current polygon if the point p_i lies to the exterior of P_{i-1}.

LEMMA 3.2. *Let S be a convex point set, then each iteration of $RAND\_MINAP$ on S will have only one potential triangle.*

PROOF. In each iteration i of $RAND\_MINAP$ algorithm, the point set is divided into i sub sets where each sub set, sub_set_k has only one unique edge e_k (e_k belongs to the current polygon P_{i-1}) with which the points from sub_set_k may form a potential triangle. We term this edge as *friend edge*. In the Figure 5, the polygon P_2 divides the point set S into sub_set_1 with friend edge e_1, sub_set_2 with friend edge e_2 and sub_set_3 with friend edge e_3. If a new point is selected, the point can form only one potential triangle as it lies in any one of the *sub_sets* and the *sub_set* has a unique friend edge *e* associated with it. This scenario is attributed by the fact that all points lie on the convex positions on the hull and a triangle consisting of a point from one *sub_set* and non friend edges can be made only at the expense of an intersection with the corresponding friend edge. This makes that triangle a non potential one. □

It is to be noted that all the potential triangles have its vertices on the convex hull of S. Since all the points lie on the convex hull, while building the MINAP, no point is trapped inside the current polygon.

LEMMA 3.3. *RAND_MINAP always returns an optimal solution for convex point sets.*

PROOF. Let S be a convex point set. The claim is easy to establish by contradiction. We assume that $RAND\_MINAP$ constructs a non-convex simple polygon, which is obviously a non-optimal solution for S. However, the vertices of potential triangle, which is the only non-intersecting triangle available in an iteration of $RAND\_MINAP$ (refer to Lemma 3.2), lies on the convex hull of S. Consequently, a non-convex polygon is never generated for S by $RAND\_MINAP$ which contradicts our assumption. Hence,

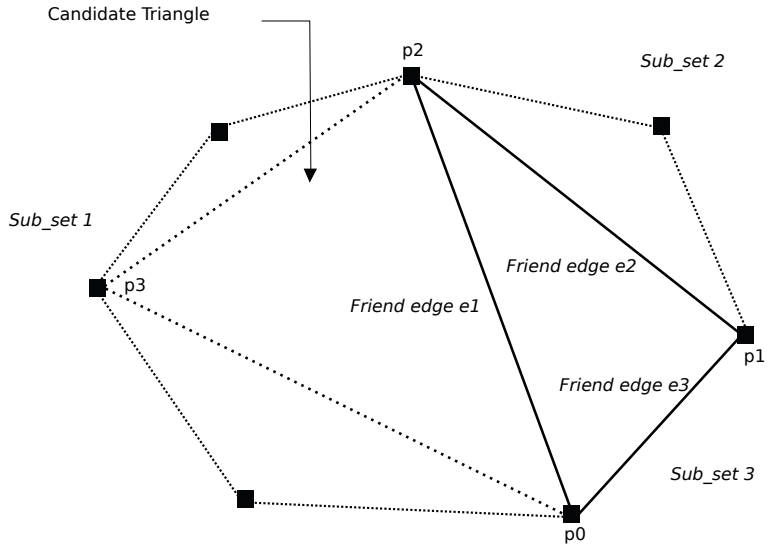


Fig. 5. Potential triangle in an iteration of $RAND\_MINAP$ for a convex point set.

$RAND\_MINAP$ always generates convex hull for convex point sets, which represents the optimal solution. □

3.3. ξ_{error} Hypothesis

Lemma 3.4 shows the existence of a permuted sequence leading to an optimal MINAP when used by the RAND_MINAP algorithm for point sets of size n (when all points lie at convex positions and all but one points lie at convex positions). This indicates that a permuted sequence producing an optimal polygon exists for any point sets mentioned in the above category. A poit set S is referred to as *convex points* if all the points in S are co-located on the convex hull of S. The sequence producing optimal MINAP is referred to as *optimal sequence*.

LEMMA 3.4. *Given a set of n points in the general position, all of which are at convex positions or all but one point are at convex positions, there exists at least one permuted sequence of points which when subjected to the rules of RAND_MINAP algorithm will generate an optimal MINAP of S*

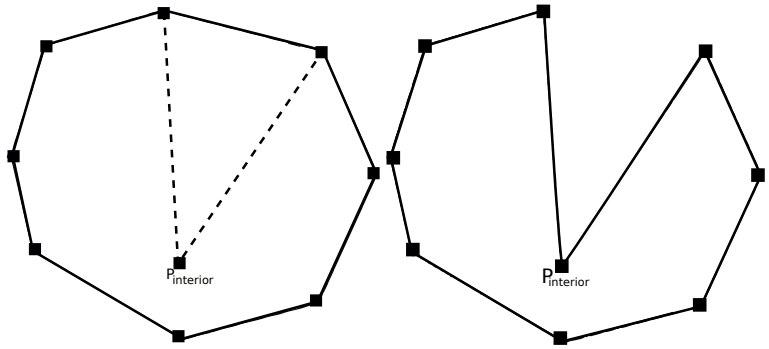
PROOF.

(1) **Base case, n=4**: There arise two cases as the following.
   (a) **case 1**:All points lie at convex position
   When all points lie at convex position making up a convex quadrilateral, any permuted sequence will generate the optimal MINAP as stated in the Lemma 3.3
   (b) **case 2**:One point lies interior to the other three convex points
   In this case, a permuted sequence consists of the points on the convex positions followed by the point at non-convex position. This will clearly produce an optimal MINAP.
(2) When point set size is n
   (a) **case 1**:All points lie at convex position
   This is again a direct consequence of Lemma 3.3. When all points lie at convex

position making up a convex n-gon, any permuted sequence will generate the optimal MINAP as stated in the Lemma 3.3.

(b) **case 2**:One point lies interior to the other $n-1$ convex points
When only one point lies to the interior of the other points (convex (n-1)-gon), any permuted sequence of points of convex (n-1)-gon followed by the point lying interior will generate the optimal MINAP as shown in the Figure 6.

□



(a) Point set 9 (b) Optimal MINAP

Fig. 6. One point lying interior to a 8-gon and the corresponding optimal MINAP. The permuted sequence consists of all points on the convex 8-gon followed by the interior point

Given a set of n points, there exists a convex hull of k points where k ranges from 3 to n. We have examined for the existence of an optimal permuted sequence for point sets of size n when the convex hull size, k=n or k=$n-1$ in the Lemma 3.4. To explore the remaining cases, when k=3 to $n-2$, we use the notation ξ_{error}, which is defined as the difference between the area of the convex hull of the given point set and area of the MINAP generated by the RAND_MINAP. Intuitively, the largest ξ_{error} should produce the optimal MINAP.

ξ_{error} is computed as follows. Assume that the convex hull has been computed from the point set by the RAND_MINAP algorithm. When the selected point lies interior to the previous polygon, the area of excluded triangle is added to ξ_{error}. When the point lies exterior to the previous polygon, the area of the triangle included is subtracted from ξ_{error}. Finally we get the total ξ_{error}. Using ξ_{error} measure, we hypothesis that a sequence of points on the convex hull in any order followed by a sequence of interior points, which generates the largest ξ_{error} may generate the optimal MINAP. We illustrate this principle for a point set of size 5 in the Figure 7.

4. EXPERIMENTAL STUDY

In this section, a detailed experimental study on the behavior of the algorithms has been carried out. We conducted several experiments on different point sets to evaluate the quality of RAND_MINAP and RAND_MAXAP algorithms. All computations were performed on a machine having Intel Core i3-2330M processor with 2.20 GHz and 2GB RAM. 20 trials were used to generate MINAP or MAXAP of each of the above point sets. One trial consists of 100 execution which implies that, for a point set, the algorithm

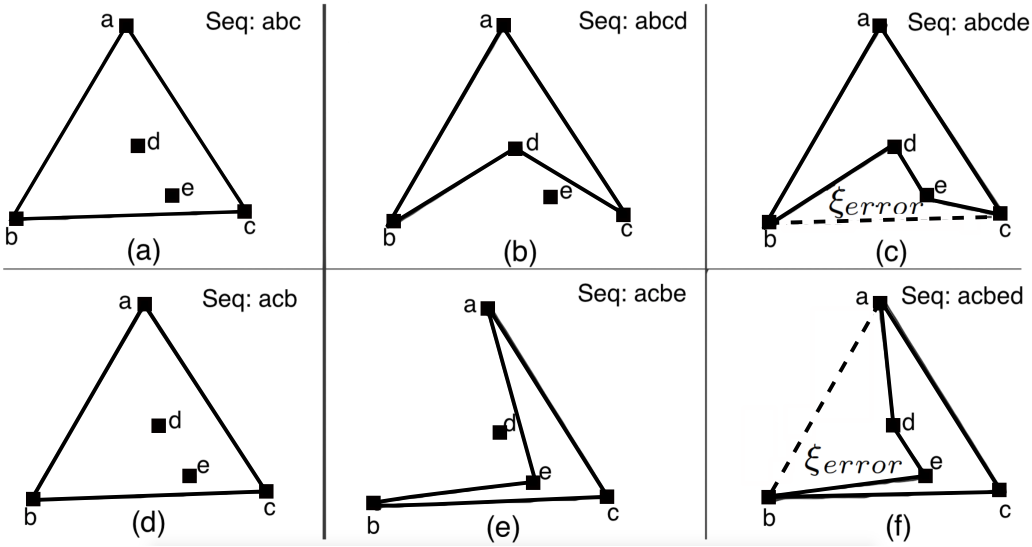


Fig. 7. Illustration of the relationship between optimal sequence and the largest ξ_{error} for set of 5 points. The sequence producing largest ξ_{error} results in optimal MINAP as shown in Figure 7(f) as compared to the one in Figure 7(c)

was run for 2000 times before making a conclusion on the lower bound/upper bound on the optimal areas.

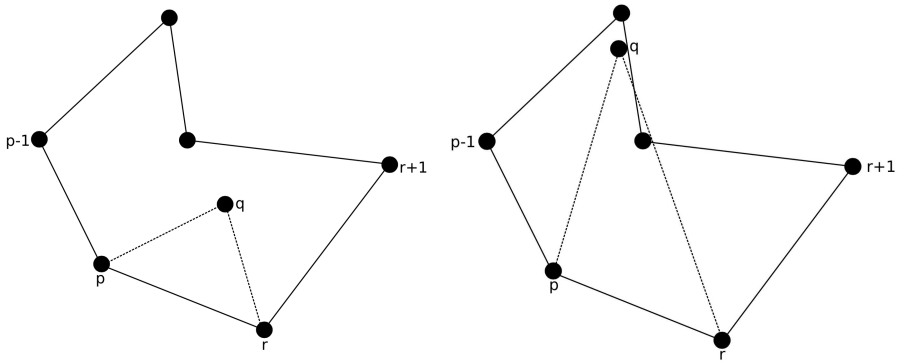
We generated random point sets of sizes 100, 150, 200, 250, 300, 400, 500 and 1000 for the experimental purpose. Some convex point sets (square(196), circle shape(70), Elliptical shape(70) and Octagonal shape(100)) were also generated. In our experiments, we were able to generate random MINAP and random MAXAP for point sets of appreciable sizes, which include point sets of sizes 500 and 1000.

4.1. Implementation

The algorithms RAND_MINAP and RAND_MAXAP are implemented using C++ and OpenGL in MS Visual Studio 2008. Random point selection is realized using the functions srand() and rand() available in the header file stdlib.h. The function srand() is used to initialize the pseudo-random number generator by passing the argument seed. System time is used as the seed to the srand() function. When ever a point is added to the polygon, algorithm mark it as *used* so that the point will be ignored when selecting a random point in the future iterations. The major subroutines of the program consists of point in polygon checking and intersection checking which are explained in Section 4.1.1.

4.1.1. Subroutines. The intersection between the edges is determined through classical line-sweeping algorithm [Berg et al. 2008]. Ray tracing [O'Rourke 1998] is used to check whether the selected point lies interior or exterior to the current polygon. Important subroutines used by algorithm 1 are the following.

— **Interior**(P_{i-1},p_i):This function returns TRUE if the point p_i lies inside or on the boundary of the current polygon P_{i-1}. Time complexity is $O(n)$.
— **Area**(p,q,r):The function area(p,q,r) computes and returns the area of the triangle formed by the points p,q and r in constant time. The function uses the expression for the area of the triangle as a function of its vertex coordinates as given by the formula: $A(T) = \frac{1}{2} \times (q_0 - p_0)(r_1 - p_1) - (r_0 - p_0)(q_1 - p_1)$ [O'Rourke 1998].



(a) Non-intersecting (b) Intersection

Fig. 8. Different types of intersections of the triangle with the previous polygon.

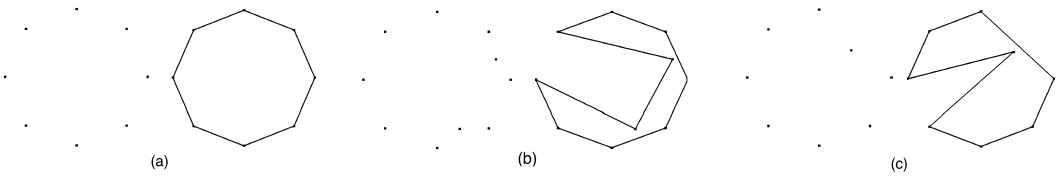


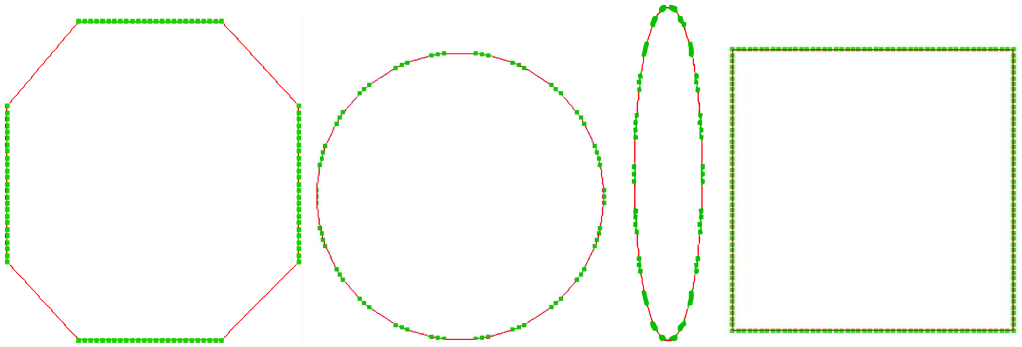
Fig. 9. Results of octagonal (regular polygon) point sets and its variants.

— **Line_sweep**(E_{i-1}, L):It updates the list L with the non-intersecting edges that p_i forms with the vertices of P_{i-1}. In the non-intersecting case, the segments $\overline{qp}$ and $\overline{qr}$ meet the vertices p and r respectively as shown in the Figure 8(a). When the segments $\overline{qp}$ and $\overline{qr}$ cuts any of the previous polygonal edges, then it is considered as an intersection. An example for intersection is shown in Figure 8(b).
— **Walk**(P_{i-1}, E_{i-1}, L): It picks up the largest/smallest area valid triangle by walking along the previous polygon in linear time.
— **Update**(P_{i-1},p_i): Update function inserts the selected point in the appropriate position in the ordered list of points. The point will be inserted between the two neighboring points with which p_i forms a valid minimal/maximal area triangle. The complexity of Update is $O(n)$.

4.2. Computational Results

4.2.1. Random Point Sets. Figure 9 show the minimum area polygonizations for octagonal point set (an example of a regular polygon) and the variant point sets induced from it. Additional points in the form of perturbance have been added to the octagonal point set in Figure 9 (b) and an octagonal vertex itself is slightly moved towards the interior to generate the point set of Figure 9 (c). Please note that, RAND_MINAP algorithm generates the optimal results for all the three point sets, for which intuitive optimal solutions are known to us.

Table I & II report the maximum and minimum areas obtained in each trial of the respective algorithm for various point sets. The last rows list the minimum and maximum areas along with the average, median and standard deviations of areas for 20



(a) Octagon (100 Points) (b) Circle (70 Points) (c) Ellipse (70 Points) (d) Square (196 Points)

Fig. 10. Shapes generated by RAND_MINAP for convex n-gonal point sets.

Table I. Minimum Area Table for Various Point Sets(in Sq. Units)

Trial No.	Pointset100.	Pointset250.	Pointset300.	Pointset500.	Pointset1000.
1	624.5	13624.5	2702.5	43890.0	267088.0
2	635.5	15406.0	2640.0	46655.0	275839.5
3	618.5	16212.0	2859.5	40126.0	279085.0
4	623.5	16483.5	2768.5	43317.5	259935.5
5	608.0	14437.5	2824.0	41005.0	270605.0
6	655.0	18923.5	2819.0	43874.5	265872.5
7	667.5	16145.5	2810.5	41001.5	277434.0
8	661.5	16261.5	2758.5	44932.0	274999.5
9	695.0	14981.0	2901.5	41185.5	262591.0
10	624.5	15683.0	2677.0	41603.5	270501.0
11	644.5	15135.5	2940.0	43811.0	270207.5
12	647.0	15928.5	2769.0	41457.0	280472.0
13	604.5	15145.5	2507.0	41286.0	286047.0
14	635.0	17966.0	2585.5	43236.0	274134.0
15	771.0	14965.5	2648.5	42123.5	274364.0
16	618.5	16273.0	3059.5	41352.0	285153.5
17	617.0	16361.0	2573.5	42000.5	271355.5
18	594.0	16639.0	2809.5	43371.5	276404.0
19	630.5	15269.0	2808.0	43555.0	279136.0
20	629.0	14884.5	2803.0	44333.5	265091.5
Area $\leq$	**594.0**	**13624.5**	**2507.0**	**40126.**	**259935.5**
Average	**640.225**	**15836.275**	**2763.225**	**42705.825**	**273315.8**
Median	**629.75**	**15805.75**	**2786.0**	**42679.75**	**274249.0**
Std. deviation	**38.771**	**1186.879**	**132.778**	**1642.119**	**7043.885**

trials. Recall that one trial consists of 100 executions of RAND_MINAP algorithm. So, the algorithm was run for 2000 times before taking the the bound value for the optimal area for each of the point sets. Best area (minimum) value out of the 100 values has been presented for each trial in the table. Our bound value on the optimal minimal area for each point set is listed in the last row of the Table I.

4.2.2. SPAETH Cluster Data. We experimented with the point set data taken from SPAETH cluster analysis database [Spaeth 2014]. Each input consists of differ-

Table II. Maximum Area Table for Various Point Sets (in Sq. Units)

Trial No.	Pointset100.	Pointset250.	Pointset300.	Pointset500.	Pointset1000.
1	1697.5	43316.5	7208.0	111640.5	799223.0
2	1700.0	45230.5	6982.5	115282.0	724297.0
3	1677.5	42431.5	7300.5	116491.0	867238.5
4	1654.5	42866.5	7386.0	116191.5	717051.5
5	1712.5	43479.5	7098.5	112744.0	756729.5
6	1746.5	43267.5	7311.0	112834.0	713686.0
7	1731.5	44896.0	7088.0	116306.0	739367.0
8	1620.0	43626.5	7243.0	116415.0	715490.0
9	1705.5	44269.0	6924.0	115242.0	733364.0
10	1604.5	44789.5	7126.0	113844.0	714224.0
11	1767.5	44415.5	7255.5	113075.0	728068.0
12	1684.5	43757.5	7219.	114297.5	723059.0
13	1619.0	42907.5	7066.0	114219.0	746503.5
14	1638.5	43468.0	7058.0	111989.5	727794.0
15	1631.5	43406.5	7186.	113897.0	728596.0
16	1692.5	45086.0	7204.5	112821.0	721769.0
17	1680.5	44259.5	7001.0	113459.5	753409.0
18	1733.5	42628.0	7176.0	117897.5	969552.0
19	1666.0	44503.0	7083.0	111819.5	783217.0
20	1683.5	42810.5	7150.5	111598.0	711122.5
Area $\geq$	**1767.5**	**45230.5**	**7386.0**	**117897.5**	**969552.0**
Average	**1682.35**	**43770.75**	**7153.35**	**114103.175**	**753688.025**
Median	**1684.0**	**43553.0**	**7163.25**	**113870.5**	**728332.0**
Std. deviation	**44.741**	**857.72**	**118.167**	**1865.21**	**63105.373**

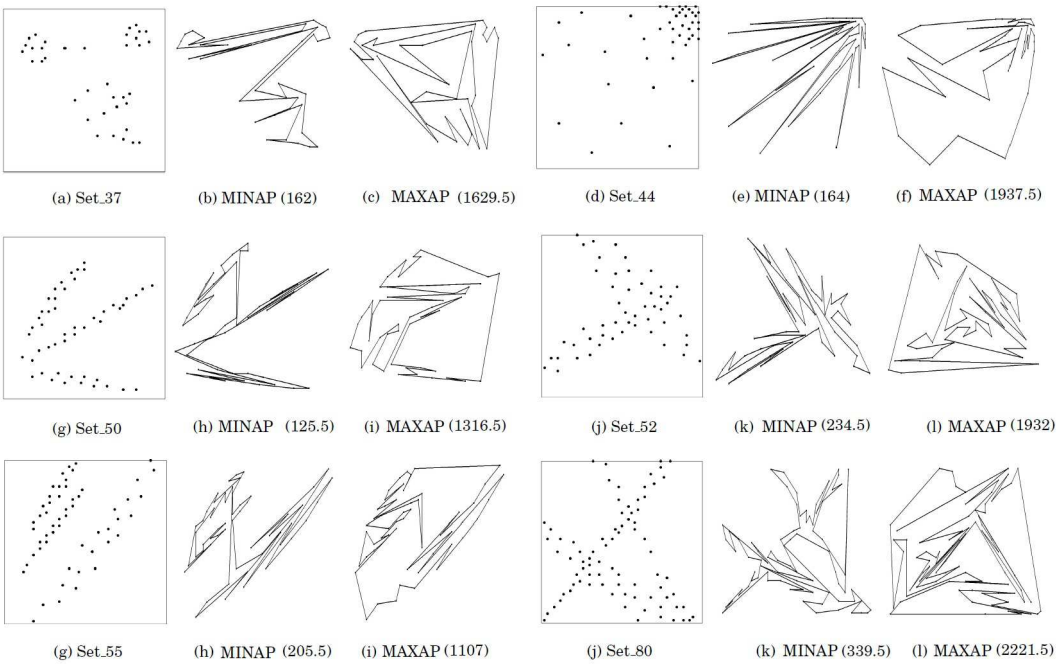


Fig. 11. Results generated for point sets taken from [Spaeth 2014].

ent shaped clusters and hence represents a challenging input to evaluate the proposed heuristics. Figure 11 illustrates the results generated by RAND_MINAP and

Table III. Approximate minimum areas and running times by RAND_MINAP() for the test instances taken from TSPLIB [Reinelt 2014] (areas in Sq. Units and time in seconds)

Test data	Execution time.	Approximate minimum area.	Scaling factor
berlin52	2.022	405.719	25
bayg29	2.006	1030.319	25
att48	2.014	8264.27	25
a280	4.536	7.12	25
burma14	2.001	7.07	1
ch130	2.271	146.5	25
ts225	2.975	1443.75	200
rat99	2.153	150.73	6
kroB100	2.14	674.84	50
pr439	10.567	11.32	1000
lin318	4.564	972.02	50
ulysses22	2.002	18.26	1
u159	2.465	592.0	100
st70	2.037	0.858	50

RAND_MAXAP for points having different clusters (area of each result is also mentioned in the brackets of corresponding figure).

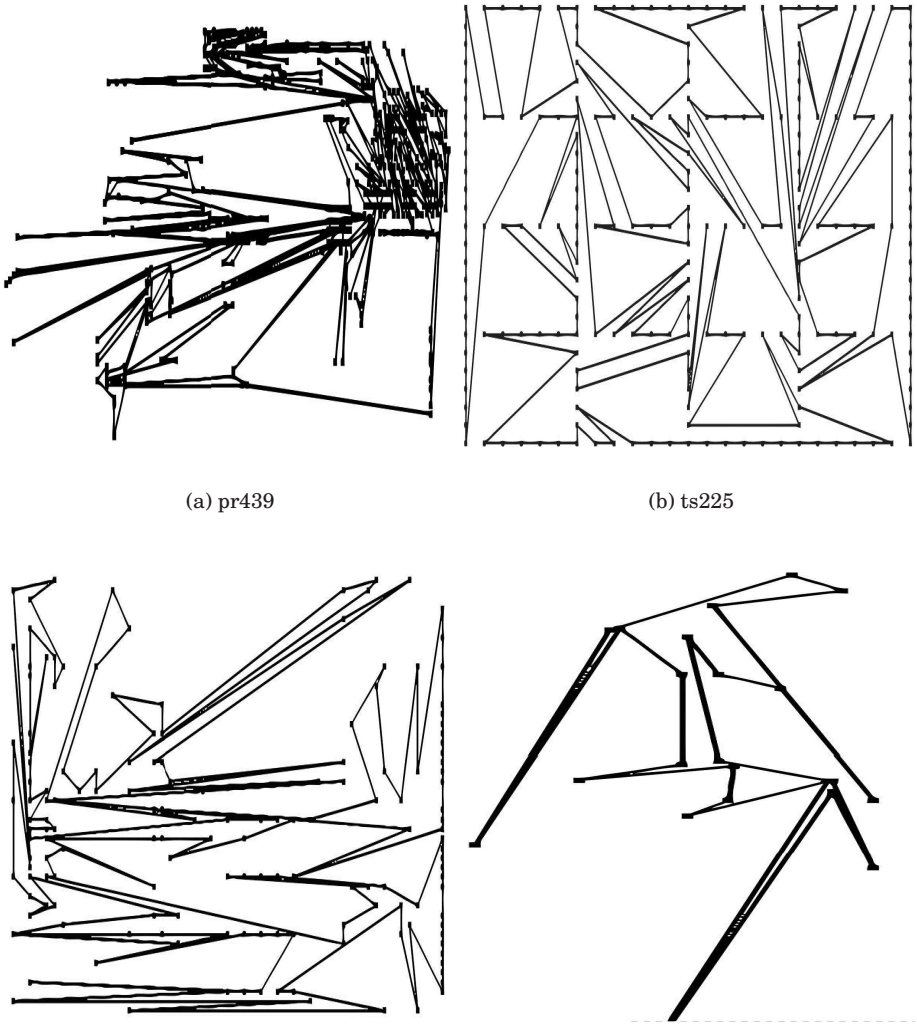
4.2.3. TSPLIB Benchmark Data. One of the major difficulty in experimenting with heuristics for MINAP is the absence of a publicly available benchmark data. Even generating test data for MINAP is a good contribution in this area. Apart from the random test data that we generated, we also experimented with the data sets available in TSPLIB benchmark data [Reinelt 2014] which is mainly meant for evaluating TSP heuristics. We used symmetric TSP data that includes national TSPs (A national TSP consists of points representing cities of a country, For eg. Burma), VLSI data sets etc.. We tested the proposed RAND_MINAP() using a collection of 14 TSP instances. Table III reports on best minimum areas obtained for various test instances along with the running time for one execution of the RAND_MINAP() algorithm. For a few point sets, the area of the polygonization were too large to be accommodated in the data type and hence we scaled down the coordinates of points using a factor (scaled down factor for each point set is mentioned in the last column of Table III). Figure 12 shows few interesting results from our experiment on TSPLIB benchmark data sets.

4.3. Area Fluctuation

In order to study the behavior of RAND_MINAP and RAND_MAXAP algorithms for different data sets, we constructed area fluctuation graphs from the Table I, II and the minimal areas of convex point sets. An area fluctuation graph is a line graph showing the area values in all the 20 trials for each of the mentioned point sets. A straight horizontal line in the graph increases the probability of that area being the optimal area. In the graph 13(a) of general point sets, none of the lines are steady as opposed to the straight lines of convex point sets shown in 13(b). Steady lines in the area fluctuation graph of convex points can be justified by Lemma 3.3, which states that RAND_MINAP always returns the optimal results for convex point sets. Area fluctuation graph for RAND_MAXAP is shown in Figure 13(c). We can observe non-straight lines in area fluctuation graph of RAND_MINAP/RAND_MAXAP, for general point sets. This is because, there might be several random sequences leading to an exponential number of different randomized MINAPs/MAXAPs resulting in non-straight lines in the area fluctuation graphs. It is an expected behavior of the algorithm for general point sets.

4.4. CPU Time Distribution among Subroutines

Table IV lists the CPU time allocation among three major sub procedures for point sets of sizes 300, 500 and 1000. Line sweeping and walking subroutines with complexities



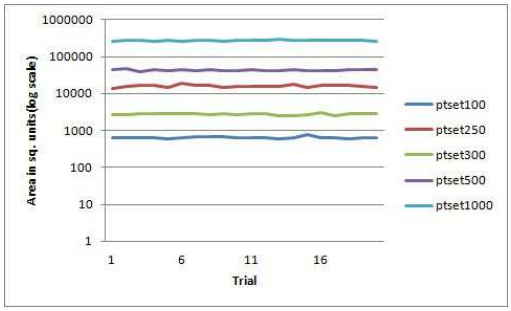
(a) pr439 (b) ts225

(c) u159 (d) ulysses22

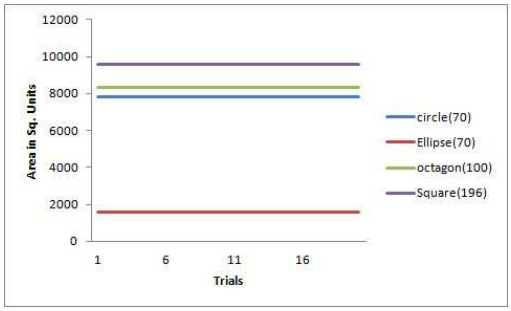
Fig. 12. Randomized MINAPs for different test data from TSPLIB [Reinelt 2014].

of $O(n \log n)$ and $O(n)$ respectively, steal a major portion of the overall execution time of both the algorithms. The term *other* in Table IV refers to all the remaining work done by the randomized algorithm which include random point selection and counter clock wise orientation of the points. All these methods account for the next portion in the CPU time. The procedures Interior() and Update() uses negligible amount of CPU time.

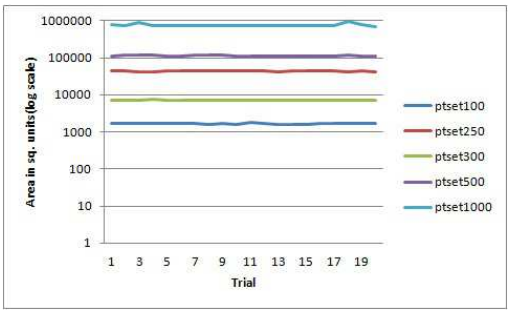
Another observation that can be drawn from Table IV is regarding the point set size and CPU time usage by different procedures. As the point set size becomes larger, the percentage of CPU time spent in the intersection checking grows proportionately. CPU time usage by the methods in other category has an inverse proportionate relation



(a) MINAP of General Point Sets



(b) MINAP of Convex Point Sets



(c) MAXAP of General Point Sets

Fig. 13. Area Fluctuation Graphs for MINAP and MAXAP based on the areas given in Tables I and II. 13(b) is constructed from another minimum area table for convex point sets given in Appendix

with the point set size. So it is evident from the Table IV that improving valid triangle picking (through Line_sweeping and walking) procedure should yield a considerable overall speed up for both the algorithms.

Table IV. CPU Time Distribution Among Different Subroutines

Algorithm	Subroutine	% of CPU Time		
		Pointset300	Pointset500	Pointset1000
RAND_MINAP	Line_sweep & Walk	52.103 %	67.842%	77.592%
	Interior	0.039%	0.043%	0.018%
	Update	0.598%	0.064%	0.033%
	Other	47.297%	32.050%	22.357%
RAND_MAXAP	Line_sweep & Walk	55.175%	72.849%	89.941%
	Interior	0.114%	0.039%	0.047%
	Update	0.114%	0.019%	0.041%
	Other	44.596%	27.092%	9.969%

5. COMPARISON

To the best of our knowledge, there do not exist benchmarks publicly available for MINAP problem that allow us to compare our results. Hence, we compare RAND_MINAP algorithm with an approximation algorithm [Muravitskiy and Tereshchenko 2011] and

Table V. Running Time of MINAP Algorithms for Smaller Point Sets

Point set size	Execution Time(Seconds)		
	RAND_MINAP	PERMUTE_REJECT	APPROXIMATE_MINAP
8	2	2.423	2.001
9	2	6.544	2.002
10	2	55.773	2.002
11	2	682.033	2.003
12	2	9315.623	2.003

Table VI. Comparison of RAND_MINAP with APPROX_MINAP for Larger Point sets

Point set size	Execution Time(Seconds)		Minimum Area (Sq. Units)		Speed up factor
	RAND_MINAP	APPROXIMATE_MINAP	RAND_MINAP	APPROXIMATE_MINAP	
150	2.28	30.06	2160.5	2593.5	13.18
200	3.19	58.58	2597.5	2673.0	18.38
250	3.94	170.99	13624.5	17824.0	43.36
400	8.63	1358.63	2657.0	2961.0	157.43

a brute force algorithm in Section 5.1. We also evaluate RAND_MINAP algorithm for its performance on different sized point sets. Further, we measure the speed up that RAND_MINAP gained against APPROXIMATE_MINAP and PERMUTE_REJECT as the point set size increased. The details are presented in Section 5.1.2

5.1. Comparison of MINAP Algorithms

5.1.1. Complexity of the algorithms. The minimal area polygonization (APPROXIMATE_MINAP) proposed by Muravitskiy et al. [Muravitskiy and Tereshchenko 2011] is a greedy approximation algorithm with a non-constant approximation factor. The worst case time complexity of APPROXIMATE_MINAP is $O(n^4)$. A preliminary preprocessing of the set of points can further improve the time complexity of the approximation algorithm to $O(n^3)$ but at the expense of increased memory usage [Muravitskiy and Tereshchenko 2011]. Though the authors [Muravitskiy and Tereshchenko 2011] talk about the optimization of the proposed greedy algorithm, it is not so clear whether they have incorporated the optimization in their implementation and hence we have implemented the $O(n^4)$ greedy approximation algorithm for the comparison purpose.

As there are no other heuristics available for the MINAP generation problem, we implemented an exhaustive search algorithm with a time complexity of $O((n-1)!)$ for the comparison purpose. Brute force technique is referred to as PERMUTE_REJECT, which can be summarized as follows:

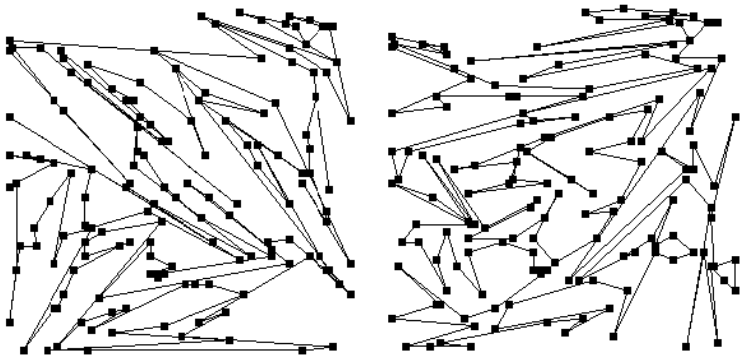
(1) Generates all the permutations of the point set, S
(2) Chooses a sequence which forms a simple polygon with minimum area

In terms of the computational complexity, the proposed algorithm ($O(n^2 \log n)$) performs better than APPROXIMATE_MINAP ($O(n^3)$) and PERMUTE_REJECT ($(n-1)!$) algorithms.

RAND_MINAP, APPROXIMATE_MINAP and PERMUTE_REJECT algorithms take $O(n)$ space where as the optimized greedy approximation algorithm takes $O(n^2)$ space.

5.1.2. Performance Evaluation. We evaluate the RAND_MINAP algorithm for its performance on various instances of point sets. Table V reports running times of all three MINAP algorithms for smaller sized point sets. As PERMUTE_REJECT takes exponentially large time for larger point sets, we had to restrict our comparison for point set instances of size at most 12. Table V indicates that RAND_MINAP and APPROXIMATE_MINAP take almost the same running time (approximately 2 seconds) for smaller point sets, whereas PERMUTE_REJECT takes much larger running time in each of the point sets. This is certainly because of its exhaustive searching nature.

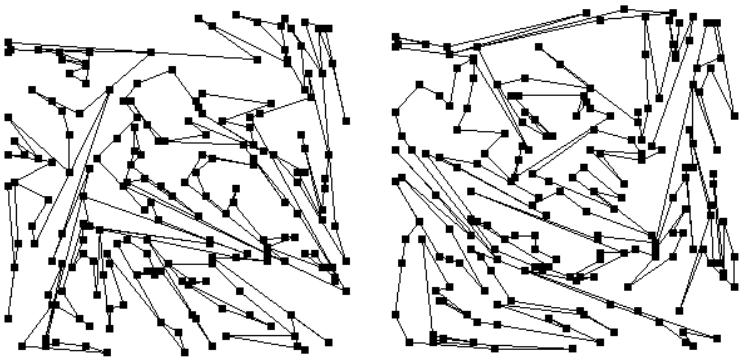
5.1.3. Comparison of computed area. In the case of larger point sets, it is not straight forward to verify optimal MINAP and hence RAND_MINAP returns a polygonization with an upper bound on the minimal area after repeated executions and updating on the minimal area. It is to be noted that the PERMUTE_REJECT takes longer time for large point sets and no other method exists for verification. So we restricted our comparison among RAND_MINAP and APPROXIMATE_MINAP for larger point sets. Figure 14 through 17 show the comparison of RAND_MINAP with APPROXIMATE_MINAP [Muravitskiy and Tereshchenko 2011] for a larger point set.



(a) Randomized MINAP (2160.5 Sq. units)

(b) Approximate MINAP (2593.5 Sq. units)

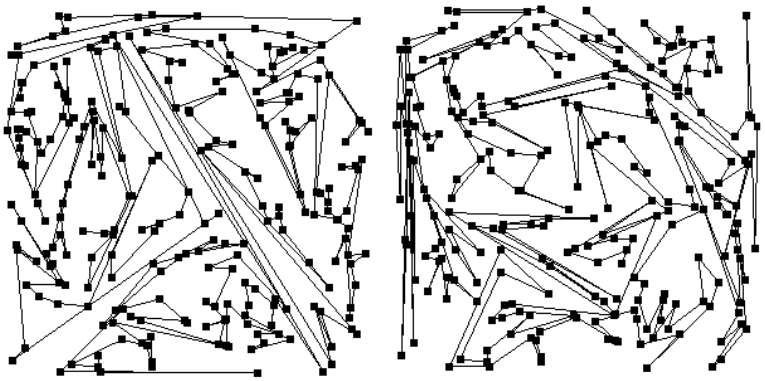
Fig. 14. Polygons Generated by RAND_MINAP and APPROXIMATE_MINAP for Point set 150



(a) Randomized MINAP (2597.5 Sq. units)

(b) Approximate MINAP (2673 Sq. units)

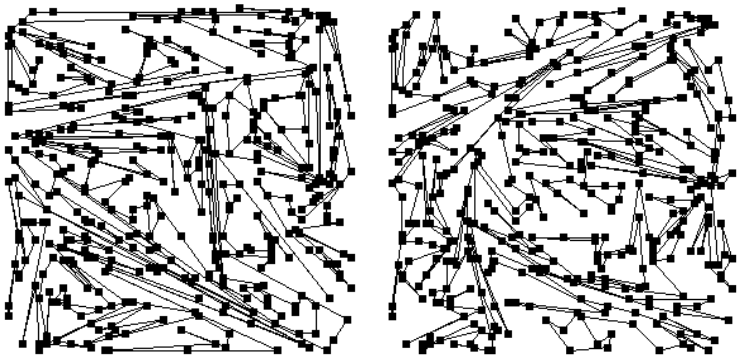
Fig. 15. Polygons Generated by RAND_MINAP and APPROXIMATE_MINAP for Point set 200



(a) Randomized MINAP (13624.5 Sq. units)

(b) Approximate MINAP (17824 Sq. units)

Fig. 16. Polygons Generated by RAND_MINAP and APPROXIMATE_MINAP for Point set 250



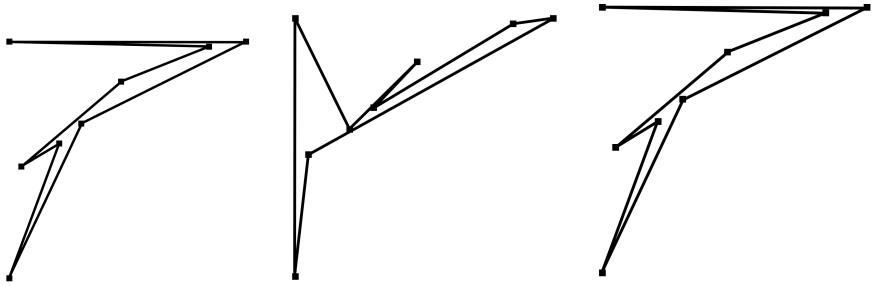
(a) Randomized MINAP (2657 Sq. units)

(b) Approximate MINAP (2961 Sq. units)

Fig. 17. Polygons Generated by RAND_MINAP and APPROXIMATE_MINAP for Point set 400

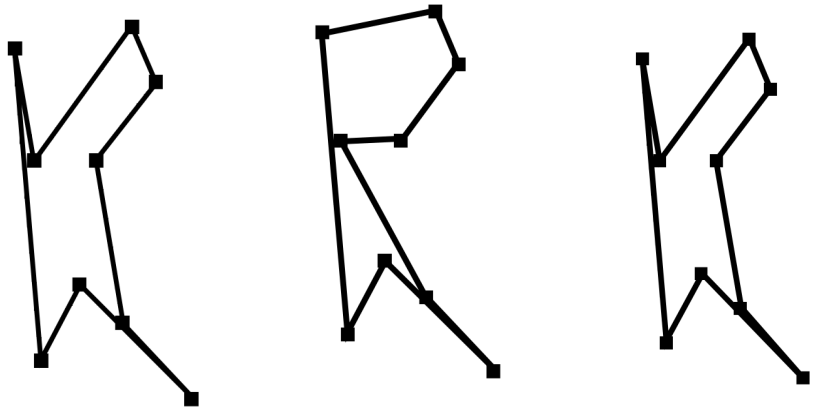
Point set sizes have been restricted to a maximum of 12 for RAND_MINAP Vs PERMUTE_REJECT comparison as anything above it took enormous amount of time for the completion of PERMUTE_REJECT method. This is clear from the Table V. PERMUTE_REJECT on a point set of size 12 took around 9315.623 seconds$\simeq$2.58 hours for its completion. The results returned by the RAND_MINAP were verified against the optimal solution generated by the brute force method. RAND_MINAP generated optimal MINAP for point sets of smaller sizes. Figures 18 to 20 visualize some of the results generated using all the three methods along with their areas.

Table VI lists minimal areas obtained by RAND_MINAP and APPROXIMATE_MINAP for various point sets. The readings clearly favor the RAND_MINAP algorithm as compared to the APPROXIMATE_MINAP algorithm. For all the point



(a) Permute MINAP (712.5 sq.units) (b) Approximate MINAP (974.5 sq.units) (c) Randomized MINAP (712.5 sq.units)

Fig. 18. Point set of size 8 and corresponding PERMUTE_MINAP, APPROXIMATE_MINAP and RAND_MINAP.

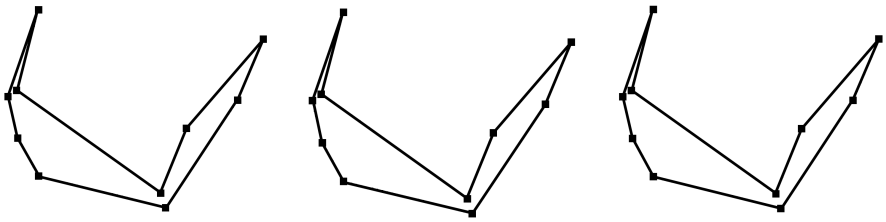


(a) Permute MINAP (553.53 sq.units) (b) Approximate MINAP (601.72 sq.units) (c) Randomized MINAP (553.53 sq.units)

Fig. 19. Point set of size 9 and corresponding PERMUTE_MINAP, APPROXIMATE_MINAP and RAND_MINAP

sets, RAND_MINAP provides a tighter upper bound on optimal minimal area as compared to the bound generated by APPROXIMATE_MINAP.

5.1.4. Speed up factor. In order to further assess the performance of our algorithm with other MINAP algorithms, we consider the *speed up* factor of the proposed algorithm. Speed up factor is defined as the ratio of running time of RAND_MINAP to running time of the other MINAP algorithm under consideration. For example, if we want to measure and compare the performance of RAND_MINAP with APPROXIMATE_MINAP for any point set S, we compute the speed up factor as in the equation



(a) Permute MINAP (b) Approximate MINAP (c) Randomized MINAP

Fig. 20. Point set of size 10 and corresponding PERMUTE_MINAP, APPROXIMATE_MINAP and RAND_MINAP (all three with an area of 15395.5 sq. units)

1.

$$\text{speed up}_{APPROXIMATE\_MINAP} = \frac{\text{running time of APPRIXIMATE\_MINAP for S}}{\text{running time of RAND\_MINAP for S}} \quad (1)$$

Similarly we can define speed up$_{PERMUTE_REJECT}$.

For larger point sets, we compared RAND_MINAP with APPROXIMATE_MINAP. Table VI lists running times and minimal areas obtained by these two algorithms for various point sets. We consider only one execution of RAND_MINAP for performance analysis. Speed up factor is also mentioned in the last column of Table VI. An interesting observation that can be drawn from the Table VI is on the relationship of speed up factor with point set size. Both are directly proportional. It is obvious that the speed up should be of $\frac{O(n^4)}{O(n^2 \log n)}$. We claim that such a speed up factor will be achieved at some instant as the point set size increases further. The graph plotted between speed up factor and point set size is presented in the Figure 21. The speed up curve further reassures our claim on the convergence of speed up factor to $O(\frac{n^2}{\log n})$.

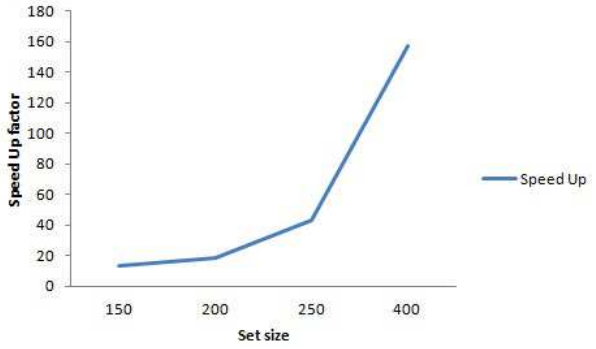


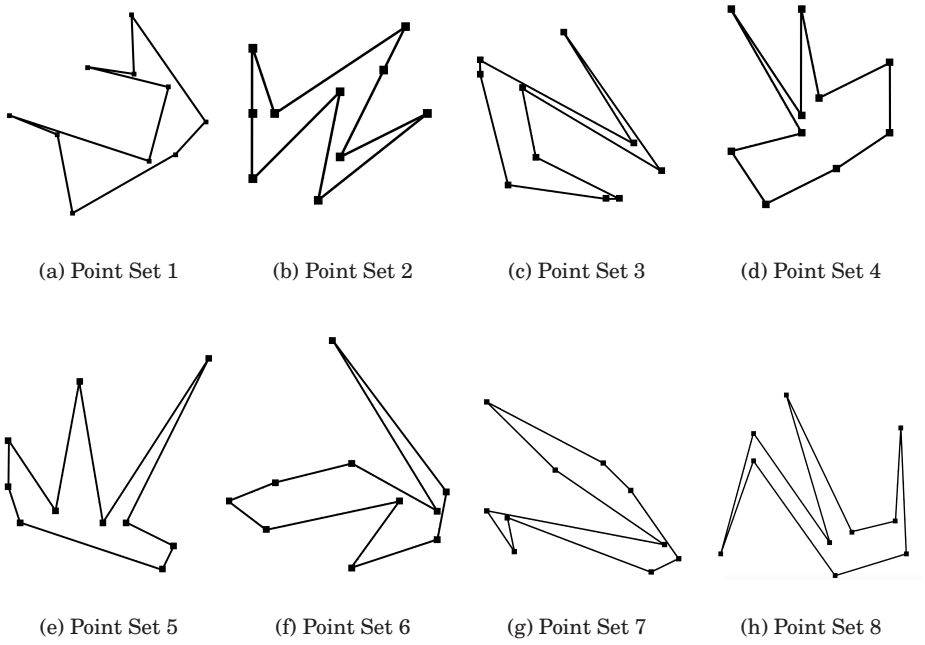
Fig. 21. Comparison of RAND_MINAP with APPROX_MINAP in terms of Speed up achieved in an Execution for Different Point sets

5.1.5. Extension to higher dimensions. Extension to higher dimensions is a primary concern for geometric algorithms. An algorithm gains more credibility if it can be easily extended to higher dimensions or to 3D at least. RAND_MINAP and RAND_MAXAP have been already extended to three dimensions [Peethambaran et al. 2015]. Instead

of triangles, both the algorithms use tetrahedra for constructing minimal volume polyhedrons from the three dimensional point sets. The authors have implemented PERMUTE_REJECT algorithm in three dimension. However, the experiments showed that it is impractical for point sets of larger sizes(size ≥ 7). In fact, the estimated computational time by PERMUTE_REJECT() for a point set of size 7 was more than half an hour [Peethambaran et al. 2015]. This huge computational time is incurred due to the exhaustive searching for all possible combinations of triangles [Veltkamp 1995]. The time complexity of brute force method in 3D is at least $\Omega(n_v^{5.n_v})$ [Veltkamp 1995] where n_v is the number of points in the set.

Table VII summarizes the comparison of the three algorithms for minimum area polygonization.

As a final validation experiment, we took several point sets of size 10 and generated optimal MINAP using PERMUTE_REJECT. Then we repeated the experiment on RAND_MINAP for these point sets. Though RAND_MINAP took several runs, it generated optimal MINAP for all the point sets. Figure 22 shows some of the optimal minimal area polygons that we obtained during our experiments.



(a) Point Set 1 (b) Point Set 2 (c) Point Set 3 (d) Point Set 4

(e) Point Set 5 (f) Point Set 6 (g) Point Set 7 (h) Point Set 8

Fig. 22. Optimal Minimal Area Polygons generated by RAND_MINAP and verified using PERMUTE_REJECT for various point sets of size 10

Table VII. Summary of the Comparison

Point of comparison	**PERMUTE_REJECT**	**APPROXIMATE_MINAP**	**RAND_MINAP**
Nature of the Algorithm	Brute force method	Greedy approximation	Randomized greedy
Time complexity	$O((n-1)!)$	$O(n^3)$	$O(n^2 \log n)$
Space complexity	$O(n)$	$O(n^2)$	$O(n)$
Extension to higher dimension	Difficult	Relatively easy	Easy
Optimal result	always	not always	not always

6. SUMMARY AND CONCLUSIONS

In this paper, we have presented a simple, randomized and greedy algorithm for computing the minimal area simple polygons (similarly, maximal area simple polygon) of planar point sets. The proposed algorithm is guaranteed to construct optimal solutions for certain point sets such as convex point sets and all-but-one convex point sets. The proposed method performs better than the available MINAP algorithms in terms of computational complexity of single execution.

One of the major difference between the proposed method and the existing MINAP/MAXAP algorithms is the type of input sequence used for polygon construction. Our method works on random sequences of input points whereas the existing algorithms work on fixed sequence formed out of the algorithmic rules. Being a randomized meta-heuristic, our strategy enjoys the flexibility of choosing the best candidate from the solution space. On the contrary, one of the main advantage as well as the disadvantage of existing MINAP or MAXAP algorithms is its adherence to a fixed sequence of points for computing the solutions. While the fixed sequence helps in generating the solution quickly, it also limits the algorithms ability to further explore and find out the best candidate from the solution space.

We conducted an empirical study, taking into account several performance measures for evaluating the proposed method. The proposed algorithm has been evaluated on test data taken from standard repositories such as SPAETH [Spaeth 2014] and TSPLIB [Reinelt 2014], a few large sized random point sets, convex point sets and a few challenging synthetically generated data. In our study, the proposed RAND_MINAP algorithm always found to perform better than its counterparts for all the input data used in our tests (Please refer to Figures 14-20). Since there exist no method for generating optimal minimal area polygonization to date, we firmly believe that heuristics such as RAND_MINAP with the ability of giving an upper bound on optimal minimal area for polygonization is quite relevant. Further, this can be used for generating polygonization test instances from large sized point sets (containing 1000 or 5000 points). These test instances are very useful to evaluate other geometric algorithms that use polygons as input data.

Based on our experimental results, Lemma 3.4 and the ξ_{error} principle (Section 3.3), we conjecture (Conjecture 6.1) on the existence of a permuted sequence leading to optimal polygonizations for any set of planar points.

CONJECTURE 6.1. *Given a set of* n *points in the general positions and k of which lie at convex position where* $3 \leq k \leq n-2$*, there exists at least one permuted sequence for which RAND_MINAP algorithm will generate an optimal MINAP of S*

It remains to be proved or disproved the Conjecture 6.1.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their valuable comments to improve the manuscript. Thanks to the anonymous reviewer for his/her suggestions to incorporate the plane sweep technique to enhance the algorithmic performance.

REFERENCES

AUER, T. AND HELD, M. 1998. Rpg - heuristics for the generation of random polygons. In *Proc. 8th Canad. Conf. Comput. Geom*. 38–44.

BERG, M. D., CHEONG, O., KREVELD, M. V., AND OVERMARS, M. 2008. *Computational Geometry: Algorithms and Applications* 3rd ed. Ed. Springer-Verlag TELOS, Santa Clara, CA, USA.

BOYCE, J. E., DOBKIN, D. P., III, R. L. S. D., AND GUIBAS, L. J. 1985. Finding extremal polygons. *SIAM Journal on Computing 14,* 1, 134–147.

DENEE, L. 1988. Polygonizations of point sets in the plane. *Discrete Comput. Geom. 3*, 77–87.

EPPSTEIN, D., OVERMARS, M., ROTE, G., AND WOEGINGER, G. 1992. Finding minimum area k-gons. *DISCRETE COMPUT. GEOM 7*, 45–58.

FEKETE, S. P. 1992. Ph.D. thesis. Ph.D. thesis, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON.

FEKETE, S. P. 2000. On simple polygonalizations with optimal area. *Discrete & Computational Geometry 23,* 1, 73–110.

GOYAL, S. 2010. A survey on travelling salesman problem. *Midwest Instruction and Computing Symposium*, 1–9.

MURAVITSKIY, V. AND TERESHCHENKO, V. 2011. Generating a simple polygonalizations. In *Proceedings of the 2011 15th International Conference on Information Visualisation*. IV '11. IEEE Computer Society, Washington, DC, USA, 502–506.

O'ROURKE, J. 1998. *Computational Geometry in C* 2nd Ed. Cambridge University Press, New York, NY, USA.

PEETHAMBARAN, J., PARAKKAT, A. D., AND MUTHUGANAPATHY, R. 2015. A randomized approach to volume constrained polyhedronization problem. *ASME J. Comput. Inf. Sci. Eng. 15,* 1.

REINELT, G. 2014. Tsplib database, http://www.iwr.uni-heidelberg.de/groups/comopt/software/tsplib95/.

SHARIR, M., SHEFFER, A., AND WELZL, E. 2011. Counting plane graphs: Perfect matchings, spanning cycles, and kasteleyn's technique. *CoRR abs/1109.5596*.

SPAETH, H. 2014. Spaeth cluster analysis datasets, http://people.sc.fsu.edu/ jburkardt/datasets/spaeth/spaeth.html.

TARANILLA, M. T., GAGLIARDI, E. O., AND PENALVER, G. H. 2011. Approaching minimum area polygonization. In *XVII Congreso Argentino de Ciencias de la Computacin*.

VELTKAMP, R. C. 1995. Boundaries through scattered points of unknown density. *Graph. Models Image Process. 57,* 6, 441–452.

ZHU, C., SUNDARAM, G., SNOEYINK, J., AND MITCHELL, J. S. 1996. Generating random polygons with given vertices. *Computational Geometry 6,* 5, 277 – 290.