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

JIJU PEETHAMBARAN, AMAL DEV PARAKKAT,
and RAMANATHAN MUTHUGANAPATHY, Advanced Geometric Computing Lab,
Department of Engineering Design, Indian Institute of Technology, Madras, India-600036

While random polygon generation from a set of planar points has been widely investigated in the literature, very few works address the construction of a 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 article, we present a greedy heuristic for computing the 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, hence very easy for implementation and maintenance. We report on various experimental studies on the behavior of a randomized heuristic on different point set instances. Test data have been taken from the 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.

CCS Concepts: • **Theory of computation → Randomness, geometry and discrete structures**; **Computational geometry;**

Additional Key Words and Phrases: Randomized algorithm, minimal area polygon, maximal area polygon, incremental algorithm, convex n-gons

ACM Reference Format:
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), 24 pages.
DOI: http://dx.doi.org/10.1145/2896849

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 their applications in pattern recognition and image processing [Fekete 2000; Denee 1988]. Peethambaran et al. [2015] point out that the three-dimensional versions of area optimization, that is, minimum/maximum volume simple polyhedronization, have the potential 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 a simple polygon with a given set of points for which the enclosed area attains the minimum [Fekete 1992]. Perimeter

Authors' addresses: J. Peethambaran, A. D. Parakkat, and R. Muthuganapathy, Advanced Geometric Computing Lab, Department of Engineering Design, Indian Institute of Technology, Madras, Chennai, India-600036; emails: {jijupnair2000, adp.upasana, emry01}@gmail.com.

optimization of polygonization, commonly known as the geometric travelling salesman problem (TSP) focuses 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 has been determined to be more difficult than the minimization of its perimeter. Boyce et al. [1985] 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 [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 a very thin triangle. Further difficulty arise from self-intersections of polygons. While self-intersections do not occur automatically in the case of a polygon with a 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 the same size. For instance, it is found to be exponential in n for some point sets [Denee 1988], whereas 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]; 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.

This is little previous work regarding minimum area polygonizations of fixed points in the plane, although polygonizations, especially randomized polygonizations, have 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 [2000], establishes the NP-completeness of minimum weight polygons or maximum weight polygons of a vertex set, a result which leads to the NP-completeness proof of corresponding area optimization problems. 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].

Taranilla et al. [2011] suggest three different strategies (greedy-MAP, RS-MAP, and ACO-MAP) to approximate the MINAP of planar point sets. The described heuristics employ either a locally optimal choice on the point selection (greedy-MAP), a random search over the space of feasible solutions (RS-MAP), or an 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 article, we present simple, randomized, and greedy algorithms for constructing optimal area (both minimum and maximum) polygonizations of planar point sets of any size. In addition to the intrinsic interest in obtaining the MINAP 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 article is organized as follows. In Section 2, we explain the incremental construction of minimal area simple polygons and maximal area simple polygons with runtime 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 present our conclusions in Section 6.

2. INCREMENTAL CONSTRUCTION

We start with an overall idea of MINAP construction. Let S $= \{p^0, p^1, p^2, \ldots, p^{n-1}\}$ denote a set of n points; each point p^i is represented by its x and y coordinates. P_i is an ordered set of points on the polygon in counterclockwise (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 the randomized incremental algorithm for MAXAP and MINAP, respectively. Simple polygons 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, $\triangle_2 = \mathrm{P}_2 = \{p_0, p_1, p_2\}$ (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 $\mathrm{E}_2 = \{e_0, e_1, e_2\}$. 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 $\mathbf{S} - \mathbf{P}_{i-1}$.
(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 nonintersecting 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 $p_q, p_r \in \mathbf{P}_{i-1}$. Remove $\triangle_i$ from the polygon $\mathbf{P}_{i-1}$ by adding the edges and the point p_i at the appropriate position. Update the edge set and vertex set as $\mathbf{E}_i = \mathbf{E}_{i-1} \cup \{p_q, p_i\} \cup \{p_i, p_r\} - \{p_q, p_r\}$ and $\mathbf{P}_i = \mathbf{P}_{i-1} \cup p_i$.
(4) If the point lies in the exterior of/on the $\mathbf{P}_{i-1}$, then find the smallest-area nonintersecting triangle $\triangle_i$ that the point p_i makes with the edges of $\mathbf{P}_{i-1}$. Let us denote the triangle as $\triangle_i = \{p_q, p_i, p_r\}$ $p_q, p_r \in \mathbf{P}_{i-1}$. Add $\triangle_i$ to the polygon $\mathbf{P}_{i-1}$ by adding the edges and the point p_i at the appropriate position. Update the edge set and vertex set as $\mathbf{E}_i = \mathbf{E}_{i-1} \cup \{p_q, p_i\} \cup \{p_i, p_r\} - \{p_q, p_r\}$ and $\mathbf{P}_i = \mathbf{P}_{i-1} \cup p_i$.

The various steps in generating the MINAP of a point set are depicted in Figure 1. A set of six points is given in Figure 1(a). An initial triangle $\mathrm{P}_2 = \{p_0, p_1, p_2\}$ 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 nonintersecting triangles formed by p_3 with the edges of P_2, as shown by the dashed and dark lines in 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 is comprised of two nonintersecting and two intersecting triangles, as shown in Figure 1(d). The smallest-area triangle from the nonintersecting 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 leads to the MINAP generation of the given point set in Figure 1(f).

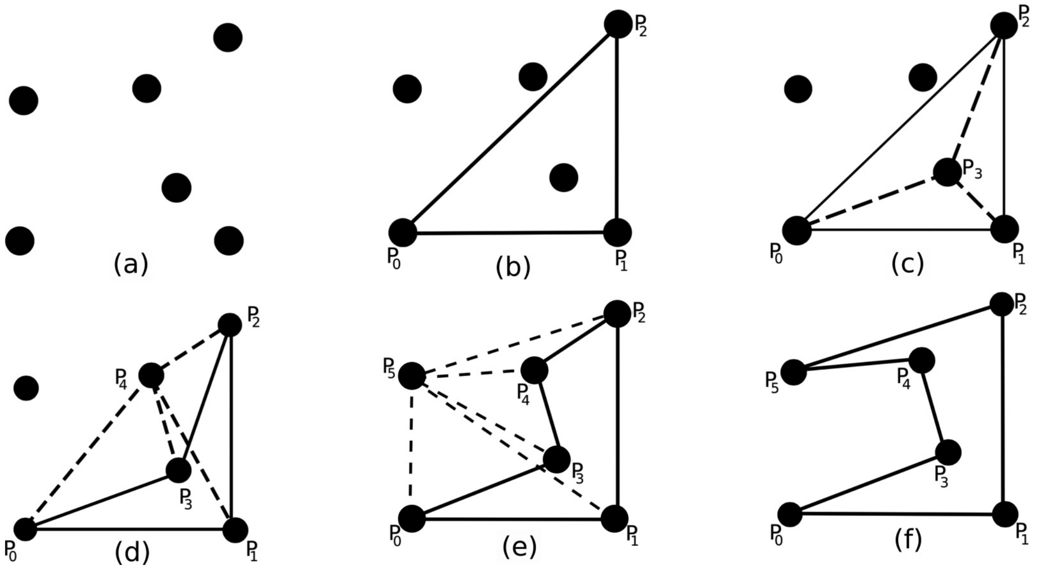


Fig. 1. Incremental construction of minimal area simple polygon. (a) Point set. (b) The initial random triangle. (c) - (e) Incremental construction. (f) Final polygon.

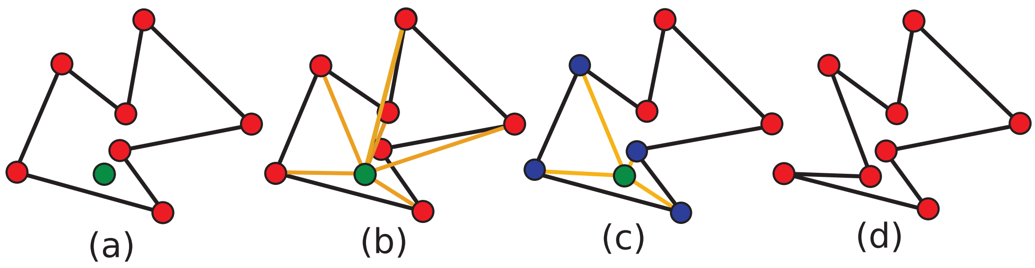


Fig. 2. Valid triangle removal through line-sweeping and walking techniques.

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 that does not have its edges intersected with the edges of the previous polygon (referred to as a **valid triangle**) is either added to the polygon or removed from the polygon. A valid triangle is determined by employing a classical line-sweeping algorithm [de Berg et al. 2008], described in Section 2.2.

2.2. Intersection Checking Through Line-Sweep Technique

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 [de Berg et al. 2008]. The idea can be described as follows. Assume that a 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 line segments in L share a common end vertex p_i; therefore, no mutual line segment intersections occur

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 nonintersecting 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 $E_{n-1}$;
```

in L (assuming general position). Hence, the algorithm uses a line-sweep method to determine the edges from L that intersect with the edges in E_{i-1} (see Figure 2(b)). All intersecting edges are discarded to get the set of edges (Figure 2(c)) that possibly form **valid** triangles with the previous polygonal edges.

The vertices of P_{i-1} that induce nonintersecting 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-area) valid triangle that 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 determine the largest-area (or smallest-area) triangle, the triangle area information is also updated during this walk. Once the required triangle is identified, it is removed (or 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 pseudocode 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 Section 2.1. The only difference occurs when excluding or including the triangles in each iteration. Instead of excluding the largest-area nonintersecting triangle if the point lies inside the current polygon, MAXAP construction excludes the smallest area nonintersecting triangle. If the point is exterior to the current polygon, the algorithm includes the largest-area nonintersecting triangle to the current polygon. A modified version of the pseudocode (RAND_MINAP) given in Algorithm 1 can be used for the MAXAP construction as well.

Fekete [2000] put forward an $O(n \log n)$ algorithm (APPROX_MAXAP) to obtain MAXAP. His construction employs a slope-based sorting of input points with respect to a point on the convex hull, 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 the convex hull, the algorithm returns P; otherwise, a complementary simple polygon with respect to the corresponding convex hull is returned. This simple construction

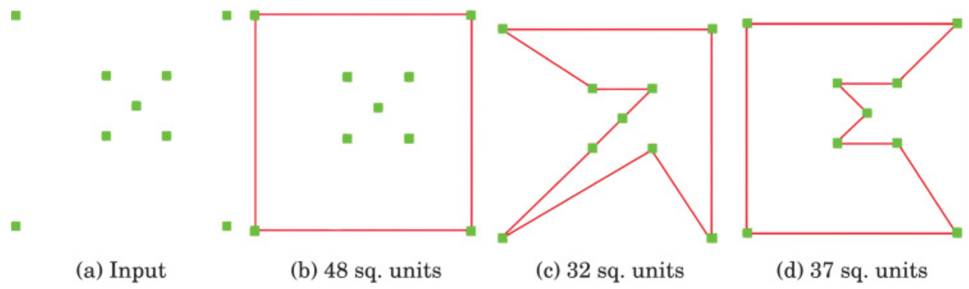


Fig. 3. Maximum area polygonizations generated by different algorithms. (a) Point set. (b) Convex hull. (c) Result by Fekete [2000]. (d) Our result. The area of each polygon is mentioned, along with the subfigure.

guarantees generation of 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; 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 a better solution 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 runtime. On the contrary, one of the main advantages, 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 algorithm's ability to further explore and determine 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 straightforward 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 nonintersecting edges that the selected point forms with the vertices of P_{i-1}. Walking over the polygon and picking the largest-area (or smallest-area) triangle is achieved in $O(n)$. Thus, the overall asymptotical runtime 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*; 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; 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 a large-sized point set in $O(n^2 \log n)$, which is a clear advantage, especially when considering that the MINAP problem has only a few approximate algorithms and brute-force technique. On space complexity, the algorithm maintains three lists—P_i, E_i, and L—each having a size of $O(n)$, hence taking no more than $O(n)$ space overall.

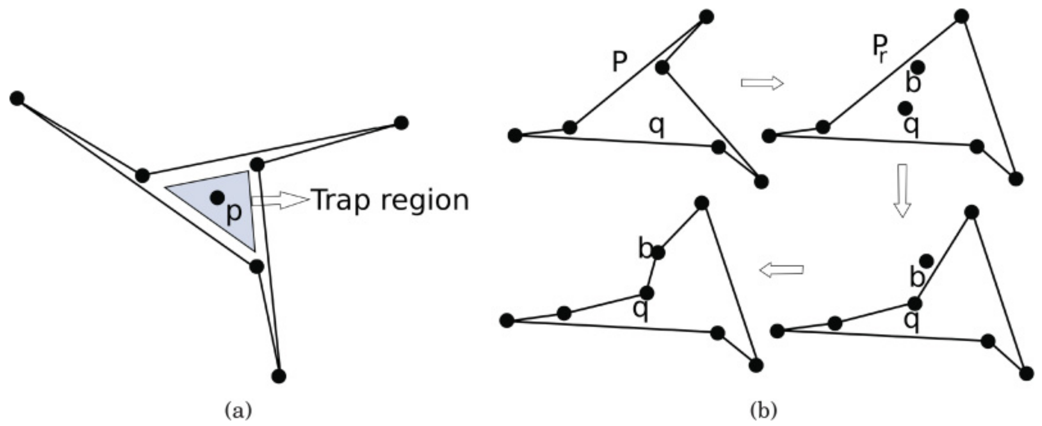


Fig. 4. (a) An example of a trap region (gray-colored region). (b) Illustration of a local rearrangement. In Figure 4(b), P is the previous polygon, q is the trapped point, and $\{b\}$ represents the minimal set of blocking points.

3. CORRECTNESS ANALYSIS

3.1. Existence of Polygonization

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 that, 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 which no edges of P are completely visible (refer to the gray-colored region in Figure 4(a)). In an iteration of *RAND_MINAP*(), if the selected point belongs to any of the trap regions of the previous polygon (as shown in Figure 4(a)), it forms only intersecting triangles with the previous polygon; 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; 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.

To resolve a trap region that may arise in any of the iterations of *RAND_MINAP*, we employ a local rearrangement technique on blocking vertices of the previous polygon, P. Vertices and edges of P that block the visibility of a point $q \subseteq T(P)$ are 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, 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 (see Proposition 3.1).

PROPOSITION 3.1. *For any finite set of planar points S where $|S| = n$, there exists an ordering of points, which, when subjected to the rules of the 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 hypothesize that this 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 subsets of points, $\{S_r, B\}$, where $B = \{b_1, b_2, \ldots b_m\}$ 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 the 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 that leads to the polygonization of S_r, followed by q, followed by $b_1, b_2, \ldots b_m$, 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 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 a *convex point set* if all the points in S are colocated on the convex hull of S. In this section, we show that the 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 the RAND_MINAP algorithm. Then, a *potential triangle* can be either of the following:

—a nonintersecting 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 nonintersecting 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 the *RAND_MINAP* algorithm, the point set is divided into i subsets, for which each subset, 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 the *friend edge*.

In 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 to 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 nonfriend edges can be made only at the expense of an intersection with the corresponding friend edge. This makes that triangle a nonpotential one. □

Note that all potential triangles have their vertices on the convex hull of S. Since all 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 nonconvex simple polygon, which is

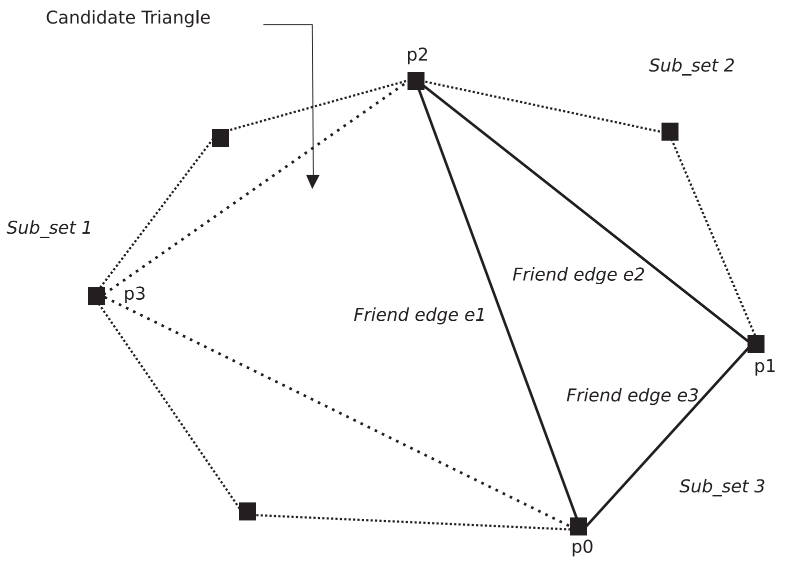


Fig. 5. Potential triangle in an iteration of *RAND_MINAP* for a convex point set.

obviously a nonoptimal solution for S. However, the vertices of a potential triangle, which is the only nonintersecting triangle available in an iteration of *RAND_MINAP* (see Lemma 3.2), lies on the convex hull of S. Consequently, a nonconvex polygon is never generated for S by *RAND_MINAP*, which contradicts our assumption. Hence, *RAND_MINAP* always generates a 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 point lie at convex positions). This indicates that a permuted sequence producing an optimal polygon exists for any point sets mentioned in the earlier category. A point set S is referred to as *convex points* if all points in S are colocated on the convex hull of S. The sequence producing optimal MINAP is referred to as the *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 that, when subjected to the rules of the RAND_MINAP algorithm, will generate an optimal MINAP of S.*

PROOF.

(1) **Base case, n=4:** There arise two cases, as follows.
   (a) **Case 1:** All points lie at the convex position.
   When all points lie at the 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 the nonconvex position. This will clearly produce an optimal MINAP.

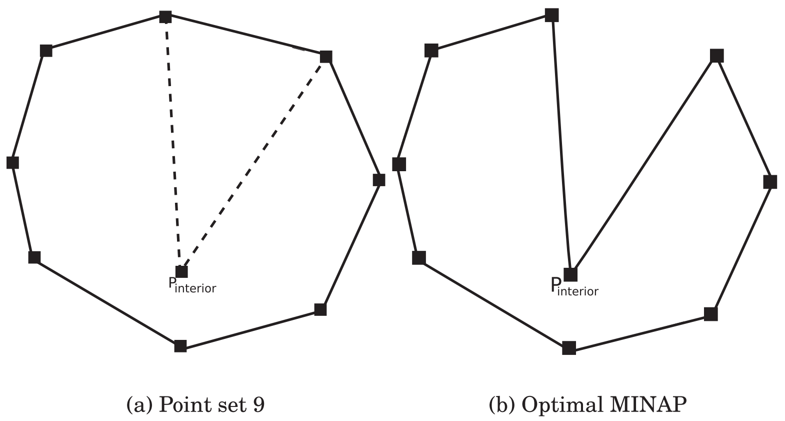


Fig. 6. One point lying interior to an 8-gon and the corresponding optimal MINAP. The permuted sequence consists of all points on the convex 8-gon, followed by the interior point.

(2) When point set size is n, there are two cases, as follows.
   (a) **Case 1:** All points lie at the convex position.
   This is, again, a direct consequence of Lemma 3.3. When all points lie at the 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. □

Given a set of n points, there exists a convex hull of k points, for which 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 the 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 the ξ_{error} measure, we hypothesize 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 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 with an Intel Core i3-2330M processor with 2.20GHz and 2GB RAM. Twenty trials were done to generate MINAP or MAXAP of each of the point sets delineated earlier. One trial consists of 100 executions, which implies that, for a point set, the algorithm was run 2000 times before making a conclusion on the lower bound/upper bound on the optimal areas.

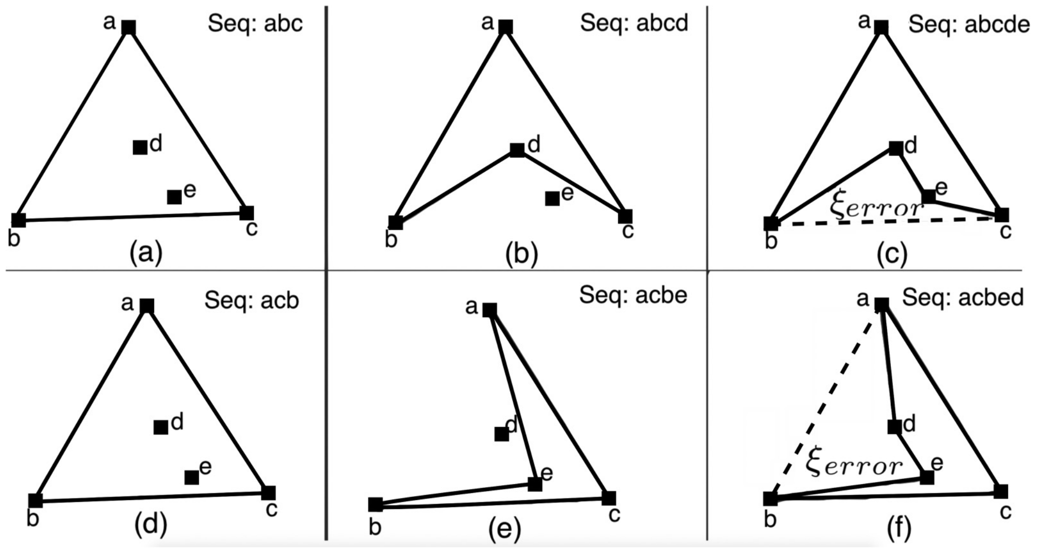


Fig. 7. Illustration of the relationship between optimal sequence and the largest ξ_{error} for a set of 5 points. The sequence producing the largest ξ_{error} results in the optimal MINAP, as shown in Figure 7(f), as compared to the one in Figure 7(c).

We generated random point sets of sizes 100, 150, 200, 250, 300, 400, 500, and 1000 for experimental purposes. 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. Whenever a point is added to the polygon, the algorithm marks it as *used* so that the point will be ignored when selecting a random point in future iterations. The major subroutines of the program consist 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 a classical line-sweeping algorithm [de 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}. The 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].
—**Line_sweep**(E_{i-1}, L): It updates the list L with the nonintersecting edges that p_i forms with the vertices of P_{i-1}. In the nonintersecting case, the segments $\overline{qp}$ and $\overline{qr}$

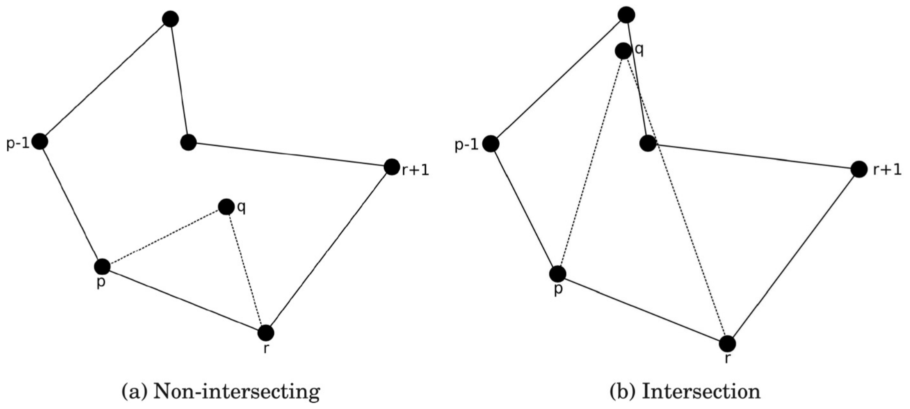


Fig. 8. Different types of intersections of the triangle with the previous polygon.

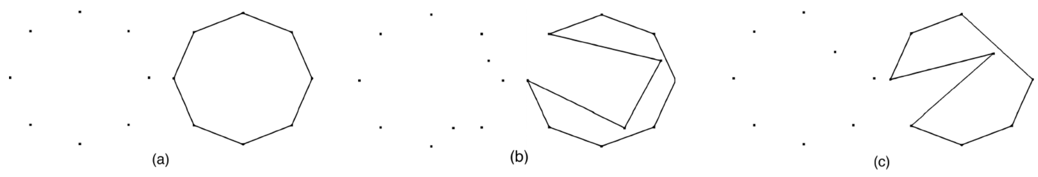


Fig. 9. Results of octagonal (regular polygon) point sets and its variants.

meet the vertices p and r, respectively, as shown in Figure 8(a). When the segments $\overline{qp}$ and $\overline{qr}$ cut any of the previous polygonal edges, it is considered as an intersection. An example of 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): The 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 the update is $O(n)$.

4.2. Computational Results

4.2.1. Random Point Sets. Figure 9 shows the MINAPs for an 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). Note that the RAND_MINAP algorithm generates the optimal results for all three point sets for which intuitive optimal solutions are known to us. Similarly, Figure 10 shows the optimal solutions generated for a few convex point sets.

Tables I and 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 trials. Recall that one trial consists of 100 executions of the RAND_MINAP algorithm. Thus, the algorithm was run 2000 times before taking 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 tables. Our bound value on the optimal minimal area for each point set is listed in the last row of Table I.

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

Trial No.	Point set 100	Point set 250	Point set 300	Point set 500	Point set 1000
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**

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

Trial No.	Point set 100	Point set 250	Point set 300	Point set 500	Point set 1000
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**

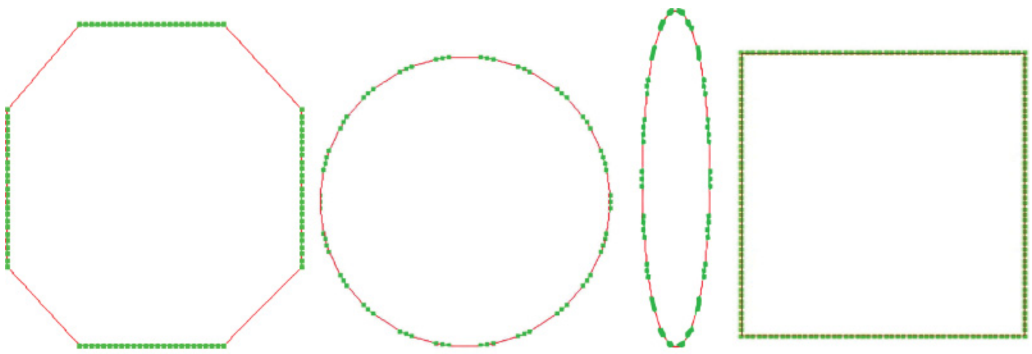
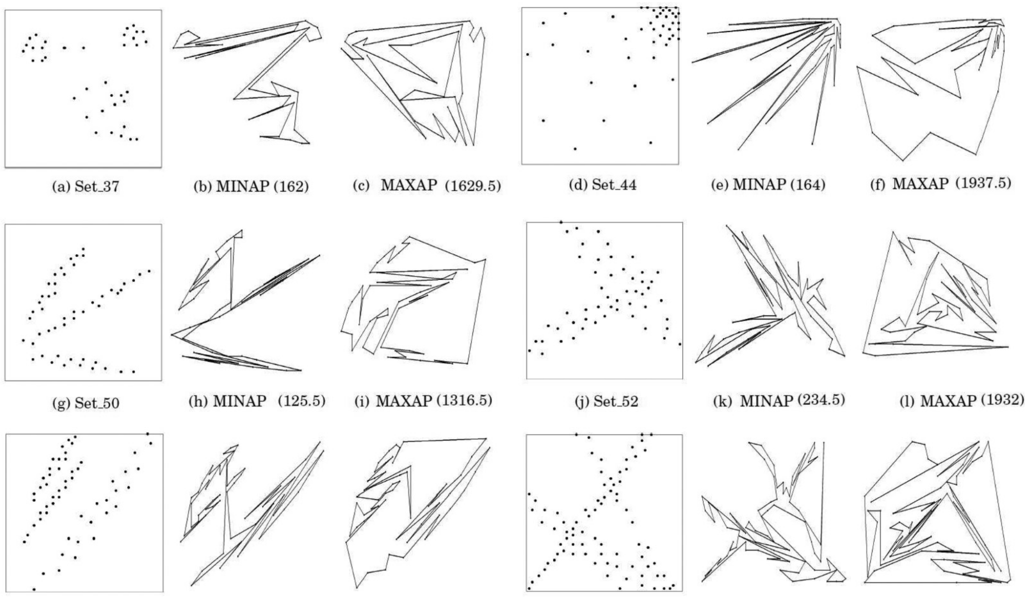

(a) Octagon (100 Points) (b) Circle (70 Points) (c) Ellipse (70 Points) (d) Square (196 Points)

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


(a) Set_37 (b) MINAP (162) (c) MAXAP (1629.5) (d) Set_44 (e) MINAP (164) (f) MAXAP (1937.5)

(g) Set_50 (h) MINAP (125.5) (i) MAXAP (1316.5) (j) Set_52 (k) MINAP (234.5) (l) MAXAP (1932)

(g) Set_55 (h) MINAP (205.5) (i) MAXAP (1107) (j) Set_80 (k) MINAP (339.5) (l) MAXAP (2221.5)

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

4.2.2. SPAETH Cluster Data. We experimented with the point set data taken from the SPAETH cluster analysis database [Spaeth 2014]. Each input consists of different-shaped clusters, hence represents a challenging input to evaluate the proposed heuristics. Figure 11 illustrates the results generated by RAND_MINAP and RAND_MAXAP for points having different clusters (the area of each result is also mentioned in the brackets of corresponding figure).

4.2.3. TSPLIB Benchmark Data. One of the major difficulties in experimenting with heuristics for MINAP is the absence of 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 datasets available in TSPLIB benchmark data [Reinelt 2014], which is mainly meant for evaluating TSP

Table III. Approximate Minimum Areas and Runtimes 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

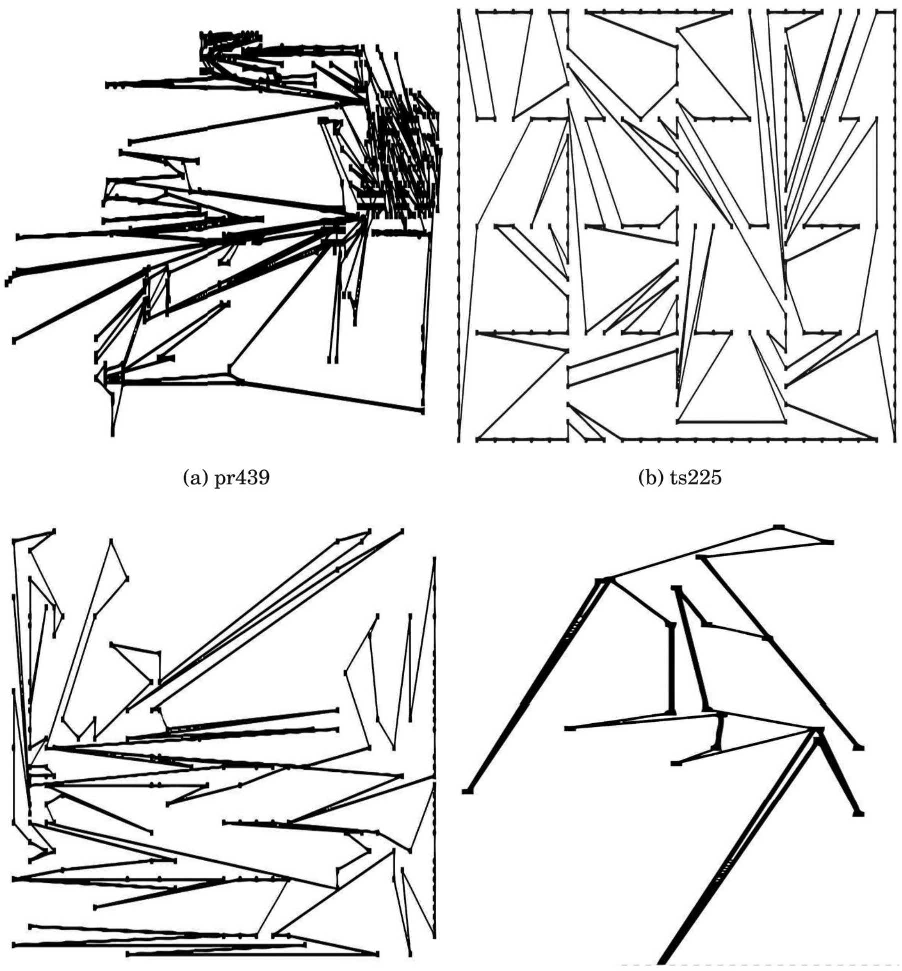
heuristics. We used symmetric TSP data that includes national TSPs (a national TSP consists of points representing cities of a country, e.g., Burma), VLSI datasets, and so on. We tested the proposed RAND_MINAP() using a collection of 14 TSP instances. Table III reports on best minimum areas obtained for various test instances, along with the runtime for one execution of the RAND_MINAP() algorithm. For a few point sets, the areas of the polygonization were too large to be accommodated in the data type; hence, we scaled down the coordinates of points using a factor (the scaled down factor for each point set is mentioned in the last column of Table III). Figure 12 shows a few interesting results from our experiment on TSPLIB benchmark datasets.

4.3. Area Fluctuation

In order to study the behavior of RAND_MINAP and RAND_MAXAP algorithms for different datasets, we constructed area fluctuation graphs from Tables I and II and the minimal areas of convex point sets. An area fluctuation graph is a line graph showing the area values in all 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 of general point sets of Figure 13(a), none of the lines is steady as opposed to the straight lines of the convex point sets shown in Figure 13(b). Steady lines in the area fluctuation graph of the convex points can be justified by Lemma 3.3, which states that RAND_MINAP always returns the optimal results for convex point sets. The area fluctuation graph for RAND_MAXAP is shown in Figure 13(c). We can observe nonstraight lines in the 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 nonstraight 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 subprocedures for point sets of sizes 300, 500, and 1000. Line-sweeping and walking subroutines with complexities of $O(n \log n)$ and $O(n)$, respectively, steal a major portion of the overall execution time of both algorithms. The term *other* in Table IV refers to the remaining work done by the randomized algorithm, which includes random point selection and CCW orientation

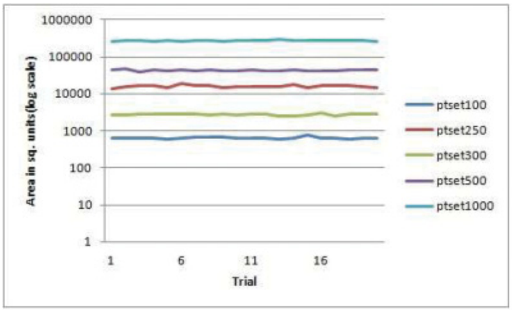

(a) pr439 (b) ts225

(c) u159 (d) ulysses22

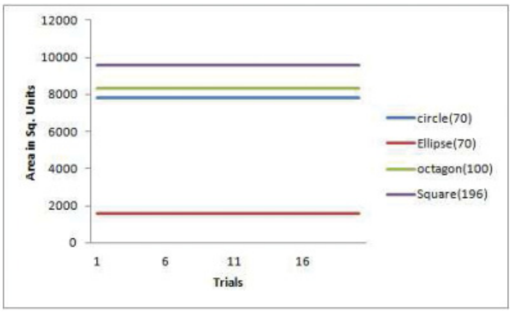
Fig. 12. Randomized MINAPs for different test data from TSPLIB [Reinelt 2014].

Table IV. CPU Time Distribution Among Different Subroutines

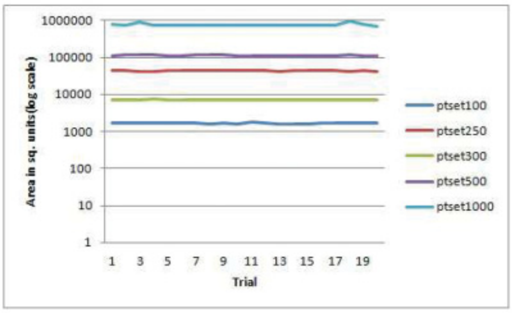
		% of CPU Time		
Algorithm	Subroutine	Point set 300	Point set 500	Point set 1000
RAND_MINAP	Line_sweep & Walk	52.103%	67.842%	77.592%
	Interior	0.039%	0.043%	0.018%
	Update	0.598%	0.064%	0.033%
	Other	47.297%	32.050%	22.357%
RAND_MAXAP	Line_sweep & Walk	55.175%	72.849%	89.941%
	Interior	0.114%	0.039%	0.047%
	Update	0.114%	0.019%	0.041%
	Other	44.596%	27.092%	9.969%



(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. Figure (b) is constructed from another minimum area table for convex point sets given in the Appendix.

of the points. All these methods account for the next portion in the CPU time. The procedures Interior() and Update() use a 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 the *other* category has an inverse proportionate relation with the point set size. Thus, it is evident from the Table IV that improving the valid triangle picking (through Line_sweeping and walking) procedure should yield a considerable overall speedup for both algorithms.

5. COMPARISON

To the best of our knowledge, there are no publicly available benchmarks for the MINAP problem that allow us to compare our results. Hence, we compare the RAND_MINAP algorithm with an approximation algorithm [Muravitskiy and Tereshchenko 2011] and a brute-force algorithm in Section 5.1. We also evaluate the RAND_MINAP algorithm for its performance on different-sized point sets. Further, we measure the speedup 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 and Tereshchenko [2011] is a greedy approximation algorithm with a nonconstant approximation factor. The worst-case time complexity of APPROXIMATE_MINAP is $O(n^4)$. Preliminary preprocessing of the set

Table V. Runtime of MINAP Algorithms for Smaller Point Sets

	Execution Time (Seconds)		
Point set size	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

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 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. Hence, we have implemented the $O(n^4)$ greedy approximation algorithm for comparison purposes.

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 comparison purposes. The 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 that forms a simple polygon with minimum area

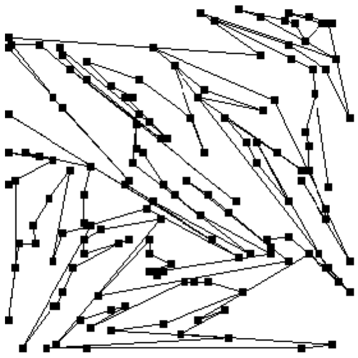
In terms of computational complexity, the proposed algorithm ($O(n^2 \log n)$) performs better than the APPROXIMATE_MINAP ($O(n^3)$) and PERMUTE_REJECT ($(n-1)!$) algorithms.

The RAND_MINAP, APPROXIMATE_MINAP and PERMUTE_REJECT algorithms take $O(n)$ space, whereas 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 runtimes 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 runtime (approximately 2s) for smaller point sets, whereas PERMUTE_REJECT takes much larger runtime 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 straightforward to verify optimal MINAP; 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 PERMUTE_REJECT takes longer for large point sets and no other method exists for verification. Thus, we restricted our comparison among RAND_MINAP and APPROXIMATE_MINAP for larger point sets. Figures 14–17 show the comparison of RAND_MINAP with APPROXIMATE_MINAP [Muravitskiy and Tereshchenko 2011] for a larger point set.

Point set sizes have been restricted to a maximum of 12 for RAND_MINAP versus PERMUTE_REJECT comparison, as anything above it took an enormous amount of time for the completion of the PERMUTE_REJECT method. This is clear from Table V. PERMUTE_REJECT on a point set of size 12 took around 9315.623s $\simeq$ 2.58h 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


(a) Randomized MINAP (2160.5 Sq. units)

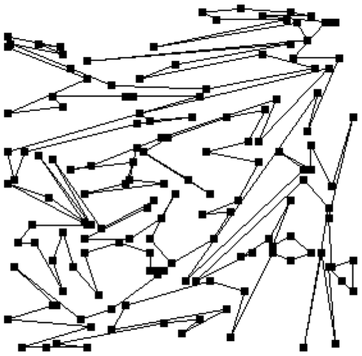
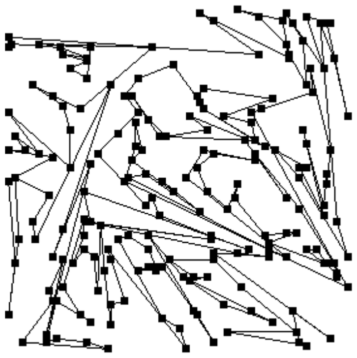

(b) Approximate MINAP (2593.5 Sq. units)

Fig. 14. Polygons generated by RAND_MINAP and APPROXIMATE_MINAP for point set 150.


(a) Randomized MINAP (2597.5 Sq. units)

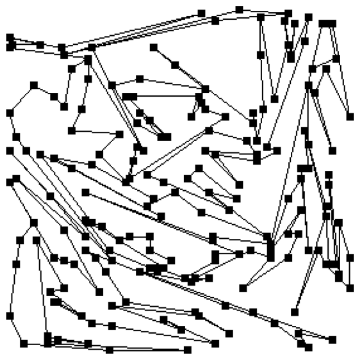
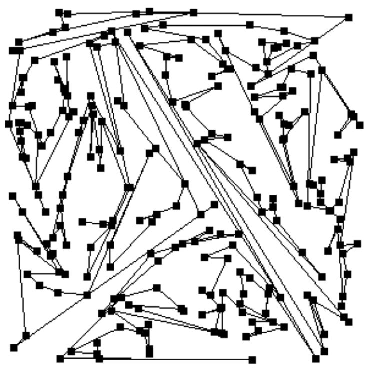

(b) Approximate MINAP (2673 Sq. units)

Fig. 15. Polygons generated by RAND_MINAP and APPROXIMATE_MINAP for point set 200.


(a) Randomized MINAP (13624.5 Sq. units)

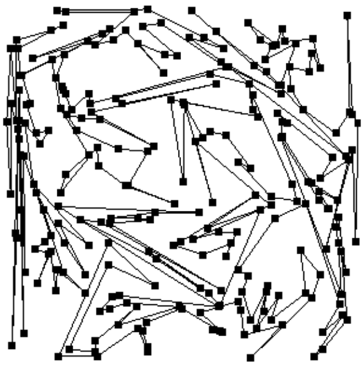

(b) Approximate MINAP (17824 Sq. units)

Fig. 16. Polygons generated by RAND_MINAP and APPROXIMATE_MINAP for point set 250.

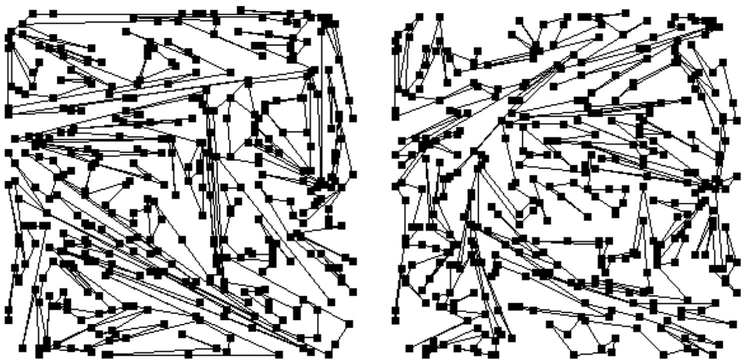

(a) Randomized MINAP (2657 Sq. units) (b) Approximate MINAP (2961 Sq. units)

Fig. 17. Polygons generated by RAND_MINAP and APPROXIMATE_MINAP for point set 400.

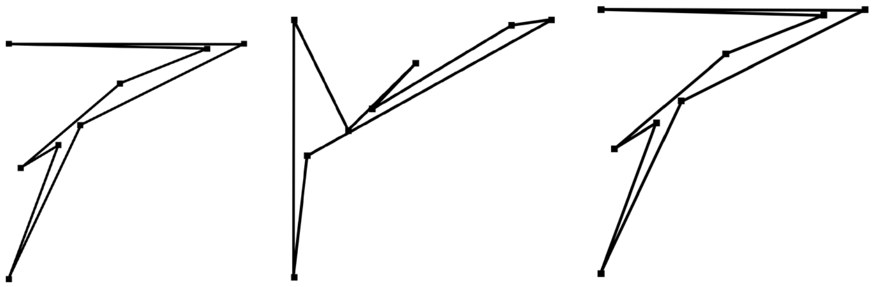

(a) Permute MINAP (712.5 sq.units) (b) Approximate MINAP (974.5 sq.units) (c) Randomized MINAP (712.5 sq.units)

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

optimal MINAP for point sets of smaller sizes. Figures 18 to 20 visualize some of the results generated using all 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 point sets, RAND_MINAP provides a tighter upper bound on optimal minimal area as compared to the bound generated by APPROXIMATE_MINAP.

5.1.4. Speedup Factor. In order to further assess the performance of our algorithm with other MINAP algorithms, we consider the *speedup* factor of the proposed algorithm. Speedup factor is defined as the ratio of the runtime of RAND_MINAP to the runtime 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 speedup factor as in Equation (1).

$$\text{speedup}_{APPROXIMATE\_MINAP} = \frac{\text{runtime of APPROXIMATE\_MINAP for S}}{\text{runtime of RAND\_MINAP for S}}. \quad (1)$$

Similarly, we can define $\text{speedup}_{PERMUTE\_REJECT}$.

For larger point sets, we compared RAND_MINAP with APPROXIMATE_MINAP. Table VI lists runtimes and minimal areas obtained by these two algorithms for various

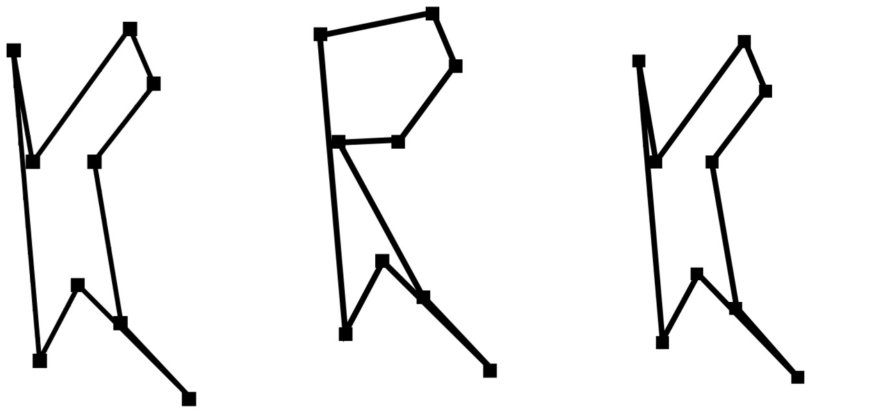

(a) Permute MINAP (553.53 sq.units)
(b) Approximate MINAP (601.72 sq.units)
(c) Randomized MINAP (553.53 sq.units)

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

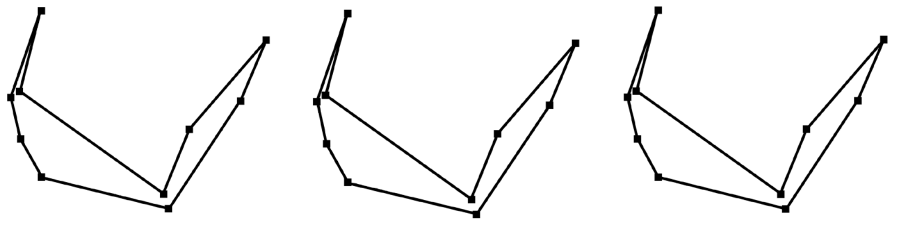


Fig. 20. Point set of size 10 and corresponding PERMUTE_MINAP, APPROXIMATE_MINAP, and RAND_MINAP (all three with an area of 15395.5 sq. units).

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

	Execution Time (Seconds)		Minimum Area (Sq. Units)		Speedup
Point set size	RAND_MINAP	APPROXIMATE_MINAP	RAND_MINAP	APPROXIMATE_MINAP	factor
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

point sets. We consider only one execution of RAND_MINAP for performance analysis. Speedup factor is also mentioned in the last column of Table VI. An interesting observation that can be drawn from Table VI is on the relationship of the speedup factor with the point set size. Both are directly proportional. It is obvious that the speedup should be of $\frac{O(n^4)}{O(n^2 \log n)}$. We claim that such a speedup factor will be achieved at some instant as the point set size increases further. The graph plotted between speedup factor and point set size is presented in Figure 21. The speedup curve further validates our claim on the convergence of speedup factor to $O(\frac{n^2}{\log n})$.

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

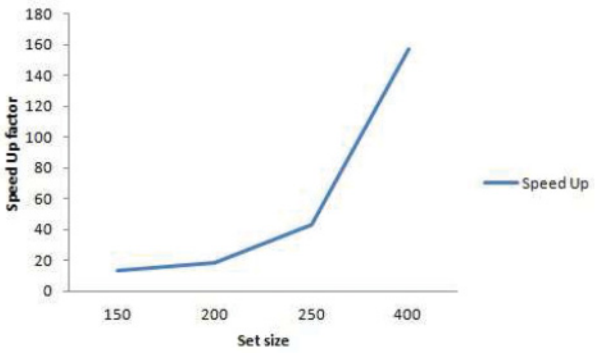


Fig. 21. Comparison of RAND_MINAP with APPROX_MINAP in terms of speedup achieved in an execution for different point sets.

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

have already been extended to three dimensions [Peethambaran et al. 2015]. Instead of triangles, both algorithms use tetrahedra for constructing minimal-volume polyhedrons from the three-dimensional point sets. Peethambaran et al. [2015] have implemented the PERMUTE_REJECT algorithm in three dimensions. However, the experiments showed that it is impractical for point sets of larger sizes (size $\geq$7). In fact, the estimated computational time by PERMUTE_REJECT() for a point set of size 7 was more than 30min [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 the 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 point sets. Figure 22 shows some of the optimal minimal area polygons that we obtained during our experiments.

6. SUMMARY AND CONCLUSIONS

In this article, we have presented a simple, randomized, and greedy algorithm for computing the minimal area simple polygons (similarly, maximal area simple polygons) 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 differences 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

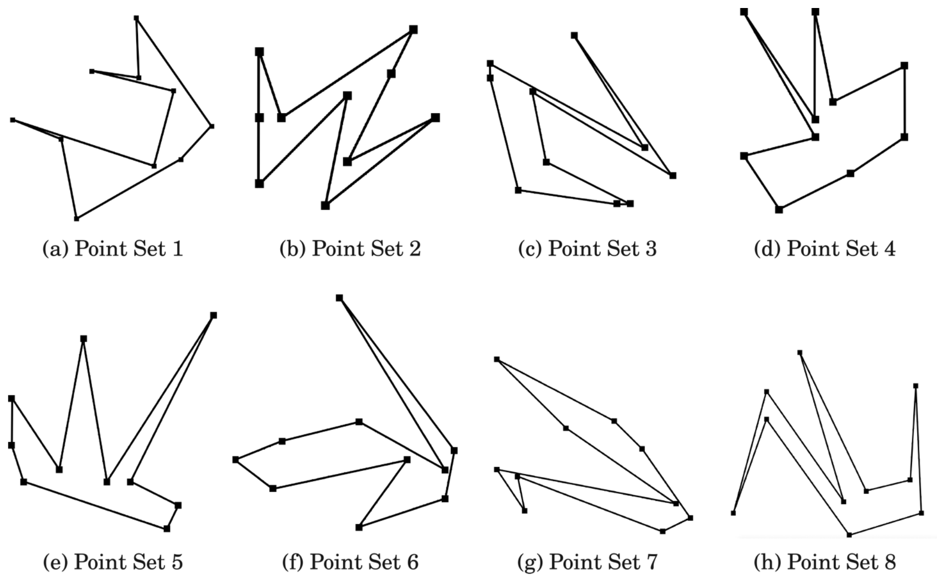


Fig. 22. Optimal minimal area polygons generated by RAND_MINAP and verified using PERMUTE_REJECT for various point sets of size 10.

algorithms work on fixed sequences formed out of the algorithmic rules. Being a randomized metaheuristic, our strategy enjoys the flexibility of choosing the best candidate from the solution space. On the contrary, one of the main advantages, as well as disadvantages, 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 algorithm's ability to further explore and determine 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 was always found to perform better than its counterparts for all the input data used in our tests (see Figures 14–20). Since there is no method for generating optimal MINAP to date, we firmly believe that heuristics such as RAND_MINAP, with the ability of giving an upper bound on optimal MINAP, 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 a convex position where $3 \leq k \leq n\text{-}2$, there exists at least one permuted sequence for which the RAND_MINAP algorithm will generate an optimal MINAP of S.*

Conjecture 6.1 remains to be proved or disproved.

ACKNOWLEDGMENTS

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

REFERENCES

Thomas Auer and Martin Held. 1998. RPG—heuristics for the generation of random polygons. In *Proceedings of the 8th Canadian Conference on Computational Geometry*. 38–44.

Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. *Computational Geometry: Algorithms and Applications* (3rd ed.). Springer-Verlag TELOS, Santa Clara, CA.

James E. Boyce, David P. Dobkin, Robert L. (Scot) Drysdale III, and Leo J. Guibas. 1985. Finding extremal polygons. *SIAM Journal on Computing* 14, 1, 134–147. DOI:http://dx.doi.org/10.1137/0214011

Linda Denee. 1988. Polygonizations of point sets in the plane. *Discrete and Computational Geometry* 3, 1, 77–87. DOI:http://dx.doi.org/10.1007/BF02187898

David Eppstein, Mark Overmars, Günter Rote, and Gerhard Woeginger. 1992. Finding minimum area k-gons. *Discrete and Computational Geometry* 7, 45–58.

Sándor P. Fekete. 1992. *Geometry and the Travelling Salesman Problem*. Ph.D. Thesis. Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON.

S. P. Fekete. 2000. On simple polygonalizations with optimal area. *Discrete & Computational Geometry* 23, 1, 73–110. DOI:http://dx.doi.org/10.1007/PL00009492

Sanchit Goyal. 2010. A survey on travelling salesman problem. *Midwest Instruction and Computing Symposium*. 1–9.

V. Muravitskiy and V. Tereshchenko. 2011. Generating a simple polygonalization. In *Proceedings of the 15th International Conference on Information Visualisation (IV'11)*. IEEE Computer Society, Washington, DC, 502–506. DOI:http://dx.doi.org/10.1109/IV.2011.88

Joseph O'Rourke. 1998. *Computational Geometry in C* (2nd ed.). Cambridge University Press, New York, NY.

Jiju Peethambaran, Amal Dev Parakkat, and Ramanathan Muthuganapathy. 2015. A randomized approach to volume constrained polyhedronization problem. *Journal of Computing and Information Science in Engineering* 15, 1, DOI:http://dx.doi.org/10.1115/1.4029559

Gerhard Reinelt. 2014. TSPLIB database. Retrieved March 27, 2016 from http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

Micha Sharir, Adam Sheffer, and Emo Welzl. 2011. Counting plane graphs: Perfect matchings, spanning cycles, and Kasteleyn's technique. *CoRR* abs/1109.5596.

Helmuth Spaeth. 2014. SPAETH Cluster Analysis Datasets. Retrieved March 27, 2016 from http://people.sc.fsu.edu/~jburkardt/datasets/spaeth/spaeth.html.

Maria Teresa Taranilla, Edilma Olinda Gagliardi, and Gregario Hernandez Penalver. 2011. Approaching minimum area polygonization. In *XVII Congreso Argentino de Ciencias de la Computacin*.

Remco C. Veltkamp. 1995. Boundaries through scattered points of unknown density. *Graphic Models and Image Processing* 57, 6, 441–452. DOI:http://dx.doi.org/10.1006/gmip.1995.1038

Chong Zhu, Gopalakrishnan Sundaram, Jack Snoeyink, and Joseph S. B. Mitchell. 1996. Generating random polygons with given vertices. *Computational Geometry* 6, 5, 277–290.

Received August 2012; revised October 2015; accepted February 2016