Available online at www.sciencedirect.com

ScienceDirect

Procedia Technology 10 (2013) 356 – 364

Procedia Technology

International Conference on Computational Intelligence: Modeling, Techniques and Applications (CIMTA) 2013)

Random Polygon Generation through Convex Layers

Sanjib Sadhu[a,*], Niraj Kumar[a], Bhudev Kumar[a]

[a]*National Institute of Technology, Durgapur-713209, India*

Abstract

In this paper we have designed a randomized algorithm to generate a random polygon P from a given set $S=\{p_1, p_2, \ldots, p_n\}$ of n points lying on a two dimensional plane. As a preprocessing task, we first compute the convex hull layers from the set S in $O(n^2)$ time by modifying the well known Jarvis march algorithm. Next, we execute our algorithm taking the convex hull layers as input to generate random polygons over given point set. This is a Las-Vegas randomized algorithm with $O(n\ell ogn)$ time which is much improved over the existing one. We also give a procedure to count the total number of different polygons that can be generated by our algorithm. This number help us to calculate the probability of generating a unique polygon in each execution which measure the randomness of our algorithm.

Selection and peer-review under responsibility of the University of Kalyani, Department of Computer Science & Engineering

Keywords: Simple Polygon, Convex Hull, Convex Layers, End Visibility, Directed Acyclic Graph.

1. Introduction

The problem of generating simple polygon draws attention of researcher community extensively because of its wide area of application, both theoretical applicability as well as for testing and verification of various geometry algorithms. Generating sufficient test cases from user world for testing computational geometry algorithms, is a challenging task. To ease testing procedure, an algorithm can be designed to randomly generate polygons.

Unfortunately, no polynomial-time algorithm is known to generate all possible simple polygons with a given vertex set. All previous works are based on applying some heuristic or generating a specific class of polygon. Although there exists a lot of heuristics to generate a random polygon from n points, still it is an open problem to generate it uniformly at random, i.e. generating a polygon with probability $(\frac{1}{j})$ if there exists j simple polygons on set S in total. Much efforts has also been applied to generate a specific class of polygons like monotone polygons[1], star shaped polygons [2]. In this paper, we have designed a new algorithm to generate a random polygon from a set $S=\{p_1, p_2, \ldots, p_n\}$ of n points which lie on a 2-dimensional plane. Our main objective is to reduce the time complexity of the existing heuristic. Our algorithm is based on the computation of convex layers of the given point set S and the visibility of the two edges lying on the two successive convex hull layers CL_i and CL_{i+1}.

* Corresponding author
E-mail address: sanjib.sadhu@cse.nitdgp.ac.in

Selection and peer-review under responsibility of the University of Kalyani, Department of Computer Science & Engineering
doi:10.1016/j.protcy.2013.12.371

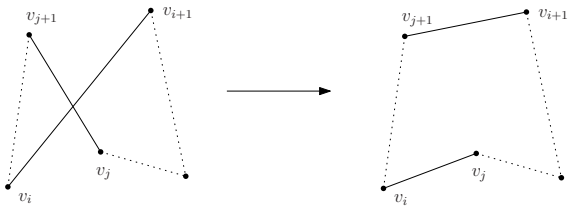

Fig. 1. An example of a 2-opt move [2].

Recently, in our previous work [3], we designed and implemented a heuristic, known as "GRP_CH" heuristic for generating a random simple polygon from a set of n points which takes $O(n^3)$ time. The running time complexity to generate a random polygon has been improved to $O(n\ \ell ogn)$ in this work. Therefore, the presented algorithm runs faster than the "GRP_CH" [3] and the "2-Opt Move" [2] heuristics.

The organisation of paper is as follows. In section 2, we briefly describe the related work. Section 3 defines the basic terminologies. In section 4, we state our problem. The preprocessing task and the algorithm to generate a random polygon are described in Section 4.2 and 4.3 respectively. In Section 4.2, we briefly describe how the Jarvis March can be modified to generate the convex layers of given point set required for subsequent processing. In section 5, we sketch a procedure to count total number of possible polygons that can be generated by this algorithm so that we can estimate probability of generating a new polygon. Finally, we conclude in Section 6 with future work.

2. Related Work

From 1992, the generation of geometric objects became an interesting research topic to the researchers for its different applications. Epstein [4] studied random generation of triangulation. Zhu [1] designed an algorithm to generate an x-monotone polygon on a given set of vertices uniformly at random. A heuristic [5] for generating simple polygons was investigated in 1991. However, the vertices move while creating a polygon in their algorithm. The "2-Opt Move" heuristic was first proposed to solve the traveling salesman problem by J.van Leeuwen et al. [6]. In 1996, Thomas Auer et al. [2] presented a study of all heuristics present at that time and reported a variety of comparison among them. The existing heuristics for random polygon generating simple polygon are discussed below:

- Permute & Reject
  The algorithm "Permute & Reject" [2] creates a permutation of S and check whether this permutation corresponds to a simple polygon. If the polygon is simple then it is output; otherwise a new polygon is generated. Obviously, the actual running time of this method mainly depends on how many polygons need to be generated in order to encounter a simple polygon. Clearly, "Permute & Reject" produce all possible polygons with a uniform distribution, but this algorithm is not applicable to anything but extremely small set of points.
- Steady Growth
  "Steady Growth" has been designed by Thomas Auer et al [2]. As initialization, "Steady Growth" randomly select three points $p_1,\ p_2,\ p_3$ from the set S such that no other point of S lies within convex hull, CH($p_1,\ p_2,\ p_3$). This convex hull is taken as a start polygon. In each of the following iteration steps a point p is chosen in such a way that by appending p to the polygon, it's convex hull again does not contain any further points of the point set. Then an edge (u, v) of the polygon that is completely visible from p is searched and replaced by the chain (u, p, v). This way the polygon is extended with the point p. By using "Steady Growth", one can compute a simple polygon in at most $O(n^2)$ time. Unfortunately "Steady Growth" does not generate every possible polygon on S.
- 2-Opt Move
  It was also designed by Thomas Auer et. al.[2]. This algorithm first generates a random permutation of S, which again is regarded as the initial polygon P. Any self intersections of P are removed by applying so called "2-Opt Move". Every "2-Opt move" replaces a pair of intersecting edges (v_i,v_{i+1}), (v_j,v_{j+1}) with the edges (v_{j+1},v_{i+1}) and (v_j,v_i) as shown in Fig 1. In this application, at each iteration of the algorithm one pair of intersecting edge is chosen at random and the intersection is removed. Leeuwen et al.[6] has shown that for obtaining a simple

polygon, at most $O(n^3)$ "2-Opt move" are required to be applied. Thus, an overall time complexity of $O(n^4)$ can be achieved and it is very high. The "2-Opt Move" heuristic will produce all possible polygons, but not with a uniform distribution[1]. However, this is reported as the best heuristic[2] among all the existing ones in the sense that it produces variety of different simple polygons from the point set S.

- GRP_CH Heuristic
  In our previous work [3], the convex hull, $CH(S)$ was computed from a given set S of n points. This convex Hull was taken as an initial polygon P. Then a point $q \in \{S - CH(S)\}$ was selected randomly. After that the set of edges which are completely visible from the point q, were computed. One of such visible edge e is selected randomly for deletion and its two end points v_1 and v_2 are connected with the point q after deleting that edge e. After such modification, the number of edges in the polygon P increases by one. This process is repeated until all the points within the convex hull $CH(S)$ are exhausted. Finally a random polygon P is generated. This algorithm takes $O(n^3)$ time.

3. Preliminaries

We begin by presenting some definitions. We use the term points and vertices interchangeably throughout the paper. A polygon P is said to be **simple** [7] if it consists of straight, non-intersecting line segments, called edges that are joined pair-wise to form a closed path. The adjacent edges of polygon meet only at their common end point known as vertices. An edge connecting two points p and q are denoted by $e(p,q)$. The x-coordinate and y-coordinate of a point p are denoted by $x(p)$ and $y(p)$ respectively. Here and throughout the paper, unless qualified otherwise, we take polygon to mean simple polygon on the plane.

A subset S of the plane is called **convex** [7] if and only if for any pair of points $(p,q) \in S$ the line segment $\ell(p,q)$ is completely contained in S. The **Convex Hull** $CH(S)$ of a point set S is the smallest convex set that contains S. Convex hull of a point set S is represented by set of vertices that defines hull edges. Many algorithms have been presented [8] [9] [10] [11] for finding convex hull. Lower bound to find $CH(S)$ of a given point set with n points, is $\Omega(n\ell ogn)$ [12]. One of the output sensitive algorithm to find convex hull is Jarvis March [11]. It is based on the idea of finding hull edges instead of hull vertices.

A point p is said to be **visible** [13] from a point q if the line segment $\ell(p,q)$ does not intersect any other line segment or does not pass through a third point r.

Definition 1 (End Visibility)**.** *The "End Visibility" of two edges ($e(p,q)$ and $e(u,v)$) are defined with respect to their end points. Edges $e(u,v)$ and $e(p,q)$ are said to be "End Visible" if and only if u is visible from p(or q) and v is visible from q(or p).*

Fig 2 shows that the edge $e(p,q)$ and $e(u,v)$ are "End Visible", whereas the edges $e(p,q)$ and $e(u,x)$ are not "end visible". Similarly, the two edges 9 and 11 are "End Visible", although no points of these two edges, except their end points, are visible to each other.

4. Problem Statement

We are given a set $S = \{p_1, p_2, \ldots, p_n\}$ of n points lying on a 2-dimensional plane. The objective is to generate a random polygon with less time complexity. Our algorithm starts after a preproceesing is done.

4.1. Assumption

For two points p and q, the point p will be at the left of point q if $x(p) < x(q)$ or, $y(p) < y(q)$ when $x(p) = x(q)$. The point set S lie on a plane in general position, that means no three points or vertices are collinear.

4.2. Preprocessing task

In preprocessing phase, we compute set of convex layers $CL(S) = (CL_1, CL_2, \ldots, CL_m)$ of point set S, where m(say) is the number of convex layers(Refer to Fig 3). To find $CL(S)$, Jarvis march can be modified as follows. We apply

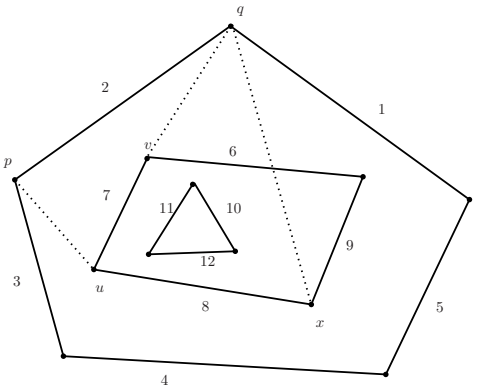

Fig. 2. The edges $e(p,q)$ and $e(u,v)$ are end visible, whereas the edges $e(p,q)$ and $e(u,x)$ are not end visible.

Jarvis's March algorithm to find out the Convex Hull $CH(S)$ of the point set S. This becomes the outermost Convex Layer CL_1. Then, we delete all the points $p \in CL_1$ from S and update S. Again we apply the Jarvis March algorithm on the updated point set S to get the next convex layer CL_2 and so on until the set S becomes Φ. If S contains a single point p or two points p and q, we take that single point or line segment $\ell(p,q)$ as a convex layer. This preprocessing will take $O(n^2)$ time. The algorithm 1 shows the modified Jarvis algorithm.

Algorithm 1: Generating Convex Layers $CL(S)$

Input : A Set $S = \{p_1, p_2, \ldots, p_n\}$ of n points lying on a 2-dimensional plane
Output: The list of m Convex Layers $CL(S) = \{CL_1, CL_2, \ldots, CL_m\}$

$m \leftarrow 1$;
repeat
  Find the lowest point(smallest y coordinate) from S;
  Let i_0 be its index, and set $i \leftarrow i_0$;
  Take an extra point $p_0(-\infty, y)$ at extreme left;
  Consider $e(p_0, p_i)$ as hull edge;
  `/* Do not add this hull edge in convex layer` CL_m `*/`
  repeat
    `/*` j `is an index of a point` $p \in S$ `*/`
    for *each* $j \neq i$ **do**
      Compute counterclockwise angle θ from previous hull edge;
      Let k be the index of the point with the smallest θ;
      Add $e(p_i, p_k)$ as a hull edge in CL_m ;
      $i \leftarrow k$;
  until $i \neq i_0$;
  $S \leftarrow S - \{Vertices\ on\ CL_m\}$;
  $m \leftarrow (m+1)$;
until $S \neq \Phi$;
return $CL(S)$;

4.3. Algorithm

The preprocessing step is computed only once and remains same unless the point set S changes.
Observation: With respect to an edge $e(p,q) \in CL_i$, there must exist at least one "end visible" edge $e(u,v) \in CL_{i+1}$.

The following algorithm 2, if executed, generates a random polygon. This algorithm takes as an Input the set of convex layers $CL(S)$ (computed in preprocessing step). An edge $e(p,q)$ is chosen randomly from a layer CL_k(say). Then compute the set **EV(e(p,q))** which contains the edges from layer CL_{k+1} which are "**end visible**" with respect to the edge "$e(p,q)$". Now, randomly choose an edge, say $e(u,v)$, from the set "$EV(e(p,q))$". Edges $e(p,q)$ and $e(u,v)$ are removed from CL_k and CL_{k+1} respectively and connect their end points. The process is continued until we reach to the inner most layer. Finally, a random polygon P will be generated. The Fig 3 shows how to generate a random polygon P from a set S after computing its convex layers.

Algorithm 2: Generating Random Polygon P

Input : Set of convex layers $CL(S) = \{CL_1, CL_2, \ldots, CL_m\}$
Output: Polygon P

$i \leftarrow 1$;
$P \leftarrow CL_i$;
while $(i \neq m)$ **do**
  Select an edge $e(p,q)$ randomly from CL_i;
  Find the tangents to the CL_{i+1} from p and q of CL_i;
  Compute the set of edges $EV(e(p,q)) \in CL_{i+1}$ which are "**end visible**" with respect to $e(p,q) \in CL_i$;
  Randomly select an edge $e(u,v) \in EV(e(p,q))$;
    /* Delete the edges $e(p,q)$ and $e(u,v)$ from CL_i and CL_{i+1} respectively; */
  $CL_{i+1} \leftarrow CL_{i+1} - e(u,v)$;
  $CL_i \leftarrow CL_i - e(p,q)$;
    /* Connect the points p with u and q with v respectively; */
  $P \leftarrow \{P - e(p,q) + CL_{i+1} + e(p,u) + e(q,v)\}$;
  $i \leftarrow (i+1)$;
return P;

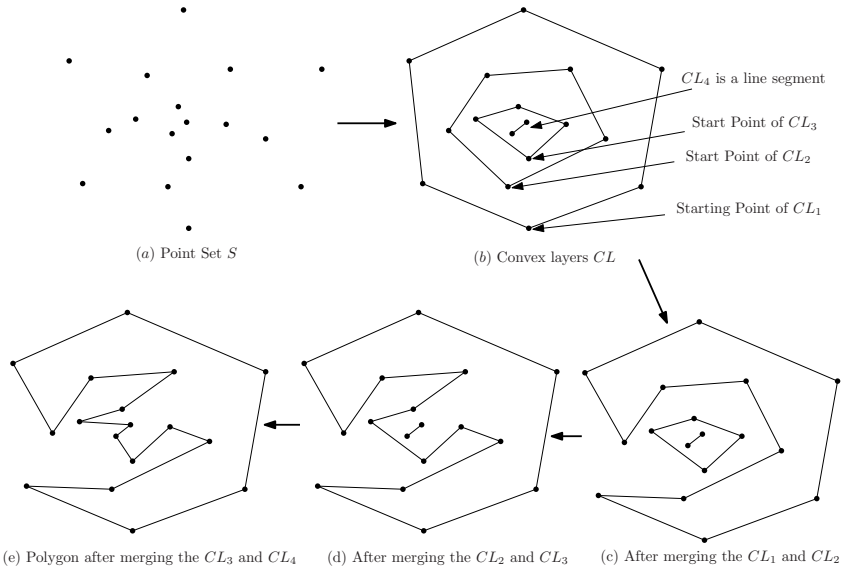

Fig. 3. (a) A given Point Set S. (b) The Convex Hull Layers. (c),(d),(e) The steps to generate a random polygon P

4.4. Analysis

If we store all the vertices of a convex layer CL_i in counterclockwise or clockwise direction, two tangents from a point p outside the CL_i can be computed in $O(\ell og\ n)$ time using binary search. Hence from the endpoints p and q of an edge $e(p,q) \in CL_i$, we can compute all the four tangents to its next inner convex layers CL_{i+1} in $O(\ell og\ n)$ time. Therefore we can compute the set of "End Visible" edges in CL_{i+1} from an edge $e(p,q)$ in $O(\ell og\ n)$ time. Now we have to compute the set of "End Visible" edges for one edge in each layer of $CL(S)$, except the last one. Since the number of layers can be $O(n)$, overall time complexity to get a random polygon is $O(n\ell og\ n)$. The space complexity of the algorithm 2 is $O(n)$.

4.5. Limitation

The Fig 4 shows a polygon which cannot be generated using the algorithm 2. The polygon which looks like star may not be generated using this algorithm. This is the limitation of our heuristic whose removal will be a future work.

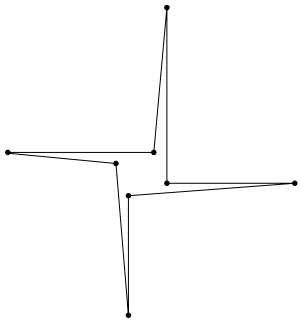

Fig. 4. This polygon cannot be generated using the Algorithm 2

5. Counting the number of polygons generated

It is likely that the same set of edges be selected for deletion in different runs of algorithm, which results in generating the same polygon. Here, we sketch a procedure to count all possible polygons that can be generated by our algorithm, so that we can estimate the probability of generating distinct polygon in each run of the algorithm 2. For counting procedure, we represent the "end visibility" of the edges in $CL(S)$ by a DAG (directed acyclic graph) data structure. Each convex layer CL_i(where, $i \in \{1,2,\ldots,m\}$) in $CL(S)$ corresponds to a *level i* in DAG. In *level* 0, we add only a single dummy node which is connected with every node in *level* 1. An edge e in convex layer CL_i corresponds to a node v in *level i* of DAG. There will be a directed edge from a node $v_j \in$ *level k*(representing edge $e_j \in CL_k$) to a node $v_i \in$ *level* $(k+1)$(representing edge $e_i \in CL_{k+1}$) in DAG if the edge $e_i \in EV(e_j)$, where $EV(e_j)$ is the set of edges on CL_{k+1} which are "end visible" with respect to the edge $e_j \in CL_k$. The edges in DAG are always directed from a node in upper *level* to a node in next lower *level*. A node $v \in$ *level i* must be connected with some nodes of *level* $(i+1)$ which are termed as **next_nbr(v)**. All edges $e \in CL_i$ corresponds to the nodes lying on the same level in DAG and are termed as **immediate_nbr** to each other. So, in Fig 5(a) node $next\_nbr(2) = \{6,7\}$, while $immediate\_nbr(2) = \{1,3,4,5\}$. The Fig 5(a) depicts the DAG corresponding to polygon of Fig 2.

For a given node v, its *immediate_nbr* and *next_nbr* nodes are stored in the array $immediate\_nbr[v][1..n]$ and $next\_nbr[v][1..n]$ respectively. Its *immediate_nbr* can be computed by traversing corresponding convex layer in $O(k)$ time, where k is number of edges in that convex layer. We have already explained in Section 4.4, that the $next\_nbr(v)$ of any node v can be computed in $O(\ell og\ n)$ time. Traversal of an edge $e(v_i,v_j)$ in DAG corresponds to deletion of corresponding edge pair (e_i,e_j) in the convex layer set (here nodes v_i and v_j are termed as **source node** and **destination node** respectively). We select exactly one edge between each adjacent level from *level* 0 to *level m* in such a way that no two edges have a common endpoint and this traversal corresponds to a polygon. If we reach on *level i* through an edge $\overrightarrow{e(v_1,v_2)}$, one of the $immediate\_nbr(v_2)$ is selected as source to reach the next *level* $(i+1)$. This process continues until we reach the last level. The purpose of root node is to start traversal from any of nodes

of first level. Hence we will not consider the edge traversed from *level* 0 to *level* 1 in DAG. For example, let the path being traversed is $(0 \rightarrow 1 \rightarrow 6 \rightarrow 8 \rightarrow 12)$ which implies that the edge pairs (e_1, e_6) and (e_8, e_{12}) are deleted (shown in bold in Fig 5(b)) and hence the polygon is generated as in Fig 5(b). The collection of all such deletions represents all possible traversals of DAG and hence, the total number of possible polygons that can be constructed by our algorithm.

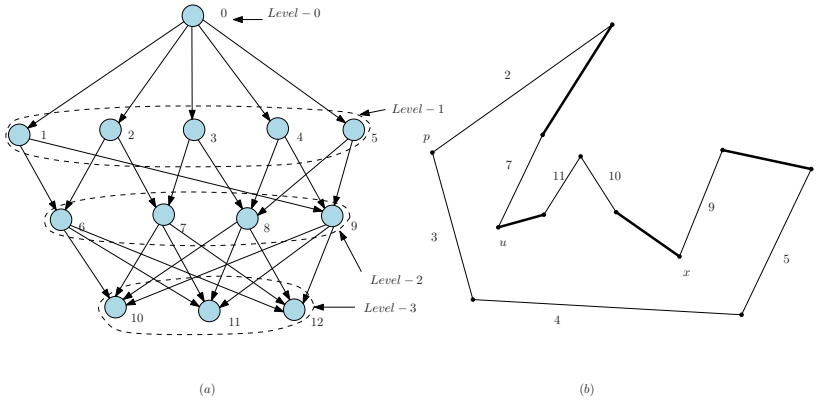

Fig. 5. (a)DAG corresponding to Fig 2 (b) Representation of result of a traversal in DAG

Observation: One edge(node) from outermost layer CL_1 (*level* 1) and one from the innermost layer CL_m(*level m*) is required to be deleted whereas exactly two edges(nodes) are to be deleted from each of the remaining layers CL_2, $CL_3, \ldots,$ CL_{m-1} (*level* $2, \ldots,$ *level m*) to get a polygon from the convex layers $CL(S)$.

Definition 2 (Path)**.** *A path traversed from a node u* $\in$ *level i to node v* $\in$ *level j indicates a set of selected edges with exactly one edge between each adjacent levels in such a way that the end points of the selected edges in adjacent levels are not common and the path starts from node u and ends in v.*

Now, we associate two attributes with each node, namely the contribution to parent, denoted by **cont_par** and contribution to sibling, denoted by **cont_sib**. The **cont_sib(v)** is total number of possible paths from the node $v \in$ *level i* to all the nodes in last *level m*. The **cont_par(v)** is defined as the sum of total number of possible paths from each node $u \in$ *immediate_nbr*(v) to each node in last level *m*. To compute *cont_sib*(v) (where $v \in$ *level i*) *v* is selected as the source node to reach a node in *level* $(i+1)$, whereas in *cont_par*(v), the node *v* was the destination node to arrive on *level i* from a node in *level* $(i-1)$. In Fig 5(a), if node 6 has been selected as destination, then one of the *immediate_nbr* node (7, 8 *or* 9) will be selected as source for next edge to be traversed. Then the *cont_par* value for node 6 will be 9 which is all possible paths from node 7, 8 *and* 9 to the last level. The node 8 will contribute the *cont_sib*(8) to each of its siblings (6, 7 *and* 9) when that sibling was selected as destination node from its upper level. It can be observed that, nodes in outermost layer have *cont_sib* only. For the nodes of last level, *cont_par* value has been initialized to 1 since we are already in last level and the *cont_sib* value initialized to 0 because no node can be selected as source from this level. The algorithm 3 describes the counting procedure in which the *cont_par* value is propagated in bottom-up manner. So,the probability *Pr* that a unique polygon *P* will be generated from a given point set *S* is given by $Pr = \{1 - (\frac{k}{count})\}$, where *k* is number of different polygons generated so far and *count* is total number of polygons that can be generated by this algorithm. Thus, for the first time execution of the algorithm, this probability *Pr* is one and it decreases in successive execution.

A node $v \in$ *level i* can have $O(n)$ number of *immediate_nbr* nodes and *next_nbr* nodes. Also the number of convex layers *m* in $CL(S)$ is $O(n)$. Therefore the $for\ loop$ in line 5 may take $O(n^3)$ time. So the total time complexity of "CountPolygon()" procedure is $O(n^3)$. The space complexity is mainly for storing the *next_nbr* and *immediate_nbr* array. Hence the space complexity will be $O(n^2)$.

Algorithm 3: CountPolygon(DAG, $next\_nbr[1..n][1..n]$, $immediate\_nbr[1..n][1..n]$, m)

Input : A DAG with $m+1$ levels $\{level\ 0, level\ 1, level\ 2, \ldots, level\ m\}$; Array $next\_nbr[1..n][1..n]$ and $immediate\_nbr[1..n][1..n]$ containing the information $next\_nbr$ and $immediate\_nbr$ of each node; m is the number of convex layers.

Output: Total number of possible polygons(returned as *count*) that can be generated by algorithm 2

```
1  count ← 0
2  for each node v ∈ level m do
3  |  cont_par(v) ← 1;
4  |  cont_sib(v) ← 0;
5  for i = (m − 1) to 2 do
6  |  sum ← 0;
7  |  sum1 ← 0;
8  |  for every node v1 ∈ level i do
9  |  |  for every node v2 ∈ next_nbr(v1) do
10 |  |  |  sum ← sum + cont_par(v2);
11 |  |  cont_sib(v1) ← sum;
12 |  for every node v1 ∈ level i do
13 |  |  for every node v2 ∈ immediate_nbr(v1) do
14 |  |  |  sum1 ← sum1 + cont_sib(v2)
15 |  |  cont_par(v1) ← sum1
16 sum ← 0
17 for each node v1 ∈ level 1 do
18 |  for every node v2 ∈ next_nbr(v1) do
19 |  |  sum ← sum + cont_par(v2)
20 |  cont_sib(v1) ← sum;
21 for each node v ∈ level 1 do
22 |  count ← Σ(cont_sib(v));
23 return count;
24   /* count indicates the total number of possible polygons */
```

6. Conclusion and Future Work

We have presented a randomized algorithm for generating random simple polygons. Our algorithm is based on a preprocessing task of computing the convex layers of the given point set S. The Convex Layers $CL(S)$ has been generated using modified version of the Jarvis March Algorithm. After that a random polygon is generated by deleting and inserting the edges in the convex layers. Our randomized algorithm takes $O(n\ell ogn)$ time which has been improved over the existing heuristics. We have shown how to count the number of unique polygon generated by our algorithm.

From a theoretical point of view, it remains an open problem to generate polygons on a given set of points uniformly at random. The algorithm designed here has a limitation and the removal of that limitation will be a future work.

References

[1] C. Zhu, G. Sundaram, J. Snoeyink, J. S. B. Mitchel, Generating random polygons with given vertices, Computational Geometry: Theory and Application (1996) 277–290.

[2] T. Auer, M. Held, RPG: Heuristics for the generation of random polygons, in: Proc. 8th Canadian Conference Computational Geometry, 1996, pp. 38–44.

[3] S. Sadhu, S. Hazarika, K. K. Jain, S. Basu, T. De, GRP_CH heuristic for generating random simple polygon, in: International Workshop on Combinatorial Algorithm (IWOCA), 2012, pp. 293–302.
[4] P. Epstein, J. Sack, Generating triangulation at random, ACM Transaction on Modeling and Computer Simulation VOL 4 NO 3 (1994) 267–278.
[5] J. O'Rourke, M. Virmani, Generating random polygons, in: Technical Report 011,CS Dept., Smith College, Northampton, MA 01063, 1991, pp. 38–44.
[6] J. V. Leeuwen, A. A. Schoone, Untangling a travelling salesman tour in the plane, in: 7th Conference Graph-theoretic Concepts in Computer Science, 1982, pp. 87–98.
[7] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry : Algorithms and Applications 3rd ed, Springer-Verlag TELOS Santa Clara, CA, USA, 2008.
[8] R. L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, Information Processing Letter (1972) 132–133.
[9] F. P. Preparata, S. J. Hong, Convex hulls of finite sets of points in two and three dimensions, Commun. ACM (1977) 87–93.
[10] F. P. Preparata, M. I. Shamos, Computational Geometry - An Introduction, Springer-Verlag.2nd printing, 1998.
[11] R. A. Jarvis, On the identification of the convex hull of a finite set of points in the plane, Information Processing Letter (1973) 18–21.
[12] A. C. Yao, A lower bound to finding convex hulls., J. ACM (1981) 780–787.
[13] S. K. Ghosh, Visibility Algorithm in the Plane, Cambridge University press, 2007.