Algorithmic Enumeration of Surrounding Polygons

Katsuhisa Yamanaka[1], Takashi Horiyama[2], Yoshio Okamoto[3,4], Ryuhei Uehara[5], and Tanami Yamauchi[1]

1 **Iwate University, Japan**
yamanaka@cis.iwate-u.ac.jp, tanami@kono.cis.iwate-u.ac.jp
2 **Saitama University, Japan**
horiyama@al.ics.saitama-u.ac.jp
3 **The University of Electro-Communications, Japan**
okamotoy@uec.ac.jp
4 **RIKEN Center for Advanced Intelligence Project, Japan**
5 **Japan Advanced Institute of Science and Technology, Japan**
uehara@jaist.ac.jp

Abstract

We are given a set S of points in the Euclidean plane. We assume that S is in general position. A simple polygon P is a *surrounding polygon* of S if each vertex of P is a point in S and every point in S is either inside P or a vertex of P. In this paper, we present an enumeration algorithm of the surrounding polygons for a given point set. Our algorithm is based on reverse search by Avis and Fukuda and enumerates all the surrounding polygons in polynomial delay.

1 Introduction

Enumeration problems are fundamental and important in computer science. Enumerating geometric objects are studied for triangulations [2, 3, 9], non-crossing spanning trees [9], pseudoline arrangements [20], non-crossing matchings [19], unfoldings of Platonic solids [8], and so on. In this paper, we focus on an enumeration problem of simple polygons of a given point set. We are given a set S of n points in the Euclidean plane. A *surrounding polygon* of S is a simple polygon P such that each vertex of P is a point in S and every point in S is either inside the polygon or a vertex of the polygon. A surrounding polygon P of S is a *simple polygonization*[1] of S if every point of S is a vertex of P. See Figure 1 for examples.

Simple polygonizations are studied from various perspectives. As for the counting, the current fastest algorithm was given by Marx and Miltzou [10], and it runs in $n^{O\sqrt{n}}$ time when a set of n points is given. It is still an outstanding open problem to propose a polynomial-time algorithm that counts the number of simple polygonizations of a given point set [12]. Much attention has been paid for combinatorial counting, too. A history on the lower and upper bounds is summarized by Demaine [4] and O'Rourke *et al.* [14]. Let b_P be the number of simple polygonizations of a point set P, and let b_n be the maximum of b_P among all the sets P of n points. The current best lower and upper bounds for b_n are 4.64^n [5] and 54.55^n [15], respectively.

Another research topic is a random generation of simple polygonizations. Since no polynomial-time counting algorithm is known for simple polygonizations, it seems to be a hard task to propose a polynomial-time algorithm that uniformly generates simple polygonizations. However, uniformly random generations are known for restricted classes: x-monotone polygons [21] and star-shaped polygons [16]. These uniform random generations are based

[1] The simple polygonizations are also called spanning cycles, Hamiltonian polygons, and planar traveling salesman tours.

35th European Workshop on Computational Geometry, Utrecht, The Netherlands, March 18–20, 2019.
This is an extended abstract of a presentation given at EuroCG'19. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

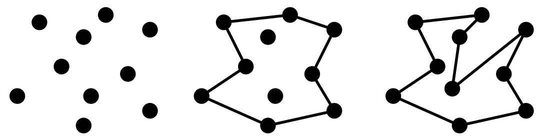

Figure 1 (a) A point set S. (b) A surrounding polygon of S. (c) A simple polygonization of S.

on counting. For general simple polygonizations, heuristic algorithms are known [1, 17, 21]. Those algorithms efficiently generate simple polygons, but not uniformly at random.

On the other hand, nothing is known for the problem of enumerating all the simple polygonizations, as mentioned in [18]. A trivial enumeration is to generate all the permutations of given points, then output only simple polygonizations. However, this is clearly a time-consuming algorithm. It is an interesting and challenging question whether all the simple polygonizations of a given point set can be enumerated efficiently (for example, in output-polynomial time[2] or in polynomial delay[3]).

As the first step toward the question, we consider the problem of enumerating the surrounding polygons of a given point set S. From the definition, the set of surrounding polygons of S includes the set of simple polygonizations of S. We show that, for this enumeration problem, the reverse search by Avis and Fukuda [2] can be applied. First, we introduce an "embedding" operation: deleting a vertex from a surrounding polygons and putting it inside the polygon. Then, using this operation, we define a rooted tree structure among the set of surrounding polygons of S. We show that, by traversing the tree, one can enumerate all the surrounding polygons. The proposed algorithm enumerates them in polynomial delay.

Due to space limitation, all the proofs and some details are omitted.

2 Preliminaries

A *simple polygon* is a closed region of the plane enclosed by a simple cycle of edges. Here, a simple cycle means that two adjacent line segments intersect only at their common endpoint and no two non-adjacent line segments intersect. An *ear* of a simple polygon P is a triangle such that one of its edges is a diagonal of P and the remaining two edges are edges of P. The following theorem for ears is known.

▶ **Theorem 2.1** ([11]). *Every simple polygon with $n \geq 4$ vertices has at least two non-overlapping ears.*

Let S be a set of n points in the Euclidean plane. We assume that S is in general position, i.e., no three points are collinear. The *upper-left point* of S is the point with the minimum x-coordinate. If a tie exists, we choose the point with the maximum y-coordinate among them. A *surrounding polygon* of S is a simple polygon such that every point in S is either inside the polygon or a vertex of the polygon. For example, the convex hull of S is a

[2] The running time of an enumeration algorithm A for an enumeration problem is *output-polynomial* if the total running time of A is bounded by a polynomial in the input and output size of the problem.

[3] The running time of an enumeration algorithm A for an enumeration problem is *polynomial-delay* if the delay, which is the maximum computation time between any two output, of A is bounded by a polynomial in the input size of the problem.

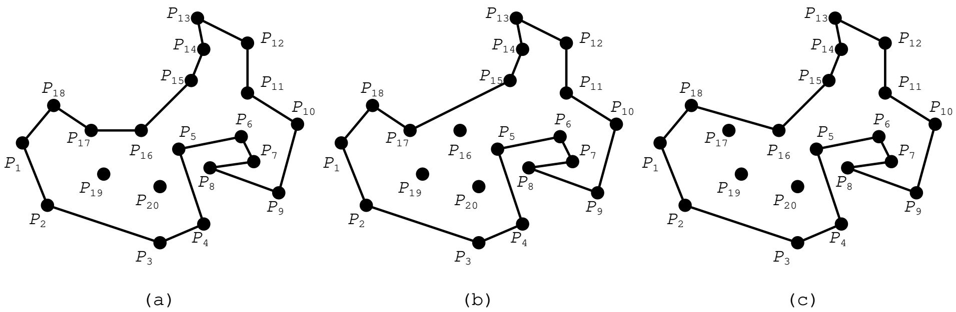

Figure 2 (a) A surrounding polygon, where $p_6, p_7, p_{11}, p_{14}, p_{15}, p_{16}$, and p_{17} are embeddable. (b) The surrounding polygon obtained by embedding p_{16}. The point p_{16} is embedded inside the polygon. (c) The parent of the polygon in (a), which is obtained by embedding p_{17}.

surrounding polygon of S. Note that any surrounding polygon has the upper-left point in S as a vertex.

We denote by $\mathcal{P}(S)$ the set of surrounding polygons of S, and denote by $\mathsf{CH}(S)$ the convex hull of S. We denote a surrounding polygon of S by a (cyclic) sequence of the vertices in the surrounding polygon. Let $P = \langle p_1, p_2, \ldots, p_k \rangle$ be a surrounding polygon of S. Throughout this paper, we assume that p_1 is the upper-left point in S, the vertices on P appear in counterclockwise order, and the successor of p_k is p_1. Let p be a vertex of a surrounding polygon P of S. We denote by $\mathsf{pred}(p)$ and $\mathsf{succ}(p)$ the predecessor and successor of p on P, respectively.

3 Family tree

Let S be a set of n points in the Euclidean plane, and let $\mathcal{P}(S)$ be the set of surrounding polygons of S. In this section, we define a tree structure over $\mathcal{P}(S)$ such that its nodes correspond to the surrounding polygons. To define a tree structure, we first define the parent of a surrounding polygon using the "embedding operation" defined below. Then, using the parent-child relationship, we define the tree structure rooted at $\mathsf{CH}(S)$.

Now, we introduce some notations. Let $P = \langle p_1, p_2, \ldots, p_k \rangle$ be a surrounding polygon of S. Recall that p_1 is the upper-left vertex on P and the vertices on P are arranged in the counterclockwise order. We denote by $p_i \prec p_j$ if $i < j$ holds, and we say that p_j is *larger than* p_i. The vertex p of P is *embeddable* if the triangle consisting of $\mathsf{pred}(p)$, p, and $\mathsf{succ}(p)$ does not intersect the interior of P. See examples in Figure 2(a). In the figure, $p_6, p_7, p_{11}, p_{14}, p_{15}, p_{16}$, and p_{17} are embeddable.

▶ **Lemma 3.1.** *Let S be a set of points, and let P be a surrounding polygon in $\mathcal{P}(S) \setminus \{\mathsf{CH}(S)\}$. Then, P has at least one embeddable vertex.*

Now, let us define an operation that makes another surrounding polygon from a surrounding polygon. Let p be an embeddable vertex on P. An *embedding operation* is to remove the two edges $(\mathsf{pred}(p), p)$ and $(p, \mathsf{succ}(p))$ and insert the edge $(\mathsf{pred}(p), \mathsf{succ}(p))$. Intuitively, an embedding operation "embeds" a vertex into the interior of P. See Figure 2.

We denote by $\mathsf{larg}(P)$ the largest embeddable vertex on P. The *parent* of P, denoted by $\mathsf{par}(P)$, is the polygon obtained by embedding $\mathsf{larg}(P)$ on P. Note that $\mathsf{par}(P)$ is also a surrounding polygon of S. By repeatedly finding the parents from P, we obtain a sequence of surrounding polygons. The *parent sequence* $\mathsf{PS}(P) = \langle P_1, P_2, \ldots, P_\ell \rangle$ of P is a sequence of

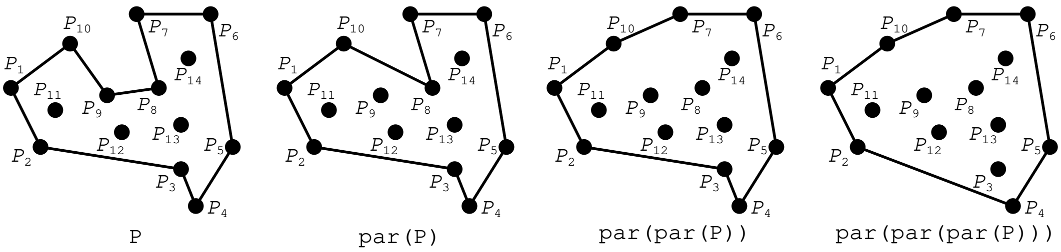

Figure 3 A parent sequence.

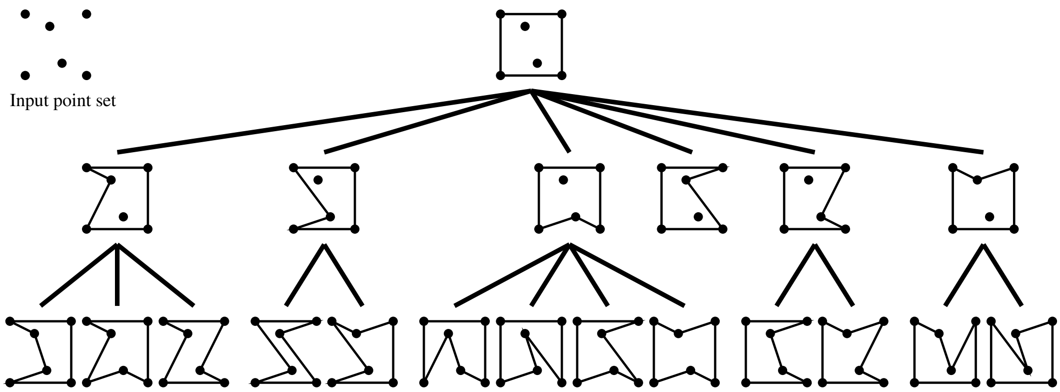

Figure 4 An example of a family tree.

surrounding polygons such that the first polygon is P itself and P_i is the parent of P_{i-1} for each $i = 2, 3, \ldots, \ell$. See Figure 3. As we can see in the following lemma, the last polygon in a parent sequence is always $\mathsf{CH}(P)$.

▶ **Lemma 3.2.** *Let S be a set of n points in the Euclidean plane, and let P be a surrounding polygon in $\mathcal{P}(S) \setminus \{\mathsf{CH}(S)\}$. The last polygon of $\mathsf{PS}(P)$ is $\mathsf{CH}(S)$.*

From Lemma 3.2, for any surrounding polygon, the last polygon of its parent sequence is the convex hull. By merging the parent sequences for all surrounding polygons in $\mathcal{P}(S)$, we have the tree structure rooted at $\mathsf{CH}(S)$. We call such a tree the *family tree*. An example of the family tree is shown in Figure 4.

4 Enumeration algorithm

In this section, we present an algorithm that, for a given set S of n points, enumerates all the surrounding polygons in $\mathcal{P}(S)$. In the previous section, we defined the family tree among $\mathcal{P}(S)$. We know that the root of the family tree is the convex hull of S. Hence, we have the following enumeration algorithm. We first construct the convex hull of S. Then, we traverse the (implicitly defined) family tree with depth first search. This algorithm can enumerate all the surrounding polygons in $\mathcal{P}(S)$. To perform the search, we design an algorithm that finds all the children of any surrounding polygon of S. Starting from the root, we apply the child-enumeration algorithm recursively, and then we can traverse the family tree.

To describe how to construct children, we introduce some notations. Let $P = \langle p_1, p_2, \ldots, p_k \rangle$ be a surrounding polygon in $\mathcal{P}(S)$. For an edge (p_i, p_{i+1}) of P and a point p inside P, we denote by $P(p_i, p_{i+1}; p)$ the polygon obtained by removing (p_i, p_{i+1}) and inserting two edges

(p_i, p) and (p, p_{i+1}). Intuitively, this operation is the reverse one of embedding operation. We call it a *dig operation.* Any child of P is described as $P(p_i, p_{i+1}; p)$ for some p, p_i, and p_{i+1}. Hence, for all possible $P(p_i, p_{i+1}; p)$, if we can check whether or not $P(p_i, p_{i+1}; p)$ is a child, then one can enumerate all the children. We have the following observation.

▶ **Lemma 4.1.** *Let P be a surrounding polygon of a set of points. For an edge (p_i, p_{i+1}) of P and a point p inside P, $P(p_i, p_{i+1}; p)$ is a child of P if*

(1) $P(p_i, p_{i+1}; p)$ is a surrounding polygon of S and

(2) $\mathsf{par}(P(p_i, p_{i+1}; p)) = P$ holds.

Note that the condition (2) in Lemma 4.1 can be rephrased as follows: p is the largest embeddable vertex in $P(p_i, p_{i+1}; p)$. Using the conditions in Lemma 4.1, we obtain the child-enumeration algorithm. For every possible $P(p_i, p_{i+1}; p)$, we check whether or not $P(p_i, p_{i+1}; p)$ is a child of P. We apply the algorithm recursively starting from the convex hull. Thus, we can traverse the family tree. In this way, one can enumerate all the surrounding polygons. In each recursive call, there are $O(n^2)$ child candidates $P(p_i, p_{i+1}; p)$. We can check whether or not $P(p_i, p_{i+1}; p)$ is a child in $O(\log n)$ time using triangular range query [6] with $O(n^2)$-time preprocessing and $O(n^2)$ additional space for an input point set and shortest path query [7] with $O(n)$-time preprocessing for each surrounding polygon. Thus, each recursive call takes $O(n^2 \log n)$ time. Now we have the following theorem.

▶ **Theorem 4.2.** *Let S be a set of n points in the Euclidean plane. One can enumerate all the surrounding polygons in $\mathcal{P}(S)$ in $O(n^2 \log n\, |\mathcal{P}(S)|)$-time and $O(n^2)$ space.*

From the theorem above, one can see that our algorithm is output-polynomial. Using the even-odd traversal in [13], we have a polynomial-delay enumeration algorithm. In the traversal, the algorithm outputs polygons with even depth when we go down the family tree and output polygons with odd depth when we go up. See [13] for further details. We have the following corollary.

▶ **Corollary 4.3.** *Let S be a set of n points in the Euclidean plane. There is an $O(n^2 \log n)$-delay and $O(n^2)$-space algorithm that enumerates all the surrounding polygons in $\mathcal{P}(S)$.*

Acknowledgments. Part of this work has been discussed during the Japan-Austria Bilateral Seminar: Computational Geometry Seminar with Applications to Sensor Networks in November 2018. The authors thank the organizers for providing an encouraging atmosphere. They also thank the anonymous referees for their valuable comments. This work was supported by JSPS KAKENHI Grant Numbers JP15K00009, JP15H05711, JP16K00002, JP17H06287, JP18H04091, JP18K11153. The third author is also supported by JST CREST Grant Number JPMJCR1402 and Kayamori Foundation of Informational Science Advancement.

References

1 Thomas Auer and Martin Held. Heuristics for the generation of random polygons. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 38–43, 1996.

2 David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996.

3 Sergei Bespamyatnikh. An efficient algorithm for enumeration of triangulations. *Computational Geometry Theory and Applications*, 23(3):271–279, 2002.

4 Erik D. Demaine. `http://erikdemaine.org/polygonization/`, 2012.

5 Alfredo García, Marc Noy, and Javier Tejel. Lower bounds on the number of crossing-free subgraphs of K_N. *Computational Geometry*, 16(4):211–221, 2000.

6 Partha P. Goswami, Sandip Das, and Subhas C. Nandy. Triangular range counting query in 2D and its application in finding k nearest neighbors of a line segment. *Computational Geometry*, 29(3):163 – 175, 2004.

7 Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126 – 152, 1989.

8 Takashi Horiyama and Wataru Shoji. Edge unfoldings of platonic solids never overlap. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry*, pages 65–70, 2011.

9 Naoki Katoh and Shin-ichi Tanigawa. Enumerating edge-constrained triangulations and edge-constrained non-crossing geometric spanning trees. *Discrete Applied Mathematics*, 157(17):3569–3585, 2009.

10 Dániel Marx and Tillmann Miltzow. Peeling and nibbling the cactus: Subexponential-time algorithms for counting triangulations and related problems. In *32nd International Symposium on Computational Geometry, SoCG 2016*, pages 52:1–52:16, 2016.

11 Gary H. Meisters. Polygons have ears. *American Mathematical Monthly*, 82(6):648–651, 1975.

12 Joseph S. B. Mitchell and Joseph O'Rourke. Computational geometry column 42. *International Journal of Computational Geometry and Applications*, 11(5):573–582, 2001.

13 Shin-ichi Nakano and Takeaki Uno. Generating colored trees. *Proceedings of the 31th Workshop on Graph-Theoretic Concepts in Computer Science, (WG 2005)*, LNCS 3787:249–260, 2005.

14 Joseph O'Rourke, Subhash Suri, and Csaba D. Tóth. Polygons. In *Handbook of Discrete and Computational Geometry, Third Edition.*, pages 787–810. Chapman and Hall/CRC, 2017.

15 Micha Sharir, Adam Sheffer, and Emo Welzl. Counting plane graphs: Perfect matchings, spanning cycles, and Kasteleyn's technique. *J. Comb. Theory Ser. A*, 120(4):777–794, 2013.

16 Christian Sohler. Generating random star-shaped polygons. In *Proceedings of the 11th Canadian Conference on Computational Geometry*, pages 174–177, 1999.

17 Sachio Teramoto, Mitsuo Motoki, Ryuhei Uehara, and Tetsuo Asano. Heuristics for generating a simple polygonalization. IPSJ SIG Technical Report 2006-AL-106(6), Information Processing Society of Japan, May 2006.

18 Emo Welzl. Counting simple polygonizations of planar point sets. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry*, 2011. URL: `http://www.cccg.ca/proceedings/2011/papers/invited3.pdf`.

19 Manuel Wettstein. Counting and enumerating crossing-free geometric graphs. *Journal of Computational Geometry*, 8(1):47–77, 2017.

20 Katsuhisa Yamanaka, Shin-ichi Nakano, Yasuko Matsui, Ryuhei Uehara, and Kento Nakada. Efficient enumeration of pseudoline arrangements. In *Proceedings of European Workshop on Computational Geometry 2009*, pages 143–146, March 2009.

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