Enumerating Surrounding Polygons

Katsuhisa Yamanaka[1,a] Takashi Horiyama[2,b] Yoshio Okamoto[3,4,c] Ryuhei Uehara[5,d]
Tanami Yamauchi[1,e]

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 point of P is a point in S and every point in S is either inside P or a point of P. In this paper, we present an enumeration algorithm of 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], [7], non-crossing spanning trees [7], pseudoline arrangements [17], non-crossing matchings [16], unforldings of Platonic solids [6], 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 Euclidean plane. A *surrounding polygon* of S is a simple polygon P such that each point of P is a point in S and every point in S is either inside the polygon or a point of the polygon. A surrounding polygon P of S is a *simple polygonization*[*1] if the vertices on P is precisely the set of S. See Fig. 1 for examples. A simple polygonization is one of theoretically-appealing geometric objects and studied on counting and random generations.

The main topic on counting simple polygonizations is to give lower and upper bounds of the number of them. There is a history on investigating the number of simple polygonizations of n points. The current best lower and upper bounds are 4.64^n [5] and 54.55^n [12], respectively. See surveys for further details [4], [11]. 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 [9].

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 [18] and star-shaped polygons [13]. For general simple polygonizations, heuristic algorithms are known [1], [14], [18]. Those algorithms efficiently generate simple polygons, but not uniformly at random.

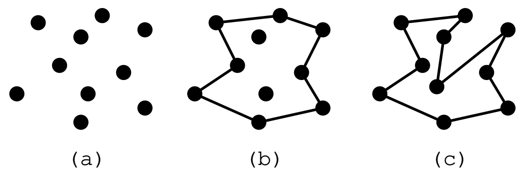

Fig. 1 (a) A point set S. (b) A surrounding polygon of S. (c) A simple polygonization of S.

On the other hand, nothing is known for the problem of enumerating all the simple polygonizations, as mentioned in [15]. 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 simple polygonizations of a given point set can be enumerated efficiently (for example, output-polynomial time[*2] or polynomial-delay[*3]).

As the first step toward the question, we consider the problem of enumerating 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 point from a surrounding polygons and putting it inside

[1] Iwate University, Morioka, Iwate 020–8515, Japan
[2] Saitama University, Saitama, Saitama 338–8570, Japan
[3] The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[4] RIKEN Center for Advanced Intelligence Project, Chuo-ku, Tokyo 103–0027, Japan
[5] Japan Advanced Institute of Science and Technology, Nomi, Ishikawa 923–1211, Japan
[a] yamanaka@cis.iwate-u.ac.jp
[b] horiyama@al.ics.saitama-u.ac.jp
[c] okamotoy@uec.ac.jp
[d] uehara@jaist.ac.jp
[e] tanami@kono.cis.iwate-u.ac.jp

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

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

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

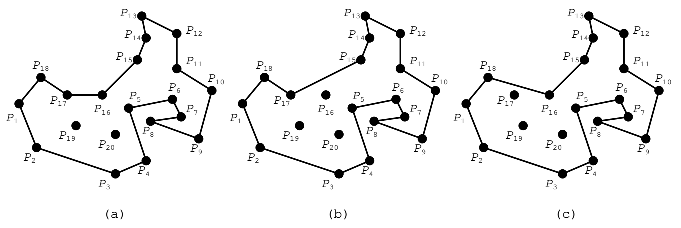

Fig. 2 (a) A surrounding polygon, where $p_6, p_7, p_{11}, p_{14}, p_{15}, p_{16}$, and p_{17} are embeddable. (b) The surrounding polygon obtained by embedding p_{16}. The point p_{16} is embedded inside the polygon. (c) The parent of the polygon in (a), which is obtained by embedding p_{17}.

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 structure, one can enumerate all the surrounding polygons and the proposed algorithm can enumerate them in polynomial-delay.

All the proofs and some details are omitted in this report.

2. Preliminaries

A *simple polygon* is a closed region of the plane enclosed by a simple cycle of straight line segments. 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 straight line segments is a diagonal of P and the remaining two straight line segments are straight line segments of P. The following theorem for ears is known.

Theorem 1 ([8]) Every simple polygon with $n \geq 4$ vertices has at least two non-overlapping ears.

Let S be a set of n points in 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 top point among the leftmost points in S. A *surrounding polygon* of S is a simple polygon such that every point in S is either inside the polygon or a point of the polygon. From now on, we call a point on a surrounding polygon a *vertex* and a straight line segment an *edge*. For example, the convex hull of S is a surrounding polygon of S. Note that any surrounding polygon includes the upper-left point in S.

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 Euclidean plane, and let $\mathcal{P}(S)$ be a set of surrounding polygons of S. In this section, we define a tree structure over $\mathcal{P}(S)$ such that its nodes correspond to 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 $\mathsf{succ}(p)$ includes no inside region of P. See examples in Fig. 2.

Lemma 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 removes 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 inside of P. See Fig. 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 repeating to find the parents starting 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 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 Fig. 3. As we can see in the following lemma, the last polygon in a parent sequence is always $\mathsf{CH}(P)$.

Lemma 2 Let S be a set of n points in Euclidean space, 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 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 structure the *family tree*. An example of the family tree is shown in Fig. 4.

4. Enumeration algorithm

In this section, we present an enumeration algorithm that, for a given set S of n points, enumerates all the polygons in $\mathcal{P}(S)$. In the previous section, we define 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 do the search, we design an algorithm that finds all the children

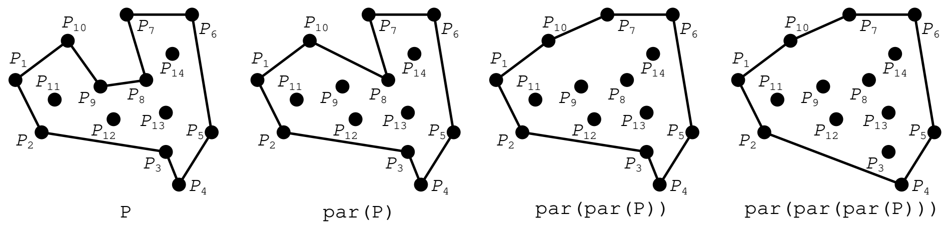

Fig. 3 A parent sequence.

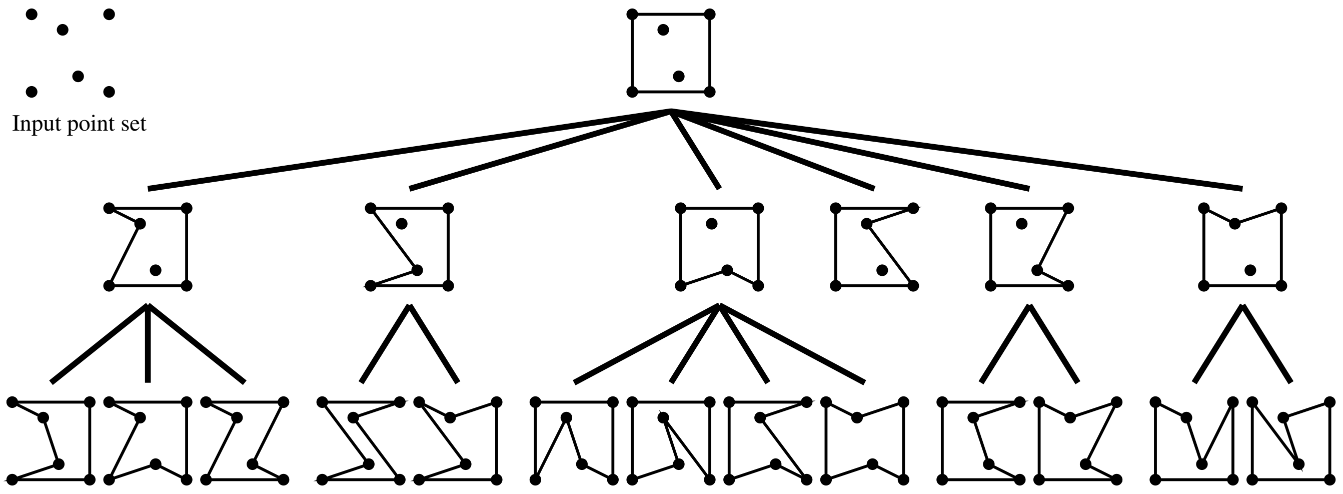

Fig. 4 An example of a family tree.

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 $P = \langle p_1, p_2, \dots, p_k \rangle$ be a surrounding polygon in $\mathcal{P}(S)$. For an edge (p_i, p_{i+1}) on 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 3 Let P be a surrounding polygon of a set of points. For an edge (p_i, p_{i+1}) on 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 3 can be rephrased as follows: p is the largest vertex in $P(p_i, p_{i+1}; p)$. Using the conditions in Lemma 3, 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 $O(n)$ time. Thus, each recursive call takes $O(n^3)$ time. Now we have the following theorem.

Theorem 2 Let S be a set of n points in Euclidean plane. There is an $O(n^3 |\mathcal{P}(S)|)$-time algorithm that enumerates all the surrounding polygons in $\mathcal{P}(S)$.

From the theorem above, one can see that our algorithm is output-polynomial. Using the even-odd traversal in [10], we have a polynomial-delay enumeration algorithm. In the traversal, the algorithm output polygons with even depth when we go down the family tree and output polygons with odd depth when we go up. See [10] for further details. We can have the following corollary.

Corollary 1 Let S be a set of n points in Euclidean plane. There is an $O(n^3)$-delay algorithm that enumerates all the surrounding polygons in $\mathcal{P}(S)$.

Acknowledgements

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. This work was supported by JSPS KAKENHI Grant Numbers JP15K00009, JP15H05711, JP16K00002, JP17H06287, JP18H04091, JP18K11153. The third author is supported by JST CREST Grant Number JPMJCR1402 and Kayamori Foundation of Informational Science Advancement.

References

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

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

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

[4] E. D. Demaine. `http://erikdemaine.org/polygonization/`,

2012.

[5] A. García, M. Noy, and J. Tejel. Lower bounds on the number of crossing-free subgraphs of kn. *Computational Geometry*, 16(4):211–221, 2000.

[6] T. Horiyama and W. Shoji. Edge unfoldings of platonic solids never overlap. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry*, pages 65–70, 2011.

[7] N. Katoh and S. Tanigawa. Enumerating edge-constrained triangulations and edge-constrained non-crossing geometric spanning trees. *Discrete Applied Mathematics*, 157(17):3569–3585, 2009.

[8] G. H. Meisters. Polygons have ears. *American Mathematical Monthly*, 82(6):648–651, 1975.

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

[10] S. Nakano and T. Uno. Generating colored trees. *Proceedings of the 31th Workshop on Graph-Theoretic Concepts in Computer Science, (WG 2005)*, LNCS 3787:249–260, 2005.

[11] J. O'Rourke, S. Suri, and C. D. Tóth. Polygons. In *Handbook of Discrete and Computational Geometry, Third Edition.*, pages 787–810. Chapman and Hall/CRC, 2017.

[12] M. Sharir, A. Sheffer, and E. Welzl. Counting plane graphs: Perfect matchings, spanning cycles, and kasteleyn's technique. *J. Comb. Theory Ser. A*, 120(4):777–794, May 2013.

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

[14] S. Teramoto, M. Motoki, R. Uehara, and T. Asano. Heuristics for generating a simple polygonalization. IPSJ SIG Technical Report 2006-AL-106(6), Information Processing Society of Japan, May 2006.

[15] E. Welzl. Counting simple polygonizations of planar point sets. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry*, 2011.

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

[17] K. Yamanaka, S. Nakano, Y. Matsui, R. Uehara, and K. Nakada. Efficient enumeration of pseudoline arrangements. In *Proceedings of European Workshop on Computational Geometry 2009*, pages 143–146, Mar. 2009.

[18] C. Zhu, G. Sundaram, J. Snoeyink, and J. S. B. Mitchell. Generating random polygons with given vertices. *Computational Geometry: Theory and Applications*, 6:277–290, 1996.