PAPER *Special Section on Discrete Mathematics and Its Applications*

Enumerating Empty and Surrounding Polygons

Shunta TERUI[†], ***Nonmember***, **Katsuhisa YAMANAKA**[†a)], **Takashi HIRAYAMA**[†b)], **Takashi HORIYAMA**[††c)], **Kazuhiro KURITA**[†††d)], ***and*** **Takeaki UNO**[††††e)], ***Members***

SUMMARY We are given a set S of n points in the Euclidean plane. We assume that S is in general position. A simple polygon P is an *empty polygon* of S if each vertex of P is a point in S and every point in S is either outside P or a vertex of P. In this paper, we consider the problem of enumerating all the empty polygons of a given point set. To design an efficient enumeration algorithm, we use a reverse search by Avis and Fukuda with child lists. We propose an algorithm that enumerates all the empty polygons of S in $O(n^2 |\mathcal{E}(S)|)$-time, where $\mathcal{E}(S)$ is the set of empty polygons of S. Moreover, by applying the same idea to the problem of enumerating surrounding polygons of a given point set S, we propose an enumeration algorithm that enumerates them in $O(n^2)$-delay, while the known algorithm enumerates in $O(n^2 \log n)$-delay, where a *surrounding polygon* of S is a polygon such that each vertex of the polygon is a point in S and every point in S is either inside the polygon or a vertex of the polygon.

key words: *enumeration algorithm, reverse search, simple polygon, empty polygon, surrounding polygon*

1. Introduction

Enumeration problems are fundamental and important in computer science and have applications including data mining, bioinformatics, and artificial intelligence. A lot of enumeration algorithms for enumeration problems have been proposed [1]. Among them, enumeration problems for geometric objects have been studied. For example, enumeration algorithms have been proposed for triangulations [2]–[5], non-crossing spanning trees [2], [4]–[6], non-crossing spanning cycles [5], [6], non-crossing convex partitions [5], non-crossing perfect matchings [5], non-crossing convex subdivisions [5], pseudoline arrangements [7], unfoldings of Platonic solids [8], Archimedian solids [9], and so on.

In this paper, we focus on the enumeration problem of generating simple polygons with the following property.

Manuscript received September 15, 2022.
Manuscript revised January 12, 2023.
Manuscript publicized April 3, 2023.

[†]The authors are with Iwate University, Morioka-shi, 020-8551 Japan.
[††]The author is with Hokkaido University, Sapporo-shi, 060-0814 Japan.
[†††]The author is with Nagoya University, Nagoya-shi, 464-8601 Japan.
[††††]The author is with National Institute of Informatics, Tokyo, 101-8430 Japan.
a) E-mail: yamanaka@iwate-u.ac.jp
b) E-mail: hira6377@iwate-u.ac.jp
c) E-mail: horiyama@ist.hokudai.ac.jp
d) E-mail: kurita@i.nagoya-u.ac.jp
e) E-mail: uno@nii.jp

DOI: 10.1587/transfun.2022DMP0007

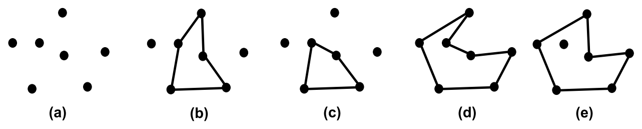

Fig. 1 (a) A point set S. (b) An empty polygon, (c) an empty convex polygon, (d) a non-crossing spanning cycle, and (e) a surrounding polygon of S.

We are given a set S of n points in the Euclidean plane and in general position. An *empty 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 outside the polygon or a vertex of the polygon. Fig. 1(b) is an example of an empty polygon of the point set in Fig. 1(a). The class of empty polygons includes two important classes of simple polygons: empty convex polygons and non-crossing spanning cycles.

An empty polygon of S is an *empty convex polygon* of S if the polygon is convex. See Fig. 1(c) for an example. The empty convex polygons have been studied in the contexts of counting and enumeration. For counting, Rote et al. [10] presented an $O(n^{k-2})$-time algorithm that counts the number of the empty convex k-gons of a given point set S for any $k \geq 4$, where a k-gon of S is a simple polygon such that the polygon consists of k vertices and each vertex is a point in S. Rote and Woeginger [11] improved the running time of the algorithm to $O(n^{\lceil k/2 \rceil})$-time. Moreover, Mitchell et al. [12] improved the running time. Their algorithm is based on a dynamic programming and runs in $O(kn^3)$-time. Very recently, Bae proposed $O(k\gamma_k(S))$-time counting algorithm, where $\gamma_k(S)$ is the number of the empty convex k-gons of S. For enumeration, Dobkin et al. [13] proposed an enumeration algorithm that enumerates all the empty convex k-gons of a given point set S and runs in $O(\gamma_3(S) + k\gamma_k(S))$-time.

An empty polygon of a point set S is called a *non-crossing spanning cycle* of S if every point S is a vertex of the polygon. See Fig. 1(d) for an example. The non-crossing spanning cycles of a point set are known as appealing objects in the area of computational geometry and have been studied in the context of counting [5], [6], [14], random generation [15]–[18], and enumeration [5], [6].

Recently, Yamanaka et al. [19] proposed surrounding polygons as a new class of simple polygons. For a point set S, a *surrounding polygon* of S is a simple polygon such that each vertex of the polygon is a point in S and every point in S is either inside the polygon or a vertex of the polygon.

See Fig. 1(e) for an example. They showed that one can enumerate all the surrounding polygons of S in $O(n^2 \log n)$-delay and $O(n^2)$-space, where the delay of an enumeration algorithm is the worst-case running time between two consecutive outputs in the algorithm.

Our contributions are as follows. We first design an algorithm that enumerates all the empty polygons of a given set of n points in $O(n^2 |\mathcal{E}(S)|)$-time, where $\mathcal{E}(S)$ is the set of empty polygons of S. The algorithm is based on the reverse search by Avis and Fukuda [2] with the "child list" technique, which is often used to design efficient enumeration algorithms ([20]). Moreover, we apply the technique to the problem of enumerating surrounding polygons of S. We present an algorithm that enumerates all the surrounding polygons of S in $O(n^2)$-delay and $O(n^2)$-space, while the known algorithm enumerates them in $O(n^2 \log n)$-delay and $O(n^2)$-space.

2. Preliminary

2.1 Notations

In this section, we define some notations required in this paper.

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. A line segment connecting two vertices on a simple polygon P is a *diagonal* if the line segment is included inside P. 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. We say that two ears in P are *non-overlapping* if their inside regions are disjoint. Otherwise, we say that they are *overlapping*. The following theorem for ears is known.

Theorem 1 ([21]): 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. Throughout this paper, 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 sequence $P = \langle p_1, p_2, \ldots, p_k \rangle$, $(k \leq n)$, of points in S is a *simple polygon of S* if the alternating sequence of points and line segments

$$p_1, (p_1, p_2), p_2, (p_2, p_3), \ldots, p_k, (p_k, p_1)$$

forms a simple polygon. Let $P = \langle p_1, p_2, \ldots, p_k \rangle$ be a simple polygon of S. We suppose that the vertices on P appear in counterclockwise order. We denote by $\mathsf{in}(P) \subseteq S$ and $\mathsf{out}(P) \subseteq S$ the sets of the points inside and outside P, respectively. We denote by $p_i \prec p_j$ if $i < j$ holds, and we say that p_j is *larger than* p_i on P. $\mathsf{pred}(p_i)$ and $\mathsf{succ}(p_i)$ denote the predecessor and successor of p_i on P, respectively. Note that the successor of p_k is p_1. Suppose that P has 4 or more vertices. A *removal* of p_i on P is to remove two line segments $(\mathsf{pred}(p_i), p_i)$ and $(p_i, \mathsf{succ}(p_i))$ and insert the line segment $(\mathsf{pred}(p_i), \mathsf{succ}(p_i))$. We denote by $\mathsf{rem}(P, p_i)$ the simple polygon obtained from P by applying the removal of p_i to P. Intuitively, the remove operation picks out a point from a simple polygon. Note that $\mathsf{rem}(P, p_i)$ may or may not be a simple polygon of S. An *insertion* of $p \in \mathsf{in}(P) \cup \mathsf{out}(P)$ after p_i on P is to remove the line segment $(p_i, \mathsf{succ}(p_i))$ and insert the two line segments (p_i, p) and $(p, \mathsf{succ}(p_i))$. We denote by $\mathsf{ins}(P, p_i, p)$ the simple polygon obtained from P by applying the insertion of p after p_i on P. Intuitively, the insertion operation adds a point as a vertex of a simple polygon.

The *convex hull*, denoted by $\mathsf{CH}(S)$, of S is the simple polygon with the smallest area that contains all the points in S. An *empty polygon* of S is a simple polygon such that every point in S is either outside the polygon or a vertex of the polygon. 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.

2.2 Reverse Search with Child Lists

The reverse search by Avis and Fukuda [22] is a framework for designing enumeration algorithms. In this section, we redescribe the reverse search in the context of our paper. In this paper, we consider problems of enumerating simple polygons with the designated properties. Moreover, to design efficient algorithms, we use "child lists", which is an idea for speeding up reverse-search-enumeration algorithms [20].

Let S be a set of n points in the Euclidean plane. Let $C(S, Q)$ be the set of simple polygons of S with a property Q. We consider designing an enumeration algorithm for all the polygons in $C(S, Q)$ by using the reverse search. In the reverse search, we first define an undirected simple graph $G = (V, E_G)$ such that each node in V corresponds to a simple polygon in $C(S, Q)$ and each branch in E_G corresponds to a relation between two simple polygons in $C(S, Q)$. The relation between two simple polygons is defined by using "basic operations". For two simple polygons P and P', the node corresponding to P is adjacent to that corresponding to P' in G if P is obtained from P' by applying the basic operation and P' is obtained from P by applying the inverse operation of the basic operation. That is, one of P and P' is constructed from the other by applying an operation. (In this paper, we use the removal and insertion as a basic operation and its inverse operation for the problems of enumerating empty and surrounding polygons.) Next, we define a spanning forest $F = (V, E_F)$ of G with the root node set $V_R \subseteq V$. Let denote the parent of $v \in V$ in F by $\mathsf{par}(v)$. A reverse-search-enumeration algorithm visits every node in V by traversing F and enumerates all the polygons in $C(S, Q)$.

For a node $v \in V$, $N_G(v)$ and $C_F(v)$ denote the set of the neighbour of v in G and the set of the children of v in F, respectively. To traverse F, we have to generate all the children of a node v in F. A naïve way to generate all the children is as follows. First, we check whether $v = \mathsf{par}(u)$

Algorithm 1: REVERSE-SEARCH(S)

```
1 foreach $r \in V_R$ do
2     NAÏVE-FIND-CHILDREN($r$)
```

Algorithm 2: NAÏVE-FIND-CHILDREN(v)

```
1 Output $v$
2 foreach $u \in N_G(v)$ do
3     if $v = \mathsf{par}(u)$ then
4         NAÏVE-FIND-CHILDREN($u$)
```

Algorithm 3: FIND-CHILDREN-WITH-LIST(v, $L(v)$)

```
1 /* $L(v)$ and $L(u)$ is the lists of the children of
     $v$ and $u$, respectively.                          */
2 Output $v$
3 foreach $u \in L(v)$ do
4     Construct $L(u)$ from $L(v)$
5     FIND-CHILDREN-WITH-LIST($u$, $L(u)$)
```

holds for each $u \in N_G(v)$. If the condition is true, u is a child of v. Note that $C_F(v) \subseteq N_G(v)$ holds, since F is a spanning forest of G. A pseudo-code of such an algorithm is shown in **Algorithm 1** and **Algorithm 2**. In the algorithms, we enumerate all the roots in V_R. Let t_R be the running time to enumerate all the roots in V_R in **Algorithm 1**. Let t_u be the running time to perform **Algorithm 2** for a node u of G and let $t_{C(S,Q)}$ be the worst-case running time for t_u (thus, $t_u \leq t_{C(S,Q)}$ for any $u \in V$). Then, we have the following theorem.

Theorem 2: **Algorithm 1** and **Algorithm 2** enumerate all the polygons in $C(S,Q)$ in $O(t_R + t_{C(S,Q)} |C(S,Q)|)$ time.

To accelerate the algorithm for generating children, we maintain the list of the children of the current polygon. The pseudo-code is shown in **Algorithm 3**. In **Algorithm 3**, before calling **Algorithm 3** for a child u of v, we construct the list of the children of u. The main routine of **Algorithm 3** is almost the same as **Algorithm 1** (for each root node, we construct the child list of the root and call **Algorithm 3**). Let t'_R be the running time to enumerate all the roots in V_R and construct their child lists. Let t'_u be the running time to construct the child list for the polygon corresponding to the node u and let $t'_{C(S,Q)}$ be the worst-case running time for t'_u (thus, $t'_u \leq t'_{C(S,Q)}$ for any $u \in V$).

Theorem 3: One can enumerate all the polygons in $C(S,Q)$ in $O(t'_R + t'_{C(S,Q)} |C(S,Q)|)$ time.

3. Enumeration of Empty Polygons

Let S be a set of n points in the Euclidean plane, and let $\mathcal{E}(S)$ be the set of empty polygons of S. In this section, we propose enumeration algorithms of empty polygons of S. An empty polygon P is an *empty triangle*[†] if P has three vertices. Let $\mathcal{T}(S) \subseteq \mathcal{E}(S)$ be the set of empty triangles of S. Our algorithms are based on reverse search, shown in Sect. 2.2. In Sect. 3.1, we define a forest structure over $\mathcal{E}(S)$ such that each node corresponds to an empty polygon in $\mathcal{E}(S)$ and each root node corresponds to an empty triangle in $\mathcal{T}(S)$. Next, in Sect. 3.2, we design an enumeration algorithm by using the standard reverse search. The algorithm traverses the forest structure in depth-first manner and enumerates all the empty polygons in $\mathcal{E}(S)$. Finally, in Sect. 3.3, we design a more efficient algorithm by using the reverse search with child lists.

3.1 Family Forest

Now, we introduce some notations. Let $P = \langle p_1, p_2, \dots, p_k \rangle$ be an empty polygon in $\mathcal{E}(S) \setminus \mathcal{T}(S)$. A vertex p_i of P is *ear-central* if $\mathsf{pred}(p_i), p_i, \mathsf{succ}(p_i)$ form an ear of P. We have the following observation.

Observation 1: Let $P = \langle p_1, p_2, \dots, p_k \rangle$ be an empty polygon in $\mathcal{E}(S) \setminus \mathcal{T}(S)$. Let p_i be an ear-central vertex of P. Then, $\mathsf{rem}(P, p_i)$ is an empty polygon with $k-1$ vertices.

Proof. Immediate from the definition of an ear-central vertex. □

Next, we define a parent of P. Since $P \in \mathcal{E}(S) \setminus \mathcal{T}(S)$, P has four or more vertices. From Theorem 1, P has two or more non-overlapping ears. Thus, we have the following observation.

Observation 2: An empty polygon with four or more vertices has two or more ear-central vertices.

Let $\mathsf{l\text{-}ear}(P)$ be the largest ear-central vertex on P. We define $\mathsf{rem}(P, \mathsf{l\text{-}ear}(P))$ as the *parent* of P. For convenience, we denote the parent of P by $\mathsf{epar}(P) := \mathsf{rem}(P, \mathsf{l\text{-}ear}(P))$. We say that P is a child of $\mathsf{epar}(P)$. The following lemma shows the properties of the parent of P.

Lemma 1: Let S be a point set in the Euclidean plane, and let P be an empty polygon in $\mathcal{E}(S) \setminus \mathcal{T}(S)$. Then,
(1) $\mathsf{epar}(P)$ is an empty polygon with one less vertex,
(2) $\mathsf{epar}(P)$ always exists, and
(3) $\mathsf{epar}(P)$ is unique.

Proof. The properties (1) and (2) are immediate from Observations 1 and 2. The property (3) holds, since the largest ear-central of P is defined uniquely. □

By repeatedly finding the parents from P, we obtain a sequence of empty polygons. For an empty polygon $P \in \mathcal{E}(S)$, the *parent sequence* $\mathsf{EPS}(P) = \langle P_1, P_2, \dots, P_\ell \rangle$ of P is a sequence of empty polygons such that the first polygon is P itself and $P_i = \mathsf{epar}(P_{i-1})$ for each $i = 2, 3, \dots, \ell$. See an example in Fig. 2. Note that the parent sequence of an empty triangle P is $\mathsf{EPS}(P) = \langle P \rangle$. As we can see in the following

[†]Empty triangles of a point set are equivalent to the empty convex 3-gons of the point set. In this paper, we use the terminology "empty triangles".

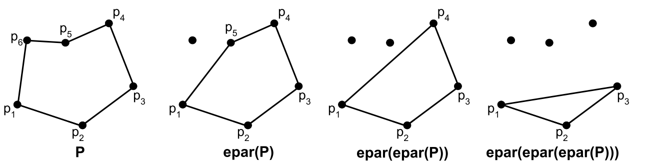

Fig. 2 A parent sequence of empty polygon.

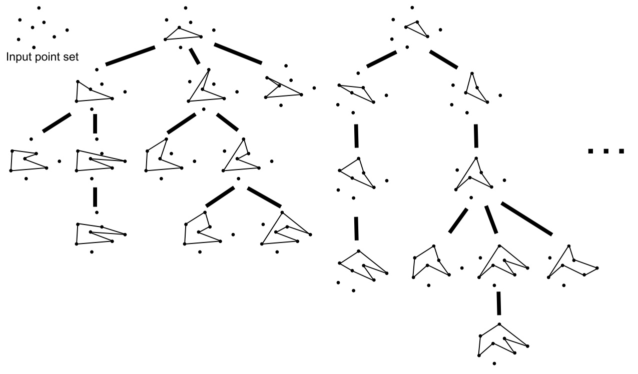

Fig. 3 An example of a family forest. An input point is illustrated in the top-left of the figure. In this figure, only two trees are illustrated in the family forest.

lemma, the last polygon in a parent sequence is always an empty triangle in $\mathcal{T}(S)$.

Lemma 2: Let S be a set of n points in the Euclidean plane, and let P be an empty polygon in $\mathcal{E}(S)$. The last polygon in $\mathsf{EPS}(P)$ is an empty triangle in $\mathcal{T}(S)$.

Proof. If P is an empty triangle, the claim holds clearly. Hence, we assume that P is not an empty triangle. Let $\mathsf{EPS}(P) = \langle P_1, P_2, \ldots, P_\ell \rangle$ be the parent sequence of P. For a polygon P_i in $\mathsf{EPS}(P)$, we define a function $\phi(P_i)$ as the number of the vertices on P_i. Note that, for any $T \in \mathcal{T}(S)$, $\phi(T) = 3$ and $\phi(T) \leq \phi(P_i)$ holds. It can be observed that $\phi(P_i) = 3$ if and only if $P_i \in \mathcal{T}(S)$ holds. From the condition (1) in Lemma 1, it can be observed that $\phi(P_i) = \phi(P_{i-1}) - 1$ for each $i = 2, 3, \ldots, \ell$. Moreover, from the condition (2) in Lemma 1, the parent of an empty polygon in $\mathcal{E}(S) \setminus \mathcal{T}(S)$ always exists. Therefore, $P_\ell \in \mathcal{T}(S)$ holds. □

From Lemma 2, for any empty polygon, the last polygon of its parent sequence is an empty triangle. By merging parent sequences of all the empty polygons in $\mathcal{E}(S)$, we obtain a forest structure, called a *family forest* of $\mathcal{E}(S)$. A family forest consists of trees, each of which is rooted at an empty triangle in $\mathcal{T}(S)$. Each empty triangle in $\mathcal{T}(S)$ appears as the root of a tree in a family forest. An example of a family forest is shown in Fig. 3.

3.2 Enumeration by Reverse Search

In this subsection, we present an algorithm that, for a given set S of n points, enumerates all the empty polygons in $\mathcal{E}(S)$. We use the standard reverse search technique, shown in Sect. 2.2. We propose an $O((n \log n)\,|\mathcal{T}(S)| + (n^2 \log n)\,|\mathcal{E}(S)|)$-time and $O(n^2)$-space enumeration algorithm.

In the previous subsection, we defined the family forest over $\mathcal{E}(S)$. We know that the roots of the family forest are the empty triangles in $\mathcal{T}(S)$. Hence, we obtain the following enumeration algorithm. We first enumerate the empty triangles in $\mathcal{T}(S)$. Then, we traverse the tree rooted at each empty triangle in depth-first manner by recursively generating children from the current empty polygons. This algorithm can enumerate all the empty polygons in $\mathcal{E}(S)$.

Now, we describe how to generate all the children of an empty polygon. Let $P = \langle p_1, p_2, \ldots, p_k \rangle$ be an empty polygon in $\mathcal{E}(S)$. An empty polygon P' is a child of P if $P = \mathsf{epar}(P')$ holds. Hence, a child of P is obtained by applying an insertion to P, which is a reverse operation of removal Thus, any child of P is described as $\mathsf{ins}(P, p_i, p)$ for p on P and $p_i \in \mathsf{out}(S)$. It is easy to observe that, for all possible $\mathsf{ins}(P, p_i, p)$, if we check whether or not $\mathsf{ins}(P, p_i, p)$ is a child, then one can generate all the children of P. We have the following lemma.

Lemma 3: Let $P = \langle p_1, p_2, \ldots, p_k \rangle$ be an empty polygon of a set S of points. For a vertex p_i $(1 \leq i \leq k)$ on P and a point $p \in \mathsf{out}(P)$, $\mathsf{ins}(P, p_i, p)$ is a child of P if

(1) $\mathsf{ins}(P, p_i, p)$ is an empty polygon of S and
(2) $\mathsf{epar}(\mathsf{ins}(P, p_i, p)) = P$ holds.

Actually, using the conditions in Lemma 3, we obtain the child-enumeration algorithm. However, we give a more detailed case analysis for child generation. Clearly, if $\mathsf{ins}(P, p_i, p)$ is not an empty polygon, then $\mathsf{ins}(P, p_i, p)$ is not a child of P. Hence, in the case analysis below, we only consider the cases that $\mathsf{ins}(P, p_i, p)$ is an empty polygon.

Case 1: $p_i \prec \mathsf{pred}(\mathsf{l\text{-}ear}(P))$.

If we apply an insertion to the vertex p_i smaller than $\mathsf{pred}(\mathsf{l\text{-}ear}(P))$, in the polygon $\mathsf{ins}(P, p_i, p)$, the largest ear-central vertex is still $\mathsf{l\text{-}ear}(P)$, that is, $\mathsf{l\text{-}ear}(P) = \mathsf{l\text{-}ear}(\mathsf{ins}(P, p_i, p))$ holds. Hence, $\mathsf{epar}(\mathsf{ins}(P, p_i, p)) \neq P$ holds. Therefore, $\mathsf{ins}(P, p_i, p)$ is not a child of P. See Fig. 4(b) for an example.

Case 2: $p_i = \mathsf{pred}(\mathsf{l\text{-}ear}(P))$.

If $\mathsf{l\text{-}ear}(P)$ is still the largest ear-central vertex in $\mathsf{ins}(P, p_i, p)$, then $\mathsf{epar}(\mathsf{ins}(P, p_i, p)) \neq P$ holds, since $\mathsf{epar}(\mathsf{ins}(P, p_i, p))$ is obtained from $\mathsf{ins}(P, p_i, p)$ by removing $\mathsf{l\text{-}ear}(P)$. Hence, $\mathsf{ins}(P, p_i, p)$ is not a child of P. See Fig. 4(c). However, if $\mathsf{l\text{-}ear}(P)$ is not ear-central in $\mathsf{ins}(P, p_i, p)$, $\mathsf{l\text{-}ear}(\mathsf{ins}(P, p_i, p)) = p$. Hence, $\mathsf{epar}(\mathsf{ins}(P, p_i, p)) = P$ holds. Thus, $\mathsf{ins}(P, p_i, p)$ is a child of P. See Fig. 4(d).

Case 3: $\mathsf{pred}(\mathsf{l\text{-}ear}(P)) \prec p_i$

In this case, $p_i = \mathsf{l\text{-}ear}(\mathsf{ins}(P, p_i, p))$ always holds. Hence, $P = \mathsf{epar}(\mathsf{ins}(P, p_i, p))$ holds. Thus, $\mathsf{ins}(P, p_i, p)$ is a child of P. See Fig. 4(e).

From the above case analysis and Lemma 3, we can generate all the children of P.

Now, we describe our enumeration algorithm in **Algo-**

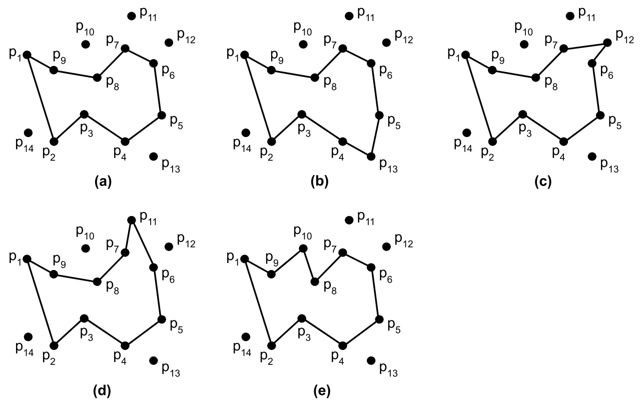

Fig. 4 (a) An empty polygon P, where $\text{l-ear}(P) = p_7$. (b) $\text{ins}(P, p_4, p_{13})$ is not a child of P, since $p_{13} \neq \text{l-ear}(\text{ins}(P, p_4, p_{13}))$. (c) $\text{ins}(P, p_6, p_{12})$ is not a child of P. (d) $\text{ins}(P, p_6, p_{11})$ is a child of P, since p_7 turns to be a non-ear. (e) $\text{ins}(P, p_6, p_{10})$ is a child of P.

rithm 4 and **Algorithm 5**. In **Algorithm 4**, we enumerate all the empty triangles by checking whether or not any three points $p, q, r \in S$ form an empty triangle. Then, from each empty triangle, we traverse the family forest by recursively applying **Algorithm 5**. We have the following theorem.

Theorem 4: Let S be a set of n points in the Euclidean plane. One can enumerate all the empty polygons of S in $O((n \log n)\,|\mathcal{T}(S)| + (n^2 \log n)\,|\mathcal{E}(S)|)$-time and in $O(n^2)$-space.

Proof. We first show that **Algorithm 4** and **Algorithm 5** enumerate all the empty polygons in $\mathcal{E}(S)$ without duplicates. For any empty polygon in $\mathcal{E}(S)$, the parent is always defined (Lemma 1), and its parent sequence ends up with an empty triangle (Lemma 2). Hence, every empty polygon is included in the family forest of $\mathcal{E}(S)$. Moreover, for any empty polygon in $\mathcal{E}(S) \setminus \mathcal{T}(S)$, the parent is defined uniquely (Lemma 1). Hence, **Algorithm 5** outputs each empty polygon exactly once. Therefore, the algorithm enumerates all the empty polygons in $\mathcal{E}(S)$ without duplicates.

Next, we estimate the running time of our enumeration algorithm. In **Algorithm 4**, we enumerate all the empty triangles. This can be done in $O((n \log n)\,|\mathcal{T}(S)|)$-time by using triangular range query [23], as follows. Three points $p, q, r \in S$ form an empty triangle in $\mathcal{T}(S)$ if $\triangle(p, q, r)$ has no point in its inside. This can be checked in $O(\log n)$-time by the triangular range query with $O(n^2)$-preprocessing and $O(n^2)$-additional space. This preprocessing is performed to an input point set S. That is, throughout the enumeration algorithm, the preprocessing for the triangular range query is performed once. For any three points in S, we check the above condition and enumerate all the empty triangles in $\mathcal{T}(S)$. We take $O(n^3 \log n)$-time for this enumeration. However, we can have a tighter estimation. Let (p, q) be a line segment connecting two points in S and let r be the closest point from (p, q). Then, $\triangle(p, q, r)$ is an empty triangle in $\mathcal{T}(S)$. Thus, for each line segment connecting two points in S, there exists an empty triangle in $\mathcal{T}(S)$. Moreover, any empty triangle corresponds to 3 line segments. This implies

Algorithm 4: ENUM-EMPTY-POLYGONS(S)

```
1 /* $S$ is a set of $n$ points in the Euclidean
   plane.                                          */
2 foreach Three points $p, q, r \in S$ do
3     if $\triangle(p, q, r)$ is an empty triangle then
4         FIND-CHILDREN($\triangle(p, q, r)$)
```

Algorithm 5: FIND-CHILDREN(P)

```
1  /* $P$ is an empty polygon of a point set $S$.    */
2  Output $P$.
3  Set $p_i$ = pred(l-ear($P$)).
4  foreach $p \in$ out($P$) do                       // Case 2
5      if l-ear($P$) is not ear in ins($P, p_i, p$) then
6          if ins($P, p_i, p$) is an empty polygon then
7              FIND-CHILDREN(ins($P, p_i, p$))
8  foreach $p_i$ on $P$ with pred(l-ear($P$)) $\prec p_i$ do   // Case 3
9      foreach $p \in$ out($P$) do
10         if ins($P, p_i, p$) is an empty polygon then
11             FIND-CHILDREN(ins($P, p_i, p$))
```

that $|\mathcal{T}(S)| \in \Omega(n^2)$ holds. Therefore, we can observe that the enumeration algorithm for the empty triangles in $\mathcal{T}(S)$ takes $O((n \log n)\,|\mathcal{T}(S)|)$-time.

In **Algorithm 5**, each recursive call checks whether $\text{ins}(P, p_i, p)$ is a child of P for each vertex p_i of P and each point $p \in \text{out}(P)$. If $\text{ins}(P, p_i, p)$ is a child of P, we recursively call **Algorithm 5** to $\text{ins}(P, p_i, p)$. Here, we estimate the running time for checking whether $\text{ins}(P, p_i, p)$ is a child of P. To do this, we check whether the following two conditions are satisfied: (1) $\text{ins}(P, p_i, p)$ has no intersection and (2) the triangle $\triangle(p_i, p, \text{succ}(p_i))$ has no point in its inside. The condition (1) can be checked by using ray-shooting query in $O(\log n)$-time with $O(n \log n)$-time preprocessing and $O(n)$-additional space [24], [25], which is performed for each empty polygon (thus, each recursive call of **Algorithm 5**). The condition (2) can be checked by using triangular range query. The above two conditions can be checked in $O(\log n)$-time. Hence, each recursive call takes $O(n^2 \log n)$-time. **Algorithm 5** stores the input point set S, the current empty polygon P and $\text{l-ear}(P)$ in global memory. Hence, each recursive call of **Algorithm 5** uses $O(n)$-space. The height of the family tree of $\mathcal{E}(S)$ is bounded by $O(n)$, the whole of our algorithm uses $O(n^2)$-space. □

3.3 Improvement: Enumeration by Reverse Search with Child Lists

In this subsection, we improve the running time by modifying **Algorithm 5** in Theorem 4. To improve the running time, we use the reverse search with child lists, shown in Sect. 2.2. **Algorithm 5** checks whether or not $\text{ins}(P, p_i, p)$ is a child for every vertex p_i on P and every point $p \in \text{out}(P)$ of an empty polygon P. On the other hand, the algorithm proposed in this subsection maintains, for every outside point p, a list of every vertex p_i such that $\mathsf{ins}(P, p_i, p)$ is a child. In **Algorithm 5**, each recursive call takes $O(n^2 \log n)$-time. We improve this running time to $O(n^2)$-time.

Now, we introduce some notations. Let S be a set of n points in the Euclidean plane. Let $P = \langle p_1, p_2, \ldots, p_k \rangle$ be an empty polygon in $\mathcal{E}(S)$. A vertex p_i on P for $p \in \mathsf{out}(P)$ is *active* if $\mathsf{ins}(P, p_i, p)$ is a child of P. We define the *active sequence* of $p \in \mathsf{out}(P)$, denoted by $\mathsf{eact}(P, p)$, as the cyclic sequence of the active vertices for $p \in \mathsf{out}(P)$. In $\mathsf{eact}(P, p)$, the active vertices are arranged in clockwise order around p.

If we have $\mathsf{eact}(P, p)$ for every $p \in \mathsf{out}(P)$, it is easy to generate all the children of P. Thus, our algorithm maintains active sequences for the current empty polygon in the traversal of a family forest. To achieve this, we update the active sequences, when a child is generated. Below, we explain how to update the active sequences. Suppose that $\mathsf{ins}(P, p_i, p)$ is a child of P. When $\mathsf{ins}(P, p_i, p)$ is generated as a child of P, for each $q \in \mathsf{out}(P) \setminus \{p\}$, some vertices may be deleted from $\mathsf{eact}(P, q)$ and some vertices may be inserted into $\mathsf{eact}(P, q)$. Then, $\mathsf{eact}(\mathsf{ins}(P, p_i, p), q)$ is obtained. See Fig. 5 for examples. We consider the deletion and insertion of vertices for active sequences, respectively.

Deletion

Let q be a point in $\mathsf{out}(P) \setminus \{p\}$. Here, we consider finding and deleting every vertex p_j in $\mathsf{eact}(P, q)$ such that p_j is non-active in $\mathsf{ins}(P, p_i, p)$.

Phase 1: Check the largest ear-central vertex.

In $\mathsf{ins}(P, p_i, p)$, p is the largest ear-central vertex. Hence, every vertex p_j in $\mathsf{eact}(P, q)$ with $p_j \prec \mathsf{pred}(p)$ is non-active in $\mathsf{ins}(P, p_i, p)$ and deleted from $\mathsf{eact}(P, q)$. Suppose that $p_j = \mathsf{pred}(p)$ holds. In this case, we delete p_j from $\mathsf{eact}(P, q)$, since $(p_j, \mathsf{succ}(p_j))$ in P is removed. (In the insertion process described later, p_j may be inserted in active sequence of q.)

Phase 2: Check intersections.

By inserting p to P, the two line segments (p_i, p) and $(\mathsf{succ}(p_i), p)$ are inserted. Hence, p_j in $\mathsf{eact}(P, p_i)$ is deleted if (1) (p_j, q) intersects with (p_i, p) or $(\mathsf{succ}(p_i), p)$ or (2) $(\mathsf{succ}(p_j), q)$ intersects with (p_i, p) or $(\mathsf{succ}(p_i), p)$.

The above two phases can be done in $O(n)$-time for updating the active sequence of $\mathsf{out}(P) \setminus \{p\}$. Hence, we take $O(n^2)$-time for the deletions of all the active sequences.

Insertion

Suppose that a child $\mathsf{ins}(P, p_i, p)$ of P is generated from P. For any $q \in \mathsf{out}(P) \setminus \{p\}$, each of p_i and p can be active in $\mathsf{ins}(P, p_i, p)$. Hence, we have to insert them into the active sequence $\mathsf{eact}(P, q)$ of q if they are active for q. The details are as follows. First, we check whether p_i is active for q. Note that the vertex p_i is active for q in $\mathsf{ins}(P, p_i, p)$ if neither (q, p_i) nor (q, p) has an intersection and the triangle $\triangle(q, p_i, p)$

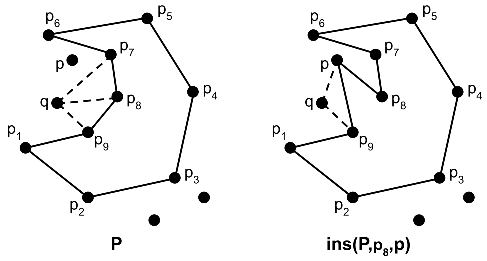

Fig. 5 Illustration for updating the active sequence of q. Each dashed line connects q and its active vertex. In the empty polygon P on the left side, the active sequence of q is $\mathsf{eact}(P, q) = \langle p_7, p_8, p_9 \rangle$. In $\mathsf{ins}(P, p_8, p)$ on the right side, the active sequence of q is $\mathsf{eact}(\mathsf{ins}(P, p_8, p), q) = \langle p, p_9 \rangle$.

Algorithm 6: FIND-CHILDREN-WITH-LISTS(P)

```
1 /* P is an empty polygon of a point set S.   */
2 Output P.
3 foreach $p \in \mathsf{out}(P)$ do
4     foreach $p_i$ in $\mathsf{eact}(P, p)$ do
5         Construct the active sequences of $\mathsf{ins}(P, p_i, p)$ by
            updating the active sequences of $P$.
6         FIND-CHILDREN-WITH-LISTS($\mathsf{ins}(P, p_i, p)$)
```

includes no point in its inside. This check can be done in $O(\log n)$-time using ray-shooting query [24], [25] and triangular range query [23]. Recall that the preprocessing of the ray-shooting query takes $O(n \log n)$-time for $\mathsf{ins}(P, p_i, p)$ and the preprocessing of the triangular range query takes $O(n^2)$-time for an input point set. If p_i is active for q, we insert it into $\mathsf{eact}(q, P(p_i, p))$. This can be done in $O(n)$-time. Next, we do the same process for checking whether p is active for q and insert into $\mathsf{eact}(q, P(p_i, p))$ if p is active. See an example of an insertion in Fig. 5. Since the active sequence of each $q \in \mathsf{out}(P) \setminus \{p\}$ takes $O(n)$-time for the insertion. Hence, we take $O(n^2)$-time for the insertions of all the active sequences.

Now, we re-describe our algorithm, which is shown in **Algorithm 6**. First, we output the current empty polygon P. Next, for each point p_i in the active sequence of each $p \in \mathsf{out}(P)$, we generate a child $\mathsf{ins}(P, p_i, p)$. Once a child is generated, we update all active sequences and obtain active sequences of $\mathsf{ins}(P, p_i, p)$. Last, we recursively call **Algorithm 6** to $\mathsf{ins}(P, p_i, p)$. Note that the main routine is almost the same as **Algorithm 4**. We enumerate all the empty triangles in $\mathcal{T}(S)$, construct the active sequences for each empty triangle, and call **Algorithm 6** for each empty triangle $T \in \mathcal{T}(S)$.

Below, we estimate the running time and space of our enumeration algorithm. Suppose that a child $\mathsf{ins}(P, p_i, p)$ is generated from an empty polygon P. To generate $\mathsf{ins}(P, p_i, p)$ and its active sequences, we take $O(n^2)$-time for the deletion and insertion processes. Next, we estimate the space complexity of our algorithm. Remember that the main routine (**Algorithm 4**) enumerates all the empty polygons in $\mathcal{T}(S)$ in $O((n \log n)\,|\mathcal{T}(S)|)$-time. A call of **Algorithm 6** to empty polygon P uses $O(n^2)$-space to store child lists for all the points in $\mathsf{out}(P)$. When the algorithm visits a deep recursive call, we have to store the information (including child lists) for all the ancestor recursive calls. Since the depth of the family forest is bounded above by $O(n)$, such information uses $O(n^3)$-space. However, this can be improved to $O(n^2)$-space, as follows. In the traversal of a family tree, let P be the current empty polygon in $\mathcal{E}(S)$. We store the active sequence of each $q \in \mathsf{out}(P)$ in global memory and update it while the algorithm traverses the family tree. When a child P' of P is generated, we store the difference of the active sequence for each q. The difference consists of (1) the list of deleted active vertices, which is stored in the clockwise order, and (2) the list of inserted active vertices. Note that only at most two active vertices can be inserted into the sequence. When the algorithm turns back from the recursive call of P' to the one of P, active sequence of each $q \in \mathsf{out}(P)$ can be recovered in $O(n)$-time. Note that both the active sequence of q and the list of deleted active vertices are stored in the clockwise order. The total space for storing differences from the current recursive call to the recursive call of the root node is bounded by $O(n^2)$, since the size of a list of deleted active vertices is bounded by $O(n)$. Hence, we have the following theorem.

Theorem 5: Let S be a set of n points in the Euclidean plane. One can enumerate all the empty polygons in $\mathcal{E}(S)$ in $O((n \log n)\,|\mathcal{T}(S)| + n^2\,|\mathcal{E}(S)|)$-time and $O(n^2)$-space.

4. Enumeration of Surrounding Polygons

Let S be a set of n points in the Euclidean plane, and let $\mathcal{S}(S)$ be the set of surrounding polygons of S. Yamanaka et al. [19] proposed an algorithm that enumerates all the surrounding polygons in $\mathcal{S}(S)$ in $O(n^2 \log n)$-delay and $O(n^2)$-space. The enumeration algorithm by Yamanaka et al. [19] was designed using the (standard) reverse search. In this section, we modify the algorithm such that it uses the reverse search with child lists. Our algorithm enumerates all the surrounding polygons in $\mathcal{S}(S)$ in $O(n^2)$-delay and $O(n^2)$-space.

We first explain a tree structure over $\mathcal{S}(S)$ and the child-enumeration algorithm proposed by Yamanaka et al. [19] briefly. Next, we explain an enumeration algorithm by using the reverse search with child lists. The content of this section is similar to Sect. 3. However, for self-containment of this section, we describe an enumeration algorithm in the context of enumerating surrounding polygons.

4.1 Family Tree

Now, we introduce some notations. Let $P = \langle p_1, p_2, \ldots, p_k \rangle$ be a surrounding polygon of S. The vertex p_i on P is *embeddable* if the triangle consisting of $\mathsf{pred}(p_i)$, p_i, and $\mathsf{succ}(p_i)$ does not intersect the interior of P. See examples in Fig. 6(a). Note that, for an embeddable vertex p_i, $\mathsf{rem}(P, p_i)$ is a surrounding polygon. A surrounding polygon of S except $\mathsf{CH}(S)$

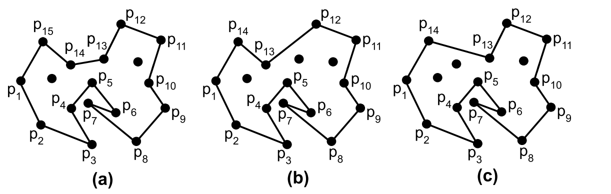

Fig. 6 (a) A surrounding polygon, where p_{10}, p_{13}, and p_{14} are embeddable. (b) The surrounding polygon obtained by removing p_{13}. The point p_{13} is embedded inside the polygon. (c) The parent of the polygon in (a), which is obtained by removing p_{14}.

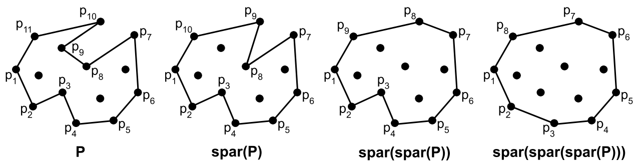

Fig. 7 A parent sequence of surrounding polygon.

always has an embeddable vertex, as mentioned in the following lemma.

Lemma 4 ([19]): Let S be a set of points, and let P be a surrounding polygon in $\mathcal{S}(S) \setminus \{\mathsf{CH}(S)\}$. Then, P has at least one embeddable vertex.

We denote by $\mathsf{l\text{-}emb}(P)$ the largest embeddable vertex on P. We define the *parent* of P, denoted by $\mathsf{spar}(P)$, as $\mathsf{rem}(P, \mathsf{l\text{-}emb}(P))$. For convenience, we denote $\mathsf{spar}(P) := \mathsf{rem}(P, \mathsf{l\text{-}emb}(P))$. Note that $\mathsf{spar}(P)$ is also a surrounding polygon of S. By repeatedly finding the parents from P, we obtain a sequence of surrounding polygons of S. The *parent sequence* $\mathsf{SPS}(P) = \langle P_1, P_2, \ldots, P_\ell \rangle$ of P is a sequence of surrounding polygons such that the first polygon is P itself and $P_i = \mathsf{spar}(P_{i-1})$ holds for each $i = 2, 3, \ldots, \ell$. See Fig. 7. As we can see in the following lemma, the last polygon in a parent sequence is always $\mathsf{CH}(S)$.

Lemma 5 ([19]): Let S be a set of n points in the Euclidean plane, and let P be a surrounding polygon in $\mathcal{S}(S)$. The last polygon of $\mathsf{SPS}(P)$ is $\mathsf{CH}(S)$.

From Lemma 5, for any surrounding polygon, the last polygon of its parent sequence is the convex hull. By merging the parent sequences for all the surrounding polygons in $\mathcal{S}(S)$, we have the tree structure rooted at $\mathsf{CH}(S)$. We call such a tree the *family tree* of $\mathcal{S}(S)$. An example of a family tree is shown in Fig. 8.

4.2 Enumeration by Reverse Search with Child Lists

In this subsection, we first describe the enumeration algorithm by Yamanaka et al. [19], which uses the reverse search. Then, we modify the algorithm such that it uses the reverse search with child lists.

Let $P = \langle p_1, p_2, \ldots, p_k \rangle$ be a surrounding polygon in $\mathcal{S}(S)$. From the definition of the parent, it can be observed that any child of P is described as $\mathsf{ins}(P, p_i, p)$ for p_i on P and $p \in \mathsf{in}(P)$. However, $\mathsf{ins}(P, p_i, p)$ may or may not be a

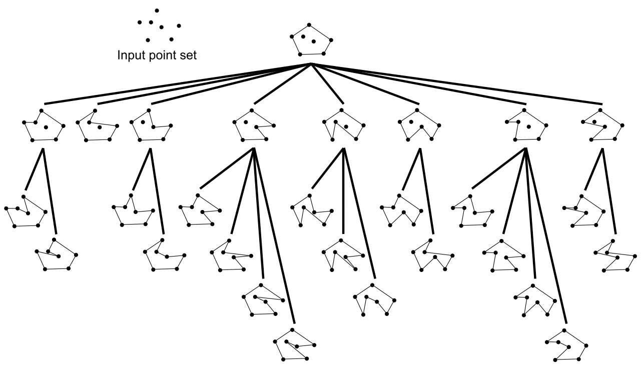

Fig. 8 An example of a family tree.

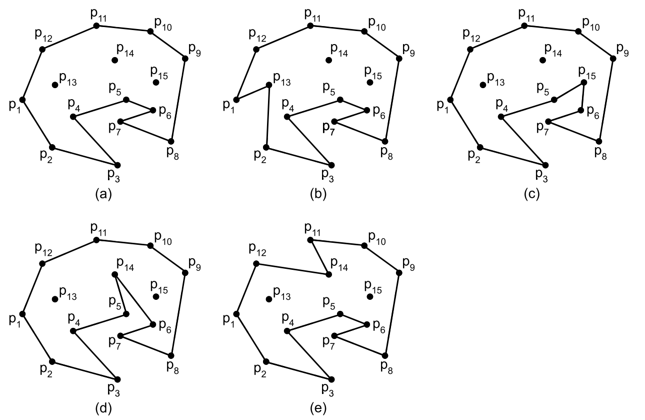

Fig. 9 (a) A surrounding polygon P, where l-ear(P) = p_6. (b) ins(P, p_1, p_{13}) is not a child of P, since p_{13} is not the largest embeddable vertex. (c) ins(P, p_5, p_{15}) is not a child of P, since p_{15} is not the largest embeddable vertex. (d) ins(P, p_5, p_{14}) is a child of P, since p_6 turns to be non-embeddable. (e) ins(P, p_{11}, p_{14}) is a child.

child of P. By using the following lemma, one can check whether ins(P, p_i, p) is a child of P.

Lemma 6 ([19]): Let $P = \langle p_1, p_2, \ldots, p_k \rangle$ be a surrounding polygon of a set S of points. For a point p_i ($1 \leq i \leq k$) on P and a point $p \in$ in(P), ins(P, p_i, p) is a child of P if

(1) ins(P, p_i, p) is a surrounding polygon of S and
(2) spar(ins(P, p_i, p)) = P holds.

Now, we describe the details of the condition (2) using a case analysis. If ins(P, p_i, p) is not a surrounding polygon, then ins(P, p_i, p) is not a child of P. Hence, in the case analysis below, we only consider the cases that ins(P, p_i, p) is a surrounding polygon.

Case 1: $p_i \prec$ pred(l-emb(P)).

If we apply an insertion to a point p_i smaller than pred(l-emb(P)), in the polygon ins(P, p_i, p), the largest embeddable is still l-emb(P), that is, l-emb(P) = l-emb(ins(P, p_i, p)) holds. Hence, spar(ins(P, p_i, p)) $\neq P$ holds. Therefore, ins(P, p_i, p) is not a child of P. See Fig. 9(b) for an example.

Case 2: p_i = pred(l-emb(P)).

If l-emb(P) is still embeddable in ins(P, p_i, p), then ins(P, p_i, p) is not a child of P. See Fig. 9(c). However, if l-emb(P) is non-embeddable in ins(P, p_i, p), then ins(P, p_i, p) is a child. See Fig. 9(d).

Case 3: pred(l-emb(P)) $\prec p_i$

In this case, p = l-emb(ins(P, p_i, p)) always holds. Hence, P = spar(ins(P, p_i, p)) holds. Thus, ins(P, p_i, p) is a child of P. See Fig. 9(e).

From the above case analysis and Lemma 6, we can generate all the children of P. Since the enumeration algorithm using the child-enumeration above is almost the same as **Algorithm 4** and **Algorithm 5**, we omit the description of their pseudo-codes.

Now, we describe an enumeration algorithm with child lists for surrounding polygons. As for empty polygons, we maintain the list of every vertex p_i on P for a point $p \in$ in(P) such that ins(P, p_i, p) is a child of P.

Below, we re-define active vertices and active sequences in the context of surrounding polygons. A vertex p_i on P for $p \in$ in(P) is *active* if ins(P, p_i, p) is a child of P. We define the *active sequence* of $p \in$ in(P), denoted by sact(P, p), as the cyclic sequence of the active vertices for $p \in$ in(P). In sact(P, p), the active vertices are arranged in counterclockwise order around p.

Next, we explain how to update the active sequences, when a child is generated. Suppose that ins(P, p_i, p) ($p \in$ in(P)) is a child of P. We consider the deletion and insertion of vertices for sact(P, p), respectively.

Deletion

Let q be a point in in(P) \ $\{p\}$. Here, we consider finding and deleting every vertex p_j in sact(P, q) such that p_j is non-active in ins(P, p_i, p).

Phase 1: Check the largest embeddable vertex.

In ins(P, p_i, p), p is the largest embeddable vertex. Hence, every vertex p_j in sact(P, q) with $p_j \prec$ pred(p_i) is non-active in ins(P, p_i, p) and deleted from sact(P, q). Suppose that p_j = pred(p_i) holds. In this case, we delete p_j from sact(P, q), since (p_j, succ(p_j)) in P is removed. (In the insertion process described later, p_j may be inserted in the active sequence of q.)

Phase 2: Check intersections.

By inserting p_i to P, the two line segments (p_i, p) and (succ(p_i), p) are inserted. Hence, p_j in sact(P, p_i) is deleted if (1) (p_j, q) intersects with (p_i, p) or (succ(p_i), p) or (2) (succ(p_j), q) intersects with (p_i, p) or (succ(p_i), p).

The above two phases can be done in $O(n^2)$-time for updating all the active sequences of P.

Insertion

For any $q \in$ in(P) \ $\{p\}$, each of p_i and p can be active for q in ins(P, p_i, p). Hence, we have to insert them into the active sequence sact(P, q) of q if they are active. The details are as

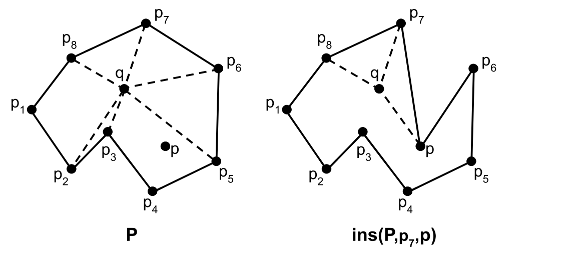

Fig. 10 Illustration for updating the active sequence of q. Each dashed line connects q and its active vertex. In the surrounding polygon P on the left side, the active sequence of q is $\mathsf{sact}(P, q) = \langle p_2, p_3, p_5, p_6, p_7, p_8\rangle$. In $\mathsf{ins}(P, p_6, p)$ on the right side, the active sequence of q is $\mathsf{sact}(\mathsf{ins}(P, p_6, p), q) = \langle p, p_7, p_8\rangle$.

follows.

First, we check whether p_i is active for q. Note that the vertex p_i is active for q in $\mathsf{ins}(P, p_i, p)$ if neither (q, p_i) nor (q, p) has an intersection and the triangle $\triangle(q, p_i, p)$ includes no point in its inside. This check can be done in $O(\log n)$-time using ray-shooting query [24], [25] and triangular range query [23]. If p_i is active for q, we insert it into $\mathsf{sact}(\mathsf{ins}(P, p_i, p), q)$. This can be done in $O(n)$-time. Next, we do the same process to check whether p is active for q. See Fig. 10 for an example.

From the deletion and insertion above, we can design a child-enumeration algorithm with child lists. The pseudocode of the algorithm is almost the same as **Algorithm 6**. Hence, we omit the description of the pseudo-code. Note that the root of the family tree of $\mathcal{S}(S)$ is $\mathsf{CH}(S)$ which can be computed by a standard algorithm for finding convex hulls in $O(n \log n)$-time.

Now, we estimate the running time and space of our enumeration algorithm. Suppose that a child $\mathsf{ins}(P, p_i, p)$ is generated from a surrounding polygon P. We estimate the running time for generating $\mathsf{ins}(P, p_i, p)$ and its active sequences. To generate $\mathsf{ins}(P, p_i, p)$ and its active sequences, we take $O(n^2)$-time for the insertion and deletion processes. Next, we estimate the space complexity of our algorithm. Similar to Theorem 5, we store only differences of active sequences when a child is generated. This attains $O(n^2)$-space. Hence, we have the following theorem.

Theorem 6: Let S be a set of n points in the Euclidean plane. One can enumerate all the surrounding polygons in $\mathcal{S}(S)$ in $O(n \log n + n^2 |\mathcal{S}(S)|)$-time and $O(n^2)$-space.

From the theorem above, one can see that the running time of our algorithm is output-polynomial[†]. Using the alternative output method by Nakano and Uno [26], we have a polynomial-delay enumeration algorithm. In the traversal, the algorithm outputs polygons with even depth when we go down a family tree and outputs polygons with odd depth when we go up. More precisely, we modify our algorithm so that the algorithm outputs the current surrounding polygon P before the children of P in even depth and outputs P after the children of P in odd depth. It is easy to see that the modified algorithm outputs a surrounding polygon once at most three edge traversals in a family. See [26] for further details.

[†]A running time is output-polynomial if the running time is a polynomial of an input size and an output size.

Corollary 1: Let S be a set of n points in the Euclidean plane. One can enumerate all the surrounding polygons in $\mathcal{E}(S)$ in $O(n^2)$-delay and $O(n^2)$-space.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers JP18H04091, JP19H01133, JP20H00595, JP20H00605, JP20H05962, JP20H05964, JP21H03559, JP21K17812, and JP21H05861 and JST ACT-X Grant Number JPMJAX2106, Japan.

References

[1] K. Wasa, “Enumeration of enumeration algorithms,” CoRR, vol.abs/1605.05102, 2016.

[2] D. Avis and K. Fukuda, “Reverse search for enumeration,” Discrete Applied Mathematics, vol.65, no.1-3, pp.21–46, 1996.

[3] S. Bespamyatnikh, “An efficient algorithm for enumeration of triangulations,” Computational Geometry, vol.23, no.3, pp.271–279, 2002.

[4] N. Katoh and S. Tanigawa, “Enumerating edge-constrained triangulations and edge-constrained non-crossing geometric spanning trees,” Discrete Applied Mathematics, vol.157, no.17, pp.3569–3585, 2009.

[5] M. Wettstein, “Counting and enumerating crossing-free geometric graphs,” Journal of Computational Geometry, vol.8, no.1, pp.47–77, 2017.

[6] Y. Nakahata, T. Horiyama, S. Minato, and K. Yamanaka, “Compiling crossing-free geometric graphs with connectivity constraint for fast enumeration, random sampling, and optimization,” CoRR, vol.abs/2001.08899, 2020.

[7] K. Yamanaka, S. Nakano, Y. Matsui, R. Uehara, and K. Nakada, “Efficient enumeration of pseudoline arrangements,” Proc. European Workshop on Computational Geometry 2009, pp.143–146, March 2009.

[8] T. Horiyama and W. Shoji, “Edge unfoldings of platonic solids never overlap,” Proc. 23rd Annual Canadian Conference on Computational Geometry, pp.65–70, 2011.

[9] T. Horiyama, M. Miyasaka, and R. Sasaki, “Isomorphism elimination by zero-suppressed binary decision diagrams,” Proc. 30th Canadian Conference on Computational Geometry, CCCG 2018, Aug. 2018, University of Manitoba, Winnipeg, Manitoba, Canada, S. Durocher and S. Kamali, eds., pp.360–366, 2018.

[10] G. Rote, G.J. Woeginger, B. Zhu, and Z. Wang, “Counting k-subsets and convex k-gons in the plane,” Inf. Process. Lett., vol.38, no.3, pp.149–151, 1991.

[11] G. Rote and G.J. Woeginger, “Counting convex k-gons in planar point sets,” Inf. Process. Lett., vol.41, no.4, pp.191–194, 1992.

[12] J.S.B. Mitchell, G. Rote, G. Sundaram, and G.J. Woeginger, “Counting convex polygons in planar point sets,” Inf. Process. Lett., vol.56, no.1, pp.45–49, 1995.

[13] D.P. Dobkin, H. Edelsbrunner, and M.H. Overmars, “Searching for empty convex polygons,” Algorithmica, vol.5, no.4, pp.561–571, 1990.

[14] D. Marx and T. Miltzow, “Peeling and nibbling the cactus: Subexponential-time algorithms for counting triangulations and related problems,” 32nd International Symposium on Computational Geometry, SoCG 2016, June 2016, Boston, MA, USA, pp.52:1–52:16, 2016.

[15] T. Auer and M. Held, "Heuristics for the generation of random polygons," Proc. 8th Canadian Conference on Computational Geometry, pp.38–43, 1996.
[16] C. Sohler, "Generating random star-shaped polygons," Proc. 11th Canadian Conference on Computational Geometry, pp.174–177, 1999.
[17] 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.
[18] C. Zhu, G. Sundaram, J. Snoeyink, and J.S.B. Mitchell, "Generating random polygons with given vertices," Computational Geometry, vol.6, no.5, pp.277–290, 1996.
[19] K. Yamanaka, D. Avis, T. Horiyama, Y. Okamoto, R. Uehara, and T. Yamauchi, "Algorithmic enumeration of surrounding polygons," Discrete Applied Mathematics, vol.303, pp.305–313, 2021.
[20] K. Yamanaka and S. Nakano, "Listing all plane graphs," Proc. Second International Workshop on Algorithms and Computation, (WALCOM 2008), LNCS, vol.4921, pp.210–221, 2008.
[21] G.H. Meisters, "Polygons have ears," American Mathematical Monthly, vol.82, no.6, pp.648–651, 1975.
[22] D. Avis, "Generating rooted triangulations without repetitions," Algorithmica, vol.16, pp.618–632, 1996.
[23] P.P. Goswami, S. Das, and S.C. Nandy, "Triangular range counting query in 2D and its application in finding k nearest neighbors of a line segment," Computational Geometry, vol.29, no.3, pp.163–175, 2004.
[24] B. Chazelle, H. Edelsbrunner, M. Grigni, L.J. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink, "Ray shooting in polygons using geodesic triangulations," Algorithmica, vol.12, no.1, pp.54–68, 1994.
[25] B. Chazelle and L.J. Guibas, "Visibility and intersection problems in plane geometry," Discrete Comput. Geom., vol.4, pp.551–581, 1989.
[26] S. Nakano and T. Uno, "Generating colored trees," Proc. 31th Workshop on Graph-Theoretic Concepts in Computer Science, (WG 2005), LNCS, vol.3787, pp.249–260, 2005.

Shunta Terui received his B.E. and M.E. degrees from Iwate University in 2020 and 2022, respectively. His research interests include combinatorial algorithms and computational geometry.

Katsuhisa Yamanaka is a professor of Faculty of Science and Engineering, Iwate University. He received B.E., M.E. and Dr. Eng. degrees from Gunma University in 2003, 2005 and 2007, respectively. His research interests include combinatorial algorithms and graph algorithms.

Takashi Hirayama received his B.E., M.E., and Dr. Eng. degrees in computer science from Gunma University in 1994, 1996, and 1999, respectively. From 1999 to 2001 he was a research assistant in the Department of Electrical and Electronics Engineering, Ashikaga Institute of Technology. He is currently a associate professor of Faculty of Science and Engineering, Iwate University. His research interests include high level and logic synthesis and design for testability of VLSIs.

Takashi Horiyama received the B.E. and M.E. degrees in information science and Ph.D. in informatics from Kyoto University, Kyoto, Japan in 1995, 1997 and 2004, respectively. He was a research associate at Nara Institute of Science and Technology from 1999, a research associate at Kyoto University from 2002, and an associate professor at Saitama University from 2007. Since 2019, he is a professor at Hokkaido University. His current interests include computational geometry and algorithm design.

Kazuhiro Kurita is an assistant professor of Graduate School of Informatics, Nagoya University. He received B.E., M.E. and Dr. Eng. degrees from Hokkaido University in 2015, 2017 and 2020, respectively. His research interests include enumeration algorithms, graph algorithms, and the design and analysis of algorithms in these fields.

Takeaki Uno received the Ph.D. degree (Doctor of Science) from Department of Systems Science, Tokyo Institute of Technology Japan, 1998. He was an assistant professor in Department of Industrial and Management Science in Tokyo Institute of Technology from 1998 to 2001, and was an associate professor of National Institute of Informatics Japan, from 2001 to 2013. He is currently a professor of National Institute of Informatics Japan, from 2014. His research topic is discrete algorithms, especially enumeration algorithms, algorithms on graph classes, and data mining algorithms. On the theoretical part, he studies low degree polynomial time algorithms, and hardness proofs. In the application area, he works on the paradigm of constructing practically efficient algorithms for large scale data that are data oriented and theoretically supported. In an international frequent pattern mining competition in 2004 he won the best implementation award. He got Young Scientists' Prize of The Commendation for Science and Technology by the Minister of Education, Culture, Sports, Science and Technology in Japan, 2010.