Algorithmic Enumeration of Surrounding Polygons[☆]

Katsuhisa Yamanaka[a], David Avis[b,c], Takashi Horiyama[d], Yoshio Okamoto[e,f], Ryuhei Uehara[g], Tanami Yamauchi[a]

[a] *Iwate University, Japan.*
[b] *McGill University, Canada.*
[c] *Kyoto University, Japan.*
[d] *Hokkaido University, Japan.*
[e] *The University of Electro-Communications, Japan.*
[f] *RIKEN Center for Advanced Intelligence Project, Japan*
[g] *Japan Advanced Institute of Science and Technology, Japan*

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 and quadratic space. It also provides the first space efficient method to generate all simple polygonizations on a given point set in exponential time. By relating these two problems we provide an upper bound on the number of surrounding polygons.

Keywords: polygons, surrounding polygons, simple polygonizations, enumeration algorithms, polynomial delay

[☆]A preliminary version appeared in the proceedings of the 35th European Workshop on Computational Geometry (EuroCG 2019), pp. 1:1–1:6, 2019.

Email addresses: yamanaka@cis.iwate-u.ac.jp (Katsuhisa Yamanaka), avis@i.kyoto-u.ac.jp (David Avis), horiyama@ist.hokudai.ac.jp (Takashi Horiyama), okamotoy@uec.ac.jp (Yoshio Okamoto), uehara@jaist.ac.jp (Ryuhei Uehara), tanami@kono.cis.iwate-u.ac.jp (Tanami Yamauchi)

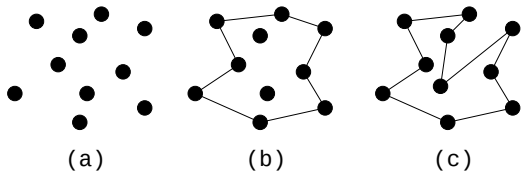

Figure 1: (a) A point set S. (b) A surrounding polygon of S. (c) A simple polygonization of S.

1. Introduction

Enumeration problems are fundamental and important in computer science. Enumerating geometric objects are studied for triangulations [4, 5, 13], non-crossing spanning trees [4, 13], pseudoline arrangements [25], non-crossing matchings [24], unfoldings of Platonic solids [12], and so on. In this paper, we focus on the enumeration problem of generating simple polygons with the following additional property. 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 [14], and it runs in $n^{\mathrm{O}(\sqrt{n})}$ 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 [16]. Much attention has been paid for combinatorial counting, too. A history on the lower and upper bounds is summarized by Demaine [8] and O'Rourke *et al.* [19]. 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 [9] and 54.55^n [20], respectively.

Another research topic is a random generation of simple polygonizations. Since no polynomial-time counting algorithm is known for simple polygo-

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

nizations, 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 [26] and star-shaped polygons [21]. These uniform random generations are based on counting. For general simple polygonizations, heuristic algorithms are known [3, 22, 26]. Those algorithms efficiently generate simple polygons, but not uniformly at random.

At the SPA Workshop in Yugawara in 1991, Avis asked whether the set of all simple polygonizations on a planar n point set is connected by local 2-edge flip operations, allowing their enumeration by the reverse search technique [4]. Such a technique works, for example, for triangulations and non-crossing spanning trees. Unfortunately, Hernando, Houle and Hurtado [11] showed the answer is no. Whether allowing k-edge flips for larger k is sufficient remains open. In fact, very little is known about the problem of enumerating all the simple polygonizations, as mentioned in [23]. A trivial enumeration is to generate all the permutations of given points, then output only simple polygonizations. However this method requires $\Omega(n!)$ time whereas it is known that the number of polygonizations is exponentially bounded in n. The algorithm we give for enumerating surrounding polygons uses reverse search and $O(n^2)$ space. The output contains a list of all simple polygonizations on S, providing a partial answer to Avis' problem. In Section 5, by relating the two problems, we will see that the algorithm's running time is exponentially bounded in n providing a big improvement on the trivial algorithm. We note here that a very recent result of Nakahata et al. [17] also provides an exponential algorithm for enumerating all simple polygonizations using a completely different technique. The space used is exponential in n. There remains the interesting and challenging question of whether all the simple polygonizations of a given point set can be enumerated in output-polynomial time[2] or in polynomial delay[3].

In this paper we consider the problem of enumerating the surrounding polygons of a given point set S. From the definition, the set of surrounding

[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.

polygons of S includes the set of simple polygonizations of S. We show that, for this enumeration problem, the reverse search can be applied. First, we introduce an "embedding" operation: deleting a vertex from a surrounding polygon 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.

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 1 ([15])**.** *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 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 in $\mathcal{P}(S)$. 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 2. *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.*

Proof. Let us define some notations. The *complement*, denoted by $\overline{P}$, of P is the set of regions outside P and inside $\mathsf{CH}(S)$: $\overline{P} = \mathsf{CH}(S) \setminus P$. Note that the complement consists of zero or more simple polygons (if $P = \mathsf{CH}(S)$, $\overline{P}$ is the empty set).

It can be observed that a vertex p on P is embeddable if and only if the triangle consisting of $\mathsf{pred}(p)$, p, and $\mathsf{succ}(p)$ is either (1) a simple polygon of 3 vertices in $\overline{P}$ or (2) an ear of a simple polygon in $\overline{P}$.

If $\overline{P}$ includes a simple polygon of 3 vertices, then P includes an embeddable vertex. Otherwise, let us assume that $\overline{P}$ has no simple polygon of 3 vertices. Then, a simple polygon Q with 4 or more vertices in $\overline{P}$ exists, since $P \neq \mathsf{CH}(S)$ holds. From Theorem 1, Q has at least two non-overlapping ears. At least one of them includes an embeddable vertex on P. □

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* to p 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 embeddable vertex of largest index on P. The *parent* of P, denoted by $\mathsf{par}(P)$, is the polygon obtained by embedding $\mathsf{larg}(P)$ in the interior of P. Now, we have the following lemma.

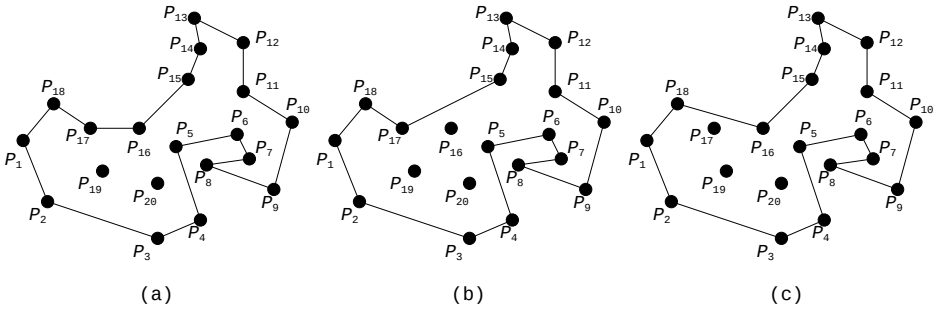

Figure 2: (a) A surrounding polygon, where $p_6, p_7, p_{11}, p_{14}, p_{15}, p_{16}$, and p_{17} are embeddable. (b) The surrounding polygon obtained by embedding p_{16}. The point p_{16} is embedded inside the polygon. (c) The parent of the polygon in (a), which is obtained by embedding p_{17}.

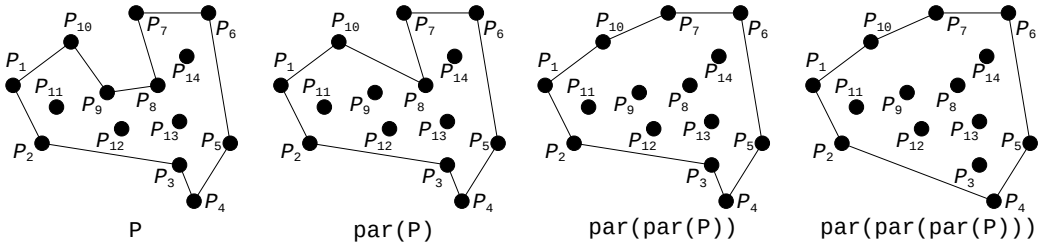

Figure 3: A parent sequence.

Lemma 3. *Let S be a set of n points in the Euclidean plane, and let P be a surrounding polygon in $\mathcal{P}(S) \setminus \{\mathsf{CH}(S)\}$. Then, the parent $\mathsf{par}(P)$ of P always exists and is unique.*

Proof. From Lemma 2, P has at least one embeddable vertex. Hence, $\mathsf{larg}(P)$ is always defined. Moreover, $\mathsf{larg}(P)$ is unique for P from its definition. Therefore the claim holds. □

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 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 4. *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 $\mathsf{CH}(S)$.*

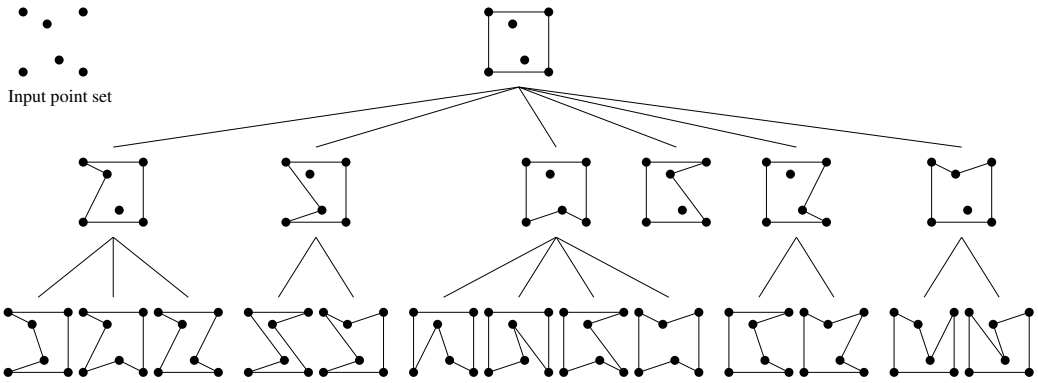

Figure 4: An example of a family tree.

Proof. Let $\mathsf{PS}(P) = \langle P_1, P_2, \ldots, P_\ell \rangle$ be the parent sequence of P. For a polygon P_i in $\mathsf{PS}(P)$, we define a function $\phi(P_i)$ as the number of edges on P_i. Note that $\phi(\mathsf{CH}(S)) \leq \phi(P_i)$ holds and $\phi(P_i) = \phi(\mathsf{CH}(S))$ if and only if $P_i = \mathsf{CH}(S)$. From the definition of the parent, it can be observed that $\phi(P_i) = \phi(P_{i-1}) - 1$ for each $i = 2, 3, \ldots, \ell$. From Lemma 2, the parent of a surrounding polygon always exists unless the surrounding polygon is the convex hull. Therefore, $P_\ell = \mathsf{CH}(S)$ holds. □

From Lemma 4, 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 enumeration 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 over $\mathcal{P}(S)$. We know that the root of the family tree is the convex hull of S. Hence, we obtain 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 of any surrounding polygon of S. Starting from the root, we apply the 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 $(p_i, \mathsf{succ}(p_i))$ of P and a point p inside P, we denote by $P(p_i, p)$ the polygon obtained by removing $(p_i, \mathsf{succ}(p_i))$ and inserting two edges (p_i, p) and $(p, \mathsf{succ}(p_i))$. Intuitively, this operation is the reverse one of the embedding operation. We call it a *digging operation*. Any child of P is described as $P(p_i, p)$ for some p_i and p. Hence, for all possible $P(p_i, p)$, if we can check whether or not $P(p_i, p)$ is a child, then one can enumerate all the children. We have the following observation.

Lemma 5. *Let $P = \langle p_1, p_2, \ldots, p_k \rangle$ be a surrounding polygon of a set of points. For a point p_i $(1 \leq i \leq k)$ on P and a point p inside P, $P(p_i, p)$ is a child of P if*

(1) $P(p_i, p)$ is a surrounding polygon of S and

(2) $\mathsf{par}(P(p_i, p)) = P$ holds.

Actually, using the conditions in Lemma 5, we obtain the child-enumeration algorithm. However, let us give a more detailed case analysis to describe our algorithm. If $P(p_i, p)$ is not a surrounding polygon, then $P(p_i, p)$ is not a child of P (see Figure 5(b)). Hence, in the case analysis below, we only consider the cases that $P(p_i, p)$ is a surrounding polygon.

Case 1: $p_i \prec \mathsf{pred}(\mathsf{larg}(P))$.

Here, $\mathsf{pred}(\mathsf{larg}(P))$ is the predecessor of $\mathsf{larg}(P)$. In this case, if we apply a digging operation to a point p_i smaller than $\mathsf{pred}(\mathsf{larg}(P))$, in the polygon $P(p_i, p)$, the embeddable vertex of largest index is still $\mathsf{larg}(P)$, that is, $\mathsf{larg}(P) = \mathsf{larg}(P(p_i, p))$ holds. Hence, $\mathsf{par}(P(p_i, p)) \neq P$ holds. Therefore, $P(p_i, p)$ is not a child of P. See Figure 5(c) for an example.

Case 2: $p_i = \mathsf{pred}(\mathsf{larg}(P))$.

If $\mathsf{larg}(P)$ is still embeddable in $P(p_i, p)$, then $P(p_i, p)$ is not a child of P. See Figure 5(d). However, if $\mathsf{larg}(P)$ is not embeddable in $P(p_i, p)$, then $P(p_i, p)$ is a child. See Figure 5(e).

Case 3: $\mathsf{pred}(\mathsf{larg}(P)) \prec p_i$

In this case, $p = \mathsf{larg}(P(p_i, p))$ always holds. Hence, $P = \mathsf{par}(P(p_i, p))$ holds. Thus, $P(p_i, p)$ is a child of P. See Figure 5(f).

The above case analysis gives the algorithm shown in **Algorithm 1**. We apply the algorithm recursively starting from the convex hull, and we can

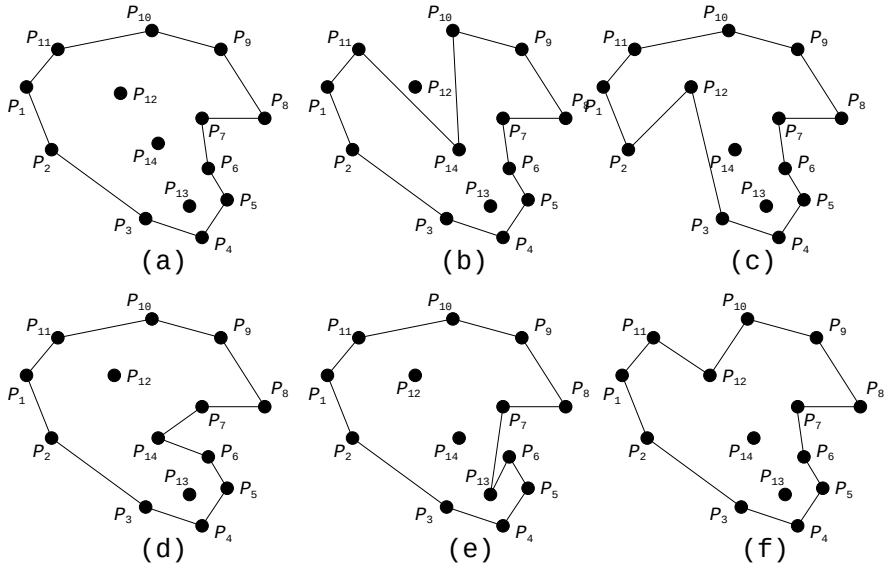

Figure 5: (a) A surrounding polygon P, where $\mathsf{larg}(P) = p_7$. (b) $P(p_{10}, p_{14})$ is not a child of P, since it is not a surrounding polygon. (c) $P(p_2, p_{12})$ is not a child of P. (d) $P(p_6, p_{14})$ is not a child of P. (e) $P(p_6, p_{13})$ is a child of P, since p_7 is not embeddable in $P(p_6, p_{13})$. (f) $P(p_{10}, p_{12})$ is a child of P.

traverse the family tree. one can enumerate all the surrounding polygons in $\mathcal{P}(S)$.

Theorem 6. *Let S be a set of n points in Euclidean plane. **Algorithm 1** enumerates all the surrounding polygons in $\mathcal{P}(S)$ in $O((n^2 \log n)\,|\mathcal{P}(S)|)$-time and $O(n^2)$-space.*

Proof. We first show that **Algorithm 1** enumerates all the surrounding polygons in $\mathcal{P}(S)$ without duplicate. For any surrounding polygon in $\mathcal{P}(S)$, the parent is always defined (Lemma 3) and its parent sequence ends up with the convex hull (Lemma 4). Hence, every surrounding polygon is included in the family tree of $\mathcal{P}(S)$. Moreover, for any surrounding polygon in $\mathcal{P}(S) \setminus \mathsf{CH}(S)$, the parent is defined uniquely (Lemma 3). Hence, **Algorithm 1** outputs each surrounding polygon exactly once. Therefore, the algorithm enumerates all the surrounding polygons in $\mathcal{P}(S)$ without duplicate.

Next, we analyze the running time of our enumeration algorithm. We estimate the running time for one recursive call of **Algorithm 1** in worst case. Since the double for-loops in Lines 8–11 dominate the running time of a

Algorithm 1: FIND-CHILDREN(P)

```
1  /* P is a surrounding polygon of a point set S and S is
      stored in a global variable.                          */
2  Output P.
3  Set p_i = pred(larg(P)).
4  foreach point p inside P do                              // Case 2
5      if larg(P) is not embeddable in P(p_i, p) then
6          if P(p_i, p) is a surrounding polygon then
7              FIND-CHILDREN(P(p_i, p))
8  foreach p_i with pred(larg(P)) ≺ p_i do                  // Case 3
9      foreach point p inside P do
10         if P(p_i, p) is a surrounding polygon then
11             FIND-CHILDREN(P(p_i, p))
```

recursive call, we analyze the running time of only this part. Here, the outer for-loop takes $O(n)$ iterations. Similarly, the inner for-loop takes $O(n)$ iterations. The if-condition can be checked in $O(\log n)$ time, as follows. Recall that P is a surrounding polygon. Hence, if (1) the triangle consisting of p_i, $\mathsf{succ}(p_i)$ and p has no point inside and (2) each of line segments (p_i, p) and $(\mathsf{succ}(p_i), p)$ has no intersection with edges of P, $P(p_i, p)$ is a surrounding polygon (the first condition in Lemma 5). The condition (1) can be checked using a triangular range query [10], which asks the designated three points include a point inside. This query can be done in $O(\log n)$ time with $O(n^2)$-time preprocessing and $O(n^2)$-additional space for an input point set. The condition (2) can be checked using a ray-shooting query, which asks the first intersection of a query ray (a half line) with the polygon. It is easy to observe that the first intersection point is further than p from p_i (and $\mathsf{succ}(p_i)$) if and only if (p_i, p) (and $(\mathsf{succ}(p_i), p)$) has no intersection, respectively. A ray-shooting query can be done in $O(\log n)$ time with $O(n \log n)$-time preprocessing and $O(n)$-additional space for each surrounding polygon [6, 7]. For the triangular range queries, the algorithm requires $O(n^2)$-time preprocessing only once and $O(n^2)$-additional space. For the ray-shooting queries, each recursive call in the algorithm requires $O(n \log n)$-time preprocessing and $O(n)$-additional space. Therefore, the running time of each recursive call is

$O(n^2 \log n)$ time and the whole of the algorithm requires $O(n^2)$ space. □

From the theorem above, one can see that the running time of our algorithm is output-polynomial. Using the alternative output method in [18], we have a polynomial-delay time 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 **Algorithm 1** 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 tree. See [18] for further details.

Corollary 7. *Let S be a set of n points in Euclidean plane. Algorithm 1 enumerates all the surrounding polygons in $\mathcal{P}(S)$ in $O(n^2 \log n)$-delay and $O(n^2)$-space.*

5. The number of surrounding polygons and simple polygonizations

In this section we relate the number of simple polygonizations and surrounding polygons on a given point set S. Using known results on the latter allows us to give a worst case upper bound on the output size of our algorithm. Since the relationship is quite tight this also shows that our algorithm gives a relatively good method to enumerate simple polygonizations, which is space efficient.

Let S be a set of n points in Euclidean plane. We denote by $\mathcal{SP}(S)$ the set of its simple polygonizations and by $\mathcal{P}(S)$ the set of its surrounding polygons.

Proposition 1. *Let S be a set of n points in Euclidean plane. Suppose the number of simple polygonizations of S is at most c^n for a constant c, that is $|\mathcal{SP}(S)| \leq c^n$. Then the number of surrounding polygons of S is at most $(c+1)^n$, that is $|\mathcal{P}(S)| \leq (c+1)^{n+1}$.*

Proof. Let $\mathcal{P}(S, p)$ be the set of surrounding polygons of S with p inside points. Then we have the following inequality:

$$|\mathcal{P}(S, p)| \leq \binom{n}{p} c^{n-p}.$$

Hence, we have

$$\begin{aligned}|\mathcal{P}(S)| &= \sum_{p=1}^{n-3} |\mathcal{P}(S,p)| \\ &\leq \sum_{p=1}^{n} \binom{n}{p} c^{n-p} \\ &= (c+1)^n.\end{aligned}$$

□

For any point set, the number of simple polygonizations is bounded above by $O^*(54.55^n)$ [20]. Hence the proposition implies that for any point set the number of surrounding polygons is bounded above by $O^*(55.55^n)$. As remarked earlier, a very recent result [17] gives a method using $O^*(4^n)$ time and exponential space to count the number of simple polygonizations. Their method can also be used to generate them.

Note that since the base of the exponentials differ by one, Algorithm 1 may not give either an output polynomial or polynomial time delay method for enumerating simple polygonizations. Although the analysis above is surely not tight, we present empirical evidence in the next section that shows the output sizes of the two problems do in fact grow at different rates.

6. Experimental results

We implemented our enumeration algorithm in Section 4. This section shows experimental results for small point sets. We ran our enumeration algorithm for random point sets and point sets representing order types.

Our experimental environment is as follows: Intel Xeon(R) W-2102 CPU 2.90GHz x 4 processor, 32GB memory, Ubuntu 18.04 OS, GNU C++ compiler, and CGAL 4.11-2build1.

We generated random point sets in Euclidean plane, as follows. For each $k = 3, 4, \ldots, 12$ and $p = 0, 1, \ldots, 9$ with $k + p \leq 12$, we randomly generated 100 point sets in an unit grid such that the convex hull of each point set consists of k vertices and includes p points inside. We enumerated all the surrounding polygons for every generated point set for each k and p. We calculated the numbers of surrounding polygons and simple polygonizations for

each point set to compare them. Besides, we calculated the number of leaves in the family tree. It can be observed that, in Figure 4, almost all leaves are simple polygonizations. More precisely, a surrounding polygon P is a simple polygonization if and only if P is a leaf with the maximum depth in a family tree. To practically investigate how many leaves are simple polygonizations, we also calculated the number of leaves of the family tree for each point set. Our enumeration algorithm (**Algorithm 1**) can be used to enumerate all the simple polygonizations, if we modify the algorithm so that only simple polygonizations are output. Note that all the simple polygonizations appear as leaves in a family tree. To estimate the running time for such an enumeration, the ratio between the number of surrounding polygons and leaves of a family tree plays an important role. Indeed, we gave a upper bound for the number of surrounding polygons using bounds for the number of simple polygonizations (Proposition 1). In this section, we investigate the behavior of the ratio empirically.

Table 1 shows our experimental results. Each cell consists of three numbers: (1) the average number of surrounding polygons ($\#\mathsf{surp}(k,p)$), (2) the average number of simple polygonizations ($\#\mathsf{simp}(k,p)$), and (3) the average number of leaves of family trees ($\#\mathsf{leaves}(k,p)$) for 100 point sets for each k and p.

First, we can see the following basic observations from Table 1. When we fix k, as p increases, both $\#\mathsf{surp}(k,p)$, $\#\mathsf{simp}(k,p)$, and $\#\mathsf{leaves}(k,p)$ increase, respectively. Similarly, if we fix p, as k increases, the three values increase, respectively. When we fix $k+p$, as k decreases and p increases, the three values increase, respectively.

Next, let us compare $\#\mathsf{simp}(k,p)$ with $\#\mathsf{leaves}(n,p)$ and $\#\mathsf{surp}(n,p)$, respectively. It is easy to see that $\#\mathsf{simp}(k,p) \leq \#\mathsf{leaves}(k,p) \leq \#\mathsf{surp}(k,p)$ holds. Let us focus on the ratio $\frac{\#\mathsf{leaves}(k,p)}{\#\mathsf{simp}(k,p)}$. For very small k and p, $\#\mathsf{simp}(k,p)$ and $\#\mathsf{leaves}(k,p)$ take close values. However, when we fix k, as p increases, $\frac{\#\mathsf{leaves}(k,p)}{\#\mathsf{simp}(k,p)}$ increases. Next, let us see the ratio $\frac{\#\mathsf{surp}(k,p)}{\#\mathsf{simp}(k,p)}$. When we fix k, as p increases, $\frac{\#\mathsf{surp}(k,p)}{\#\mathsf{simp}(k,p)}$ increases. In our experiments, for a fixed k, as p grows, both $\#\mathsf{leaves}(k,p)$ and $\#\mathsf{surp}(k,p)$ grow faster than $\#\mathsf{simp}(k,p)$. Therefore, it is easy to imagine that the above two ratios tend to increase monotonically. Proving or disproving this an interesting open problem.

Finally, we ran our algorithm for point sets representing order types [2]. Points sets representing all the order types are published in Aichholzer's web page [1]. The purpose of these experiments is to investigate the ex-

k \ p	0	1	2	3	4	5	6	7	8	9
3	1	4	13	43	149	548	1970	7291	28632	111851
	1	3	8	22	70	230	735	2438	8556	30349
	1	3	8	26	90	319	1132	4159	16226	63121
4	1	5	19	73	272	1013	4066	16172	60910	-
	1	4	12	40	129	424	1505	5327	17906	-
	1	4	13	47	168	616	2398	9471	35580	-
5	1	6	27	113	463	1894	7710	31794	-	-
	1	5	18	64	224	797	2865	10436	-	-
	1	5	20	75	297	1176	4681	19005	-	-
6	1	7	37	173	764	3329	14081	-	-	-
	1	6	26	101	381	1439	5305	-	-	-
	1	6	27	121	503	2121	8791	-	-	-
7	1	8	47	247	1207	5423	-	-	-	-
	1	7	34	150	623	2390	-	-	-	-
	1	7	36	175	823	3544	-	-	-	-
8	1	9	60	342	1754	-	-	-	-	-
	1	8	45	214	929	-	-	-	-	-
	1	8	48	251	1216	-	-	-	-	-
9	1	10	74	459	-	-	-	-	-	-
	1	9	57	297	-	-	-	-	-	-
	1	9	60	343	-	-	-	-	-	-
10	1	11	89	-	-	-	-	-	-	-
	1	10	70	-	-	-	-	-	-	-
	1	10	73	-	-	-	-	-	-	-
11	1	12	-	-	-	-	-	-	-	-
	1	11	-	-	-	-	-	-	-	-
	1	11	-	-	-	-	-	-	-	-
12	1	-	-	-	-	-	-	-	-	-
	1	-	-	-	-	-	-	-	-	-
	1	-	-	-	-	-	-	-	-	-

Table 1: Experimental results. Each cell consists of the average number of surrounding polygons, the average number of simple polygonizations, the average number of the leaves in family trees for each $k = 3, 4, \ldots, 12$ and $p = 0, 1, \ldots, 9$ such that $k + p \leq 12$. The numbers after decimal points are removed in the table.

n	#point sets	$\#\mathsf{surp}_{\mathsf{ord}}(n)$		$\#\mathsf{simp}_{\mathsf{ord}}(n)$		Time(s)
		Ave	Max	Ave	Max	
4	2	2.0	4	2.0	3	< 0.01
5	3	6.3	13	4.3	8	< 0.01
6	16	25.1	54	14.6	29	0.04
7	135	91.7	193	46.9	92	1.54
8	3315	338.5	783	153.9	339	123.27
9	158817	1281.1	3182	521.5	1282	22801.60

Table 2: The number of point sets representing different order types (from Aichholzer's webpage [1]), the average and maximum numbers of the surrounding polygons ($\#\mathsf{surp}_{\mathsf{ord}}(n)$) and the average and maximum numbers of simple polygonizations ($\#\mathsf{simp}_{\mathsf{ord}}(n)$) for $n = 4, 5, 6, 7, 8, 9$. Each average number is rounded down to one decimal place. Time column shows running times to enumerate all the surrounding polygons for all point sets representing order types for each $n = 4, 5, 6, 7, 8, 9$ (< 0.01 means less than 0.01 seconds).

act maximum numbers of surrounding polygons and simple polygonizations among points sets representing all order types. For $n = 4, 5, 6, 7, 8, 9$, we calculated the average and maximum numbers of the surrounding polygons ($\#\mathsf{surp}_{\mathsf{ord}}(n)$) and the simple polygonizations ($\#\mathsf{simp}_{\mathsf{ord}}(n)$), which are shown in Table 2. Running times are also shown in the table. We plot the maximum numbers in the table in Figure 6 and fit the plotted values using an exponential function $d(c^n)$, where c and d are constants, using Gnuplot 5.2. As a result, we obtained two fitting functions: $0.0106(4.0577^n)$ for surrounding polygons and $0.0085(3.7597^n)$ for simple polygonizations. From these fitting functions, it seems that the maximum numbers of surrounding polygons grows exponentially faster than the one of simple polygonizations. Thus, we conjecture that the enumeration algorithm for simple polygonizations using **Algorithm 1** is neither polynomial-delay nor output-polynomial.

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, JP17H06287, JP18H04091, JP18K11153.

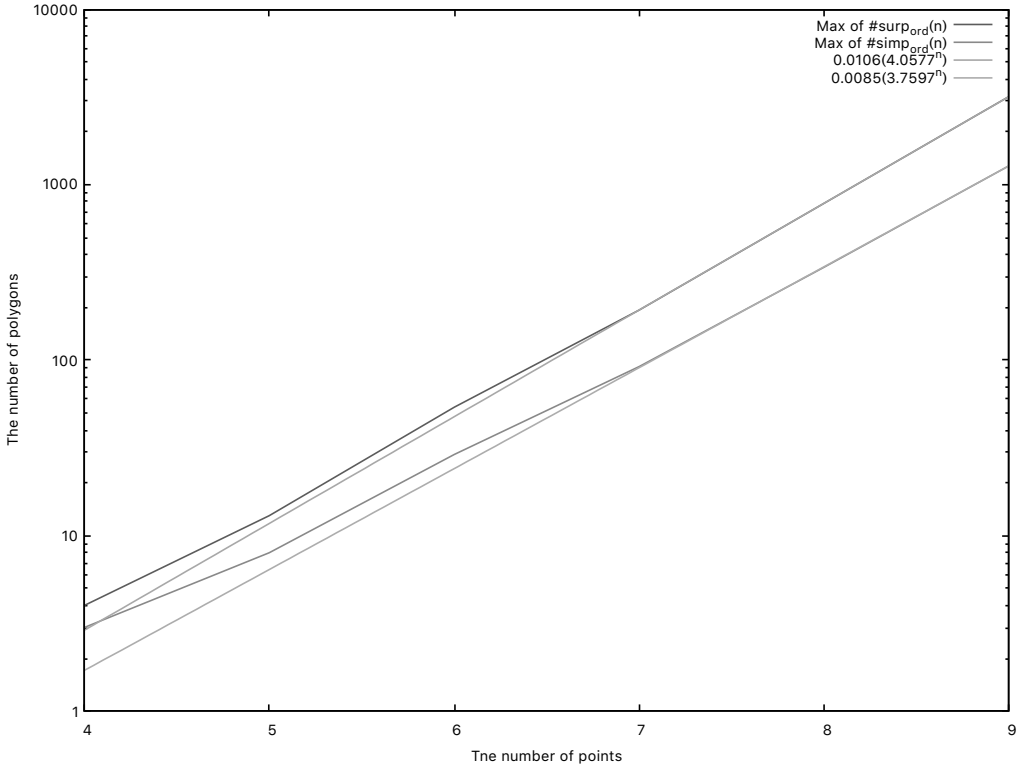

Figure 6: A semi-log graph for the maximum numbers of surrounding polygons and simple polygonizations for $n = 4, 5, 6, 7, 8, 9$. The purple line connects the plotted values for the maximum numbers of surrounding polygons. The light blue dotted line is a fitting function, $0.0106(4.0577^n)$, for the values. The green line connects the plotted values for the maximum numbers of simple polygonizations. The yellow dotted line is a fitting function, $0.0085(3.7597^n)$, for the values.

JP19K11812, The fourth author is also supported by JST CREST Grant Number JPMJCR1402 and Kayamori Foundation of Informational Science Advancement.

References

[1] O. Aichholzer. `http://www.ist.tugraz.at/staff/aichholzer/research/rp/triangulations/ordertypes/`, 2006.

[2] O. Aichholzer, F. Aurenhammer, and H. Krasser. Enumerating order types for small point sets with applications. *Order*, 19(3):265–281, 2002.

[3] 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.

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

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

[6] B. Chazelle, H. Edelsbrunner, M. Grigni, L. J. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

[7] B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. *Discrete & Computational Geometry*, 4:551–581, 1989.

[8] E. D. Demaine. `http://erikdemaine.org/polygonization/`, 2012.

[9] 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.

[10] 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*, 29(3):163 – 175, 2004.

[11] M. C. Hernando, M. E. Houle, and F. Hurtado. On local transformation of polygons with visibility properties. *Theor. Comput. Sci.*, 289(2):919–937, 2002.

[12] 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.

[13] 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.

[14] D. Marx and T. Miltzow. Peeling and nibbling the cactus: Subexponential-time algorithms for counting triangulations and related problems. In *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA*, pages 52:1–52:16, 2016.

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

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

[17] 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*, abs/2001.08899, 2020.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

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

[25] 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.

[26] 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.