Fast Enumeration Algorithms for Non-crossing Geometric Graphs

Naoki Katoh and Shin-ichi Tanigawa

Department of Architecture and Architectural Engineering, Kyoto University,
Kyoto Daigaku Katsura, Nishikyo-ku, Kyoto 615-8540 Japan,
{naoki,is.tanigawa}@archi.kyoto-u.ac.jp

Abstract

A non-crossing geometric graph is a graph embedded on a given set of points in the plane with non-crossing straight line segments. In this paper we present a new general framework for enumerating non-crossing geometric graphs for a given point set. By applying our idea to specific enumeration problems, we obtain faster algorithms for enumerating plane straight-line graphs, non-crossing spanning connected graphs, non-crossing spanning trees and non-crossing minimally rigid frameworks. Furthermore, we also obtain efficient enumeration algorithms for non-crossing geometric graph classes, for which no enumeration algorithm has been reported so far, such as non-crossing matchings, non-crossing blue-and-red matchings, non-crossing k-vertex or k-edge connected graphs or non-crossing directed spanning trees. The proposed idea is relatively simple, and can be potentially applied to various other enumeration problems of non-crossing geometric graphs.

1 Introduction

Given a graph $G = (V, E)$ with n vertices and m edges where $V = \{1, \ldots, n\}$, an embedding of the graph on a set of points $P = \{p_1, \cdots, p_n\} \subset \mathbf{R}^2$ is a mapping of the vertices to the points in the Euclidean plane $i \mapsto p_i$. A *geometric graph (on P)* is a graph embedded on P such that each edge (i, j) of G is mapped to a straight line segment (p_i, p_j). A set of embedded segments is called *non-crossing* if any pair of elements does not have a point in common except possibly their endpoints, and so a geometric graph is non-crossing if its corresponding straight line segments is non-crossing.

In this paper we assume that a given point set P is *fixed* in $\mathbf{R}^2$, and an embedding $V \leftarrow P$ is given. This means that the property whether $G = (V, E)$ is non-crossing or not is combinatorially determined. A graph class is a collection of graphs that is defined by a property that all its members share. By imposing to a graph class the additional requirement that embedded segments are non-crossing, we can define a non-crossing geometric graph class, such as non-crossing spanning trees or non-crossing perfect matchings. Let us denote by $\mathcal{NGG}$ a specific non-crossing geometric graph class. In this paper we extensively study the following *enumeration problem*:

Input: A *fixed* point set P in the plane with n points.

Output: The list of all the non-crossing geometric graphs belonging to $\mathcal{NGG}$ on P.

Since the output of the problem may consist of exponentially many graphs in terms of the input size, the efficiency of the *enumeration algorithm* is measured customarily in both the input and output sizes. In particular, if the computational time can be bounded by a polynomial in the input size and by a linear function in the output, the algorithm is said to work in *polynomial time (on average)*.

In this paper we will present a new general framework for enumerating non-crossing geometric graphs. Our new framework provides faster algorithms for various enumeration problems compared with existing ones, such as those for *plane straight-line graphs*, *non-crossing spanning connected graphs*, *non-crossing spanning trees* and *non-crossing minimally rigid frameworks*. Moreover the idea is quite simple. So the technique could be easily applied to many enumeration problems, for

Table 1: Time complexities of new algorithms and previous ones.

	New results	Previous best results
plane graphs	$O(\mathsf{pg}(P))$	$O(n \log n \cdot \mathsf{pg}(P))$ [2]
non-crossing spanning connected graphs	$O(\mathsf{cg}(P))$	$O(n \log n \cdot \mathsf{cg}(P))$ [2]
non-crossing spanning trees	$O(n \cdot \mathsf{tri}(P) + \mathsf{st}(P))$	$O(n \log n \cdot \mathsf{st}(P))$ [2]
non-crossing minimally rigid frameworks	$O(n^2 \cdot \mathsf{mrf}(P))$	$O(n^3 \cdot \mathsf{mrf}(P))$ [7, 8]
non-crossing perfect matchings	$O(n^{3/2} \cdot \mathsf{tri}(P) + n^{5/2}\mathsf{pm}(P))$	—

which enumeration algorithms were not known to the best of our knowledge, such as *non-crossing matchings*, *non-crossing red-and-blue matchings*, *non-crossing k-vertex or k-edge connected graphs*, or *non-crossing directed geometric graphs*. In Table 1 we list the time complexities of (a part of) new algorithms obtained in this paper, where we use the following notations to denote the numbers of graphs on a given point set P; $\mathsf{pg}(P)$: plane straight-line graphs, $\mathsf{cg}(P)$: non-crossing spanning connected graphs, $\mathsf{st}(P)$: non-crossing spanning trees, $\mathsf{mrf}(P)$: non-crossing minimally rigid frameworks, $\mathsf{tri}(P)$: triangulations and $\mathsf{pm}(P)$: non-crossing perfect matchings.

The key idea is to use triangulations because every non-crossing geometric graph is a subgraph of some triangulation. Let us consider enumerating all non-crossing spanning trees for example. Enumerating all non-crossing spanning trees in a triangulation is easily done by applying the existing algorithm such as [18, 31] for enumerating spanning trees in a given (abstract) graph since every subgraph of a triangulation is non-crossing. Moreover, efficient enumeration algorithms for triangulations are already known [6, 11]. Therefore, by enumerating spanning trees in every triangulation, we will obtain all non-crossing spanning trees. However, some non-crossing spanning tree might be produced more than once since it could be a subgraph of more than one triangulation. In order to avoid duplicate generation, we will do as follows.

A geometric graph containing a specified set of segments F is called *F-constrained*. We first apply the algorithm for enumerating all triangulations [6, 11]. Then each triangulation T generated by the algorithm can be viewed as the F-constrained lexicographically largest triangulation for some $F \subset T$ (that is a triangulation of the lexicographically largest edge list among all F-constrained triangulations). Our algorithm determines such F so that F is a minimal set that produces T as F-constrained lexicographically largest triangulation and enumerates only spanning trees that are containing F as their subsets and contained in T. We will show that this slightly modified algorithm correctly enumerate all non-crossing spanning trees without repetitions.

The overall idea of our techniques will be described in two algorithms, Algorithm 1 and Algorithm 2, in Sections 3 and 4, respectively. Let $\mathsf{ngg}(P)$ be the total number of graphs of $\mathcal{NGG}$ to be enumerated. Then, Algorithm 1 enumerates all the non-crossing geometric graphs belonging to $\mathcal{NGG}$ without repetitions in $O(f(n) \cdot \mathsf{tri}(P) + g(n) \cdot \mathsf{ngg}(P))$ time, where $f(\cdot)$ and $g(\cdot)$ are polynomial functions. By applying Algorithm 1, we obtain algorithms for enumerating plane straight-line graphs, non-crossing spanning connected graphs, non-crossing spanning trees and non-crossing perfect matchings (see Table 1). We remark that, for plane straight-line graphs or non-crossing spanning connected graphs, since we could show that $\mathsf{pg}(P)$ and $\mathsf{cg}(P)$ are exponentially larger than $\mathsf{tri}(P)$ for *every* point set P, the term of $f(n) \cdot \mathsf{tri}(P)$ is dominated by $\mathsf{pg}(P)$ or $\mathsf{cg}(P)$. Therefore, we can say that Algorithm 1 enumerates all plane straight-line graphs in $O(\mathsf{pg}(P))$ or all non-crossing spanning connected graphs in $O(\mathsf{cg}(P))$ time (details are discussed in Section 3.3). These results improve the running time of the previous best ones by Aichholzer et al. [2]. For the set of non-crossing spanning trees, we strongly believe that our algorithm also enumerates all non-crossing spanning trees in $O(\mathsf{st}(\mathsf{P}))$ time.

Although Algorithm 1 enumerates all graphs of $\mathcal{NGG}$ efficiently in terms of $\mathsf{tri}(P)$ and $\mathsf{ngg}(P)$, its time complexity cannot be bounded by $O(f(n) \cdot \mathsf{ngg}(P))$ in general. In fact, its complexity is dominated by $\mathsf{tri}(P)$ when $\mathsf{tri}(P)$ is exponentially larger than $\mathsf{ngg}(P)$. The next proposed algorithm Algorithm 2 overcomes this drawback by enumerating only the triangulations satisfying some specified property, and the number of triangulations to be enumerated is reduced appropriately. By

applying Algorithm 2, we obtain an enumeration algorithm that works in $O(n^2 \cdot \mathsf{mrf}(P))$ time for non-crossing minimally rigid frameworks. This result improves the previous one by Avis et al. [7] by an $O(n)$ factor.

Enumerating combinatorial objects is a fundamental problem, and several algorithms have been developed for non-crossing geometric graphs, e.g. triangulations [6, 11], non-crossing spanning trees [2, 6, 20], pseudo-triangulations [9, 12] and non-crossing minimally rigid frameworks [7, 8]. However, all the previous algorithms make use of an property of each graph class, and there exists no general framework for enumerating non-crossing geometric graphs efficiently.

Two objects of $\mathcal{NGG}$ are *connected* if they can be transformed to each other by a *transformation*, which generates one graph from the other by a certain specified operation. In particular, it is sometimes called *(k-)flip* if they have all but k edges in common. Define a graph $\mathcal{G}_{\mathcal{NGG}}$ on $\mathcal{NGG}$ with a set of edges connecting between objects that can be transformed to each other by one transformation. Then the natural question is how we can design the transformation so that $\mathcal{G}_{\mathcal{NGG}}$ is connected. Moreover, from the viewpoint of the applications to enumeration problems, the transformation should be defined *locally*, i.e., the symmetric difference between two connected objects should be as small as possible. The design of the transformation for a set of non-crossing graphs might be interesting in its own right, and there are many known results not only for the *local transformations* [2, 5, 6, 20, 22, 23, 25] but also for the *large transformations* [1, 3, 24]. In fact, almost all previous works for the enumeration discussed above are based on the local transformations. On the other hand, the proposed technique in this paper reveals that efficient enumeration of $\mathcal{NGG}$ is possible not relying on the local transformation of $\mathcal{NGG}$ directly but with only the well investigated transformation for a set of triangulations. Our enumeration technique does not rely on the property of each graph class deeply.

Let us explain why an enumeration of non-crossing geometric graphs is more difficult than those of non-geometric (abstract) graphs. A *binary partition* (or branch-and-bound technique) is a well known framework for designing enumeration algorithms. Consider, for example, the problem for enumerating all spanning trees in a graph G. Then, according to the framework of the binary partition, we can easily design an algorithm that enumerates all the spanning trees in $O(n^3)$ time per output graph as follows. The algorithm repeatedly divides the problem into two subproblems: one enumerates the spanning trees containing an edge e of G, and the other enumerates those not containing e. In the first subproblem e is contracted (and the resulting loop is removed if there exists any), while in the second subproblem e is removed. Then, the problem size is surely reduced in each subproblem. Moreover, since it can be checked in $O(n)$ time whether the resulting graph contains at least one spanning tree, the algorithm can decide correctly whether it should continue the search or not. Therefore, by going down this branch-and-bound tree in $O(n^2)$ steps, the algorithm surely detects a new spanning tree.

A binary partition could provide us with polynomial time delay enumeration algorithms for many graph classes because it just needs the polynomial time oracle that checks whether a given graph contains at least one subgraph belonging to a specified graph class or not. However, the problem of detecting a non-crossing subgraph satisfying a certain property in a given geometrically embedded graph is known to be NP-hard for most graph classes (even in the case of non-crossing spanning trees or non-crossing perfect matchings [26]). For this reason, most of the enumeration problems for non-crossing geometric graphs become non-trivial, and we need to introduce some new technique. In fact all previous works are not based on a binary partition but on the sophisticated local transformations. On the other hand, the techniques proposed in this paper are potentially capable of reducing a problem to that of an abstract graph. Once the problem is reduced to the non-geometric one, it becomes rather easier as discussed above and the efficient algorithms for enumerating abstract graphs could be applied.

2 Edge Constrained Lexicographically Largest Triangulation

Recall that a geometric graph containing a set of non-crossing segments F as its subset is called *F-constrained*. In this section, we will first introduce some notations used throughout the paper, and then provide a number of preliminary results on the F-constrained lexicographically largest triangulation (F-CLLT) which plays a crucial role in the development of our framework. Although most of the results presented in this section are already described in [21], we will provide the proof of them in order to make the paper self-contained and share the unified pictures of the F-CLLT.

2.1 Notations

Let P be a set of n points in $\mathbf{R}^2$, and for simplicity we label the points $P = \{p_1, \ldots, p_n\}$ in the increasing order of x-coordinates. We assume that the x-coordinates of all points are distinct and that no three points of P are collinear. For two vertices $p_i, p_j \in P$, we use the notation $p_i < p_j$ if $i < j$ holds, and $p_i = p_j$ if they coincide. Considering $p_i \in P$, we often pay attention only to the point set to its right, $\{p_{i+1}, \ldots, p_n\} \subseteq P$, which is denoted by P_{i+1}.

Let K_n be the complete graph embedded on P (with straight line segments). A line segment between p_i and p_j with $p_i < p_j$ is called an *edge*, denoted by (p_i, p_j). We often consider a geometric graph G as an *edge set*, and use the notation G to denote the edge set of G for simplicity when it is clear from the context.

For three points p_i, p_j and p_k the signed area $\Delta(p_i, p_j, p_k)$ of a triangle $p_ip_jp_k$ tells us that p_k is on the left or right side of a line passing through p_i and p_j when moving along the line from p_i to p_j by $\Delta(p_i, p_j, p_k) > 0$ or $\Delta(p_i, p_j, p_k) < 0$, respectively. We define a (total) ordering $\prec$ on a set of edges as follows: for $e = (p_i, p_j)$ and $e' = (p_k, p_l)$, $e \prec e'$ holds if $p_i < p_k$, or $p_i = p_k$ and $\Delta(p_i, p_j, p_l) < 0$. Let $E = \{e_1 \prec \cdots \prec e_m\}$ and $E' = \{e'_1 \prec \cdots \prec e'_m\}$ be sorted edge lists in increasing ordering. Then, E' is *lexicographically larger* than E if $e_i \prec e'_i$ for the smallest i such that $e_i \neq e'_i$.

We say that two edge (p_i, p_j) and (p_k, p_l) *properly intersect* if (p_i, p_j) and (p_k, p_l) have a point in common except for their endpoints. For two points $p_i, p_j \in P$, p_j is *visible from p_i with respect to a given non-crossing edge set F* when the edge (p_i, p_j) and no edge of F properly intersect. We assume that p_j is *visible* from p_i if $(p_i, p_j) \in F$.

Upper and *lower tangents*, (p_i, p_i^{up}) and $(p_i, p_i^{\mathrm{low}})$, *of p_i with respect to F* are defined as edges from p_i to the convex hull of the points of P_{i+1} that are visible from p_i with respect to F (see Fig. 1). Notice that each of the upper and lower tangents defines an *empty region* in which no point of P exists as described below. Let l be a line perpendicular to the x-axis passing through p_i, and let e_1 and e_2 be the edges of F first encountered when walking from p_i along the line l upwards and downwards, respectively (if such edges exist). Then, there exists no point of P inside the region bounded by l, e_1 (resp. e_2) and the line passing through p_i and p_i^{up} (resp. p_i^{low}). When e_1 (resp. e_2) does not exist, the empty region is defined by the one bounded by l and the line through p_i and p_i^{up} (resp. p_i^{low}). Thus, we have the following fact:

Observation 2.1. *Let P be a point set in the plane and F be a non-crossing edge set on P. Let e be an edge of K_n that properly intersects either upper or lower tangent of p_i with respect to F. Then, at least one of the following two facts holds:* (1) *the left endpoint of e is less than p_i and* (2) *e properly intersects some edge of F.*

2.2 Edge-constrained Lexicographically Largest Triangulations

For a non-crossing edge set F on P and a point $p_i \in P$, let us denote by $\delta_F(p_i)$ a set of edges of F whose left endpoints are p_i. Let us consider the following construction of an F-constrained geometric graph on P:

Construction 1.

0. *Repeat the following process for all $p_i \in P$ in an arbitrary order.*

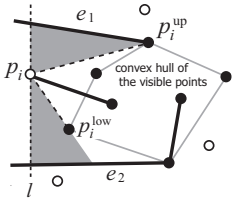


Figure 1: An example of the upper and lower tangents, denoted by (p_i, p_i^{up}) and $(p_i, p_i^{\mathrm{low}})$, respectively. Bold edges represent F. The empty regions of (p_i, p_i^{up}) and $(p_i, p_i^{\mathrm{low}})$ are shaded.

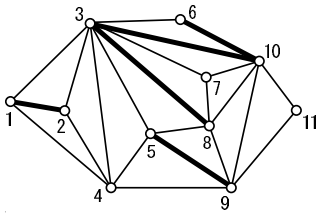


Figure 2: CLLT.

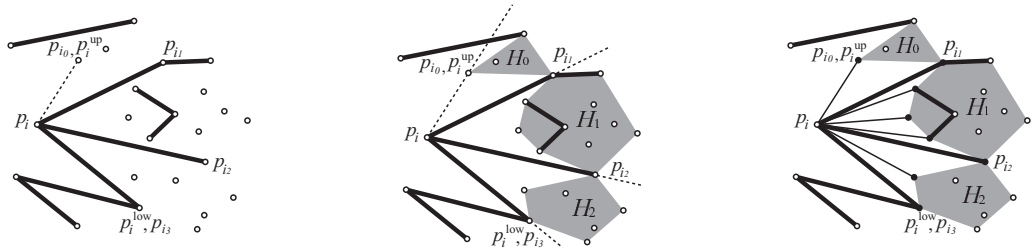


Figure 3: Construction 1 around p_i where bold edges represent F. (a)Step 1, (b)Step 2 and (c)Step 3.

1. *Let (p_i, p_i^{up}) and $(p_i, p_i^{\mathrm{low}})$ be the upper and lower tangents of $p_i \in P$ with respect to F, and denote the right endpoints of $\delta_F(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$ by $p_{i_0}, p_{i_1}, \ldots, p_{i_m}$ arranged in clockwise order around p_i, (where $p_{i_0} = p_i^{\mathrm{up}}$ and $p_{i_m} = p_i^{\mathrm{low}}$ hold) (Fig.3(a)).*
2. *Consider the cone C_k with an apex at p_i bounded by two consecutive edges (p_i, p_{i_k}) and $(p_i, p_{i_{k+1}})$ for each k with $0 \leq k \leq m-1$, where C_k contains both p_{i_k} and $p_{i_{k+1}}$, and construct the convex hull H_k of $P_{i+1} \cap C_k$ inside each C_k (Fig.3(b)).*
3. *Draw an edge from p_i to every point $p_j \in P_{i+1} \cap C_k$ such that $p_j = (p_i, p_j) \cap H_k$ for each k (Fig.3(c)).*

As will be proved in Lemma 2.3, the above algorithm never produce edge crossings, but in fact produces a triangulation. We give an example of the graph obtained by the above construction in Fig. 2. Notice that the graph obtained by Construction 1 always has the edges of $\delta_F(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$ for all $p_i \in P$. The following property of the graph obtained by Construction 1 could be easily observed.

Lemma 2.2. *Let G be the graph obtained by Construction 1 and let (p_i, p_j) be an edge of G. Then, any edge of K_n properly intersecting (p_i, p_j) properly intersects at least one edge of $\delta_F(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$.*

Proof. Let us consider Construction 1 around p_i. Then, since $(p_i, p_j) \in G$, there exists a convex hull H_k for which $p_j = (p_i, p_j) \cap H_k$. Notice that the two consecutive edges, (p_i, p_{i_k}) and $(p_i, p_{i_{k+1}})$ of $\delta_F(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$ bounding C_k in Step 2 of Construction 1, and a part of the boundary of H_k from p_{i_k} to $p_{i_{k+1}}$ (that is a convex chain) forms a simple polygon with exactly three convex vertices, p_i, p_{i_k} and $p_{i_{k+1}}$, which is so-called a pseudo-triangle. Since p_j is a vertex of such a pseudo-triangle and there exists no point of P inside of the pseudo-triangle, any edge intersecting (p_i, p_j) must intersect at least one of (p_i, p_{i_k}) and $(p_i, p_{i_{k+1}})$. □

The following lemmas describe the fundamental properties of the above defined construction.

Lemma 2.3. *The graph G obtained by Construction 1 is an F-constrained triangulation on P.*

Proof. We will prove, by induction on i from $i = n$ to 1, that (1) the subgraph of G induced by P_i, denoted by G_i, is non-crossing, and (2) all faces of G_i are triangles except for an outer face f_{out}. This implies that G is a triangulation since G clearly contains the boundary edges of the convex hull of P from the definition of Construction 1.

For the basis, G_n has no edge, and hence the statement holds. Assume that (1) and (2) hold for G_{i+1}. We first show that (1) holds for G_i. Suppose there exists an edge $(p_a, p_b) \in G_{i+1}$ with $p_a < p_b$ that properly intersects some edge of $G_i \setminus G_{i+1}$. Then, from Lemma 2.2, (p_a, p_b) properly intersects some edge of $\delta_F(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$. By Construction 1 it is obvious that (p_a, p_b) does not properly intersect any edge of F. Hence (p_a, p_b) intersects either (p_i, p_i^{up}) or $(p_i, p_i^{\mathrm{low}})$. However, this implies, by Observation 2.1, that p_a lies on the left side of p_i, which contradicts $p_a \in P_{i+1}$. Hence (1) holds for G_i.

Next, we prove (2). Let (p_i, p_a) and (p_i, p_b) be two consecutive edges of $G_i \setminus G_{i+1}$ in clockwise order around p_i. We will show that there is an edge between p_a and p_b in G_{i+1}. From the definition of Construction 1, there exists a convex hull H_k such that p_a and p_b are consecutive vertices on the boundary of H_k. Hence, an edge between p_a and p_b is one of a upper or lower tangents of p_a or p_b, and so it is contained in G_{i+1} by Construction 1. □

Lemma 2.4. *The F-constrained triangulation $T^*(F)$ obtained by Construction 1 has the lexicographically largest edge list among all F-constrained triangulations on P.*

Proof. Let $T^*(F)$ be the F-constrained triangulation obtained by Construction 1 with edge list $\{e_1^* \prec \cdots \prec e_m^*\}$. Suppose there exists an F-constrained triangulation T whose edge list $\{e_1 \prec \cdots \prec e_m\}$ is lexicographically larger than that of $T^*(F)$. Then, there exists the smallest subscript s with $e_s^* \neq e_s$ for which $e_s^* \notin T$ and $e_s^* \prec e_s$ hold.

Let $e_s^* = (p_i, p_j)$. Since s is the smallest subscript of edges for which $e_s^* \neq e_s$, a set of edges of $T^*(F)$ incident to p coincides with that of T for every $p \in \{p_1, \ldots, p_{i-1}\}$. Since T is a triangulation but does not contain e_s^*, T must contain at least one edge $e \notin T^*(F)$ that properly intersects e_s^*. By Lemma 2.2, e properly intersects some edge of $\delta_F(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$. In addition, since T is an F-constrained triangulation, e does not properly intersect any edge of $\delta_F(p_i)$, and hence e properly intersects at least (p_i, p_i^{up}) or $(p_i, p_i^{\mathrm{low}})$. However, by Observation 2.1, we found that the left endpoint of e is on the left side of p_i, which contradicts that the set of edges of $T^*(F)$ and T incident to $p \in \{p_1, \ldots, p_{i-1}\}$ coincide. □

Hence, we call the F-constrained triangulation obtained by the above construction *F-constrained lexicographically largest triangulation* (F-CLLT). In fact we can show that F-CLLT can be constructed by greedily adding the edges into F in the lexicographically descending order without violating the non-crossing property.

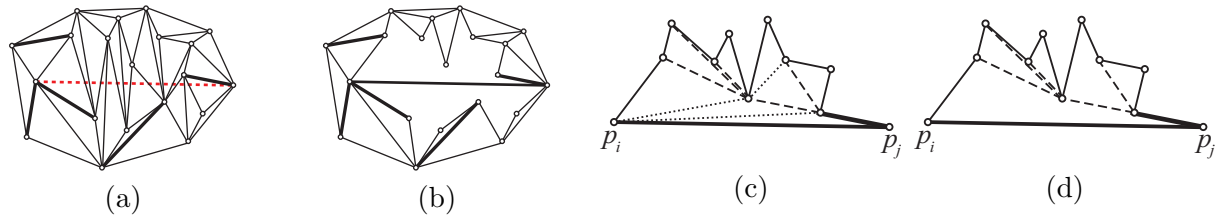
An edge e in a triangulation T is called *flippable* when two triangles incident to e in T form a *convex* quadrilateral Q. *Flipping* e in T generates a new triangulation by replacing e with the other diagonal of Q. It is known that every F-constrained triangulation can be transformed into F-CLLT by flipping $O(n^2)$ edges $e \notin F$, each of which increases the lexicographical order of the edge list [21].

2.3 Maintaining F-constrained Lexicographically Largest Triangulation

Let us discuss how to maintain the F-CLLT, denoted by $T^*(F)$, when we newly insert one constraint edge e into F. Developing the following efficient way to construct $T^*(F \cup \{e\})$ from $T^*(F)$ will be helpful for constructing the fast enumeration algorithm discussed in Section 4.1.

Lemma 2.5. *Let $T^*(F)$ be the F-CLLT on a given set of n points, and let e be an edge that does not properly intersect any edge of F. Then, it takes $O(n)$ time to construct $T^*(F \cup \{e\})$ from $T^*(F)$.*

Proof. Let $e = (p_i, p_j)$, and let I be the set of edges of $T^*(F)$ that intersect e. First we show that every edge of $T^*(F) \setminus I$, say $(p_k, p_l) \in T^*(F) \setminus I$, is still contained in $T^*(F \cup \{e\})$. Consider how $T^*(F)$ is constructed by performing Construction 1 around p_k. Let (p_k, p_k^{up}) and $(p_k, p_k^{\mathrm{low}})$ be the



(a) (b) (c) (d)

Figure 4: (a)Insertion of a new constrained edge, (b)two empty simple polygons obtained by removing the edges intersecting the inserted edge, (c)reconstruction inside the simple polygon in the upper side, where dashed and dotted edges represent the green and yellow edges, and (d)the green dashed edges.

upper and lower tangents of p_k with respect to F. Then, $(p_k, p_k^{\mathrm{up}}) \preceq e \preceq (p_k, p_k^{\mathrm{low}})$ holds, since otherwise e properly intersects some edge of F from the definition of the upper and lower tangents, which contradicts the assumption. Therefore, there exists a cone with an apex p_k considered in Step 2 of Construction 1, which contains p_l. Let H_F denote the convex hull (containing p_l) inside this cone. Similarly let H_{F+e} be the one containing p_l inside the cone with an apex p_k considered in Step 2 when constructing $T^*(F \cup \{e\})$. When inserting e, the vertices that are not visible from p_k with respect to F remain non-visible from p_k with respect to $F \cup \{e\}$ although some of the vertices visible from p_k with respect to F may become non-visible from p_k with respect to $F \cup \{e\}$. This implies $H_{F+e} \subseteq H_F$. Moreover, from $(p_k, p_l) \in T^*(F)$, $p_l = (p_k, p_l) \cap H_F$ holds now. Hence, we obtain $p_l = (p_k, p_l) \cap H_{F+e}$, and (p_k, p_l) remains in $T^*(F \cup \{e\})$.

Therefore, the update occurs only inside the two simple polygons obtained by removing the edges of I and adding e, (see Fig. 4(a)(b)). Without loss of generality, we assume that e is horizontal, and let us show an efficient algorithm to triangulate (the inside of) the polygon lying on the upper side of $e = (p_i, p_j)$ (the lower side can be treated similarly). Consider the updated triangulation of the polygon by Construction 1. There exist two types of edges: (1) lower tangent of each vertex of the polygon with respect to the boundary edges of the polygon, and (2) the others (see Fig. 4(c)). We call the type (1) and type (2) edges green (dashed) and yellow (dotted) edges, respectively.

Let us consider how to find the green edges. Let v be a vertex of the polygon which misses the lower tangent in $T^*(F) \setminus I \cup \{e\}$, i.e., the lower tangent (v, v^{low}) of v with respect to F proper intersects e (and hence (v, v^{low}) is removed). Consider a ray emanating from v to v^{low} (which first hits e before reaching v^{low}). By rotating the ray around v in counterclockwise order until it encounters a vertex of the polygon, we can find the new lower tangent $(v, \tilde{v}^{\mathrm{low}})$ of v, which is a green edge. Note that a ray emanating from v to $\tilde{v}^{\mathrm{low}}$ first hits e among the boundary edges of the polygon (except for hitting the vertex $\tilde{v}^{\mathrm{low}}$), and hence $\tilde{v}^{\mathrm{low}}$ is again a vertex missing the lower tangent in $T^*(F) \setminus I \cup \{e\}$, or coincides with p_j. This implies that the set of all green edges is a subset of the convex chain connecting between p_j and each vertex of the polygon as shown in Fig. 4(d). Besides, these convex chains indicate the shortest paths inside the polygon from p_j to the vertices of the polygon. It is known [17] that the shortest paths from a single source to all vertices inside simple polygon can be computed in $O(n)$ time although it requires an involved linear time algorithm for triangulation a simple polygon [13]. Thus, we could obtain the desired time complexity through the shortest path algorithm.

Our problem, however, can be solved easily by performing Graham scan (see e.g. [14]) only once. Let us try to construct the lower part of the convex hull of the vertices of the polygon by performing Graham scan algorithm from p_j to p_i. We remark that the algorithm scans all vertices not in the order of the coordinates as usual but in the vertex sequence order of the polygon from p_j to p_i. Then, the process of Graham scan will trace the green edges. When we encounter the new vertex p during scan, we examine the top two vertices q and r on the stack. If the angle of three points around q inside the polygon is convex, then we pop q and draw the yellow edge between p and q (if (p, q) is not the boundary edge of the polygon). Continue this process until we obtain three vertices p, q' and r' whose angle around q' inside the polygon is reflex. Then, we draw a green edge between p and q'

(if (p, q) is not the boundary edge of the polygon). Thus, repeating this process until $p = p_i$ and the stack contains only p_i and p_j, we can draw all of the green and yellow edges in linear time. □

3 Enumerating Non-crossing Geometric Graphs

3.1 General Idea

Let $\mathcal{F}$ be the collection of all non-crossing edge sets on a given point set P, and let $\mathcal{T}$ be the set of all triangulations on P. We will often treat a triangulation as an *edge set* in the subsequent discussion. We make use of the construction of the F-CLLT defined in the previous section as a function $T^* : \mathcal{F} \to \mathcal{T}$ that maps a non-crossing edge set F to the corresponding F-CLLT $T^*(F)$. Then, we define an equivalence relation $\sim$ on $\mathcal{F}$ such that, for two non-crossing edge sets F and F', $F \sim F'$ holds if and only if $T^*(F) = T^*(F')$ holds. The equivalence classes over F, denoted by $[T] = \{F \in \mathcal{F} \mid F \sim T\}$ for all $T \in \mathcal{T}$, are defined accordingly. Note that the set of all equivalence classes forms a partition of $\mathcal{F}$.

We have the following properties of the function $T^*(\cdot)$ which will be used to define a nice representative of each equivalence class.

Lemma 3.1. *Let $F \in \mathcal{F}$. Then, for every $e \in T^*(F)$, $T^*(F \cup \{e\}) = T^*(F)$ holds.*

Proof. Notice that the set of all $(F \cup \{e\})$-constrained triangulations is a subset of all F-constrained triangulations. Moreover, from $e \in T^*(F)$, $T^*(F)$ can be regarded as an $(F \cup \{e\})$-constrained triangulation as well as an F-constrained triangulation. Since $T^*(F)$ has the lexicographically largest edge list among all of the F-constrained triangulations from Lemma 2.4, $T^*(F)$ also has the lexicographically largest edge list among all of the $(F \cup \{e\})$-constrained triangulations, which implies $T^*(F \cup \{e\}) = T^*(F)$. □

Lemma 3.2. *Let $F \in \mathcal{F}$. Then, for $e = (p_i, p_j) \in F$, $T^*(F \setminus \{e\}) = T^*(F)$ holds if either (i) e is either upper or lower tangent of p_i with respect to F, or (ii) e is non-flippable in $T^*(F)$.*

Proof. First let us consider the case when $e = (p_i, p_j)$ is either upper or lower tangent of p_i with respect to F. Then, e is also either upper or lower tangent of p_i with respect to $F \setminus \{e\}$ since removing $e = (p_i, p_j)$ does not affect the visibility of p_i. Since CLLT contains upper and lower tangents for every $p \in P$ by the definition of Construction 1, it follows that $e \in T^*(F \setminus \{e\})$, which implies $T^*(F \setminus \{e\}) = T^*(F)$ from Lemma 3.1.

Next let us consider the case when e is non-flippable in $T^*(F)$. Suppose e is either upper or lower tangent of p_i with respect to F. Then the statement follows from (i). So let us assume that e is neither upper nor lower tangent. We show that e is still contained in $T^*(F \setminus \{e\})$.

Following Construction 1 for $T^*(F)$, let us denote a set of the right endpoints of $\delta_F(p_i) \cup \{(p_i, p_i^{\text{up}}), (p_i, p_i^{\text{low}})\}$ by $p_{i_0}, p_{i_1}, \ldots, p_{i_m}$ in clockwise ordering around p_i, where (p_i, p_i^{up}) and (p_i, p_i^{low}) are the upper and lower tangent of p_i with respect to F, and consider m convex hulls H_k of $P_{i+1} \cap C_k$, for $k = 0, \ldots, m-1$, bounded by the consecutive edges, (p_i, p_{i_k}) and $(p_i, p_{i_{k+1}})$, and m convex chains as the boundary of the convex hulls H_k which consist of the sequence of the vertices p satisfying $p = (p_i, p) \cap H_k$.

Since e is in F (and more precisely $e \in \delta_F(p_i)$) and e is neither upper nor lower tangent, there exists a subscript k' with $k' \neq 0, m$ for which $e = (p_i, p_{i_{k'}})$ holds. Therefore, since e is non-flippable in $T^*(F)$, by combining two convex chains, one from $p_{i_{k'-1}}$ to $p_{i_{k'}}$ and the other from $p_{i_{k'}}$ to $p_{i_{k'+1}}$, we obtain a single convex chain from $p_{i_{k'-1}}$ to $p_{i_{k'+1}}$. This implies that we obtain the convex hull H of the point set P_{i+1} inside the cone bounded by two consecutive edges $(p_i, p_{i_{k'-1}})$ and $(p_i, p_{i_{k'+1}})$ of $\delta_F(p_i) \setminus \{e\} \cup \{(p_i, p_i^{\text{up}}), (p_i, p_i^{\text{low}})\}$, (implying that H will be constructed in Construction 1 for $T^*(F \setminus \{e\})$), in which $p_{i_{k'}} = (p_i, p_{i_{k'}}) \cap H$ holds. Therefore, the edge $e = (p_i, p_{i_{k'}})$ is chosen as the edge of $T^*(F \setminus \{e\})$ in Step 3 of Construction 1. □

Lemma 3.3. *Let $F \in \mathcal{F}$. Then, for $E \subseteq F$, $T^*(F \setminus E) \neq T^*(F)$ holds if there exists an edge $e = (p_i, p_j) \in E$ that is (i) flippable in $T^*(F)$ and (ii) neither upper nor lower tangent of p_i with respect to F.*

Proof. Suppose $T^*(F \setminus E) = T^*(F)$ holds. Note that $T^*(F \setminus E) = T^*(F)$ implies that $T^*(F)$ is the $(F \setminus E)$-constrained lexicographically largest triangulation as well as the F-constrained lexicographically largest triangulation.

Consider two triangles of $T^*(F \setminus E)$ incident to $e = (p_i, p_j)$, and denote the two vertices for these triangles other than p_i and p_j by v and w. Since e is flippable in $T^*(F \setminus E)(= T^*(F))$, the quadrilateral $p_i v p_j w$ is convex. In addition, since e is neither upper nor lower tangent of p_i, both v and w lie on the right side of p_i, and hence $e \prec (v, w)$ holds. Therefore, flipping e to (v, w) produces an $(F \setminus E)$-constrained triangulation that is lexicographically larger than $T^*(F)$ from $e \prec (v, w)$, which contradicts that $T^*(F)$ is the $(F \setminus E)$-constrained lexicographically largest triangulation. □

We remark that, for any non-crossing edge set F, the upper and lower tangents of p_i with respect to F are the smallest and largest ones of $\{(p_i, q) \in T^*(F) \mid q \in \{p_{i+1}, \ldots, p_n\} = P_{i+1}\}$, respectively, from the definition of Construction 1. This implies that, for any $F \in [T]$ of a triangulation T, the upper and lower tangents with respect to F are equivalent to the smallest and largest ones of $\{(p_i, q) \in T \mid q \in P_{i+1}\}$. Using Lemmas 3.2 and 3.3, a unique minimal representative for each $[T]$ is defined as follows.

Lemma 3.4. *For a triangulation T, let F^* be the set of all flippable edges in T except for the smallest and largest edges of $\{(p_i, q) \in T \mid q \in P_{i+1}\}$ for every $p_i \in P$. Then,*

(i) $F^* \in [T]$ (i.e., $T^*(F^*) = T$), and

(ii) for any $F \in \mathcal{F}$, $F \in [T]$ holds if and only if $F^* \subseteq F \subseteq T$ holds.

Proof. Let us show (i). It is obvious that $T^*(T) = T$ holds. Note that, from the definition of F^*, every edge $e = (p_i, p_j) \in T \setminus F^*$ is either (i) non-flippable in T, or (ii) the smallest or largest edge among $\{(p_i, q) \in T \mid q \in P_{i+1}\}$ (implying that e is the upper or lower tangent of p_i with respect to T). Hence, from Lemma 3.2, removing any edge $e \in T \setminus F^*$ does not change the triangulation, i.e., $T = T^*(T) = T^*(T \setminus \{e\})$ holds. We thus eventually obtain $T = T^*(T) = T^*(T \setminus (T \setminus F^*)) = T^*(F^*)$.

Next let us show (ii). The "if-part" can be proved in the same way as in the first part. In fact, removing the edges of $F \setminus F^*$ one by one from the constraint edge set of $T^*(F)$, we obtain $T^*(F) = T^*(F \setminus (F \setminus F^*)) = T^*(F^*) = T$. Let us consider the "only-if" part. It is obvious that $F \subseteq T$ holds if $F \in [T]$. Suppose F (with $F \subseteq T$) is a counterexample, that is $T^*(F) = T$ but $F^* \setminus F \neq \emptyset$. If $F \setminus F^* \neq \emptyset$ holds, the removal of any edge of $F \setminus F^*$ from F does not change the triangulation for the same reason as in the first part, and hence we may assume that $F \setminus F^* = \emptyset$ holds, i.e. $F \subsetneq F^*$. However, since an edge $e = (p_i, p_j) \in F^* \setminus F$ is flippable in T and neither the smallest nor largest edge of $\{(p_i, q) \in T \mid q \in P_{i+1}\}$ from the definition of F^*, $T = T^*(F) = T^*(F^* \setminus (F^* \setminus F)) \neq T^*(F^*)$ holds from Lemma 3.3, which contradicts $T = T^*(F^*)$. □

Thus, we call F^* defined in Lemma 3.4 the *minimal representative set* of T, denoted by $R(T)$. Our enumeration algorithm can be easily described, which consists of two phases as follows.

Algorithm 1: Enumeration of $\mathcal{NGG}$.

Phase1: Enumerate all triangulations for a given point set P based on the fast enumeration algorithm by Bespamyatnikh [11].

Phase2: Every time a new triangulation T is found, enumerate all graphs G contained in T such that $G \in \mathcal{NGG}$ and G contains the minimal representative set $R(T)$ as its subset, i.e., G is an $R(T)$-constrained graph in T.

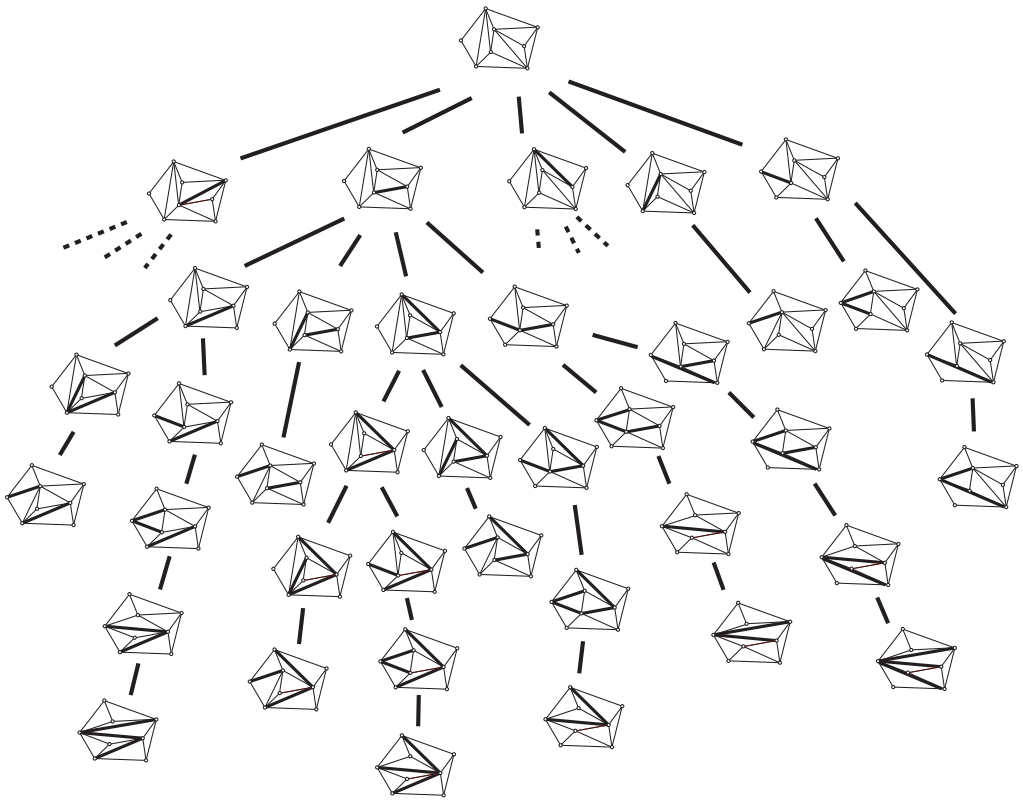


Figure 5: Search tree on the set of triangulations obtained by the algorithm by Bespamyatnikh, where each minimal representative set is drawn in bold.

Let $\mathcal{C}$ be a graph class obtained by relaxing the non-crossing constraints from a non-crossing geometric graph class $\mathcal{NGG}$ (i.e. the set of geometric graphs whose edge sets are not necessarily non-crossing but satisfy the other combinatorial property of $\mathcal{NGG}$). Notice that in Phase 2 the problem for enumerating all graphs of $\mathcal{NGG}$ is reduced to that of enumerating all elements of $\mathcal{C}$ containing $R(T)$ in a triangulation T because T is non-crossing. Then, in Phase 2, we may utilize an oracle for enumerating all the graphs of $\mathcal{C}$ containing a specified edge set F in a given (abstract) graph without repetitions. Namely, we can ignore "geometric" and "non-crossing".

The algorithm needs $R(T)$ explicitly for every T in Phase 2, and hence it will be better to maintain and update $R(T)$ during the enumeration of triangulations rather than compute it from scratch because the algorithm by Bespamyatnikh [11] creates a new triangulation by flipping one edge, and hence two triangulations generated in succession only differ in one edge. So the task of Phase 1 is in fact not only the enumeration of T but also the generation of $R(T)$. This additional task can be performed by slightly modifying the triangulation enumeration which will be discussed more formally in Section 3.2. We show an example of the enumeration of triangulations and the minimal representative sets in Fig. 5.

Theorem 3.5. *Algorithm 1 enumerates all graphs of $\mathcal{NGG}$ without repetitions.*

Proof. We just need to verify that the algorithm enumerates all graphs of $\mathcal{NGG}$ belonging to $[T]$ but not graphs belonging to the other equivalence class in Phase 2 for a triangulation T.

From Lemma 3.4 we have $R(T) \subseteq G \subseteq T$ for every $G \in \mathcal{NGG}$ belonging to $[T]$. G is hence an $R(T)$-constrained graph contained in T, and must be enumerated by the (assumed) oracle. On the other hand, suppose G' is a graph belonging to an equivalence class other than $[T]$. If $G' \not\subset T$, clearly G' is not generated in Phase 2 for T. On the other hand, if $G' \subseteq T$, then G' does not contain $R(T)$ since otherwise $R(T) \subseteq G' \subseteq T$ holds, which implies $G' \in [T]$ from Lemma 3.4. This is a contradiction. □

3.2 Time Complexity of Algorithm 1

In order to analyze the time complexity of Algorithm 1, let us briefly review the enumeration algorithm of triangulations by Bespamyatnikh [11], which is based on the reverse search technique [6]. The reverse search is a well known technique to generate all the elements of the combinatorial objects by tracing the nodes in the *search graph*, in which each node corresponds to each object to be enumerated and each edge corresponds to a *transformation* (discussed in Section 1) between two objects. To trace the search graph efficiently, the algorithm defines a *root* node and a unique *parent* for each node except for the root such that the subgraph of the search graph induced by the parent-child relation forms a rooted spanning tree. Such a spanning tree is called *search tree*, and the algorithm traces it by depth-first manner. The search graph of the algorithm by Bespamyatnikh is defined in such a way that two triangulations are connected if and only if they can be transformed to each other by a diagonal flip (see Fig. 5 or [11] for more details). The following lemma states how to efficiently maintain the minimal representative set during enumeration of triangulations.

Lemma 3.6. *Let T_1 and T_2 be two triangulations for which T_2 is a child of T_1 in the search tree of the triangulation enumeration by Algorithm 1. Then the size of the symmetric difference between $R(T_1)$ and $R(T_2)$ is constant. More specifically, only the four edges of two triangle faces incident to a flipped edge are involved in the symmetric difference.*

Proof. Recall that the minimal representative set $R(T)$ of a triangulation T was defined in Lemma 3.4 to be the edge set of all flippable edges in T except for the smallest and largest edges of $(p_i, q) \in T \mid q \in P_{i+1}$. Suppose T_2 is obtained from T_1 by replacing an edge $e_1 \in T_1$ by an edge $e_2 \in T_2$ during the enumeration. There are two cases; (Case 1:) $e = (p_i, p_j) \in R(T_1)$ becomes non-flippable in T_2, and (Case 2:) $e = (p_i, p_j)$ becomes either the smallest or largest edge among $\{(p_i, q) \in T_2 \mid q \in P_{i+1}\}$. Notice that a diagonal flip switches at most four flippable edges in T_1 into non-flippable edges in T_2, and an edge of Case 1 is clearly one of the four edges of the triangles incident to the flipped edge. Let us consider Case 2 where $e \in R(T_1)$ becomes the smallest (or largest) one of $\{(p_i, q) \in T_2 \mid q \in P_{i+1}\}$. Since e is not the smallest (or largest) one in $\{(p_i, q) \in T_1 \mid q \in P_{i+1}\}$, there exists an edge $e' = (p_i, q')$ in T_1 with $e' \prec e$ (or $e \prec e'$, respectively) such that e' and e are incident to a common triangle face of T_1. We notice that e' disappears in T_2 because e becomes the smallest (or largest) one, and hence e' is exactly e_1. So, e_1 and e are incident to the same triangle face in T_1.

The rest of the argument for $e \in R(T_2) \setminus R(T_1)$ is similar from the locality of diagonal flips. □

By Lemma 3.6, the symmetric difference of the minimal representative sets can be output in $O(1)$ time if the triangulation is maintained in a proper data structure and a flag indicating whether in the minimal representative set or not is attached to each edge. The algorithm by Bespamyatnikh [11] enumerates all triangulations in $O(\log \log n)$ time per output. Thus, we obtain the following theorem:

Theorem 3.7. *Suppose there exists an algorithm for enumerating all elements of $\mathcal{C}$ in a given graph, each of which contains a prespecified edge set, without repetitions in time $t_{\mathcal{C}}$ per output graph with preprocessing time $t_{\mathcal{C},\mathrm{pre}}$. Then all elements of $\mathcal{NGG}$ on a given point set P of n points can be enumerated without repetitions in $O((\log \log n + t_{\mathcal{C},\mathrm{pre}}) \cdot \mathsf{tri}(P) + t_{\mathcal{C}} \cdot \mathsf{ngg}(P))$ time, where $\mathsf{tri}(P)$ and $\mathsf{ngg}(P)$ are the total number of triangulations and $\mathcal{NGG}$ on P, respectively.*

3.3 Applications of Algorithm 1

3.3.1 Enumerating Non-crossing Spanning Trees

We will show how to apply Algorithm 1 to the enumeration of non-crossing spanning trees on a given point set. What we have to show here is how to enumerate all the spanning trees in a given triangulation T, each of which contains the minimal representative set $R(T)$. We notice again that, in the above process, we do not have to care about whether an output spanning tree is non-crossing because T is non-crossing. In Phase 2 of Algorithm 1, we will use the algorithm for enumerating

spanning trees on a given undirected graph developed by Kapoor and Ramesh [18] or Shioura et al. [30, 31]. These algorithms can enumerate all the spanning trees of a given graph in $O(1)$ time per output graph[1] with $O(n+m)$ preprocessing time, where n and m denote the number of vertices and edges. Since the edge-constraints can be handled easily by contracting the specified edges before calling these oracles, all the $R(T)$-constrained spanning trees in T can be enumerated in $t_{\mathcal{C}} = O(1)$ time per output graph with $t_{\mathcal{C},\mathrm{pre}} = O(n)$ preprocessing time. Thus, from Theorem 3.7, we obtain the following result:

Theorem 3.8. *Let P be a set of n points in the plane. Then the set of non-crossing spanning trees on P can be enumerated in $O(n \cdot \mathsf{tri}(P) + \mathsf{st}(P))$ time.*

Remark. Provided that there exists a constant c (> 1) for which $c^n \cdot \mathsf{tri}(P) < \mathsf{st}(P)$ holds for *every* P of n points, the above running time is dominated by $\mathsf{st}(P)$. It is known that $\mathsf{st}(P)$ becomes minimum when P is in a convex position when n is fixed. On the other hand there exists a configuration that can admit smaller number of triangulations than that of convex position (see [4]). Furthermore, the number of $\mathsf{st}(P)$ in the convex position is known to be $\Theta(6.75^n)$ [15] relative to the number of triangulations that is $\Theta(4^n)$, where we ignore polynomial factors. Hence, we strongly conjecture that there exists such a constant c with $c > 1$.

3.3.2 Enumerating Non-crossing Spanning Connected Graphs

We show here how Algorithm 1 can be applied to the enumeration of non-crossing spanning connected graphs. All the spanning connected subgraphs of a given graph can be enumerated in $O(1)$ time per output by the binary partition technique with the sophisticated amortized analysis proposed by Uno [33] for enumerating all bases of matroids (see [33, 34] for more details) with $O(n)$ preprocessing time. Let us briefly explain how to apply this algorithm to the enumeration of spanning connected subgraphs.

Consider enumerating all spanning trees (bases of graphic matroid) in a given graph by the binary partition technique discribed in Introduction. The algorithm by Uno outputs each spanning tree when it reaches a leaf of the binary partition tree (branch-and-bound tree), keeping a spanning connected subgraphs during tracing this tree. Hence, by outputting graphs not only at leafs but also at some internal nodes, we can enumerate all spanning connected subgraphs without repetitions. (More precisely, we output a spanning connected graph when the corresponding node of the binary-partition tree is obtained by removing an edge (not contracting an edge) when partitioning the problem into two subproblems, see Introduction.) The time complexity is not increased since the additional task is only output of the (symmetric difference of) graph in the internal nodes.

Since the edge-constraints can be treated easily by edge contraction as in the case of the spanning trees discussed in Section 3.3.1, all the $R(T)$-constrained spanning connected graphs of T can be enumerated in $t_{\mathcal{C}} = O(1)$ time per output with $t_{\mathcal{C},\mathrm{pre}} = O(n)$. Thus, from Theorem 3.7, Algorithm 1 enumerates all non-crossing spanning connected graphs in $O(n \cdot \mathsf{tri}(P) + \mathsf{cg}(P))$ time. Moreover, we could show the next lemma.

Lemma 3.9. *For every point set P in the plane with n points, $2^{n/2-1} \cdot \mathsf{tri}(P) \leq \mathsf{cg}(P)$ holds.*

Proof. Let T be a triangulation on P with the minimal representative set $R(T)$. We show that, for every T, there exist at least $2^{n/2-1}$ non-crossing spanning connected subgraphs in T that are not contained in the other triangulations.

Let us first show that, for every (triangle) face $p_ip_jp_k$ of T, $|\{(p_i,p_j),(p_i,p_k),(p_j,p_k)\} \cap R(T)| \leq 2$ holds. Without loss of generality, assume that $p_i < p_j < p_k$. Then, the edge (p_j,p_k) is either the lexicographically largest or smallest edge among $\{(p_j,q) \in T \mid q \in P_{j+1}\}$. Hence, (p_j,p_k) is not contained in the minimal representative set $R(T)$ from the definition of the minimal representative set described in Lemma 3.4.

[1] The algorithm outputs each enumerated graph by the *compact form*, i.e., the symmetric difference between the last found object and the current one otherwise it takes $O(n)$ time to output each graph.

Thus consider a subset S of T such that (i) S forms a spanning connected graph on P, (ii) S contains $R(T)$ as its subset and (iii) S has the minimum edge cardinality among the subsets of T satisfying (i) and (ii). Then, from the above discussion, S contains at most two edges for each face of T. Since S is $R(T)$-constrained non-crossing spanning connected graph on P, $S \cup E$ forms a distinct $R(T)$-constrained non-crossing spanning connected graphs on P for every $E \subseteq T \setminus S$ with $E \neq \emptyset$. Since the number of faces of a triangulation is known to be $2n - h - 2$, where h is the number of vertices of the convex hull of P, $|T \setminus S|$ is at least $(2n - h - 2)/2 \geq n/2 - 1$. Therefore, there exist $\Omega(2^{n/2})$ subsets of $T \setminus S$, and T contains $\Omega(2^{n/2})$ non-crossing spanning connected graphs each of which contains $R(T)$ as its subset.

We remark that every two non-crossing spanning connected graphs G_1 and G_2 with $R(T_1) \subseteq G_1 \subseteq T_1$ and $R(T_2) \subseteq G_2 \subseteq T_2$ are distinct for every distinct triangulations T_1 and T_2. This is because that $R(T_1) \subseteq G_1 \subseteq T_1$ and $R(T_2) \subseteq G_2 \subseteq T_2$ imply $G_1 \in [T_1]$ and $G_2 \in [T_2]$ from Lemma 3.4. Since $[T_1] \cap [T_2] = \emptyset$ from the definition of the equivalence class $[T]$, $G_1 \neq G_2$ holds. Thus, every triangulation contains at least $2^{n/2-1}$ non-crossing spanning connected graphs on P that are not contained in the other triangulations. □

Lemma 3.9 is of independent interest because it shows that $\mathsf{cg}(P)$ is exponentianlly larger than $\mathsf{tri}(P)$. From Lemma 3.9, the running time of Algorithm 1, which is $O(n \cdot \mathsf{tri}(P) + \mathsf{cg}(P))$ time, is dominated by $\mathsf{cg}(P)$.

Theorem 3.10. *Let P be a set of n points in the plane. Then the set of non-crossing spanning connected graphs on P can be enumerated in $O(\mathsf{cg}(P))$ time.*

3.3.3 Enumerating Plane Straight-line Graph

For any $E \subseteq T \setminus R(T)$, $E \cup R(T)$ is a plane straight-line graph containing $R(T)$. Hence, by enumerating (the symmetric differences of) all subsets of $T \setminus R(T)$, we obtain all $R(T)$-constrained plane straight-line graphs in T. Enumerating all subsets of $T \setminus R(T)$ is equivalent to generating all $|R \setminus R(T)|$-bit binary numbers with $O(n)$ preprocessing time, which can be done in constant time per output (see e.g. [29]). Algorithm 1 thus enumerates all plane straight-line graphs in $O(n \cdot \mathsf{tri}(P) + \mathsf{pg}(P))$ time.

Since a non-crossing spanning connected graph is also a plane straight-line graph, $2^{n/2-1}\mathsf{tri}(P) \leq \mathsf{cg}(P) \leq \mathsf{pg}(P)$ holds from Lemma 3.9. Thus, we obtain the following result.

Theorem 3.11. *Let P be a set of n points in the plane. Then the set of plane straigh-line graphs on P can be enumerated in $O(\mathsf{pg}(P))$ time.*

3.3.4 Enumerating Non-crossing Perfect Matchings

Given a point set P of $2n$ points, a *non-crossing perfect matching* is a non-crossing geometric graph on P such that every point of P is incident to exactly one edge of the graph.

Let us consider how to design Phase 2 of Algorithm 1. Suppose we have an algorithm for finding a perfect matching in a given (non-geometric) graph in $t_{\mathcal{PM}}$ time if it exists. Then, using this algorithm as an oracle, the naively implemented binary partition algorithm (discussed in Section 1) can enumerate all the perfect matchings in a given graph (if exists) in $O(nt_{\mathcal{PM}})$ time per output graph (see [28]). The edge constraints can be treated easily. If T has a vertex that is incident to more than one edge of $R(T)$, we report that there is no perfect matching in T. Otherwise we first remove all of $R(T)$ together with the vertices incindent to $R(T)$ and then apply the above algorithm for enumerating perfect matchings to the resulting graph. By putting $R(T)$ back to each solution, we obtain all the perfect matchings in T that contain $R(T)$. Algorithm 1 hence enumerates all the non-crossing perfect matchings on P in $O(t_{\mathcal{PM}} \cdot \mathsf{tri}(P) + nt_{\mathcal{PM}} \cdot \mathsf{pm}(P))$ time.

4 Independent Minimal Representative Set

We know that the algorithm by Bespamyatnikh [11] enumerates all triangulations efficiently, but its search tree is not nicely structured when we focus on minimal representative sets (see Fig. 5). Namely, for two triangulations T_1 and T_2 for which T_1 is a parent of T_2 in the search tree, T_2 may miss some representative edge that appears in T_1. Consider, for example, the enumeration of non-crossing matchings. Then, in Phase 2 of Algorithm 1 for a triangulation T_1, the algorithm outputs no non-crossing matching if there is a vertex incident to more than one edge of $R(T_1)$. However, since some descendant triangulation T_2 of T_1 may output non-crossing matchings, we cannot skip the enumeration of T_1 and its descendants. The next proposed algorithm avoids this inefficiecy.

We will first propose a new algorithm for enumerating triangulations whose search tree has a monotone structure with respect to the minimal representative sets such that $R(T_1) \subset R(T_2)$ holds for any triangulation T_1 and its descendant T_2 in the search tree (see Fig. 7). By using this monotonicity, we can efficiently enumerate only the minimal representative sets satisfying the specified property. Let us explain this idea more formally. Recall that $\mathcal{F}$ denotes the collection of all non-crossing edge sets on P. Let $\mathcal{I}$ be a subset of $\mathcal{F}$ satisfying the following independent system:

(I1) $\emptyset \in \mathcal{I}$,

(I2) if $F_2 \in \mathcal{I}$ and $F_1 \subseteq F_2$, then $F_1 \in \mathcal{I}$.

A non-crossing edge set $F \in \mathcal{F}$ is called *independent edge set* or *independent* (with respect to $\mathcal{I}$) if $F \in \mathcal{I}$. Similarly, the minimal representative set is called *independent minimal representative set* if it is independent. Using the monotonicity of the minimal representative sets, the proposed algorithm enumerates all independent minimal representative sets efficiently. If $\mathcal{I}$ satisfies the following condition:

(I3) for every $G \in \mathcal{NGG}$, $G \in \mathcal{I}$ holds, (where G is considered as an edge set),

then we can ensure that the minimal representative set of $T^*(G)$ is independent for every $G \in \mathcal{NGG}$. This implies that it is sufficient to enumerate only the independent minimal representative sets to enumerate all graphs of $\mathcal{NGG}$.

4.1 Enumerating Triangulations Based on Edge Insertion

Our new enumeration algorithm for triangulations is also based on the reverse search [6] whose search tree can be characterized by the root triangulation and parent-child relation (see Section 3.2 for the brief explanation of the reverse search). Here we define $T^*(\emptyset)$ as the root triangulation. So the minimal representative set of the root triangulation is empty. For each non-root triangulation T, the parent of T is defined as $T^*(R(T) \setminus \{e\})$ with the smallest edge e among $R(T)$. The correctness of our parent-child relation follows from the next lemma.

Lemma 4.1. *Let T be the triangulation of the minimal representative set $R(T) \neq \emptyset$. Then, for any $e \in R(T)$, the minimal representative set of $T^*(R(T) \setminus \{e\})$ is $R(T) \setminus \{e\}$.*

Proof. Let $T' = T^*(R(T)\setminus\{e\})$. It is sufficient to show $R(T)\setminus\{e\} \subseteq R(T')$ because $R(T') \subseteq R(T)\setminus\{e\}$ holds from Lemma 3.4.

Consider $(p_i, p_j) \in R(T) \setminus \{e\}$. Let p_ip_jv and p_ip_jw be two triangles incident to (p_i, p_j) in T, and similarly let p_ip_jv' and p_ip_jw' be those in T'. Without loss of generality, we assume that v and v', (and w and w'), lie on the lower (and the upper) side of (p_i, p_j). If $v = v'$ and $w = w'$, then the triangle faces incident to (p_i, p_j) do not differ between T and T'. Hence $(p_i, p_j) \in R(T)$ implies $(p_i, p_j) \in R(T')$.

Let us consider the case in which $v \neq v'$ holds. Let $\delta_{R(T)}(p_i)$ be a subset of $R(T)$ whose left endpoints are incident to p_i, and let (p_i, p_i^{up}) and $(p_i, p_i^{\mathrm{low}})$ be the upper and lower tangents of p_i with respect to $R(T)$. When constructing $T = T^*(R(T))$ by Construction 1, there exists a cone C with apex p_i, bounded by (p_i, p_j) and the other consecutive edge of $\delta_{R(T)}(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$ which contains both p_j and v. Let H be the convex hull of $P_{i+1} \cap C$. Then, $v = (p_i, v) \cap H$ holds

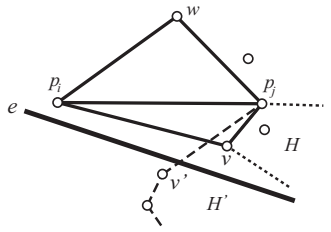


Figure 6: Illustration of the proof of Lemma 4.1, where a bold line represents the removed edge e, dotted and dashed lines represent the boundary of H and H', respectively.

since $(p_i, v) \in T^*(R(T))$. Similarly, when constructing $T' = T^*(R(T) \setminus \{e\})$, there exists a convex hull H' below $(p_i, p_j) \in R(T) \setminus \{e\}$ for which $v' = (p_i, v') \cap H'$ holds. Since every vertex visible from p_i with respect to $R(T)$ is still visible from p_i with respect to $R(T) \setminus \{e\}$, all the right endpoints of the edges of $\delta_{R(T)}(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$ are still visible from p_i with respect to $R(T) \setminus \{e\}$. Thus, $H \subseteq H'$ holds and H' contains v (see Fig. 6).

It is easily observed that, from $H \subseteq H'$, (p_i, p_j) does not become the smallest one of $\{(p_i, q) \in T' \mid q \in P_{i+1}\}$ when removing e (and it is not the largest one either). Hence, by the definition of the minimal representative set, $(p_i, p_j) \in R(T')$ holds if (p_i, p_j) is flippable in T'. Since there exists no point of P inside the triangle $p_i p_j v$, either one of the following two cases occurs depending on the position of v':(i) (p_i, v') intersects (v, p_j) or (ii) (v', p_j) intersects (p_i, v). When (i) holds, v' is properly contained in H. However, since $H \subseteq H'$, v' is also properly contained in H', which contradicts $v' = (p_i, v') \cap H'$. Thus, (ii) must hold. When (ii) holds, the inner angles $\angle p_i p_j v$ and $\angle p_i p_j v'$ satisfy $\angle p_i p_j v' \leq \angle p_i p_j v$. Applying a similar argument to the pair of w and w', we have $\angle p_i p_j w' \leq \angle p_i p_j w$. (However, it is not difficult to see $w = w'$ from the fact that e properly intersects (v', p_j) but not (p_i, p_j).) Hence the inner angle of the quadrilateral $p_i v' p_j w'$ at p_j is less than π because (p_i, p_j) is flippable in T, implying the inner angle of $p_i v p_j w$ at p_i is less than π.

Let us show that the opposite angle, that is the inner angle of the quadrilateral $p_i v' p_j w'$ at p_i, is also less than π. This can be proved from the fact that both of v' and w' are on the right side of p_i since (p_i, p_j) is neither the smallest nor largest one of $\{(p_i, q) \in T' \mid q \in P_{i+1}\}$. Hence (p_i, p_j) is flippable in T', and $(p_i, p_j) \in R(T')$ holds. □

From Lemma 4.1, $R(T) \subset R(T')$ holds for any triangulation T and its descendant T'. Moreover, since the root triangulation has an empty minimal representative set, our definition of the parent-child relation correctly induces a rooted search tree on the search graph of the set of triangulations. The algorithm traces this search tree in depth-first manner. We call this new algorithm *edge insertion algorithm* for (enumerating) triangulations. An example of the new search tree is depicted in Fig. 7.

Let us analyze the time complexity of the edge insertion algorithm. In the reverse search the most time-consuming part is to find all children T' of a triangulation T, i.e., to find all edges $e \in K_n$ for which $T' = T^*(R(T) \cup \{e\})$ is a child of T. Such e can be characterized by the following lemma.

Lemma 4.2. *Let T and T' be triangulations on P for which $T' = T^*(R(T) \cup \{e\})$ holds for $e \in K_n$, where e does not intersect any of $R(T)$. Then T' is a child of T if and only if all of the following three conditions are satisfied:*

- **(a)** *$e \notin T$,*
- **(b)** *$e \prec e_1$, where e_1 is the lexicographically smallest edge among $R(T)$, and*
- **(c)** *$R(T) \subseteq R(T')$.*

Proof. ("only if"-part:) When T' is a child of T, by the definition of the parent, we have $R(T') = R(T) \cup \{e\}$ and e is the smallest edge among $R(T')$, which implies the conditions (b) and (c). Suppose the condition (a) does not hold. Then we have $R(T) \subseteq R(T) \cup \{e\} \subseteq T$ since $e \in T$, and hence we obtain $R(T) \cup \{e\} \in [T]$ (i.e., $T' = T^*(R(T) \cup \{e\}) = T$) from Lemma 3.4, which contradicts that T' is a child of T.

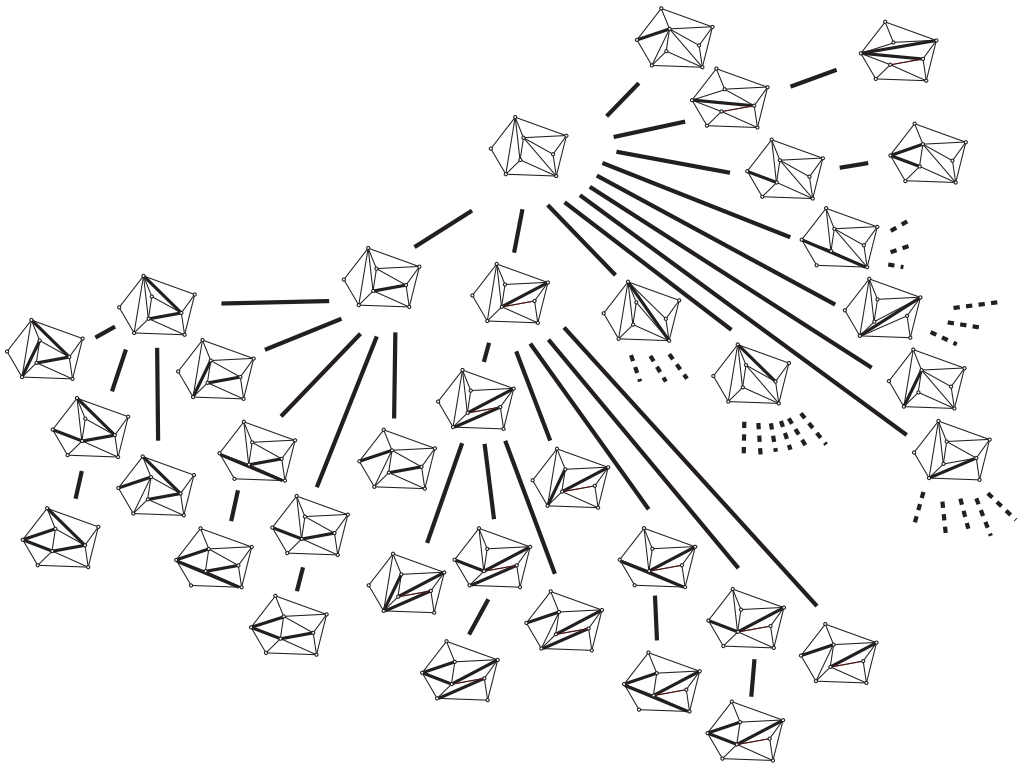


Figure 7: Search tree on the set of triangulations obtained by the edge insertion algorithm, where each minimal representative set is drawn in bold.

("if"-part:) First let us show $e \in R(T')$. Suppose $e \notin R(T')$. Then, by the definition of $R(T')$, e is either non-flippable in T', or the smallest or largest edge among $\{(p_i, q) \in T' \mid q \in P_{i+1}\}$ for the left endpoint p_i of e. We hence have, from Lemma 3.2,

$$e \in T' = T^*(R(T) \cup \{e\}) = T^*((R(T) \cup \{e\}) \setminus \{e\}) = T^*(R(T)) = T,$$

which contradicts the condition (a).

Combining $e \in R(T')$ and the condition (c), we obtain $R(T) \cup \{e\} \subseteq R(T')$. On the other hand $R(T') \subseteq R(T) \cup \{e\}$ is known from Lemma 3.4. Therefore, $R(T') = R(T) \cup \{e\}$ holds. The condition (b) says that e is the smallest edge among $R(T) \cup \{e\}$, and hence, according to the definition of the parent, $T^*(R(T') \setminus \{e\}) = T^*(R(T)) = T$ is a parent of T'. □

From Lemma 4.2, we now concentrate on how to find all edges satisfying all the conditions of Lemma 4.2 that produce children of a given triangulation T. We assume that the points are stored in increasing order of their x-coordinates. Let p_c be the left endpoint of the lexicographically smallest edge of $R(T)$ and let c be its label, which is called *critical vertex* of T.

We first show that all edges satisfying the conditions of Lemma 4.2 can be found in $O(cn^2)$ time for each T. Note that the number of edges satisfying the condition (b) can be bounded from above by $\sum_{i=1}^{c}(n-i) < cn$. The algorithm checks each of these edges one by one whether it satisfies the other conditions (a) and (c). This task can be done in $O(n)$ time for each edge e by constructing $T^*(R(T) \cup \{e\})$ based on the method of Lemma 2.5. Thus we can find all edges e that satisfy all the conditions of Lemma 4.2 in $O(cn^2)$ time.

This $O(cn^2)$ time is improved to $O(n^2/c)$ time by a simple amortized analysis as follows. Consider the point set $P' = \{p_1, \ldots, p_c\}$. We claim that, for any edge $e \in K_n \setminus T$ whose both endpoints are contained in P', e satisfies all the conditions of Lemma 4.2. Since such e clearly satisfies the conditions (a) and (b) from its definition, let us confirm that e also satisfies the condition (c). Notice that there exists no edge of $R(T)$ in the left side of p_c since p_c is defined to be the left endpoint of the smallest edge among $R(T)$. Hence, inserting e into $R(T)$ does not affect the right side of c when

constructing $T^*(R(T)\cup\{e\})$, i.e. every $e' \in R(T)$ are incident to two triangles in $T^*(R(T)\cup\{e\})$ as in $T = T^*(R(T))$, and all of $R(T)$ are still contained in the minimal representative set of $T^*(R(T)\cup\{e\})$.

The number of edges $e \in K_n \setminus T$ whose both endpoints are contained in P' is at least $c(c-1)/2 - (3c-6)$. This implies that there exist $\Omega(c^2)$ children of T. Distributing the time $O(cn^2)$ evenly to $\Omega(c^2)$ children and T itself, we obtain the result.

Theorem 4.3. *Let P be a set of n points. Then the edge insertion algorithm enumerates all the triangulations on P in $O(n^2)$ time per output graph.*

4.2 Enumerating Independent Minimal Representative Sets

Owing to the nicely structured search tree of minimal representative sets, we can now perform the efficient enumeration of independent minimal representative sets (defined at the beginning of Section 4) and the corresponding triangulations.

Algorithm 2: Enumeration of $\mathcal{NGG}$.

Phase 1: Start to enumerate triangulations based on the edge insertion algorithm from $T^*(\emptyset)$ as described in Section 4.1. Every time a new triangulation T is found, it checks whether $R(T)$ is independent or dependent. If $R(T)$ is dependent, skip the enumeration of all the descendants of T.

Phase 2: Every time a new independent $R(T)$ is found, enumerate all $R(T)$-constrained graphs of $\mathcal{NGG}$ in T.

The correctness of Algorithm 2 follows from the next lemma.

Lemma 4.4. *Let $\mathcal{I}$ be the collection of independent edge sets of $\mathcal{F}$. Then Algorithm 2 correctly enumerates all graphs of $\mathcal{NGG}$ without repetitions if $\mathcal{I}$ satisfies (I1), (I2) and (I3).*

Proof. We first note that all of independent minimal representative sets are correctly enumerated in Algorithm 2. To verify this, let us imagine the search tree which is obtained by performing the edge insertion algorithm for enumerating triangulations. The subgraph of this search tree induced by every T such that $R(T) \in \mathcal{I}$ forms a rooted tree by (I1) and (I2), and hence the algorithm enumerates every independent $R(T)$ correctly.

Let us show every $G \in \mathcal{NGG}$ is actually enumerated. From (I3) we have $G \in \mathcal{I}$. Consider the minimal representative edge set $R(T^*(G))$. Notice that $R(T^*(G))$ is a subset of G from Lemma 3.4. Hence, from (I2), $R(T^*(G))$ is independent, and thus G is enumerated in Phase 2. □

Let us analyze the time complexity of Algorithm 2 under the assumption that $\mathcal{I}$ satisfies (I1), (I2) and (I3). Assume that there exists an oracle that checks in t_{check} time whether $I \cup \{e\} \in \mathcal{I}$ or not for an independent set I and an edge e. Let $\mathcal{I}_{\text{rep}} \subseteq \mathcal{I}$ be the collection of the independent minimal representative sets on a given point set P. We can easily observe that the time to be spent in Phase 1 is $O(n^2 \cdot t_{\text{check}} \cdot |\mathcal{I}_{\text{rep}}|)$ since there exist $O(n^2)$ children for each triangulation on the search tree of the edge insertion algorithm and from Theorem 4.3. Hence, using the notations $\mathcal{C}, t_{\mathcal{C}}$ and $t_{\mathcal{C},\text{pre}}$ defined in Theorem 3.7, we obtain the following result:

Theorem 4.5. *Algorithm 2 enumerates all the elements of $\mathcal{NGG}$ on a given point set P without repetitions in $O((n^2 \cdot t_{\text{check}} + t_{\mathcal{C},\text{pre}}) \cdot |\mathcal{I}_{\text{rep}}| + t_{\mathcal{C}} \cdot \mathsf{ngg}(P))$ time. Moreover, the time complexity is bounded by $O((n^2 \cdot t_{\text{check}} + t_{\mathcal{C},\text{pre}} + t_{\mathcal{C}}) \cdot \mathsf{ngg}(P))$ if $|\mathcal{I}_{\text{rep}}| \leq \mathsf{ngg}(P)$ holds.*

4.3 Application of Algorithm 2

We show here how Algorithm 2 can be applied to the enumeration of non-crossing minimally rigid frameworks. A graph $G = (V, E)$ is *minimally rigid* if $|E| = 2|V| - 3$ and every subgraph of G induced by $V' \subseteq V$ spans at most $2|V'| - 3$ edges. An embedded minimally rigid graph on a planar point set is called *minimally rigid framework*. It is known that a set of minimally rigid graphs forms a *rigidity matroid* defined on the edge set (see e.g. [16]).

We define the *independence* on $\mathcal{F}$ in such a way that $F \in \mathcal{F}$ is independent if and only if F is independent in the rigidity matroid on K_n. Then, since the edge set of each minimally rigid framework is a base of the rigidity matroid, the collection of the independent edge sets of $\mathcal{F}$ satisfies (I1), (I2) and (I3). To bound the number of independent minimal representative sets $|\mathcal{I}_{\text{rep}}|$, we remark the following known fact:

Lemma 4.6. ([8]) *Let F be a non-crossing edge set on P that is an independent set in the rigidity matroid on K_n. Then every F-constrained triangulation on P contains an F-constrained minimally rigid framework.*

Hence, a triangulation T contains at least one $R(T)$-constrained non-crossing minimally rigid graph if $R(T)$ is independent, which implies $|\mathcal{I}_{\text{rep}}| \leq \mathsf{ngg}(P)$.

Let us consider the time complexity of Phase 1 of Algorithm 2. For a graph $G = (V, I)$ with n vertices and an independent set I of the rigidity matroid, a maximal rigid subgraph $G' = (V', I')$ of G, (i.e., a subgraph with the maximal subset $I' \subseteq I$ satisfying $|I'| = 2|V'| - 3$), is called *rigid component*. Then $I \cup \{e\}$ is independent if and only if both endpoints of e do not belong to the same rigid component. It is known that all the rigid components of G can be detected in $O(n^2)$ time (e.g. [10]). Moreover, using the data structure by Lee et al. [27] or Berg and Jordán [10] that maintains rigid components, it can be checked in $O(1)$ time whether two vertices belong to the same rigid component. Thus, the algorithm can check in $t_{\text{check}} = O(1)$ time whether a minimal representative set of a new child, that is $R(T) \cup \{e\}$, is independent or not. If $R(T) \cup \{e\}$ is independent, the algorithm enters Phase 2 while updating the rigid components in $t_{\text{update}} = O(n)$ time for each edge insertion [10, 27]. Algorithm 2 hence enumerates all independent minimal representative sets $R(T)$ (and triangulations T) in $O(n^2 \cdot t_{\text{check}} + t_{\text{update}}) = O(n^2)$ time per $R(T)$.

Next let us consider Phase 2. We enumerate all the minimally rigid graphs of a given graph containing a specified edge set by using the algorithm for enumerating all bases of a matroid. We use the known algorithm by Uno [33] to enumerate all the minimally rigid graphs of a given graph containing a specified edge set, which generates all bases of a given matroid $\mathcal{M}$ on the ground set E and rank r in $O(t_{\text{cir}}/r)$ time per base with preprocessing time t_{pre}, where t_{cir} is the time to calculate a circuit of $B \cup \{e\}$ of a base B and $e \in E \setminus B$, and t_{pre} is the time to compute the bridges of the matroid in E (where $e \in E$ is called a bridge if all bases contain e). In the case of the rigidity matroid[2], the algorithm by Berg and Jordán [10] can detect a circuit of $B \cup \{e\}$ in $t_{\text{cir}} = O(r^2)$ time for a base B and for each $e \in E \setminus B$. Moreover, the above idea by Berg and Jordán can also be utilized to detect all the bridges in E in $t_{\text{pre}} = O(r^2)$ time. It thus enumerates all the minimally rigid graphs in G that contains a specified edge set in $t_{\mathcal{C}} = O(n)$ time per output graph with $t_{\mathcal{C},\text{pre}} = O(n^2)$ preprocessing time. Putting these facts and Theorem 4.5 together gives the following result:

Theorem 4.7. *Let P be a set of n points in the plane. Then the set of non-crossing minimally rigid frameworks can be enumerated without repetitions in $O(n^2 \cdot \mathsf{mrf}(P))$ time.*

This result improves the previous one by [7], which requires $O(n^3)$ time per graph. We note that Algorithm 1 enumerates all the non-crossing minimally rigid frameworks in $O(n^2 \cdot \mathsf{tri}(P) + n \cdot \mathsf{mrf}(P))$ time.

5 Other Applications.

We proposed a new algorithm framework for efficient enumeration of non-crossing geometric graphs, and we showed improved or new algorithms for several specific graph classes by applying our technique. We briefly show below the applications of the proposed framework to the other graph classes.

[2]We remark here that the rigidity matroids are not closed under contraction, i.e. the matroid obtained after contraction may not be rigidity matroid. Actually the algorithm by Uno [33] for enumerating the matroid bases heavily utilizes the contraction. So, instead of performing the contraction during the algorithm, we just attach the flag to the edge to be contracted to use the algorithm by Berg and Jordán [10]. In fact, similar argument is discussed in [33] when enumerating bases of *transversal matroid*.

Algorithm 1 basically works in time proportional to the number of triangulations and objects to be enumerated. Meanwhile, in some problems, Algorithm 2 will works practically faster than Algorithm 1 although it seems non-trivial task to estimate its time complexity theoretically.

Non-crossing red-and-blue matchings: For a given point set P, every point is assumed to have either red or blue color. A non-crossing red-and-blue matching is a non-crossing matching on P each of whose edges is not allowed to connect points of the same color. The enumeration can be performed by using the algorithm for enumerating matchings in a (non-geometric) bipartite graph [35] in Phase 2. Algorithm 2 can enumerate all red-and-blue matchings efficiently if we define a independent set $calI$ on $\mathcal{F}$ as a set of F such that no two edges of F are incident to a vertex and no edge of F connects points of the same color.

Non-crossing k-vertex or k-edge connected graphs: A Non-crossing k-vertex (or k-edge) connected graph is a non-crossing geometric graph spanning a given point set P that remains connected after removing any $k-1$ vertices ($k-1$ edges) in the graph. Since it can be checked in polynomial time whether a given (non-geometric) graph is k-vertex connected (similarly k-edge connected) or not, according to the binary partition technique discussed in Introduction, we can enumerate k-vertex connected (or k-edge connected) graphs in a given graph. Thus, the enumeration can be performed by using this algorithm in Phase 2.

Non-crossing directed spanning trees: Each edge of a given geometric complete graph on P is assumed to have an orientation. A non-crossing directed spanning tree (or non-crossing r-arborescence) is a non-crossing spanning tree on P having a unique directed path from a rooted point r to all points of $P \setminus \{p\}$. The enumeration can be performed by using the algorithm of [19, 34] in Phase 2. Algorithm 2 can enumerate all non-crossing directed spanning trees if we define $\mathcal{I}$ as a set of F such that the directed graph D induced by F has no (directed) cycle and no vertex of D has indegree more than 1.

Edge-constrained non-crossing geometric graphs: The technique can be also applied to the enumeration of S-constrained non-crossing geometric graphs that are those containing a given specified edge set S as their subsets, e.g. S-constrained non-crossing spanning trees or S-constrained matchings. It is because that both the algorithm by Bespamyatnikh [11] and the edge insertion algorithm proposed in this paper for enumerating triangulations can be naturally extended to those for enumerating only the S-constrained triangulations by restricting the collection of non-crossing edge sets $\mathcal{F}$ to those containing S as their subset. The correctness of this extension is easily derived from the properties of CLLT shown in [21].

Acknowledgment

The first author is supported by the project *New Horizons in Computing*, Grant-in-Aid for Scientific Research on Priority Areas, NEXT Japan. The second author is supported by Grant-in-Aid for JSPS Research Fellowships for Young Scientists.

References

[1] O. Aichholzer, F. Aurenhammer, C. Huemer and H. Krasser. Transforming spanning trees and pseudo-triangulations. *Inf. Process. Lett.*, 97(1):19–22, 2006.

[2] O. Aichholzer, F. Aurenhammer, C. Huemer, and B. Vogtenhuber. Gray code enumeration of plane straight-line graphs. *Graph and Combinatorics*, 23(5):467–479, 2007.

[3] O. Aichholzer, F. Aurenhammer and F. Hurtado. Sequences of spanning trees and a fixed tree theorem. *Comput. Geom.*, 21(1-2):3–20, 2002.

[4] O. Aichholzer, T. Hackl, C. Huemer, F. Hurtado, H. Krasser, and B. Vogtenhuber. On the number of plane geometric graphs. *Graphs and Combinatorics*, 23[Suppl]:67–84, 2007.

[5] O. Aichholzer and K. Reinhardt. A quadratic distance bound on sliding between crossing-free spanning trees. *Computational Geometry: Theory and Applications*, 37:155–161, 2007.

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

[7] D. Avis, N. Katoh, M. Ohsaki, I. Streinu and S. Tanigawa. Enumerating non-crossing minimally rigid frameworks. *Graph and Combinatorics*, Vol. 23, 2007, supplement, Computational Geometry and Graph Theory, The Akiyama-Chvátal Festschrift, pp.117–134.

[8] D. Avis, N. Katoh, M. Ohsaki, I. Streinu and S. Tanigawa. Enumerating constrained non-crossing minimally rigid frameworks. to appear in *Discrete & Computational Geometry*.

[9] S. Bereg. Enumerating pseudo-triangulations in the plane. *Comput. Geom. Theory Appl.*, 30(3):207–222, 2005.

[10] A. Berg and T. Jordán. Algorithms for graph rigidity and scene analysis. In *Proc. 11th Annual European Symposium on Algorithms (ESA)*, LNCS 2832, pages 78–89. Springer, 2003.

[11] S. Bespamyatnikh. An efficient algorithm for enumeration of triangulations. *Comput. Geom. Theory Appl.*, 23(3):271–279, 2002.

[12] H. Brönnimann, L. Kettner, M. Pocchiola and J. Snoeyink. Counting and enumerating pointed pseudo-triangulations with the greedy flip algorithm *SIAM J. Comput.*, 36(3):721–739, 2006.

[13] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485-524, 1991.

[14] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, Introduction to Algorithm, Second Edition,. The MIT Press, 2001

[15] P. Flajolet and M. Noy. Analytic combinatorics of non-crossing configurations. *Discrete Math.*, 204:203–229, 1999.

[16] J. Graver, B. Servatius, and H. Servatius. *Combinatorial Rigidity*. Graduate Studies in Mathematics vol. 2. American Mathematical Society, 1993.

[17] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons, *Algorithmica*, 2:209–233, 1987.

[18] S. Kapoor and H. Ramesh. Algorithms for enumerating all spanning trees of undirected and weighted graphs, *SIAM J. Comput.*, 24(2):247–265, 1995.

[19] S. Kapoor and H. Ramesh. An algorithm for enumerating all spanning trees of a directed graph, *Algorithmica*, 27(2):120–130, 2000.

[20] N. Katoh and S. Tanigawa. Enumerating constrained non-crossing spanning trees. In *Proc. 13th Annual International Conference Computing and Combinatorics (COCOON 2007)*, LNCS 4598, pages 243–253, 2007.

[21] N. Katoh and S. Tanigawa. Enumerating edge-constrained triangulations and edge-constrained non-crossing spanning trees. *Submitted*

[22] M. C. Hernando, M. E. Houle and F. Hurtado. On local transformation of polygons with visibility properties, In *Proc.6th Annual International Conference Computing and Combinatorics,COCOON 2000*, LNCS 1858, pages 54–63. Springer, 2000.

[23] C. Hernando, F. Hurtado and M. Noy. Graphs of non-crossing perfect matchings, *Graphs and Combinatorics*, 18(3):517–532, 2002.

[24] M. E. Houle, F. Hurtado and M. Noy and E. Rivera-Campo, Graphs of triangulations and perfect matchings, *Graphs and Combinatorics*, 21(3):325-331, 2005.

[25] F. Hurtado, M. Noy and J. Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3):333–346, 1999.

[26] K. Jansen and G. J. Woeginger. The complexity of detecting crossingfree configurations in the plane. *BIT*, 33(4):580-595, 1993.

[27] A. Lee and I. Streinu. Pebble game algorithms and sparse graphs. *Discrete Mathematics*, 308(8):1425–1437, 2008.

[28] Y. Matsui, T. Matsui and K. Fukuda. A catalog of enumeration algorithms, `http://roso.epfl.ch/kf/enum/enum.html`.

[29] C. Savage. A survey of combinatorial Gray codes. *SIAM Review*, 39:605–629 1997.

[30] A. Shioura and A. Tamura. Efficiently scanning all spanning trees of an undirected graph, *Journal of the Operations Research Society of Japan*, 38(3):331–344, The Operations Research Society of Japan, 1995.

[31] A. Shioura, A. Tamura and T. Uno. An optimal algorithm for scanning all spanning trees of undirected graphs, *SIAM J. Comput.*, 26(3):678–692, 1997.

[32] T. Uno, A fast algorithm for enumerating non-bipartite maximal matchings. *Journal of National Institute of Information*, 3:89–97, 2001.

[33] T. Uno, A new approach for speeding up enumeration algorithms and its application for matroid bases. In *Proc. 5th Computing and Combinatorics Conference (COCOON '99)*, LNCS 1627, pp.349–359, 1999.

[34] T. Uno, A new approach for speeding up enumeration algorithms. In *Proc. International Symposium on Algorithm and Computation (ISAAC '98)*, LNCS 1533, pp.287–296, 1998.

[35] T. Uno, Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs. In *Proc. 8th International Symposium on Algorithms and Computation (ISAAC '97)*, LNCS 1350, pp.92–101, 1997.

[36] D. J. A. Welsh. Matroids: fundamental concepts. In *Handbook of Combinatorics Vo.I, R.L.Graham, M.Grötschel, and L.Lovász eds.*, North-Holland, 1995, 481-526.