数理解析研究所講究録
第1644巻 2009年 73-85

Enumerations of 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

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 $P = \{p_1, \cdots, p_n\} \subset \mathbb{R}^2$ is a mapping of the vertices to the points in the Euclidean plane $i \mapsto p_i$. A *geometric graph* 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 a geometric graph is called non-crossing if its corresponding straight line segments are non-crossing.

In this paper we assume that a given point set P is *fixed* in $\mathbb{R}^2$ and an embedding $V \to P$ is given. Since a *graph class* is defined in terms of the properties that all its members share, imposing the additional "non-crossing" requirement to an existing graph class, we can define a *non-crossing geometric graph class* on P, 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 [13], we have presented a new general framework for enumerating non-crossing geometric graphs on P, which 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 graphs*. The proposed framework is based on combinatorial properties of the edge-constrained lexicographically largest triangulations. In this note, we shall present a slight extension of this technique, which enumerates all non-crossing geometric graphs containing some specified segments. The problem is formulated as follows:

Input: A point set P in the plane with n points and a non-crossing straight line segments F connecting points of P.

Output: The list of all non-crossing geometric graphs on P each of which contains F and belongs to $\mathcal{NGG}$.

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)*.

Enumerating combinatorial objects is a fundamental problem, and several algorithms have been developed for non-crossing geometric graphs, e.g. triangulations [3, 7], non-crossing spanning trees [1, 3, 12], pseudo-triangulations [6, 8] and non-crossing minimally rigid graphs [4, 5]. Let us explain why the enumeration of non-crossing geometric graphs is more difficult than that of non-geometric (abstract) graphs. The *branch-and-bound technique* (or sometimes called the binary-partition technique, see e.g. [20, 21]) is a well known framework for designing enumeration algorithms. Consider, for example, the problem for enumerating all spanning trees in a (multi)graph G with n vertices and m edges. Then, we can easily design an algorithm that enumerates all spanning trees in $O(m^2)$ 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 resulting loops are 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 $O(m)$ 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 $O(m)$ steps, the algorithm surely detects a new spanning tree.

The branch-and-bound technique provides us with polynomial time enumeration algorithms for many graph classes because it just requires a polynomial time oracle that checks whether a given graph contains at least one subgraph belonging to a certain graph class. However, the problem of detecting a non-crossing subgraph in a given geometric 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 [14]). For this reason, most of the enumeration problems for non-crossing geometric graphs become non-trivial and we need to introduce some new technique.

The paper consists of seven sections. In Section 2, we shall review the preliminary results of the edge-constrained lexicographically largest triangulation. In Section 3, we shall show that every edge-constrained lexicographically largest triangulation has a "core set", called the minimal representative set, which plays a key role in our enumeration techniques. In Section 4, we will discuss two enumeration algorithms for enumerating edge-constrained triangulations. Section 5 describes our main result, general techniques for enumerating non-crossing geometric graphs and in Section 6 we shall show how to apply these techniques to some specific graph classes. Finally we will discuss some open problems in the last section.

2 The Edge-Constrained Lexicographically Largest Triangulation

A geometric graph containing a set of non-crossing straight line segments S is called *S-constrained.* In this section, we will first introduce some notations used throughout the paper, and then provide a number of preliminary results on the S-constrained lexicographically largest triangulation (S-CLLT).

Let P be a set of n points in $\mathbb{R}^2$, 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 points $p_i, p_j \in P$, 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). The line segment between p_i and p_j with $p_i < p_j$ is called *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 the triangle $p_i p_j p_k$ tells us whether p_k is on the left (or right, resp.) 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 the 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 $\Delta(p_i, p_j, p_l) < 0$ (see Fig. 1). Notice that the ordering of e and e' is determined by the clockwise ordering around p_i if $p_i = p_k$. 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 edges (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 $p_i, p_j \in P$ and a non-crossing edge set S, we say that p_j is *visible from p_i with respect to S* when the edge (p_i, p_j) does not properly intersect any edge of S, but we assume that p_j is *visible* from p_i if $(p_i, p_j) \in S$. *Upper* and *lower tangents*, (p_i, p_i^{up}) and $(p_i, p_i^{\mathrm{low}})$, *of p_i with respect to S* are defined as the supporting edges from p_i to the convex hull of the points of P_{i+1} that are visible from p_i with respect to S (see Fig. 2).

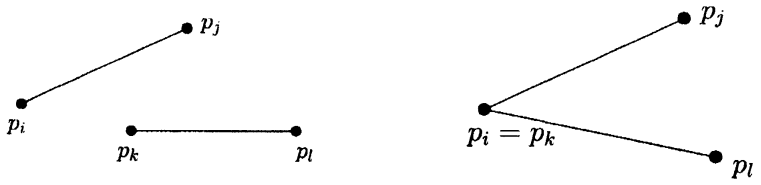

Figure 1: $(p_i, p_j) \prec (p_k, p_l)$.

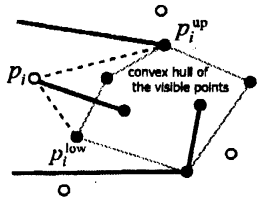

Figure 2: An example of the upper and lower tangents, denoted by (p_i, p_i^{up}) and (p_i, p_i^{low}), respectively. The bold edges represent S.

Given a point set P, and a set of non-crossing edges S on P. Consider the following construction of an S-constrained triangulation T:

Construction 1.

- *First, insert all edges of S as those of T.*
- *Then, greedily insert the edges e of $K_n \setminus S$ into T in lexicographically descending order if e does not properly intersect any edge of T.*

It is obvious that the above construction produces the triangulation on P containing S because it has the maximal number of non-crossing edges. We give an example of the graph obtained by the above construction in Fig. 3.

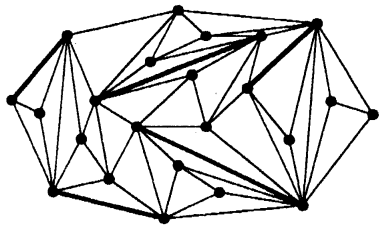

Figure 3: S-CLLT.

For $p_i \in P$ and an edge set E on P, $\delta_E(p_i)$ denotes the set of edges of E which are incident to p_i with the left endpoints.

Lemma 2.1. *The S-constrained triangulation $T^*(S)$ obtained by the above construction has the lexicographically largest edge list among all S-constrained triangulations on P.*

Proof. Let us denote the edges of $T^*(S)$ by $\{e_1^*, \ldots, e_m^*\}$ with $e_1^* \prec \cdots \prec e_m^*$. Suppose there exists an S-constrained triangulation T whose edge set $\{e_1, \ldots, e_m\}$ with $e_1 \prec \cdots \prec e_m$ is lexicographically larger than that of $T^*(S)$. Then, there exists the smallest label s with $e_s^* \neq e_s$ for which $e_s^* \notin T$ and $e_s^* \prec e_s$ hold.

Let $e_s^* = (p_i, p_j) \in T^*(S) \setminus T$. From the choice of e_s^* we have

$$\delta_{T^*(S)}(p) = \delta_T(p) \qquad \text{for every } p \in \{p_1, \ldots, p_{i-1}\}. \tag{1}$$

It is not difficult to see $e_s^* \notin S$ since otherwise T has an edge that properly intersect $e_s^* \in S$, contradicting that T is an S-constrained triangulation. 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 S. We claim the followings:

$$(p_i, p_i^{\mathrm{up}}) \text{ and } (p_i, p_i^{\mathrm{low}}) \text{ are contained in both } T^*(S) \text{ and } T. \tag{2}$$

The fact of $(p_i, p_i^{\mathrm{up}}) \in T^*(F)$ and $(p_i, p_i^{\mathrm{low}}) \in T^*(S)$ follows from the maximality of Construction 1. Since Construction 1 greedily inserts the edges from right to left, $T^*(S)$ contains the upper and lower tangents of any point of P. Suppose for a contradiction $(p_i, p_i^{\mathrm{up}}) \notin T$. Then, since T is a triangulation, there exists an edge $e \in T \setminus T^*(S)$ which properly intersects (p_i, p_i^{up}). Since T is an S-constrained, e cannot intersect any edge of S. Hence the definition of the upper tangent implies that the left endpoint of e is on the left side of p_i, which contradicts (1). Thus (2) holds.

Denote the right endpoints of $\delta_S(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$ 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). (2) and $e_s^* \notin S$ implies that $e_s \notin \delta_S(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$ and hence there exists the subscript k with $0 \leq k \leq m-1$ for which $(p_i, p_{i_k}) \prec e_s^* \prec (p_i, p_{i_{k+1}})$.

Consider the cone C with apex at p_i bounded by two consecutive edges (p_i, p_{i_k}) and $(p_i, p_{i_{k+1}})$, where C contains both p_{i_k} and $p_{i_{k+1}}$, and consider the convex hull H of $P_{i+1} \cap C$ inside C, (see Fig. 4). We focus on the convex chain (the sequence of the edges of H) between p_{i_k} and $p_{i_{k+1}}$ which is the visible part from p_i. Then, from the maximality of Construction 1, $T^*(S)$ contains this convex chain. This implies that $T^*(S)$ has a so-called *pseudo-triangle* formed by this convex chain and the two edges (p_i, p_{i_k}) and $(p_i, p_{i_{k+1}})$.

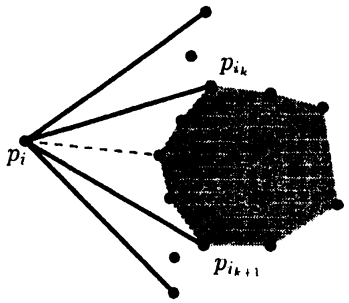

Figure 4: Proof of Lemma 2.1.

Since T is a triangulation but does not contain $e_s^* = (p_i, p_j)$, T must contain at least one edge $e' \notin T^*(S)$ that properly intersects e_s^*. Since p_j is a vertex of the pseudo-triangle mentioned above and there exists no point of P inside of this pseudo-triangle, any edge properly intersecting e_s^* must properly intersect at least one of (p_i, p_{i_k}) and $(p_i, p_{i_{k+1}})$. In addition, since T is an S-constrained triangulation, e' does not properly intersect any edge of $\delta_S(p_i)$, and consequently e' properly intersects at least (p_i, p_i^{up}) or $(p_i, p_i^{\mathrm{low}})$. This contradicts (2). □

An edge e in a triangulation T is called *flippable* if the 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. In [12], we obtained the following result.

Theorem 2.2. *Every S-constrained triangulation can be transformed into S-CLLT by flipping $O(n^2)$ edges $e \notin S$, each of which increases the lexicographical ordering.*

3 Minimal Representative Sets

Let $\mathcal{F}(S)$ be the collection of non-crossing edge set F on P satisfying $S \subseteq F$ and let $\mathcal{T}(S)$ be the collection of S-constrained triangulations on P. We define a relation $\sim$ on $\mathcal{F}$ as follows; for two

non-crossing edge sets F and F' (containing S), $F \sim F'$ holds if and only if $T^*(F) = T^*(F')$ holds. Let $[T] = \{F \in \mathcal{F}(S) \mid F \sim T\}$ for each $T \in \mathcal{T}(S)$. It is not difficult to see the following fact.

Lemma 3.1. *The relation $\sim$ is an equivalence relation on $\mathcal{F}(S)$. The collection $\{[T] \mid T \in \mathcal{T}(S)\}$ of all equivalence classes forms a partition of $\mathcal{F}(S)$.*

The following property of the function T^* is crucial for developing our general technique.

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

Proof. ("Only-if" part:) Assume, for a contradiction, that there exists $e = (p_i, p_j) \in E$ satisfying neither (i) nor (ii) of the statement when $T^*(F \setminus E) = T^*(F)$ holds. Notice 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 the two triangles of $T^*(F \setminus E)$ incident to e, and denote the two vertices appearing in 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 $T^*(F)$, which is a contradiction.

("If" part:) Let e be the upper or lower tangent of some point p_i with respect to F. Observe that flipping e in $T^*(F)$ decreases the lexicographical ordering of the edge list.

Also observe that, if e satisfies (i) and (ii), then every edge of $T^*(F) \setminus (F \setminus E)$ satisfies (i) and (ii). Hence $T^*(F)$ is a $(F \setminus E)$-constrained triangulation such that flipping any unconstreained edge (i.e. an edge of $T^*(F) \setminus (F \setminus E)$) does not increase the lexicographical ordering of the edge list. Theorem 2.2 says that any $(F \setminus E)$-constrained triangulation can be transformed to the $(F \setminus E)$-constrained lexicographically largest triangulation by diagonal flips of $e \notin F \setminus E$, each of which increases the lexicographical ordering, $T^*(F)$ is the $(F \setminus E)$-lexicographically largest triangulation. □

We say that an edge e of $F \in \mathcal{F}(S)$ is the *smallest* or *largest* one among F if it is the smallest edge, or respectively the largest edge, among F with respect to the edge ordering $\prec$. We remark that the upper tangent (and lower tangent, resp.) of p_i with respect to F is the smallest edge (and largest edge, resp.) in $\{(p_i, q) \in T^*(F) \mid q \in \{p_{i+1}, \ldots, p_n\} = P_{i+1}\}$. 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 Lemma 3.2, a unique minimal representative set for each $[T]$ is defined as follows.

Lemma 3.3. *Let T be a S-constrained triangulation on a given point set P and a non-crossing edge set S. 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) *$F^* \cup S \in [T]$ (i.e., $T^*(F^* \cup S) = T$), and*

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

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

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 $F \setminus (F^* \cup S)$, we obtain $T^*(F) = T^*(F \setminus (F \setminus (F^* \cup S))) = T^*(F^* \cup S) = T$. Let us consider the "only-if" part. It is obvious that $F \subseteq T$ holds if $F \in [T]$. Also, $S \subset F$ holds by the definition of $\mathcal{T}(S)$. Suppose F (with $S \subseteq F \subseteq T$) is a counterexample, that is $T^*(F) = T$ but $F^* \setminus F \neq \emptyset$. Since an edge $e = (p_i, p_j) \in F^* \setminus F$ is flippable in T and neither the smallest nor largest

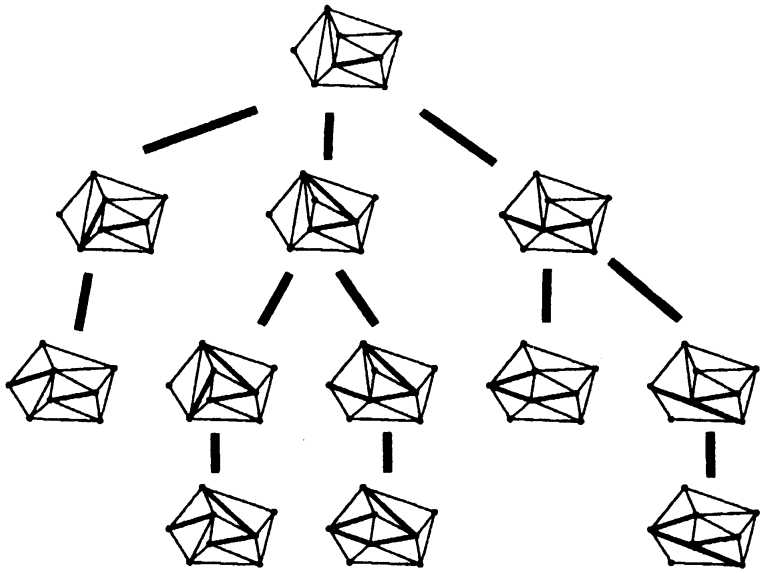

Figure 5: Search tree on the set of S-constrained triangulations obtained by the flipping algorithm, where each minimal representative set is drawn in bold.

edge among $\{(p_i, q) \in T \mid q \in P_{i+1}\}$ from the definition of F^*, $T = T^*(F) = T^*(F^* \setminus (F^* \setminus F)) \neq T^*(F^*)$ by Lemma 3.2, which contradicts $T = T^*(F^*)$. □

Thus, we call $F^* \cup S$ defined in Lemma 3.3 the *minimal representative set* of T, denoted by $R(T)$.

4 Enumerations of Edge-constrained Triangulations

This section discusses two enumeration algorithms for S-constrained triangulations on P.

4.1 A Flipping Algorithm

The first algorithm is based on diagonal flip operations, which have been proposed in [12]. In [12] we have proved that the lexicographical order of the unconstrained triangulations can be naturally extended to the edge-constrained case. The enumeration algorithm for the unconstrained case by Bespamyatnikh [7] that is based on the lexicographical order of unconstrained triangulations can be also extended to the edge-constrained case. For every S-constrained triangulation T with $T \neq T^*(S)$, let us define the *parent* of T as the triangulation obtained by flipping the smallest edge among $R(T) \setminus S$ with respect to the edge ordering $\prec$. Then, from the correctness of Theorem 2.2, these parent-child relations form the *search tree* of the S-constrained triangulations on P whose root is $T^*(S)$ (see Fig. 5).

It is known that the time complexity of the reverse search relies on the efficiency of finding the children of each object; in our case finding the children of each S-constrained triangulation. This task can be done by using the algorithm for the unconstrained case by just ignoring the edges of S in the algorithm by Bespamyatnikh [7] and thus we can obtain the algorithm that works in the same time complexity as that of the unconstrained case (see Section 4 of [7]). We obtained the following result [12]:

Theorem 4.1. *Let P be a set of n points in the plane. Then, all the S-constrained triangulations on P can be reported in $O(\log \log n)$ time per output graph with linear space.*

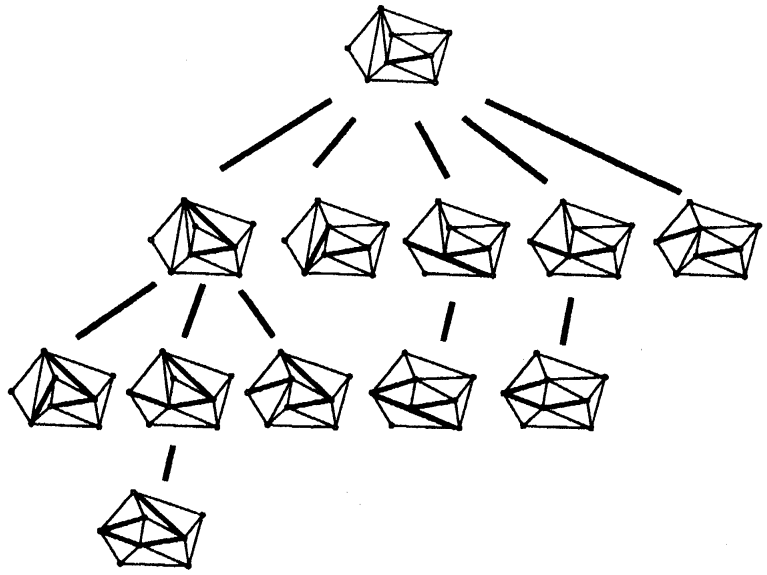

Figure 6: Search tree on the collection of S-constrained triangulations obtained by the edge insertion algorithm, where each minimal representative set is drawn in bold.

We shall refer to this algorithm as the *flipping algorithm* (for enumerating S-constrained triangulations).

4.2 An Edge Insertion Algorithm

Although the flipping algorithm enumerates all S-constrained triangulation quite efficiently, its search tree is not structured when looking it in term of the minimal representative sets. We will now develop another algorithm, whose search tree has a monotone structure with respect to the minimal representative sets such that $R(T) \subset R(T')$ holds for any triangulation T and its descendant T' (see Fig. 6). This algorithm is also based on the reverse search [3] whose search tree can be characterized by the root triangulation and the parent-child relation. Here we define the S-constrained triangulation $T^*(S)$ as the root triangulation. Hence, the minimal representative set of the root triangulation is equal to S. For each non-root S-constrained triangulation T, the parent of T is defined as $T^*(R(T) \setminus \{e\})$ with the smallest edge e among $R(T)$ with respect to the edge ordering $\prec$. The correctness of our parent-child relation follows from the next lemma, which is an application of [13, Lemma 4.1].

Lemma 4.2. *Let T be an S-constrained triangulation with $R(T) \neq S$. Then, for any $e \in R(T) \setminus S$, the minimal representative set of $T^*(R(T) \setminus \{e\})$ is $R(T) \setminus \{e\}$.*

From Lemma 4.2, $R(T) \subset R(T')$ holds for any S-constrained triangulation T and its descendant T'. Thus our definition of the parent-child relation correctly induces a rooted search tree on the collection of all S-constrained triangulations. The algorithm traces this search tree in depth-first manner. We call this new algorithm the *edge insertion algorithm* for (enumerating) S-constrained triangulations. An example of the new search tree is depicted in Fig. 6.

Let us show a sketch of the time complexity analysis 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. Since we can reconstruct $T^*(R(T) \cup \{e\})$ from T in $O(n)$ time [13], we can check whether $T^*(R(T) \cup \{e\})$ is actually child of T in $O(n)$ time for each $e \in K_n$. Thus, we obtain the following result.

Theorem 4.3. *Let P be a set of n points. Then, the edge insertion algorithm enumerates all the S-constrained triangulations on P in $O(n^3)$ time per output graph without repetition.*

5 General Techniques

5.1 Algorithm 1

Our enumeration technique for S-constrained non-crossing geometric graph can be easily described as follows.

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

Phase1: Enumerate all S-constrained triangulations for a given point set P and a non-crossing edge set S based on the fast enumeration algorithm given in Section 4.1.

Phase2: Every time a new S-constrained triangulation T is found, enumerate all graphs $G \in \mathcal{NGG}$ such that $R(T) \subseteq G \subseteq T$.

Figure 5 shows an example of the enumeration of triangulations and the minimal representative sets. The correctness of Algorithm 1 easily follows from Lemmas 3.1 and 3.3. In fact, consider an arbitrary S-constrained non-crossing geometric graph $G \in \mathcal{NGG}$ to be enumerated. Then, $T = T^*(G)$ is uniquely determined. This implies $G \in [T]$ and $G \notin [T']$ for any S-constrained triangulation $T' \neq T$ by Lemma 3.1. Since $G \in [T]$ implies $R(T) \subseteq G \subseteq T$ by Lemma 3.3, Phase 2 of Algorithm 1 for the S-constrained triangulation T enumerates G by the (assumed) oracle. Thus, any G is enumerated exactly once in Phase 2 for $T = T^*(G)$.

Let us analyze the time complexity of Algorithm 1. The flipping algorithm for enumerating S-constrained triangulations is based on the reverse search, whose search graph is defined in such a way that two S-constrained triangulations are connected if and only if they can be transformed to each other by a diagonal flip (see Fig. 5). Due to the locality of diagonal flips, we can easily show the followings: Let T_1 and T_2 be two triangulations for which T_2 is obtained from T_1 by a diagonal flip of the edge f. Then, the size of the symmetric difference between $R(T_1)$ and $R(T_2)$ is constant. More specifically, only the four edges of the two triangle faces incident to f are involved in the symmetric difference.

Thus, during Algorithm 1, 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 is attached to each edge to indicate whether it is in the minimal representative set or not. Since the edge insertion algorithm works in $O(\log \log n)$ time per output, we eventually obtained the following result:

Theorem 5.1. *Let $\mathcal{C}$ be the graph class obtained by relaxing the non-crossing constraint from $\mathcal{NGG}$. Suppose there exists an algorithm for enumerating all $R(T)$-constrained graphs of $\mathcal{C}$ in a triangulation T without repetitions in time $t_{\mathcal{C}}$ per output graph with preprocessing time $t_{\mathcal{C},\mathrm{pre}}$. Then, all S-constrained graphs of $\mathcal{NGG}$ on a given point set P can be enumerated without repetitions in $O((\log \log n + t_{\mathcal{C},\mathrm{pre}}) \cdot \mathsf{tri}(P,S) + t_{\mathcal{C}} \cdot \mathsf{ngg}(P,S))$ time, where $\mathsf{tri}(P,S)$ and $\mathsf{ngg}(P,S)$ denote the total number of S-constrained triangulations and $\mathcal{NGG}$ on P, respectively.*

5.2 Algorithm 2

We know that the flipping algorithm enumerates all triangulations efficiently, but its search tree is not nicely structured when we focus on the minimal representative sets (see Fig. 5). Namely, for two triangulations T and T' for which T is a parent of T' in the search tree, T' may miss some representative edge that appears in T. Consider, for example, the enumeration of non-crossing matchings. In Phase 2 of Algorithm 1 for a triangulation T, the algorithm outputs no $R(T)$-constrained non-crossing matching if there is a vertex incident to more than one edge of $R(T)$. However, since some descendant triangulation T' of T may not have a vertex which is incident to more than one edge of $R(T')$, T' may contain an $R(T')$-constrained non-crossing matching and thus we cannot skip the enumeration of T and its descendants.

Using the monotonicity of the search tree obtained by the edge insertion algorithm, we can efficiently enumerate only the minimal representative sets possessing the specified property, which allows us to skip the output of unnecessary triangulations. Let us explain this idea more formally. Let $\mathcal{I}$ be a subset of $\mathcal{F}(S)$ 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 $F \in \mathcal{F}(S)$ is called *independent edge set* or *independent* (with respect to $\mathcal{I}$) if $F \in \mathcal{I}$. 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 $\mathcal{NGG}$. The next proposed technique is formally described as follows:

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

Phase 1: Execute the edge insertion algorithm starting from $T^*(S)$ as described in Section 4.2 to enumerate S-constrained triangulations.

Phase 2: Every time a new S-constrained triangulation T is found, check whether $R(T)$ is independent or not. If $R(T)$ is dependent, skip the enumeration of all the descendants of T.

Phase 3: 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 5.2. *Let $\mathcal{I}$ be the collection of independent edge sets of $\mathcal{F}(S)$. 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 the 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 all T with $R(T) \in \mathcal{I}$ 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. Lemma 3.3 states $R(T^*(G)) \subseteq G \subseteq T^*(G)$. Since $G \in \mathcal{I}$ holds by (I3), $R(T^*(G)) \in \mathcal{I}$ follows by (I2). Thus G is enumerated in Phase 3 for $T^*(G)$. □

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 $e \in K_n$. Let $\mathcal{I}_{\mathrm{rep}} \subseteq \mathcal{I}$ 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 and 2 is $O(n^3 + n^2 \cdot t_{\mathrm{check}} \cdot |\mathcal{I}_{\mathrm{rep}}|)$ since there exist $O(n^2)$ children for each triangulation on the search tree and from Theorem 4.3. Hence, using the notations C, t_C and $t_{C,\mathrm{pre}}$ defined in Theorem 5.1, we obtain the following result:

Theorem 5.3. *Algorithm 2 enumerates all the elements of $\mathcal{NGG}$ on a given point set P without repetitions in $O((n^3 + n^2 \cdot t_{\mathrm{check}} + t_{C,\mathrm{pre}}) \cdot |\mathcal{I}_{\mathrm{rep}}| + t_C \cdot \mathsf{ngg}(P,S))$ time. Moreover, the time complexity is bounded by $O((n^3 + n^2 \cdot t_{\mathrm{check}} + t_{C,\mathrm{pre}} + t_C) \cdot \mathsf{ngg}(P,S))$, which is polynomial on average, if $|\mathcal{I}_{\mathrm{rep}}| \leq \mathsf{ngg}(P,S)$ holds.*

6 Applications

6.1 Edge-constrained Non-crossing Spanning Trees

We show here how to apply Algorithm 1 to the enumeration of S-constrained non-crossing spanning trees on a given point set. Based on the framework given above, the algorithm proceeds as follows:

Phase1: Enumerate all S-constrained triangulations for a given point set P based on the flipping algorithm.

Phase2: Every time a new S-constrained triangulation T is found, enumerate all spanning trees graphs of the graph obtained from T by contracting $R(T)$ by using the algorithm developed by Kapoor and Ramesh [10] or Shioura et al. [18, 19].

We remark 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, we used the algorithm for enumerating all spanning trees on a given undirected graph developed by Kapoor and Ramesh [10] or Shioura et al. [18, 19]. These algorithms can enumerate all the spanning trees of a given graph in $O(1)$ time per output graph with $O(n+m)$ preprocessing time, where n and m denote the number of vertices and edges of the given graph. Thus, from Theorem 5.1, the following result is derived: The set of S-constrained non-crossing spanning trees on P can be enumerated in $O(n \cdot \mathsf{tri}(P,S) + \mathsf{st}(P,S))$ time, where $\mathsf{st}(P,S)$ denotes the total number of S-constrained non-crossing spanning trees on P.

6.2 Edge-constrained Non-crossing Spanning Connected Graphs

We show here how Algorithm 1 can be applied to the enumeration of edge-constrained non-crossing spanning connected graphs. To efficiently perform Phase 2 of Algorithm 1, we need an algorithm for enumerating all the spanning connected subgraphs of a given graph. Although, to the best of our knowledge, previously there was no efficient enumeration algorithm for this graph class, we observe that they can be enumerated in $O(1)$ time per output with $O(n)$ preprocessing time with a slight modification of an algorithm by Uno [20], which was developed for the enumeration of all bases of a matroid (including spanning trees).

The edge constraint can be treated easily by edge contraction, and thus all the $R(T)$-constrained spanning connected subgraphs of T can be enumerated in $t_C = O(1)$ time per output with $t_{C,\mathrm{pre}} = O(n)$. Combined with Theorem 5.1, we found that Algorithm 1 enumerates all the non-crossing spanning connected graphs in $O(n \cdot \mathsf{tri}(P,S) + \mathsf{cg}(P,S))$ time, where $\mathsf{cg}(P,S)$ denotes the total number of S-constrained non-crossing spanning connected graphs on P. Moreover, since every S-constrained triangulation is a S-constrained non-crossing spanning connected graphs, we have $\mathsf{cg}(P,S) \geq \mathsf{tri}(P,S)$. Thus, we conclude that the set of non-crossing spanning connected graphs on P can be enumerated in $O(n \cdot \mathsf{cg}(P,S))$ time.

6.3 Enumerating Edge-constrained Plane Straight-line Graphs

For any $F \subseteq T \setminus R(T)$, $F \cup R(T)$ is a plane straight-line graph containing $R(T)$. Hence, by enumerating (the symmetric differences of) all subsets of $T \setminus R(T)$, we can obtain all $R(T)$-constrained plane straight-line graphs in T. Enumerating all subsets of $T \setminus R(T)$ is equivalent to generating all $|T \setminus R(T)|$-bit binary numbers with $O(n)$ preprocessing time, which can be done in constant time per output (see e.g., [16]). Algorithm 1 thus enumerates all the S-constrained plane straight-line graphs in $O(n \cdot \mathsf{tri}(P,S) + \mathsf{pg}(P,S))$ time, where $\mathsf{pg}(P,S)$ denotes the total number of S-constrained straight-line graphs on P. Since $\mathsf{tri}(P,S) \leq \mathsf{cg}(P,S) \leq \mathsf{pg}(P,S)$ holds, we obtain the following result: The set of S-constrained plane straight-line graphs on P can be enumerated in $O(n \cdot \mathsf{pg}(P,S))$ time.

6.4 Edge-constrained 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 the matchings in a (non-geometric) bipartite graph [22] in Phase 2 of Algorithm 1 or in Phase 3 of Algorithm 2, which needs $t_C = O(n)$ time per output with $t_{C,\mathrm{pre}} = O(n^{3/2})$ preprocessing time (if the edge cardinality of a given graph is $O(n)$). Hence, by Theorem 5.1, Algorithm 1 enumerates all the S-constrained non-crossing red-and-blue matchings in $O(n^{3/2} \cdot \mathsf{tri}(P,S) + n \cdot \mathsf{rbm}(P,S))$ time, where $\mathsf{rbm}(P,S)$ is the total number of S-constrained non-crossing red-and-blue matchings on P.

Algorithm 2 can enumerate all the red-and-blue matchings efficiently if we define $\mathcal{I}$ as the collection of $F \in \mathcal{F}(S)$ such that no two edges of F are incident to a vertex and no edge of F connects points of the same color. Notice that every independent minimal representative set is also an S-constrained non-crossing red-and-blue matching, which implies $|\mathcal{I}_{\mathrm{rep}}| \leq \mathsf{rbm}(P,S)$. The independence of each non-crossing edge set is trivially checked in $t_{\mathrm{check}} = O(1)$ time, and thus Algorithm 2 works in $O(n^3 \cdot \mathsf{rbm}(P,S))$ time by Theorem 5.3.

6.4.1 Edge-constrained 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 (or $k-1$ edges) from the graph. Since it can be checked in a polynomial time Q_k whether a given (non-geometric) graph is k-vertex connected (or k-edge connected) or not, according to the branch-and-bound technique discussed in the introduction, we can enumerate S-constrained k-vertex connected (or k-edge connected) subgraphs in $t_C = O(mQ_k)$ time per output with $t_{C,\mathrm{pre}} = O(n+m+Q_k)$ preprocessing time, where m denotes the number of edges in a subgraph. Thus, by using this algorithm in Phase 2, Algorithm 1 enumerates all S-constrained non-crossing k-vertex (or k-edge) connected graphs in $O((n+Q_k) \cdot \mathsf{tri}(P,S) + nQ_k \cdot \mathsf{cg}_k(P,S))$ time, where $\mathsf{cg}_k(P,S)$ denotes the total number of S-constrained non-crossing k-vertex (or k-edge) connected graphs on P.

In particular, it is known that 2-vertex (or 2-edge) connectivity of a graph can be checked in linear time (see, e.g., [17, Chapter 15.2b]). Moreover, $\mathsf{tri}(P,S) \leq \mathsf{cg}_2(P,S)$ holds for every point set P since every S-constrained triangulation is also an S-constrained non-crossing 2-vertex (or 2-edge) connected graph on P. Algorithm 1 hence enumerates all the S-constrained non-crossing 2-vertex (or 2-edge) connected graphs in $O(n^2\mathsf{cg}_2(P,S))$ time.

7 Concluding Remarks

We have proposed two enumeration techniques for edge-constrained non-crossing geometric graphs, based on our recent results on non-crossing geometric graphs [12, 13]. As an open problem the comparing the number of non-crossing spanning trees with that of triangulations might be theoretically interesting. It is known that $\mathsf{st}(P)$ becomes minimum when P is in a convex position. On the other hand, $\mathsf{tri}(P)$ is not always minimum for convex positions (see [2]). Furthermore, the number of $\mathsf{st}(P)$ in the convex position is known to be $\Theta(6.75^n)$ [9] relative to the number of triangulations, which is $\Theta(4^n)$, where we ignore polynomial factors. Hence, we strongly conjecture that there exists a constant c (> 1) for which $c^n \cdot \mathsf{tri}(P) \leq \mathsf{st}(P)$ holds for *every* $P \subset \mathbb{R}^2$ of n points.

Another open problem, which is of considerably practical importance, is to efficiently generate all the non-crossing spanning trees on a given point set that do not contain a given edge set. This problem is challenging because it is known that determining if a geometric graph contains a non-crossing spanning tree is NP-complete [14].

Acknowledgment

The first author is supported by the project *New Horizons in Computing*, Grant-in-Aid for Scientific Research on Priority Areas, MEXT Japan, and by Grant-in-Aid for Scientific Research (C), JSPS. The second author is supported by Grant-in-Aid for JSPS Research Fellowship for Young Scientists.

References

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

[2] O. Aichholzer, T. Hackl, C. Huemer, F. Hurtado, H. Krasser, and B. Vogtenhuber, On the number of plane geometric graphs. *Graphs and Combinatorics*, 23(1), 67–84 (2007).

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

[4] D. Avis, N. Katoh, M. Ohsaki, I. Streinu and S. Tanigawa, Enumerating non-crossing minimally rigid frameworks, *Graphs and Combinatorics*, 23(1), 117–134 (2007).

[5] D. Avis, N. Katoh, M. Ohsaki, I. Streinu and S. Tanigawa, Enumerating constrained non-crossing minimally rigid frameworks. *Discrete Comput. Geom.*, 40(1), 31–46 (2008).

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

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

[8] 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).

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

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

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

[12] N. Katoh and S. Tanigawa, Enumerating edge-constrained triangulations and edge-constrained non-crossing spanning trees. *Disc. Appl. Math.* (to appear).

[13] N. Katoh and S. Tanigawa. Fast enumeration algorithms for non-crossing geometric graphs. *Proc. 24th ACM Symposium on Computational Geometry*, 328–337, 2008. To appear in *Discrete Comput. Geom.*.

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

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

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

[17] A. Schrijver, Combinatorial Optimization. Polyhedra and Efficiency. Springer-Verlag, Heidelberg (2003).

[18] 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).

[19] 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).

[20] 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)*, pp. 54–63, Lecture Notes in Computer Science, vol. 1627, Springer-Verlag (1999).

[21] T. Uno, A new approach for speeding up enumeration algorithms. In *Proc. International Symposium on Algorithm and Computation (ISAAC '98)*, pp. 287–296, Lecture Notes in Computer Science, vol. 1533, Springer-Verlag (1998).

[22] 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)*, pp. 92–101, Lecture Notes in Computer Science, vol. 1350, Springer-Verlag (1997).