Enumerating Constrained Non-crossing Minimally Rigid Frameworks

David Avis [1] Naoki Katoh [2] Makoto Ohsaki [2]
Ileana Streinu [3] Shin-ichi Tanigawa [2]

April 25, 2007

Abstract

In this paper we present an algorithm for enumerating without repetitions all the non-crossing generically minimally rigid bar-and-joint frameworks under edge constraints, which we call constrained non-crossing Laman frameworks, on a given set of n points in the plane. Our algorithm is based on the reverse search paradigm of Avis and Fukuda. It generates each output graph in $O(n^4)$ time and $O(n)$ space, or, with a slightly different implementation, in $O(n^3)$ time and $O(n^2)$ space. In particular, we obtain that the set of all the constrained non-crossing Laman frameworks on a given point set is connected by flips which preserve the Laman property.

Key words: geometric enumeration; rigidity; constrained non-crossing minimally rigid frameworks; constrained Delaunay triangulation.

1 Introduction

Let G be a graph with vertices $\{1, \ldots, n\}$ and m edges. G is a *Laman graph* if $m = 2n - 3$ and every subset of $n' \leq n$ vertices spans at most $2n' - 3$ edges. An embedding of the graph $G(P)$ on a set of points $P = \{p_1, \cdots, p_n\} \subset R^2$ is a mapping of the vertices to points in the Euclidean plane $i \mapsto p_i \in P$. The edges ij of G are mapped to straight line segments p_ip_j. An embedding is *planar* or *non-crossing* if no pair of segments p_ip_j and p_kp_l corresponding to non-adjacent edges $i, j \notin \{k, l\}$ have a point in common. Where no ambiguity arises, we simply denote a vertex p_i by i and an edge p_ip_j by ij.

An embedded Laman graph on a planar point set is called a *Laman framework*. It has the special property of being *minimally rigid* when viewed as a bar-and-joint framework with fixed edge-lengths, under some rather weak conditions on the point set [13, 19]. This motivates the tremendous interest in their properties. Let F be a set of non-crossing edges (bars) on P. A Laman framework containing F is called *F-constrained*. In this paper we

[1]School of Computer Science, McGill University, Canada.

[2]Department of Architecture and Architectural Engineering, Kyoto University Katsura, Nishikyo-ku, Kyoto 615-8450 Japan, {ohsaki,naoki,is.tanigawa}@archi.kyoto-u.ac.jp. Supported by JSPS Grant-in-Aid for Scientific Research on priority areas of New Horizons in Computing.

[3]Dept. of Comp. Science, Smith College, Northampton, MA 01063, USA, streinu@cs.smith.edu. Supported by NSF grant CCF-0430990 and NSF-DARPA CARGO CCR-0310661.

(a) (b)

Figure 1: (a)Minimum pseudo-triangulation. (b)Non-crossing Laman framework.

give an algorithm for enumerating all the *F-constrained non-crossing Laman frameworks* on a given point set P.

A *pseudo-triangle* is a simple polygon with exactly three convex vertices. A *pseudo-triangulation* is the partition of the convex hull of a planar point set P into the interior disjoint pseudo-triangles, the vertices of which are points in P. It is known that a pseudo-triangulation with minimum number of edges, called a *minimum pseudo-triangulation* or a *pointed pseudo-triangulation*, is a non-crossing Laman framework [23]. However a minimum pseudo-triangulation contains all the edges of the convex hull of the underlying point set, whereas a non-crossing Laman framework need not as illustrated in Figure 1. Bereg showed that minimum pseudo-triangulations are connected via *simple flips*, in which the removal of any non-convex-hull edge leads to the choice of a *unique* other edge that can replace it, in order to maintain the pseudo-triangulation property [8]. This leads to the definition of a graph whose vertices are minimum pseudo-triangulations and whose edges are simple flips. He showed that this graph is connected and showed how to apply the reverse search technique to generate all of its vertices (minimum pseudo-triangulations). Bereg's efficient algorithm makes use of specific properties of minimum pseudo-triangulations which do not extend to arbitrary non-crossing Laman frameworks. In particular, remove-add flips are *not* unique, relative to the removed edge, in the case of a non-crossing Laman framework.

In our previous paper [5] we studied a graph whose nodes were non-crossing Laman frameworks on a given point set. Two vertices are adjacent in the graph if one can be obtained from the other by a single edge insertion and deletion in the corresponding non-crossing Laman frameworks. We showed that this graph is connected. The rather involved proof relies on some properties of the one-degree-of-freedom mechanisms that are obtained by removing a single edge from a Laman framework.

Unfortunately, it is not feasible to generate all Laman frameworks of practical interest due to the huge output size. However, certain engineering considerations, discussed later in this section, allow us to limit ourselves to Laman frameworks containing a given set F of non-crossing edges. It is not at all clear how to adapt our earlier proof to apply to these F-constrained Laman frameworks.

In this paper we use both matroids and triangulations as important tools to solve this problem. Firstly we note that although Laman graphs form the bases of a matroid defined on the edges of a given base graph (see, e.g., [13, 25]) this is not true in general for non-crossing Laman frameworks on a point set P. In fact for the two non-crossing Laman frameworks L_1 and L_2 depicted in Figure 2 (a) and (b) there exists no edge to insert into $L_1 - \{ab\}$ from $L_2 - L_1$ that maintains the non-crossing property. However, by choosing a non-crossing

(a) (b)

Figure 2: Non-crossing Laman frameworks need not form the bases of a matroid.

base graph, all subgraphs are automatically non-crossing, and hence the non-crossing Laman frameworks which are subgraphs of this graph do form the bases of a matroid. Triangulations are natural candidates for such base graphs, are connected by flip operations, and handle rather naturally the condition of being F-constrained.

The reverse search enumeration technique of Avis and Fukuda [3, 4] has been successfully applied to a variety of combinatorial and geometric enumeration problems. The necessary ingredients to use the method are an implicitly described connected graph on the objects to be generated, and an implicitly defined spanning tree in this graph. In this paper we use triangulations and matroids to supply these ingredients for the problem of generating constrained Laman frameworks. After proving the correctness of our approach, we give an implementation based on reverse search that allows the enumeration without repetitions of all the F-constrained non-crossing Laman frameworks on n points in $O(n^4)$ time and $O(n)$ space per output framework. A slightly different implementation yields $O(n^3)$ time and $O(n^2)$ space per output framework. For the unconstrained case, also using reverse search but with very different parent function, we obtain the same time and space complexity results that we obtained in [5]. The method presented here is, however, considerably simpler.

In our implementation we make use of the pebble game algorithm of Jacobs and Hendrickson [14] for 2-dimensional rigidity, see also [9]. Our complexity analysis relies on recent results, due to Lee, Streinu and Theran [20, 21], regarding the complexity of finding and maintaining rigid components during the pebble game algorithm. Indeed, the time-space trade-off of our algorithm is inherited from [21].

In the remainder of this section, we briefly describe how this problem came to our attention via the work of the third author. The motivating application described here is not essential for understanding the results of the paper, but may be of interest to readers interested in structural rigidity. Graph theoretical approaches are widely used in *structural mechanics* [17], where the edges and vertices in the graph represent the bars and rotation-free joints of a structure called a *truss*. It is well-known that the stiffest truss under static loads is statically determinate that is equivalent to a Laman graph [7].

Another bar-joint system, which is widely used in industrial applications, is a link mechanism that is unstable and generates large deformations, or changes in the direction of the nodal displacement. Applications of link mechanisms can be found in, e.g., automobile suspensions, robot hands, umbrellas, crankshafts, etc. Kawamoto et al.[18] presented a method which used the enumeration of planar graphs to find an optimal mechanism. However, their method was developed for their specific problem, and no general approach was given.

Recently, a new type of mechanism, called a compliant mechanism, has been developed

and applied mainly in the field of micro-mechanics. A compliant mechanism has flexible parts, which are not present in conventional unstable mechanisms, to stabilize the structure. Although a compliant mechanism is usually modeled as a continuum with elastic joints, it is possible to generate a similar mechanism by using a bar-joint system. Ohsaki and Nishiwaki [22] presented a method for generating compliant flexible bar-joint mechanisms using a nonlinear programming approach, and found that the optimal structure is statically determinate, i.e., minimally rigid. They utilized snapthrough behavior to generate multi-stable mechanisms that have multiple self-equilibrium states. Such a mechanism can be used as a switching device, robot hand, gripper, deployable structure, etc. In their method, the optimal locations of bars and joints are found from a highly connected initial structure that has bars between all pairs of the nodes whose distances are small enough. However, due to high nonlinearity of the analysis and optimization problems, the nonlinear programming problem should be solved many times starting from different initial solutions to obtain a few types of mechanisms.

Since the compliant bar-joint mechanism is usually *statically determinate*, the optimization problem can be solved easily if the design space is limited to statically determinate structures. Combining an implementation of our earlier method for generating unconstrained Laman frameworks [5] with this nonlinear programming approach, we could obtain many new compliant mechanisms with up to 10 joints [16]. However the number of Laman frameworks grows too rapidly to allow a complete enumeration for significantly larger examples.

In view of practical requirements, the optimal structure should not have intersecting members, and some *pre-selected members* should always exist. Therefore, the computational cost could be much reduced if the candidate set of statically determinate non-crossing trusses (non-crossing Laman frameworks) is restricted to those containing pre-selected members. Thus it is desirable to enumerate all Laman frameworks which contain a given set of specified edges.

2 Preliminaries

Let L be a non-crossing Laman framework on a given point set P. A *mechanism* is a flexible framework obtained by removing one or more edges from a Laman framework. Its *number of degrees of freedom* or *dof*'s, is the number of removed edges. We will encounter mostly *one-degree-of-freedom (1dof) mechanisms*, which arise from a Laman framework by the removal of one edge. In particular, a mechanism with k dofs has exactly $2n-3-k$ edges, and each subset of n' vertices spans at most $2n'-3$ edges. A subset of some n' vertices spanning exactly $2n'-3$ edges is called a *rigid block*. A maximal rigid block is called a *rigid component*.

An important tool in our work is the *generic rigidity matroid*, also called the *Laman matroid*, which can be defined on the edges of the complete graph K_n, see [13, 25]. A subset T of edges of K_n is independent in the matroid if for every subset $S \subseteq T$ we have $|S| \leq 2|V(S)| - 3$, where $V(S)$ is the subset of vertices spanned by the edges in S. The bases are those independent sets which contain exactly $2n-3$ edges, i.e., the Laman graphs on n vertices. Two bases L_1 and L_2 are connected by a *flip* if their edge sets agree on $2n-4$ elements. The flip is given by the pair of edges (e_1, e_2) such that $e_1 \in L_1 - L_2$, $e_2 \in L_2 - L_1$. In other words, we have $L_2 = L_1 - \{e_1\} \cup \{e_2\}$, which for simplicity we will write as $L_2 = L_1 - e_1 + e_2$ throughout the paper.

The definition of Laman matroid may be generalized by replacing the base graph K_n by any graph G. In order that the matroid be non-empty, G must contain at least one Laman subgraph. We may further extend the definition by fixing an independent set F of edges of G. The independent edge sets of G that contain F also form a matroid, which we will call the *F-constrained Laman matroid.*

Let G be an n vertex graph that contains a Laman subgraph, and let F be an independent set of edges in G. Using flips, we can define a new graph whose nodes are the bases of the F-constrained Laman matroid defined on G. Its edges correspond to bases connected by flips. It follows from the properties of a matroid that this graph is connected. But a priori, the subset of *F-constrained non-crossing Laman frameworks* may not necessarily be. We will prove this later in Section 3.

Reverse search is a memory efficient method for visiting all the nodes of a connected graph that can be defined implicitly by an adjacency oracle. It can be used whenever a spanning tree of the graph can be defined implicitly by a *parent* function. This function is defined for each vertex of the graph except a pre-specified *root*. Iterating the parent function leads to a path to the root from any other vertex in the graph. The set of such paths defines a spanning tree, known as the *search tree*.

3 Constrained Non-crossing Laman Frameworks

Let T be a triangulation on a given set of n points P in the plane, containing k triangles. The *angle vector* of T is the vector of $3k$ interior angles sorted into non-decreasing order. We say that T is an *F-constrained triangulation*, denoted $T(F)$, if it contains a given set of non-crossing edges F. Many facts about F-constrained triangulations are contained in the survey by Bern and Eppstein [10]. If F is an independent set in the Laman matroid on K_n, then a Laman framework on P containing F is called *F-constrained*, and is denoted $L(F)$. The following lemma, a well known fact about the Laman matroid, follows from rigidity considerations (see, e.g.[25]).

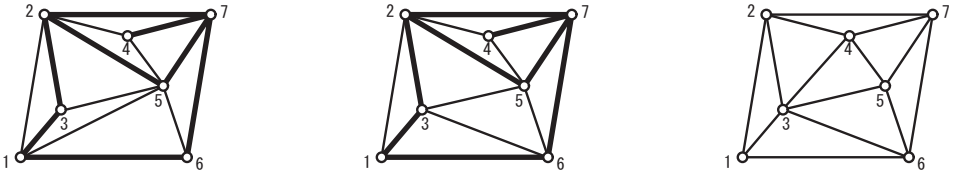
Lemma 1. *Let F be a non-crossing edge set on P that is an independent set in the Laman matroid on K_n. Every F-constrained triangulation $T(F)$ on P contains an F-constrained Laman framework $L(F)$.*

The *F-constrained Delaunay triangulation* plays an important role for developping our results, which we define by means of *legal (illegal) edge* and *D-flip* as follows:

Definition 1. (*Legal edge*) *Let T be a triangulation on a point set P. An edge ac that bounds two triangles whose edges are in T, say abc and acd, is a legal edge (with respect to T) if the circumcircle of abc does not contain d in its interior.*

Definition 2. (*D-flip*) *Let $T(F)$ be an F-constrained triangulation on a non-crossing edge set F. Let ac be an illegal edge (with respect to $T(F)$) that is not in F and is the diagonal of a convex quadrilateral $abcd$ whose edges are contained in $T(F)$. The replacement of edge ac by edge bd in $T(F)$ is called a D-flip.*

Definition 3. (*F-constrained Delaunay Triangulation*) *Let $T(F)$ be an F-constrained triangulation on a non-crossing edge set F. $T(F)$ is an F-constrained Delaunay Triangulation, denoted $DT(F)$ if it admits no D-flips. Equivalently, all edges in $T(F) - F$ are legal.*



(a) $T(F)$ (b) $DT(F)$ (c) $DT(\emptyset)$

Figure 3: (a)F-constrained triangulation, (b)F-constrained Delaunay triangulation and (c)Unconstrained Delaunay triangulation, where the bold edges denote the edges of F.

In Figure 3 (a), (b) and (c), we illustrate examples of an F-constrained triangulation $T(F)$, the F-constrained Delaunay triangulation $DT(F)$ and the unconstrained Delaunay triangulation $DT(\emptyset)$, respectively, where $F = \{13, 16, 23, 25, 27, 47, 57, 67\}$. We observe that $T(F)$ and $DT(F)$ have illegal edges $\{15, 25\}$ and $\{25\}$, respectively, and $DT(\emptyset)$ has no illegal edge. Replacing edge $\{15\}$ in $T(F)$ by edge $\{36\}$ is a D-flip.

Fact 1. *An F-constrained triangulation $T(F)$ can be converted to $DT(F)$ by at most $O(n^2)$ D-flips, taken in any order. (Lemma 4, [10]).*

If P has four or more co-circular points, using a linear transformation as described in [6], we may transform P into a point set $\bar{P}$ with a unique $DT(F)$ so that P and $\bar{P}$ have the same number of non-crossing Laman frameworks since the transformation does not change the relative order with respect to x- and y- coordinates among any three points. We will assume in what follows that P has a unique $DT(F)$.

Two points a and b are *visible* (with respect to F) if no edge of F properly intersects the segment ab. ab is *visible* to point c (with respect to F) if the triangle abc is not properly intersected by an edge of F. Then the following fact gives another characterization of F-constrained Delaunay triangulation.

Fact 2. *Let F be a non-crossing edge set on P. An F-constrained Delaunay Triangulation contains the edge ab between points a and b in P if and only if a is visible to b, and some circle through a and b contains no point of P visible to segment ab. We call ab a Delaunay-edge (with respect to F). (Definition 1, [10])*

Let $H(F)$ be a non-crossing edge set on P containing F, and let us consider $H(F)$-constrained Delaunay triangulation $DT(H(F))$. We observe the following facts that will be often utilized later. First note that all illegal edges in $DT(H(F))$ are contained in $H(F)$. This is because all edges of $DT(H(F)) - H(F)$ are legal edges by Definiton 3. We also notice that if an edge $e \in H(F)$ is not a legal edge in $DT(H(F))$, we have $DT(H(F)) = DT(H(F) - e)$ since all edges of $DT(H(F)) - (H(F) - e)$ are legal in $DT(H(F))$. On the other hand $DT(H(F) - e)$ cannot contain e if e is illegal. In fact $DT(H(F) - e)$ can be obtained from $DT(H(F))$ after performing at least one D-flip. Note in addition that a D-flip increases the angle vector lexicographically. This can be used to prove the following.

Fact 3. *$DT(F)$ has the lexicographically maximum angle vector of all F-constrained triangulations on P. (Theorem 1, [10]).*

Now let us consider non-crossing Laman frameworks.

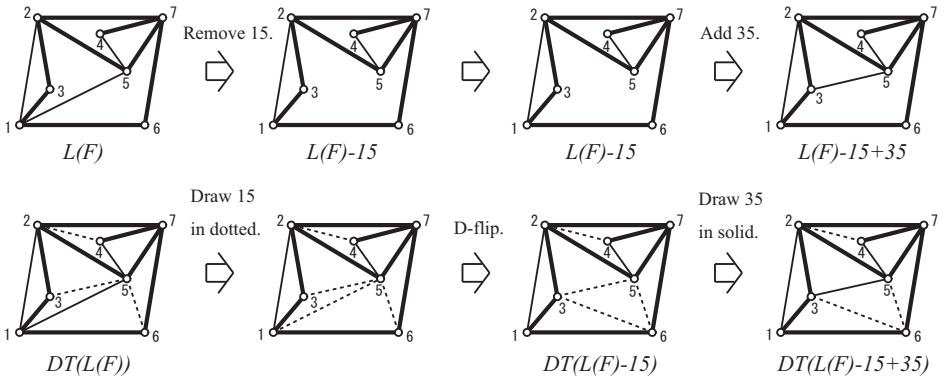


Figure 4: An example of L-flip described in the proof of Theorem 1. The bold lines represent $F = \{13, 16, 23, 25, 27, 47, 57, 67\}$, and the dotted lines represent additional edges for the constrained Delaunay triangulations.

Definition 4. *An F-constrained Delaunay Laman Framework, DL(F), is an F-constrained Laman framework that is a subset of $DT(F)$.*

Note that, unlike $DT(F)$, $DL(F)$ is not uniquely defined in general. In the sequel we will often make use of a constraining edge set F that is either a Laman framework L or a 1dof mechanism $L - e$.

Definition 5. *(L-flip) An L-flip is an edge insertion and deletion that takes a Laman framework L to a new Laman framework L'.*

Theorem 1. *Every F-constrained non-crossing Laman framework $L(F)$ can be transformed to a $DL(F)$ by at most $O(n^2)$ L-flips.*

Proof. Construct the $L(F)$-constrained Delaunay triangulation $T = DT(L(F))$. If $T = DT(F)$, then in fact $L(F)$ is a $DL(F)$ and we are done. Otherwise T contains some illegal edge $ac \notin F$ by Definition 3. Now all edges in $T - L(F)$ are legal edges with respect to T, so ac must be an edge of $L(F) - F$. Consider now the constrained Delaunay triangulation $DT(L(F)-ac)$ which contains $L(F)-ac$. We apply Lemma 1 with F and $T(F)$ in the lemma replaced by $L(F)-ac$ and $DT(L(F)-ac)$ respectively. By the lemma, $DT(L(F)-ac)$ contains a $(L(F)-ac)$-constrained Laman framework L'. Since L' contains $L(F)-ac$ it must contain one additional edge, say st, and so $L' = L(F) - ac + st$. In other words L' is obtained from $L(F)$ by an L-flip. Observe that L' is F-constrained since $ac \notin F$.

Now we can construct $DT(L(F) - ac)$ from $DT(L(F))$ by a series of D-flips, starting by deleting the illegal edge ac. Each of these D-flips lexicographically increases the angle vector. If $DT(L(F) - ac)$ is not $DT(F)$ then we repeat the above procedure. From Fact 1, using a maximum of $O(n^2)$ D-flips we will obtain $DT(F)$. The corresponding Laman framework L' is a $DL(F)$, and will have been obtained by using at most $O(n^2)$ L-flips. □

In Figure 4 we show an example of L-flip described in the proof of Theorem 1 in which $L(F)$ is not a $DL(F)$: deleting the illegal edge 15 in $DT(L(F)) - F$, and updating the constrained Delaunay triangulation to $DT(L(F) - 15)$ we find another non-crossing Laman framework shown in the rightmost and upper corner of Figure 4.

For edges $e = ij$ with $i < j$ and $e' = kl$ with $k < l$, we use the notation $e \prec e'$ or $e' \succ e$ when e is lexicographically smaller than e' i.e., either $i < k$ or $i = k$ and $j < l$, and $e = e'$ when they coincide. For an edge set A we use the notations $\max\{e \mid e \in A\}$ and $\min\{e \mid e \in A\}$ to denote the lexicographically largest and smallest edges in A, respectively.

Definition 6. (*Lexicographic edge list*) *Let $E = \{e_1 \prec e_2 \prec \ldots \prec e_m\}$ and $E' = \{e'_1 \prec e'_2 \prec \ldots \prec e'_m\}$ be the lexicographically ordered edge lists. Then E is lexicographically smaller than E' if $e_i \prec e'_i$ for the smallest i such that $e_i \neq e'_i$.*

Theorem 2. *Let $L_1(F)$ and $L_2(F)$ be two F-constrained non-crossing Laman Frameworks on a point set P. Then $L_1(F)$ can be transformed to $L_2(F)$ by at most $O(n^2)$ L-flips.*

Proof. By Theorem 1, starting from $L_1(F)$ we can perform L-flips $O(n^2)$ times to reach a $DL(F)$, say $L(F)$. Let $L^*(F)$ be the $DL(F)$ with lexicographically smallest edge list. We show that we can do edge flips from $L(F)$ to $L^*(F)$, at most $n-3$ times, maintaining the non-crossing Laman property.

Consider the Laman matroid with base graph $DT(F)$. Both $L(F)$ and $L^*(F)$ are bases in this matroid, i.e., they are Laman subgraphs of $DT(F)$. Delete from $L(F)$ the lexicographically largest edge ac in $L(F) - L^*(F)$. By the matroid properties, there will always be an edge st in $L^*(F) - L(F)$ such that $L'(F) = L(F) - ac + st$ is an F-constrained Laman framework. $L'(F)$ is non-crossing since it is a subgraph of the non-crossing graph $DT(F)$. A triangulation on n points has at most $3n-6$ edges and a Laman framework has $2n-3$ edges, so after at most $n-3$ such L-flips we reach $L^*(F)$.

A similar argument shows that we can start with $L_2(F)$ and reach $L^*(F)$ in at most $O(n^2)$ L-flips, completing the proof of the theorem. □

4 Algorithm

Let $\mathcal{L}(F)$ be the set of F-constrained non-crossing Laman frameworks on P, and $\mathcal{DL}(F)$ be the set of F-constrained Delaunay Laman frameworks. Clearly $\mathcal{DL}(F) \subseteq \mathcal{L}(F)$ holds. Let $L^*(F)$ be the $DL(F)$ with the lexicographically smallest edge list as denoted in Section 3. We define the following *parent function* $f : \mathcal{L}(F) - \{L^*(F)\} \to \mathcal{L}(F)$ based on Theorems 1 and 2.

Definition 7. (*Parent function*) *Let $L(F) \in \mathcal{L}(F)$ with $L(F) \neq L^*(F)$. Then $L'(F) = L(F) - ac + st$ is the parent of $L(F)$, where*
Case 1: $L(F) \in \mathcal{DL}(F)$,
$ac = \max\{e \mid e \in L(F) - L^*(F)\}$ and $st = \min\{e \in L^*(F) - L(F) \mid L(F) - ac + e \in \mathcal{L}(F)\}$,
Case 2: $L(F) \in \mathcal{L}(F) - \mathcal{DL}(F)$,
$ac = \max\{e \in L(F) - F \mid e$ is illegal in $DT(L(F))\}$ and $st = \min\{e \in DT(L(F) - ac) - L(F) \mid L(F) - ac + e \in \mathcal{L}(F)\}$.

To simplify the notations, we denote the parent function depending on Case 1 and Case 2 by $f_1 : \mathcal{DL}(F) - \{L^*(F)\} \to \mathcal{DL}(F)$ and $f_2 : \mathcal{L}(F) - \mathcal{DL}(F) \to \mathcal{L}(F)$, respectively.

The reverse search algorithm can be considered on the underlying graph in which each vertex corresponds to a non-crossing Laman framework and two frameworks are *adjacent* if

Algorithm Enumerating F-constrained non-crossing Laman frameworks.

1: $L^*(F) :=$ a $DL(F)$ with lexicographically smallest edge list;
2: $L'(F) := L^*(F)$; $i, j := 0$; Output($L'(F)$);
3: **repeat**
4: **while** $i \leq |L'(F)|$ **do**
5: **do** $\{i := i + 1; e_1 := \mathsf{elist}_{L'}(i);\}$ **while**$(e_1 \in F)$;
6: **while** $j \leq |K_n|$ **do**
7: **do** $\{j := j + 1; e_2 := \mathsf{elist}_{K_n}(j);\}$ **while**$(e_2 \in L'(F))$;
8: **if** $\mathrm{Adj}(L'(F), i, j) \neq$ *null* **then**
9: $L(F) := \mathrm{Adj}(L'(F), i, j)$;
10: **if** $f_1(L(F)) = L'(F)$ **or** $f_2(L(F)) = L'(F)$ **then**
11: $L'(F) := L(F)$; $i, j := 0$;
12: Output($L'(F)$);
13: **go to** line 4;
14: **end if**
15: **end if**
16: **end while**
17: **end while**
18: **if** $L'(F) \neq L^*(F)$ **then**
19: $L(F) := L'(F)$;
20: **if** $L(F) \in \mathcal{DL}(F)$ **then** $L'(F) := f_1(L(F))$;
21: **else** $L'(F) := f_2(L(F))$;
22: determine integers pair (i, j) such that $\mathrm{Adj}(L'(F), i, j) = L(F)$;
23: $i := i - 1$;
24: **end if**
25: **until** $L'(F) = L^*(F)$ **and** $i = |L'(F)|$ **and** $j = |K_n|$;

Figure 5: Algorithm for enumerating F-constrained non-crossing Laman frameworks.

and only if one can be obtained from the other by a L-flip. Then, for $L'(F) \in \mathcal{L}(F)$ the *adjacency function*, Adj, is defined as follows:

$$\mathrm{Adj}(L'(F), e_1, e_2) := \begin{cases} L'(F) - e_1 + e_2 & \text{if } L'(F) - e_1 + e_2 \in \mathcal{L}(F), \\ \text{null} & \text{otherwise,} \end{cases}$$

where $e_1 \in L'(F) - F$ and $e_2 \in K_n - L'(F)$. The number of candidate edge pairs (e_1, e_2) is $O(n^3)$.

Let $\mathsf{elist}_{L'}$ and elist_{K_n} be lists of edges of $L'(F)$ and K_n ordered lexicographically, and let $\mathsf{elist}_{L'}(i)$ and $\mathsf{elist}_{K_n}(i)$ be the i-th elements of $\mathsf{elist}_{L'}$ and elist_{K_n}, respectively. We also denote the above defined adjacency function by $\mathrm{Adj}(L'(F), i, j)$ for which $e_1 = \mathsf{elist}_{L'}(i)$ with $e_1 \notin F$ and $e_2 = \mathsf{elist}_{K_n}(j)$ with $e_2 \notin L'$. Then, based on the algorithm in [3, 4], we describe our algorithm in Figure 5. An example of the search tree on a set of F-constrained non-crossing Laman frameworks on seven points are illustrated in Figure 6.

As we will show later, both the parent function and the adjacency function need $O(n^2)$ time for each process. The while-loop from line 4 to 17 has $|L'(F)| \cdot |K_n|$ iterations which

require $O(n^5)$ time if the tests in lines 8 and 10 are performed naively. In order to improve $O(n^5)$ time to $O(n^3)$ time we will use the following two lemmas:

Lemma 2. *Let $L(F)$ and $L'(F)$ be two distinct F-constrained Delaunay Laman frameworks for which $L(F) = \mathrm{Adj}(L'(F), e_1, e_2)$ for $e_1 \in L'(F) - F$ and $e_2 \in K_n - L'(F)$. Then, $f_1(L(F)) = L'(F)$ holds if and only if e_1 and e_2 satisfy the following conditions:*

(a) $e_1 \in L^*(F)$,

(b) $e_2 \in DT(L^*(F)) - L^*(F)$,

(c) $e_1 \prec \min\{e \in L^*(F) - L'(F) \mid L'(F) - e_1 + e \in \mathcal{L}(F)\}$,

(d) $e_2 \succ \max\{e \mid e \in L'(F) - L^*(F)\}$.

Lemma 3. *Let $L(F)$ and $L'(F)$ be two distinct F-constrained non-crossing Laman frameworks for which $L(F) = \mathrm{Adj}(L'(F), e_1, e_2)$ for edges $e_1 \in L'(F) - F$ and $e_2 \in K_n - L'(F)$ with $L(F) \in \mathcal{L}(F) - \mathcal{DL}(F)$. Then, $f_2(L(F)) = L'(F)$ holds if and only if e_1 and e_2 satisfy the following conditions:*

(a) *e_1 is a legal edge in $L'(F)$,*

(b) $e_2 \in K_n - DT(L'(F))$,

(c) $e_1 \prec \min\{e \in DT(L'(F)) - L'(F) \mid L'(F) - e_1 + e \in \mathcal{L}(F)\}$.

(d) $e_2 = \max\{e \in (L'(F) - e_1 + e_2) - F \mid e$ *is illegal in* $DT(L'(F) - e_1 + e_2)\}$.

We will explain later (in the proof of Theorem 3) how Lemmas 2 and 3 are used to obtain $O(n^3)$ time for generating each output of our algorithm. Notice that for $L'(F)$ and $L(F) \in \mathcal{L}(F)$ such that $L(F) = L'(F) - e_1 + e_2$, at most one of $f_1(L(F)) = L'(F)$ and $f_2(L(F)) = L'(F)$ holds from the conditions (b) of Lemmas 2 and 3. In the following proofs of Lemmas 2 and 3 we write L for $L(F)$, L' for $L'(F)$, etc., for simplicity because the constraining set F is fixed throughout.

Proof of Lemma 2. ("only if"-part.) Since $f_1(L) = L'$, e_1 and e_2 must be chosen as st and ac in Case 1 of Definition 7. From Definition 7, $e_2(= ac) \in L - L^*$. Since $L \in \mathcal{DL}$, $L \subset DT(L^*)$ holds, and we have (b). Similarly since $e_1(= st) \in L^* - L \subset L^*$, we have (a). From $e_1 = st$, we have

$$L' - e_1 = (L - ac + st) - e_1 = L - ac. \tag{1}$$

Let $e' = \min\{e \in L^* - L' \mid L' - e_1 + e \in \mathcal{L}\}$. Suppose (c) does not hold, and $e' \prec e_1$ holds. (Note that the equality does not hold since $e_1 \in L' - F$.) Then from Eq.(1) and $e' \prec e_1 = st \prec ac$ (which comes from Definition 7),

$$\begin{aligned} e' &= \min\{e \in L^* - (L - ac + st) \mid L - ac + e \in \mathcal{L}\} && \text{(from Eq.(1))} \\ &= \min\{e \in L^* - L \mid L - ac + e \in \mathcal{L}\} && \text{(from } e' \prec st \prec ac\text{)}. \end{aligned}$$

Thus, e' would have been selected instead of e_1 when the parent function f_1 is applied to L, which contradicts $e_1 = st$. Hence, (c) holds.

Let $e'' = \max\{e \mid e \in L' - L^*\}$, and suppose that (d) does not hold. A similar argument leads a condtradiction. Thus, (d) holds.

("if"-part.) From (a) and (b), $L = L' - e_1 + e_2$ is $DL(F)$. Since $e_1 \in L^*$ from (a),

$$\begin{array}{rcll} e_2 & \succ & \max\{e \mid e \in L' - L^*\} & \text{(from (d))} \\ & = & \max\{e \mid e \in (L + e_1 - e_2) - L^*\} & \text{(from } L = L' - e_1 + e_2\text{)} \\ & = & \max\{e \mid e \in (L - e_2) - L^*\} & \text{(from } e_1 \in L^*\text{)} \end{array}$$

holds. Thus, $e_2 = \max\{e \mid e \in L - L^*\}$, and hence f_1 chooses e_2 for an edge ac to be deleted from L. From this we obtain $L - ac = L' - e_1 + e_2 - ac = L' - e_1$. Since $e_2 \notin L^*$ from (b),

$$\begin{array}{rcll} e_1 & \prec & \min\{e \in L^* - L' \mid L' - e_1 + e \in \mathcal{L}\} & \text{(from (c))} \\ & = & \min\{e \in L^* - (L + e_1 - e_2) \mid L - ac + e \in \mathcal{L}\} & \text{(from } L - ac = L' - e_1\text{)} \\ & = & \min\{e \in L^* - (L + e_1) \mid L - ac + e \in \mathcal{L}\} & \text{(from } e_2 \notin L^*\text{)} \end{array}$$

holds. Since $e_1 \in L^* - L$, we obtain $e_1 = \min\{e \in L^* - L \mid L - ac + e \in \mathcal{L}\}$. Thus, f_1 chooses e_1 for an edge to be added, and $f_1(L)$ returns L'. □

Proof of Lemma 3. ("only if"-part.) Since $f_2(L) = L'$, e_1 and e_2 must be chosen as st and ac in Case 2 of Definition 7. As in the proof of Lemma 2, we have Eq.(1) from $e_2 = ac$. Since $st \in DT(L - ac) - L$ holds from Definition 7 and from Fact 2, st is a Delaunay edge with respect to $L - ac$ and then there exists a circle through both endpoints of st containing no point visible to st with respect to $L - ac$. Since such a circle still contains no point visible to st with respect to $L - ac + st$, st is still the Delaunay edge with respect to $L - ac + st = L'$, which implies that st is not illegal in $DT(L')$. Thus, from $e_1 = st$, (a) holds. Moreover we observe that

$$DT(L') = DT(L' - e_1) = DT(L - ac) \tag{2}$$

holds since e_1 is not illegal edge in $DT(L')$.

Let us consider e_2. Since ac is illegal in $DT(L)$ from Definition 7, we have $ac = e_2 \notin DT(L - ac)$. Therfore $e_2 \notin DT(L')$ holds from Eq. (2). Thus (b) holds. Also (d) must hold since the parent function removes the lexicographically largest illegal edge in $DT(L) - F$.

Finally let us consider e_1. Let $e' = \min\{e \in DT(L') - L' \mid L' - e_1 + e \in \mathcal{L}\}$. Suppose that (c) does not hold. Then $e' \prec e_1$ holds. (Note that the equality does not hold since $e_1 \in L' - F$.) Therefore, we have

$$\begin{array}{rcll} e' & = & \min\{e \in DT(L - ac) - (L - ac + e_1) \mid L - ac + e \in \mathcal{L}\} & \text{(from Eq.(1) and (2)),} \\ & = & \min\{e \in DT(L - ac) - (L - ac) \mid L - ac + e \in \mathcal{L}\} & \text{(from } e' \prec e_1\text{).} \end{array}$$

Then, e' would have been selected when the parent function is applied to L, which contradicts $e_1 = st$. Hence (c) holds.

("if"-part.) From (a), e_1 is legal in $DT(L')$. Then, we have $DT(L') = DT(L' - e_1)$. The condition (d) says that e_2 is the lexicographically largest illegal edge in $DT(L) - F$. Thus, f_2 chooses e_2 for an edge ac to be deleted from L, and $L' - e_1 = L - ac$.

From $L' - e_1 = L - ac$ and $DT(L') = DT(L' - e_1) = DT(L - ac)$ the condition (c) implies $e_1 \prec \min\{e \in DT(L - ac) - (L - ac + e_1) \mid L - ac + e \in \mathcal{L}\}$. Thus, $e_1 = \min\{e \in DT(L - ac) - L \mid L - ac + e \in \mathcal{L}\}$. (Note that $e_1 \in DT(L - ac)$ and $ac \notin DT(L - ac)$, because $DT(L - ac) = DT(L')$, and now we have $e_1 \in DT(L')$ and $ac = e_2 \notin DT(L')$ from (a) and (b), respectively.) Thus, f_2 chooses e_1 for an edge to be added, and $f_2(L)$ returns L'. □

Using Lemmas 2 and 3, we will describe an $O(n^3)$ algorithm in the proof of the following theorem. First we give a simple observation for checking the condition (d) in Lemma 3 efficiently:

Observation 1. *Let $DT(F)$ be an F-constrained Delaunay triangulation constrained by edges of a non-crossing edge set F, and let $e_1 \in F$ be a legal edge in $DT(F)$ and $e_2 \in K_n - DT(F)$ be an edge intersecting no edge of F. Then $DT(F - e_1 + e_2) = DT(F + e_2)$.*

Proof. Since e_1 is the legal edge in $DT(F)$, we have $DT(F) = DT(F - e_1)$. Then there exists a circle through both endpoints of e_1 containing no point visible to e_1 with respect to $F - e_1$ from Fact 2. And, when inserting e_2 into $F - e_1$, this circle clearly does not contain any point visible to e_1 with respect to $F - e_1 + e_2$. Thus e_1 remains a Delaunay edge with respect to $F - e_1 + e_2$, and this implies that $DT(F - e_1 + e_2)$ contains e_1 from Fact 2 and $DT(F + e_1) = DT(F - e_1 + e_2)$. □

Theorem 3. *The set of all F-constrained non-crossing Laman frameworks on a given point set can be reported in $O(n^3)$ time per one F-constrained non-crossing Laman framework using $O(n^2)$ space, or $O(n^4)$ time using $O(n)$ space.*

Proof. As described in Section 3, we use a linear transformation if necessary to get a unique $DT(F)$. The complexity of testing the uniqueness of a $DT(F)$ is $O(n^2)$ by simply testing the circumcircle of each triangle in the $DT(F)$ to see there is another point other than vertices of the triangle on the circumcircle.

Given a non-crossing Laman framework $L'(F) \in \mathcal{L}(F)$ and $L'(F)$-constrained Delaunay triangulation $DT(L'(F))$, the algorithm will either check if $f_1(Adj(L'(F), e_1, e_2)) = L'(F)$ or if $f_2(Adj(L'(F), e_1, e_2)) = L'(F)$ at line 10 depending on the edge pair (e_1, e_2). Here we will show that each condition in Lemmas 2 and 3 can be checked in $O(1)$ time for each of the $O(n^3)$ edge pairs (e_1, e_2) by the following way.

First, for all edges $e_2 \in \mathsf{elist}_{K_n}$, we calculate the number of edges $e_1 \in L'(F)$ intersecting e_2, which we denote by $\mathsf{cross\_n}(e_2, L'(F))$. If $\mathsf{cross\_n}(e_2, L'(F)) > 1$, we delete e_2 from elist_{K_n} since $L'(F) - e_1 + e_2$ is never non-crossing for any $e_1 \in \mathsf{elist}_{L'}$. If $\mathsf{cross\_n}(e_2, L'(F)) = 1$, we associate with e_2 a pointer $\mathsf{cross\_e}(e_2, L'(F))$ to the edge e_1 which intersects it.

Next, for each $e_1 \in \mathsf{elist}_{L'}$, we attach two flags to e_1 which represent that e_1 satisfies the conditions (a) of Lemmas 2 and 3, respectively. These help us to check the conditions (a) in Lemmas 2 and 3 in $O(1)$ time. Similarly, we attach two flags to $e_2 \in \mathsf{elist}_{K_n}$ which represent that e_2 satisfies the conditions (b) of Lemmas 2 and 3 to check them in $O(1)$ time for each e_2. Additionally we calculate the lexicographically largest edge in $L'(F) - L^*(F)$ in $O(n)$ time to check the condition (d) in Lemma 2 in $O(1)$ time for each e_2. This preprocessing can be done in $O(n^2)$ time using the precomputed sorted edge list of $L^*(F)$ and $DT(L(F))$.

Now let us consider how to identify a set of edges $e_2 \in \mathsf{elist}_{L'}$ satisfying the condition (d) in Lemma 3 in $O(n^3)$ time with $O(n^2)$ space. (In the case of $O(n^4)$ time algorithm this process must be skipped, and the condition (d) in Lemma 3 will be checked simply by updating $DT(L'(F))$ to $DT(L'(F) - e_1 + e_2)$ for each pair (e_1, e_2) using $O(n)$ time and $O(n)$ space by applying the algorithm by Chin and Wang [11].) It can be done regardless of the removed edge e_1 when condition (a) in Lemma 3 is satisfied. From Observation 1 we can say that the condition (d) holds if and only if e_2 is the lexicographically largest illegal edge in $DT(L(F) + e_2) - F$ when $\mathsf{cross\_n}(e_2, L'(F)) = 0$. It is sufficient to check condition (d) only

in $DT(L(F)-\mathsf{cross\_e}(e_2, L'(F)) + e_2)$ when $\mathsf{cross\_n}(e_2, L'(F)) = 1$. Updating the Delaunay triangulation takes $O(n)$ time (see [2, 11, 12] for a linear time update of the constrained Delaunay triangulation). Thus we can attach a flag to each $e_2 \in \mathsf{elist}_{K_n}$ in $O(n)$ time which represents whether e_2 satisfies (d) of Lemma 3 or not, and this preprocessing for all edges in elist_{K_n} takes $O(n^3)$ time.

By using the above mentioned data, we will show that for a fixed $e_1 \in \mathsf{elist}_{L'}$, the inner while-loop from line 6 to 16 can be executed in $O(n^2)$. In order to efficiently test the condition (c) of Lemmas 2 and 3, we prepare the data structure proposed by Lee, Streinu and Theran [20, 21] in $O(n^2)$ time for maintaining *rigid components* of $L(F) - e_1$. This data structure supports a *pair-find* query which determines whether two vertices are spanned by a common component in $O(1)$ time using $O(n^2)$ preprocessing time with $O(n^2)$ space, or $O(n)$ time using $O(n^2)$ preprocessing time with $O(n)$ space. From this, we can calculate $Adj(L'(F), e_1, e_2)$ (i.e., determine whether $L'(F) - e_1 + e_2 \in \mathcal{L}(F)$) in $O(1)$ time with $O(n^2)$ space, or $O(n)$ time with $O(n)$ space, for each edge $e_2 \in \mathsf{elist}_{K_n}$. Also, we can compute $e' = \min\{e \in L^*(F) - L'(F) \mid L'(F) - e_1 + e \in \mathcal{L}(F)\}$ and $e'' = \min\{e \in DT(L'(F)) - L'(F) \mid L'(F) - e_1 + e \in \mathcal{L}(F)\}$ in $O(n)$ time with $O(n^2)$ space, or $O(n^2)$ time with $O(n)$ space. Using e' and e'' we can check condition (c) in Lemmas 2 and 3 in $O(1)$ time.

Thus, we have confirmed that all conditions of Lemmas 2 and 3 can be checked in $O(1)$ time for each pair (e_1, e_2) by taking $O(n^3)$ preprocessing time with $O(n^2)$ space, or $O(n)$ time for each (e_1, e_2) with $O(n)$ space.

By using the above mentioned data structure for maintaining the rigid components, we can perform both parent function and adjacency function in $O(n^2)$ time with $O(n^2)$ space, or $O(n^3)$ time with $O(n)$ space. Thus, we have an $O(n^3)$ algorithm using $O(n^2)$ space, or $O(n^4)$ algorithm using $O(n)$ space. □

5 Conclusions

We have presented an algorithm for enumerating all the constrained non-crossing Laman frameworks. We note in passing that the techniques in this paper can also be used to generate all F-constrained non-crossing spanning trees of a point set since they also form bases of the graphic matroid on any triangulation of P. The unconstrained case was considered in [1, 4].

An open problem, that is of considerable practical importance, is to generate efficiently all non-crossing Laman Frameworks that do *not* contain any edge from a given set. This is equivalent to generating all non-crossing Laman frameworks that are subgraphs of a given geometric graph. An indication that this problem may be challenging, is that it is known that determining if a geometric graph contains a non-crossing spanning tree is NP-complete [15].

6 Acknowledgements

The authors would like to thank an anonymous referee for many ideas that lead to an improved presentation of this research.

References

[1] O. Aichholzer, F. Aurenhammer, C. Huemer, and B. Vogtenhuber. Gray code enumeration of plane straight-line graphs. In *Proc. 22th European Workshop on Computational Geometry (EuroCG '06)*, pages 71–74, Greece, 2006.

[2] M. V. Anglada. An Improved incremental algorithm for constructing restricted Delaunay triangulations. *Computer and Graphics*, 21(2):215–223, 1997.

[3] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete and Computational Geometry*, 8:295–313, 1992.

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

[5] D. Avis, N. Katoh, M. Ohsaki, I. Streinu, and S. Tanigawa. Enumerating planar minimally rigid graphs. In *Proc. 12th International Computing and Combinatorics Conference (COCOON 2006)*, volume 4112 of Lecture Notes in Computer Science, pp.205–215, Springer, 2006. *Graphs and Combinatorics*, to appear.

[6] I. Beichl, and F. Sullivan. Coping with degeneracies in Delaunay triangulation. In *Modelling, Mesh Generation and Adaptive Numerical Methods for Partial Differential Equations, J.E. Flaherty et al. eds.*, pages 23–30, Springer, New York, 1995.

[7] M. P. Bendsøe and O. Sigmund. *Topology Optimization: Theory, Methods and Applications*. Springer, 2003.

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

[9] A. Berg and T. Jordán. Algorithms for graph rigidity and scene analysis. In G. D. Battista and U. Zwick, editors, *Proc. 11th Annual European Symposium on Algorithms (ESA)*, volume 2832 of *Lecture Notes in Computer Science*, pages 78–89. Springer, 2003.

[10] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. *Computing in Euclidean Geometry, 2nd Edition, Du and Hwang eds.*, 23–90, 1992.

[11] F. Chin and C. A. Wang. Finding the constrained Delaunay triangulation and constrained Voronoi diagrams of a simple polygon in linear time. SIAM J. Comput., 28(2):471–486, 1998.

[12] L. de Floriani and A. Puppo. An on-line algorithm for constrained Delaunay triangulation. *Computer Vision, Graphics and Image Processing*, 54(3):290–300, 1992.

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

[14] D. J. Jacobs and B. Hendrickson. An algorithm for two-dimensional rigidity percolation: the pebble game. *Journal of Computational Physics*, 137:346 – 365, November 1997.

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

[16] N. Katoh, M. Ohsaki, T. Kinoshita, S. Tanigawa, D. Avis and I. Streinu. Enumeration of optimal pin-jointed bistable mechanisms. In *Proc. 4th China-Japan-Korea Symp. of Structural and Mechanical Systems*, Kunming, Nov 2006.

[17] A. Kaveh. *Structural Mechanics: Graph and Matrix Methods*. Research Studies Press, Somerset, UK,, 3rd edition, 2004.

[18] A. Kawamoto, M. Bendsøe, and O. Sigmund. Planar articulated mechanism design by graph theoretical enumeration. *Struct Multidisc Optim*, 27:295–299, 2004.

[19] G. Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering Mathematics*, 4:331–340, 1970.

[20] A. Lee and I. Streinu. Pebble game algorithms and sparse graphs. In *Proc. EUROCOMB*, Berlin, September 2005.

[21] A. Lee, I. Streinu, and L. Theran. Finding and maintaining rigid components. In *Proc. Canad. Conf. Comp. Geom.*, Windsor, Canada, August 2005.

[22] M. Ohsaki and S. Nishiwaki. Shape design of pin-jointed multi-stable compliant mechanisms using snapthrough behaviour. *Struct. Multidisc. Optim.*, 30:327–334, 2005.

[23] I. Streinu. Pseudo-triangulations, rigidity and motion planning. *Discrete and Computational Geometry*, 34:587–635, December 2005.

[24] 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.

[25] W. Whiteley. Matroids from discrete geometry In *Matroid Theory, J. Bonin, J. Oxley and B. Servatius eds.* AMS Contemporary Mathematics, 171-313, 1997

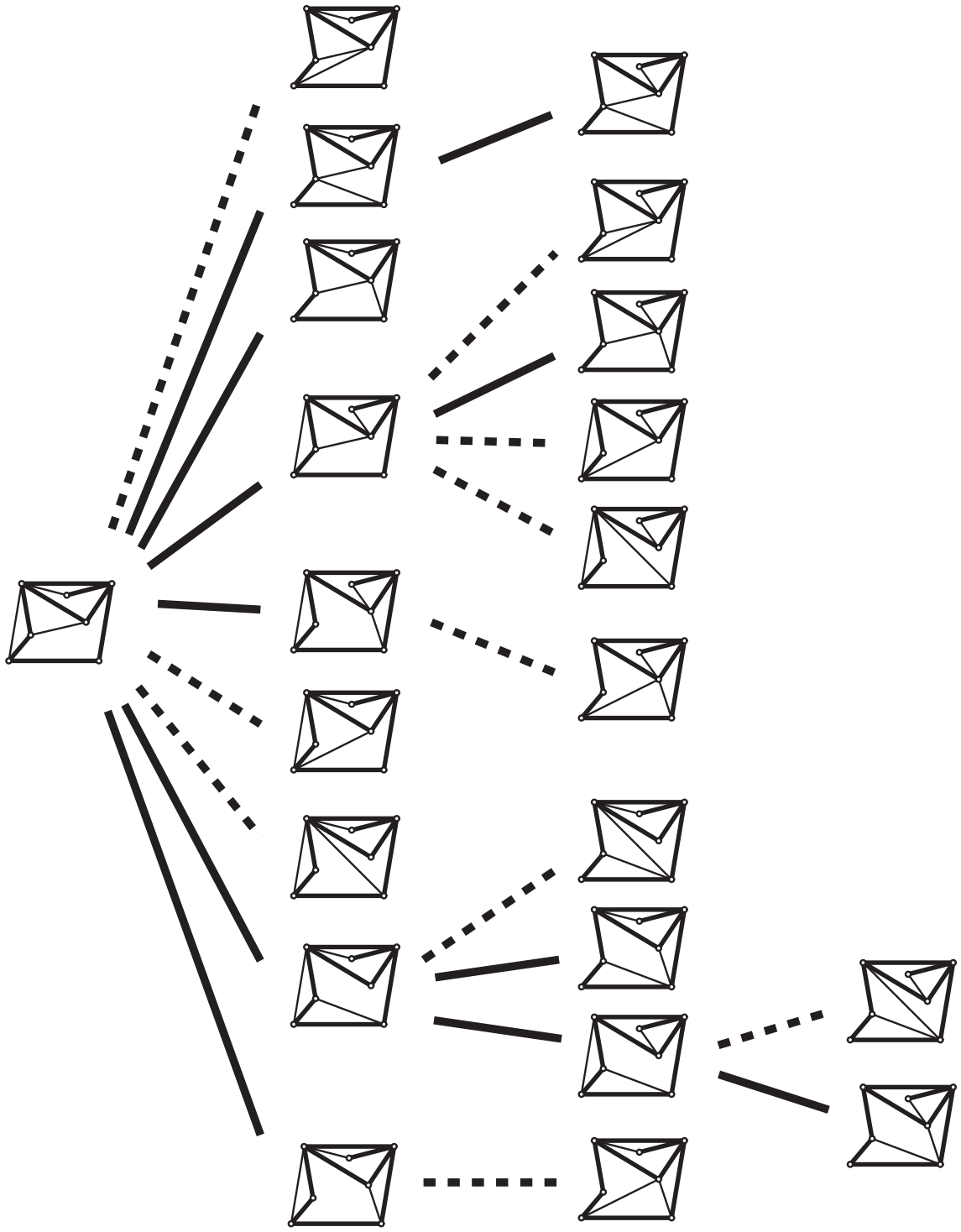


Figure 6: An example of the search tree of our algorithm on seven points. The constraint edges F are illustrated by the bold edges, and each edge of the search tree is distinguished by using the dotted or bold line according to whether it corresponds to Case 1 or Case 2 of Definition 7.