Vol.2017-AL-161 No.6
2017/1/17

Implementation of Enumerating All Edge-Constrained Triangulations without the General Position Assumption

Masatoshi Murakami[1,a)] Katsuhisa Yamanaka[1,b)] Takashi Hirayama[1] Yasuaki Nishitani[1]

Abstract: Enumeration problems of triangulations are studied in some problem settings. In this paper, without general position assumption, we enumerate all edge-constrained triangulations on a given set of points in the plane. Katoh and Tanigawa proposed an algorithm that enumerates all edge-constrained triangulations for a given point set in general position. In a point set P in general position, no three points are collinear. By extending their algorithm, we design and implement an algorithm that for a given point set P, enumerates all edge-constrained triangulations without the general position assumption(i.e. three points in P may be collinear).

1. Introduction

Triangulations of a point set are fundamental and important objects in geometry such as convex hulls and convex polyhedrons. Triangulations have many applications including mesh generations in computer graphics and finite element analysis and interpolation. Fig. 1 is an example of a triangulation.

Recently, some enumeration algorithms for triangulations of a point set were proposed [1], [3]. For a given point set, if we have an enumeration algorithm, by using this, we can construct the exhaustive list of triangulations of the point set. The list can be used for choosing good triangulations for any criteria. From this point of view, there are some results on enumerating triangulations.

Avis and Fukuda [2] presented an algorithm that enumerates all triangulations of a point set in $O(n)$ time for each, where n is the number of points. Bespamyatnikh [1] gave an algorithm that enumerates all triangulations of a given point set. The algorithm generates each triangulation in $O(\log \log n)$ time, where n is the number of points. Katoh and Tanigawa [3] extended this algorithm. They pointed out the number of all triangulation is huge and the complete enumerations essentially require much time for a large point set. Hence, we require "reasonable" constraints to reduce the number of triangulations to be enumerated. From this point of view, they improved Bespamyatnikh's an algorithm, so that, for a given edge set F, the algorithm enumerates all the triangulations including the edges in F. The algorithm also generates each triangulation in $O(\log \log n)$ time.

The papers [1], [3] assume that input points are in general position. However, triangulations of a point set without the general position assumption are also natural and important for the following reasons. First, point sets without the general position assumption, such as grids, are used frequently in application areas.

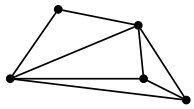

Fig. 1 A triangulation.

For example, when 3D printers read the surface of a given object, they use triangle meshes on 3-dimensional grids to represent the surface. Next, the number of the triangulations of point sets without the general position assumption tends to be smaller than the one of triangulations of point sets with the assumption. (This tendency is confirmed in this report from experimental results.) This tendency would be preferred from the viewpoint that the number of triangulations to be enumeration should be reduced by using reasonable constraints. Thus, enumerating triangulations on a point set without the general position assumption is also a natural problem setting.

In this paper, we show that the algorithm by Katoh and Tanigawa [3] still works without the general position assumption by slightly modifying the algorithm. Besides, we implemented the algorithm and measure the practical performance of the algorithm.

2. Preliminaries

Let P be a set $\{p_1, p_2, \ldots, p_n\}$ of n distinct points in the plane. We assume that the points are sorted by x-coordinate and two points with the same x-coordinate are sorted by their y-coordinates. Note that three points in P may be collinear. If all points in P are collinear, there is no triangulation for P. Thus we suppose that all points are not collinear. An ordered pair $(p_i, p_j), i < j$, of distinct two points in P is called an *edge* if the line segment connecting p_i and p_j does not contain any other point of P. A triple $\{p, q, r\}$ is a *triangle* if they are not collinear and the convex hull for p, q, and r contain no point in P. An edge in P is *external* if it is contained in the boundary of the convex hull of P, and *internal* otherwise. A *triangulation* of P is a set

[1] Faculty of Science and Engineering, Iwate University
[a)] murafei131@kono.cis.iwate-u.ac.jp
[b)] yamanaka@cis.iwate-u.ac.jp

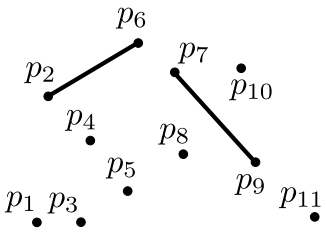

Fig. 2 The point set $P = \{p_1, p_2, \ldots, p_{11}\}$ in which p_3, p_5, and p_8 collinear and the constrained edge set $F = \{(p_2, p_6), (p_7, p_9)\}$. The point p_5 is visible from p_3, but p_8 and p_{10} are not visible.

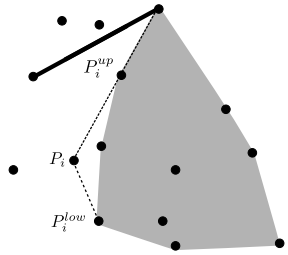

Fig. 3 The upper (p_i, p_i^{up}) tangent and the lower tangent (p_i, p_i^{low}) of p_i with respect to F.

E of edges in P such that (1) E contains all the external edges and (2) E divides the inside region of the convex hull of P into triangles. We denote $\{p_{i+1}, p_{i+2}, \ldots, p_n\}$ by P_{i+1}.

A total ordering $\prec$ on the set of edges is defined as follows: for $e = (p_i, p_j)$ and $e' = (p_k, p_l)$ (with $p_i < p_j$ and $p_k < p_l$), e is *smaller* than e' , denoted by $e \prec e'$, if and only if $p_i < p_k$, or $p_i = p_k$ and $p_j < p_l$. Let $E = \langle e_1, \ldots, e_m \rangle$ and $E' = \langle e'_1, \ldots, e'_m \rangle$ be two sorted edge lists with $e_1 \prec \cdots \prec e_m$ and $e'_1 \prec \cdots \prec e'_m$. Then, E is lexicographically *smaller* than E' if $e_i \prec e'_i$ for the smallest i such that $e_i \neq e'_i$. For two triangulations T_1 and T_2, T_1 is *smaller* than T_2 if the sorted edge list of T_1 is smaller than the one of T_2. We say that two edges (p_i, p_j) and (p_k, p_l) *properly intersect* each other if (p_i, p_j) and (p_k, p_l) have a point in common except for their endpoints. An edge set is *non-crossing* if no two edges in the set properly intersect. Let F be a non-crossing edge set on P. For a set F of non-crossing line segments on P, a triangulation containing F is called an *F-constrained triangulation*. For two points $p_i, p_j \in P$, p_j is *visible* from p_i with respect to F if (p_i, p_j) does not properly intersect any edge of F and (p_i, p_j) contains no other point (see Fig. 2). Since we do not assume the general position, it is required to clarity the visibility among collinear points. We assume that p_j is visible from p_i if $(p_i, p_j) \in F$. If a point p_x is on an edge $(p_i, p_j) \in F$, we can divide (p_i, p_j) into the two edges (p_i, p_x) and (p_x, p_y). Thus, we assume that there is no point on an edge in F.

The *upper tangent* (p_i, p_i^{up}) and the *lower tangent* (p_i, p_i^{low}) of p_i with respect to F is defined as those 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. 3

3. Enumeration algorithm

The enumeration algorithm of edge-constrained triangulations by Katoh and Tanigawa [3] works for a point set without the general position assumption using the definition of the visibility in Section 2. Their algorithm is based on the reverse search technique proposed by Avis and Fukuda [2]. We first define a tree structure called a "triangulation tree" which contains all edge-constrained triangulations of a given point set. Then the algorithm generates all triangulations by traversing the triangulation tree with the depth-first search. In this section, we briefly explain their algorithm for self-containment.

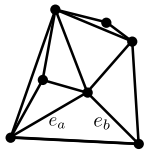

Fig. 4 The edge e_a is the flippable edge, but e_b is not the flippable edge.

3.1 Triangulation tree

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points, and let F be a set of edges in P. In this subsection, we define a tree structure $\mathcal{T} = (\mathcal{V}_\mathcal{T}, \mathcal{E}_\mathcal{T})$, called a triangulation tree, such that (1) each vertex v in $\mathcal{V}_\mathcal{T}$ corresponds to an F-constrained triangulation of a given point set P, (2) each edge e in $\mathcal{E}_\mathcal{T}$ corresponds to a parent-child relation between two triangulations, and (3) the root of $\mathcal{T}$ corresponds to the lexicographically largest triangulation T^* having the lexicographically largest edge list among all the F-constrained triangulation on P.

Let T^* denote the F-constrained lexicographically largest triangulation on P. For any F-constrained triangulation T with $T \neq T^*$, the *critical vertex* of T is the vertex having the smallest label among those incidents to some edge in $T \backslash T^*$. For an edge $e = (u, v) \in T \backslash T^*$, let uwx and uvy be the triangles incident to e. Then, e is called *flippable* if two triangles incident to e in T form a convex quadrilateral. See Fig.4 for an example. Note that if either x, u, y or x, v, y are collinear, the two triangles are not quadrilateral. Hence, (u, v) is not flippable. The *flip* of e is replacing e with the other diagonal edge of e. Flipping e in T generates a new *F-constrained triangulation*. Such an operation is called an *improving flip* if the triangulation obtained by flipping e is lexicographically larger than the previous one, and we say that e is called *improving flippable*. See Fig. 5 for an example.

Lemma 1 [3] *Let T be an F-constrained triangulation with $T \neq T^*$ and p_c be the critical vertex of T. Then, there exists at least one improving flippable edge incident to p_c in $T \backslash T^*$.*

For every F-constrained triangulation T with $T \neq T^*$, let us define the *parent* of T as the triangulation obtained by flipping the smallest improving flippable edge among $T \backslash T^*$. Note that the parent of T always exists from Lemma 1. Besides, the parent is also an F-constrained triangulation. Also, we can find the parent of the parent of T. By repeatedly find the parents, we obtain a sequence of F-constrained triangulations. The sequence always ends up with T^*. The sequence can be obtained for any F-constrained triangulation. By merging the sequence for every F-constrained triangulation, we have a tree structure, called *triangulation tree* rooted at T^*.

3.2 Constructing T^*

In the previous section, we define the triangulation tree. By traversing the triangulation tree, we enumerate all the edge-constrained triangulation. To traverse the tree, we first construct

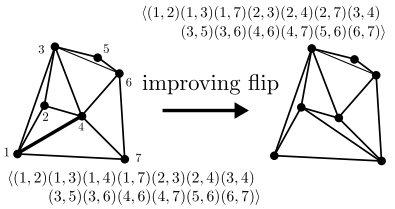

Fig. 5 The edge $(1,4)$ is improving flippable.

the root triangulation, namely the F-constrained lexicographically largest triangulation T^*. In this subsection, we explain how to construct T^*.

For $p_i \in P$ and a triangulation T of P, let us denote by $\delta_T(p_i)$ the subset of edges of T which are incident to p_i with the left (or bottom) endpoints. Similarly, for an edge set F of P, $\delta_F(p_i)$ denotes the subset of edges of F which are incident to p_i with the left (or bottom) endpoints.

Now we show a construction of T^* below. For $p_i \in P$ let (p_i, p_i^{up}) and (p_i, p_i^{low}) be the upper and lower tangents of $p_i \in P$, and denote the right (or top) endpoints of $\delta_F(p_i) \cup \{(p_i, p_i^{up}),(p_i, p_i^{low})\}$ by $p_{i_0}, p_{i_1}, \ldots, p_{i_m}$ in clockwise order around p_i, where $p_{i_0} = p_i^{up}$ and $p_{i_m} = p_i^{low}$ hold. Let C_k be the cone with apex p_i bounded by two consecutive edges (p_i, p_{i_k}) and $(p_i, p_{i_{k+1}})$ for each k with $0 \le k \le m-1$, 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. Then, we repeat the following process for all $p_i \in P$ in an arbitrary order: Connect from p_i to every point $p_j \in P_{i+1} \cap C_k$ if (1) $p_j = (p_i, p_j) \cap H_k$ holds for some k and (2) p_j is visible from p_i. Intuitively, we connect p_i with every point visible from p_i.

The difference between the construction above and the one in [3] is only the check of visibility. The triangulation obtained by the above construction is the F-constrained lexicographically largest triangulation. To show the correctness of the above construction, we can apply the same proof in [3]. We omit the proof is this manuscript.

3.3 Generating all children

Given an edge-constrained triangulation T of P, by flipping some flippable edge of T, we obtain a child of T, but flipping an arbitrary flippable edge does not always produce a child of T. We consider how to find the children of each T.

Let T be an F-constrained triangulation, and let e be a flippable edge in T. Let us denote by $\mathrm{flip}(T, e)$ the triangulation obtained from T by flipping e. It is easy to observe that $\mathrm{flip}(T, e)$ is a child if and only if the parent of $\mathrm{flip}(T, e)$ is T, that is, $\mathrm{dual}(e)$ is the smallest improving flippable edge of $\mathrm{flip}(T, e)$, where $\mathrm{dual}(e)$ is the edge obtained by flipping e. We can use the following lemma by Bespamyanikh [1] to check a flippable edge produces a child triangulation.

Lemma 2 [1] *Let $(p_a, p_b), a < b$ is the smallest improving flippable edge among $T \backslash T^*$. Let $(p_c, p_d), c < d$ is the edge obtained by flipping a flippable edge e of T. Then by flipping e, we can obtain the child of T if and only if either*

(i) $c < a$, or

(ii) $a = c$ and $d < b$, or

(iii) $a = c$, $d > b$, and (p_a, p_b) is non-flippable in $\mathrm{flip}(T, e)$ and the second lexicographically smallest flippable edge in $T \backslash T^*$ is lexicographically larger than (p_c, p_d).

Algorithm 1 ChildEdge(e, T)

1: Let $(a, b), a < b$ be the smallest improving flippable edge in $T \backslash T^*$, and let $(c, d), c < d$ be an edge obtained by flipping e
2: **if** the edge e satisfies either
(i) $c < a$ or
(ii) $a = c$ and $d < b$ or
(iii) $a = c$ and $d > b$ and the smallest improving flippable edge in $T \backslash T^*$ is non-flippable in $\mathrm{flip}(T, e)$ and second smallest flippable edge(if any) in $T \backslash T^*$ is larger than (c, d) **then**
3: **return** *true*
4: **end if**

Algorithm 2 EnumTri(T)

1: **for all** flippable edge e of $T \backslash T^*$ **do**
2: **if** ChildEdge(e, T)= *true* **then**
3: **output:** $\mathrm{flip}(T, e)$
4: EnumTri$(\mathrm{flip}(T, e))$
5: **end if**
6: **end for**

By implementing the above lemma straightforwardly, we have an algorithm shown in Algorithm 1. This algorithm determined, for a given edge, whether flipping the edge produce a child. We also have an enumeration algorithm shown in Algorithm 2. Bespamyatnikh showed that we can maintain a set of flippable edges that produce children of the current triangulation during a traversal of the triangulation tree. Such list can be updated in $O(\log \log n)$ whenever a child is generated. Hence, we have the following theorem.

Theorem 3 [1] *The triangulations of a point set in the plane with the general position assumption can be enumerated in $O(\log \log n)$ time per triangulation using linear space.*

The theorem can be applied to the triangulations of a point set without the general position assumption since the conditions of flippability are equal. (Note that a quadrangle including collinear three points is not a convex quadrilateral.) Thus we have the following corollary.

Corollary 4 *For a given point set P without the general position assumption and a set F of edges in P, one can enumerate all F-constrained triangulations of P.*

4. Experimental results

We implemented the enumeration algorithm in Section 3. In this section, we give experimental results.

4.1 Influence of the general position assumption

In this subsection, we investigate the numbers of triangulations of point sets with the general position assumption and without the assumption. Table 2 shows our experimental results. To generate data sets, we do the following process. For each $n = 10, 11, \ldots, 16$, we generated 100 sets each of which consists n random points with the general position assumption in a 100×100 square. Then, to make each set contain collinear three points, we choose three points in each point set, and change x, y-coordinates of them with (25,25), (50,50), and (75,75). Thus, each set includes exactly one collinear triple of points. Next, to generate point sets with the general position assumption, we move the center point (50,50) in each point set to (50,49). (We

Table 1 Experimental environment.

CPU	Intel®Core™i7-4770S 3.10GHz
Memory	16GB
OS	Ubuntu 14.04.1 LTS
Language	ANSI Common Lisp (SBCL 1.1.14.debian)

Table 2 Experimental results for the general position assumption. In each cell, the top value is the average number of triangulations and the bottom value is the average running time [sec] for 100 point sets.

n	With the general position	Without the general position assumption
10	1106.38 0.010	969.18 0.008
11	4389.58 0.045	3787.1 0.038
12	18090.3 0.199	15617.9 0.172
13	75771.9 0.879	69320.7 0.812
14	319642.2 3.932	283584.8 3.467
15	1557440.1 20.321	1369596.6 17.714
16	7392895.0 100.214	6995192.5 95.580

checked that this movement does not produce a collinear triple.) We performed the algorithm to each point set. Each cell in Table 2 contains the average number of triangulations (top) and the average running time [sec] (bottom). For $n = 10, 11, \ldots, 16$, one can observe that the number of triangulations without the general position assumption is smaller than the one with the assumption.

4.2 Edge-constraint

In this subsection, we investigate the numbers of edge-constrained triangulations of point sets without the general position assumption. For each $n = 10, 11, \ldots, 17$, we generate 5 random point sets each of which includes at least one collinear triple. Besides, for each point set, we choose p% edges, where $p = 0, 10, 20, 30, 40, 50$, as the constrained edges at random. Table 3 shows our experimental results. Each cell in Table 3 contains the average number of edge-constrained triangulations (top) and the average running time [sec] (bottom). For example, when $n = 17$, the average number of triangulations without edge-constraint is 8,897,090, and the average number of triangulations in which 10% edges are constrained is only 13,703. Edge-constraints tend to decrease the numbers of triangulations significantly.

4.3 Grid graphs

Finally, we performed the enumeration algorithm on grid graphs. As we refer in Section 1, grid graphs are used in practical applications such as 3-D printers. For a 3×3 grid, a 4×4 grid, and a 5×5 grid, we measured the numbers of triangulations and running time. For the case without edge-constraint, the results in Table 4 is obtained (left column). For the case with edge-constraints, we prepared the two pattern: a vertical constraint and a diagonal-constraint (Figs.6–11). The results are shown in Table 4 (center and right columns). From the results, we can observe that the number of triangulation is reduced significantly by designating constrained edges. For the 5×5 grid, the number of diagonal-constrained triangulations is only 7.4% of the no-constrained one.

5. Conclusions

We have implemented enumeration of all constrained triangulations without the general position assumption. We observed that the numbers of triangulations are significantly decreased using edge-constraint for point sets without the general position assumption. Future works include designing an algorithm that enumerates triangulations with other constraints. For instance, we enumerate all the triangulations of a given point set with bounded degrees.

Table 3 Experimental results for edge-constraint. In each cell, the top value is the average number of edge-constrained triangulations and the bottom value is the average running time [sec] for 5 point sets.

n	0%	10%	20%	30%	40%	50%
10	713.0 0.008	64.2 0.000	26.6 0.000	7.0 0.000	4.0 0.000	2.8 0.000
11	2714.6 0.033	413.0 0.004	181.6 0.001	23.8 0.000	5.4 0.000	1.2 0.000
12	6326.8 0.080	543.8 0.005	90.4 0.001	12.4 0.000	5.4 0.000	3.8 0.000
13	21962.0 0.304	1621.0 0.021	50.0 0.000	24.0 0.000	7.0 0.000	4.0 0.000
14	73727.0 1.064	5083.4 0.061	260.6 0.003	46.4 0.000	6.2 0.000	3.0 0.000
15	297122.8 4.299	24126.4 0.341	1874.0 0.021	73.0 0.021	6.0 0.000	5.8 0.000
16	1572699.4 24.292	20140.0 0.282	3818.4 0.043	1321.2 0.015	48.8 0.000	14.2 0.000
17	8897090.0 135.959	13703.2 0.176	569.6 0.006	69.4 0.000	- -	- -

Table 4 Experimental results for the grids. In each cell, the number of edge-constrained triangulations (top) and running time (bottom) of the grid.

$n \times n$	No-constraint	Vertical-constraint	Diagonal-constraint
3×3	64 0.001	36 0.001	16 0.000
4×4	46456 0.616	17040 0.187	6241 0.074
5×5	736983568 13001.669	148108900 2489.553	55115776 939.494

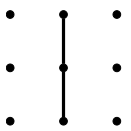

Fig. 6 A 3×3 grid with a vertical constraint.

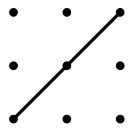

Fig. 7 A 3×3 grid with a diagonal constraint.

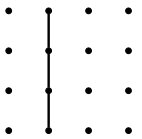

Fig. 8 A 4×4 grid with a vertical constraint.

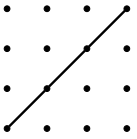

Fig. 9 A 4×4 grid with a diagonal constraint.

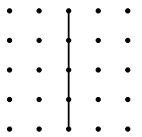

Fig. 10 A 5×5 grid with a vertical constraint.

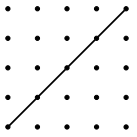

Fig. 11 A 5×5 grid with a diagonal constraint.

References

[1] S. Bespamyatnikh, An efficient algorithm for enumeration of triangulations, *Computational Geometry*, Vol. 23, No. 3, pp.271-279, 2002
[2] D. Avis and K. Fukuda, Reverse search for enumeration, *Discrete Applied Mathematics*, Vol. 65, No. 1-3, pp.21-46, 1996
[3] N. Katoh and S. Tanigawa, Enumerating edge-constrained triangulations and edge-constrained non-crossing geometric spanning trees, *Discrete Applied Mathematics*, Vol. 157, No. 17, pp.3569-3585, 2009