5P-01

Generating Edge-constrained Triangulations of Large Point Sets

Tomoaki Ishikawa† Tanami Yamauchi† Katsuhisa Yamanaka‡ Takashi Hirayama‡
†Graduate School of Science and Engineering, Iwate University ‡Iwate University

1 Introduction

An enumeration is to output all designated objects without duplication. The enumeration problem is fundamental and important in the area of theoretical computer science. In computational geometry, the problem of enumerating triangulations has been studied. Triangulations are appealing and important objects, since they have a lot of applications including mesh generation by the interpolation [1].

It is useful to enumerate all triangulations of a given point set, because we can select the optimal triangulation among the enumerated triangulations. However, the computation cost for enumeration is enormous when the given point set is large. In such case, the enumeration might be inadequate. It would be better to partly generate triangulations with desirable conditions.

Now, what is a good triangulation? Standards of good triangulations could be different depending on the situation of applications. Hence, we do not have a discussion about goodness of triangulations. However, "delaunay triangulations" are major triangulations and used in various applications. Thus, we focus on delaunay triangulations as good triangulations.

In this paper, we first evaluate the practical performance of the enumeration algorithm by Katoh and Tanigawa [2]. Their algorithm can enumerate only edge-constrained triangulations efficiently. This condition extremely reduces the number of enumerated triangulatios. However, when the number of given points is large (for example, 100 or more), the number of triangulations is too enormous. Thus, enumeration approach does not work for a large point set.

2 Definition

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n point in Euclidean plane.[1] We assume that $P = \{p_1, p_2, \ldots, p_n\}$ are arranged in the increasing order of x-coordinate. Points with the same x-coordinate are arranged in the increasing order of y-coordinate. A *triangulation* of P is a maximal planar subdivision such that point set of the triangulation is P and every edge includes no point except its endpoints. A triangulation T is a *Dalaunay triangulation* if the circumscribed circle of every triangle in T includes no point inside itself. Let F be a set of edges whose endpoints are points in P and no two edges in F intersect. A triangulation T is an *F-constrained triangulation* if T includes all the edges in F. Let $e = (x, y)$ be an edge of a triangulation T, and let $\triangle xyz$ and $\triangle xyw$ be the two triangles sharing e. A *flip* operation is to remove e and insert the diagonal $d = (w, z)$. See Figure 2. Applying a flip operation to a triangulation produces another different triangulation.

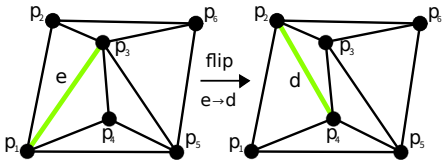

Figure 1: Flip operation.

3 Enumeration

We implemented the enumeration algorithm by Katoh and Tanigawa [2]. Their algorithm can enumerate edge-constrained triangulation in $O(\log n)$ time for each. In this section, we describe experimental results of implemented algorithms.

First we randomly generate 100 point sets of n points for each $n = 11, 12, \ldots, 16$. Then, we also randomly generate edge sets for constraints for each point set such that p % edges are constrained for each $p = 0, 10, 20, 30$. Then, we measured the running time of the algorithm. Environment of experiment is as follows: Programming language: Common lisp, Compiler: SBCL, CPU: Intel®Core™i7-4770S 3.10 GHz, and Memory: 16.0GB. The results are shown in Table 1.

From the Table 1, edge-constraint reduces the number of triangulation. As a result, the running time is reduced very well. On the other hand, when the number of points increases, running time increase enormously. Thus, from a practical viewpoint, it is almost impossible to apply enumeration algorithms to a large point set (for example, 100 points to 1000 points).

[1] We do not assume that the points in P are in general position.

Table 1: Running time for enumeration [sec]

#vertices	Ratio of edge-constraint			
	0%	10%	20%	30%
11	0.7644	0.1428	0.0371	0.0105
12	3.9967	0.6373	0.0841	0.0189
13	17.1927	2.8054	0.2216	0.0343
14	80.0496	8.7050	0.6788	0.0682
15	381.2840	13.3104	0.6337	0.1061
16	1780.8712	63.8610	3.5101	0.4447

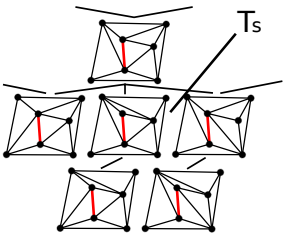

Figure 2: Generation of triangulation from a seed triangulation.

4 Generation

The previous section, we experimentally showed that an enumeration needs too high computation cost. However, we wish to obtain good triangulations. In this paper, we traverse a part of the tree structure among edge-constrained triangulations defined in [2], then generate a part of triangulations. This method allow us to generate a lot of triangulations similar to a target triangulation. That is, if we have a good triangulation, then we can obtain a lot of triangulations similar to the good triangulations. Now, we describe our generation algorithm below.

(1) We compute a "seed" triangulation T_s with desirable conditions and we then store it into a queue Q.
(2) Let T be a triangulation obtained by popping from Q. We generate a triangulation by flipping every flippable edge in T and store it into Q.
(3) We repeat the process (2) until a designated condition is satisfied (for example, a designated number of triangulations are generated or we repeat the process in designated time.

The above way makes triangulations similar to T_s, as shown in Figure 4.

In this report, we use a edge-constrained Delaunay triangulation as a seed triangulation T_s. It is know that Delaunay triangulations have the property that the interior angles and areas of triagles are as uniform as possible. This property is suitable for surfaces of 3D objects [3]. Therefore, we adopt edge-constrained Delaunay triangulations as seeds.

Table 4 shows our experimental results. Experimental environment is the same as the previous section. We uniformly generate 10 point sets of n points, for each $n = 100, 300, 500, 1000$. Let p be a ratio of constrained edges ($p = 0, 10, 20, 30$). We show the average running time that the algorithm takes to generate 1000 triangulations. The running time includes the time for constructing a seed triangulation (an edge-constrained Delaunay triangulation). We use the algorithm by Ito, et al. [1] for constructing an edge-constrained Delaunay triangulation. We allow duplications of generated triangulations. When $n = 1000$ and $p = 30$, it took much time to construct a seed triangulation, and hence we cannot measure the running time in Table 4.

Table 2: Average running time for generating 1000 triangulations [sec]

#vertices	Ratio of constrained edges			
	0%	10%	20%	30%
100	1.518	1.425	1.522	2.469
300	14.988	15.417	16.667	25.086
500	45.708	47.738	63.265	79.927
1000	371.060	252.955	399.793	-.-

Since the number of generated triangulations are restricted, the running time is faster than the enumeration before. However, when $p = 30$, the running time increased, since the algorithm take much time to generate a seed triangulation.

References

[1] T. Ito, A. Yamada, K. Inoue, T. Furuhata, and K. Shimada. An efficient implementation of the constrained delaunay triangulation method (in japanese). *Proceedings of the 55th National Convention of IPSJ*, pp. 252–253, September 1997.

[2] N. Katoh and S. Tanigawa. Enumerating edge-constrained triangulations and edge-constrained non-crossing geometric spanning trees. *Discreate Applied Mathematics*, Vol. 157, pp. 3569–3585, 2009.

[3] K. Konno T. Kinoshita, K. Yamanaka and Y. Tokuyama. A study of geometric shape of polygons for additive manufacturing. *2018 International Workshop on Advanced Image Technology*, 2018.