Listing All Connected Plane Triangulations

Zhang-Jian Li and Shin-ichi Nakano
Department of Computer Science, Gunma University, Kiryu 376-8515, Japan
E-mail: kenlee@msc.cs.gunma-u.ac.jp nakano@cs.gunma-u.ac.jp

Abstract

A "rooted" plane triangulation is a plane triangulation with one designated vertex r and one designated edge incident to r on the outer face. In this paper we first give a simple algorithm to generate all connected rooted plane triangulations with at most m edges. The algorithm uses $O(m)$ space and generates such triangulations in $O(1)$ time per triangulation without duplications. The algorithm does not output entire triangulations but the difference from the previous triangulation. By modifying the algorithm we can generate all connected (non-rooted) plane triangulations with at most m edges in $O(m^3)$ time per triangulation.

Key words: Plane triangulations, Reverse search technique, Listing.

1 Introduction

Generating all graphs with some property without duplications has many applications, including unbiased statistical analysis[M98]. A lot of algorithms to solve these problems are already known [A96, B80, LN01, M98, N01, etc]. See nice textbooks [G93, KS98].

In this paper we wish to generate all connected "rooted" plane triangulations, which will be defined precisely in Section 2, with at most m edges. Such triangulations play an important role in many algorithms, including graph drawing algorithms [FPP90, S90, etc].

To solve these all-graph-generating problems some types of algorithms are known.

Classical method algorithms [G93, p57] first generate all graphs with the given property allowing duplications, but output only if the graph has not been output yet. Thus this method requires quite a huge space to store a list of graphs that have already been output. Furthermore, checking whether each graph has already been output requires a lot of time.

Orderly method algorithms [G93, p57] do not need to store the list, since they output a graph only if it is a "canonical" representative of each isomorphism class.

Reverse search method algorithms [A96] also do not need to store the list. The idea is to implicitly define a connected graph H such that the vertices of H correspond to all graphs with the given property, and the edges of H correspond to some relation between the graphs. By traversing an implicitly defined spanning tree of H, one can find all the vertices of H, which correspond to all the graphs with the given property.

The main idea of our algorithms is that for some problems [LN01, N01] we can define a tree (not a general graph) as the graph H of reverse search method. Thus our algorithms do not need to find a spanning tree of H, since H itself is a tree. With some other ideas we give the following two simple but efficient algorithms.

Our first algorithm generates all connected rooted plane triangulations with at most m edges. A *rooted* plane triangulation means a plane triangulation with one designated "root" vertex r and one designated "root" edge incident to r on the outer face. For instance there are nine connected rooted plane triangulations with at most three edges, as shown in Fig. 1(a). The root vertices are depicted by white circles and the root edge are depicted by thick grey lines. However, there are only five connected plane triangulations with at most three edges, see Fig. 1(b). The algorithm uses $O(m)$ space and runs in $O(h(m))$ time, where $h(m)$ is the number of nonisomorphic connected rooted plane triangulations with at most m edges. The algorithm generates triangulations without duplications. So the algorithm generates each triangulation in $O(1)$ time on average. The algorithm does not output entire triangulation but the difference from the previous triangulation. Our algorithm is simple and easily implemented.

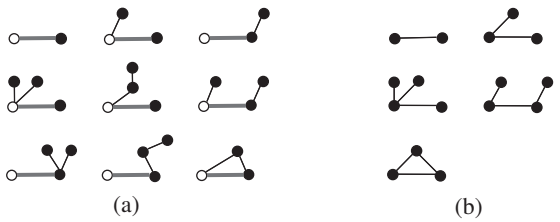

Figure 1: (a) Connected rooted plane triangulations, and (b) Connected plane triangulations.

By modifying our first algorithm we can generate all connected (non-rooted) plane triangulations with at most m edges in $O(m^3)$ time per triangulation.

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 shows a tree structure

among connected rooted plane triangulations. Section 4 presents our first algorithm. By modifying the algorithm we give one more algorithm. Finally Section 5 is a conclusion.

2 Preliminaries

In this section we give some definitions.

Let G be a connected graph with $m > 0$ edges. An edge connecting vertices u and w is denoted by (u,w). The *degree* of a vertex v is the number of neighbors of v in G.

A graph is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. A *plane* graph is a planar graph with a fixed planar embedding. A plane graph divides the plane into connected regions called *faces*. The unbounded face is called *the outer face*, and other faces are called *inner faces*. We regard *the contour* of a face as the clockwise cycle formed by the vertices on the boundary of the face. We denote the contour of the outer face of plane graph G by $C_o(G)$. For instance, in Fig. 2, $C_o(G) = v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_5, v_7, v_8, v_5, v_4, v_9, v_1, v_0(= r)$. A plane graph is called *a plane triangulation* if each inner face has exactly three edges on its contour. A *rooted* plane triangulation is a plane triangulation with one designated vertex r and one designated edge incident to r on the contour of the outer face. The designated vertex is called *the root vertex*, and the designated edge is called *the root edge*.

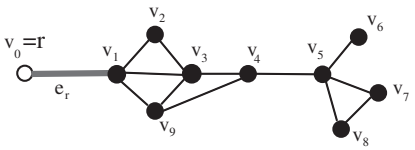

Figure 2: A connected rooted plane triangulation.

3 The genealogical tree

Let S_m be the set of all connected rooted plane triangulations with at most m edges. In this section we explain a tree structure among the triangulations in S_m.

Let G be a connected rooted plane triangulation having two or more edges. Let r be the root vertex of G, and $C_o(G) = v_0(= r), v_1, v_2, \cdots, v_{k-1}, v_0(= r)$. Let $e_r = (v_{k-1}, v_0)$ be the root edge of G.

We classify the edges on $C_0(G)$ into three types as follows. If e on $C_0(G)$ is included in a cycle of G then e is *a cycle edge*. Otherwise, if at least one end vertex of e has degree 1 then e is *a pendant*. Otherwise e is *a bridge*. If we remove a pendant or a bridge from G then the resulting triangulation is disconnected. For instance, in Fig. 2, edge (v_1, v_2) is a cycle edge, edge (v_5, v_6) is a pendant, and edge (v_4, v_5) is a bridge.

An edge $e \neq e_r$ on $C_o(G)$ is *removable* if, after removing e from G, the remaining edges induce a connected triangulation. Thus each edge $e \neq e_r$ is removable if and only if e is either a pendant or a cycle edge on $C_o(G)$.

We have the following lemma.

Lemma 3.1 *Every connected rooted plane triangulation with two or more edges has at least one removable edge.*

Proof. Let G be a connected rooted plane triangulation having two or more edges. Let $e_1 \neq e_r$ be the first edge on $C_o(G)$, then e_1 must be one of the three types as described above, a bridge, a pendant or a cycle edge. If e_1 is a pendant or a cycle edge, then it is removable, and we are done. Otherwise e_1 is a bridge, then on $C_o(G)$ the succeeding edge is either a pendant, a bridge or a cycle edge. By repeating this procedure we can find at least one pendant or cycle edge, which is removable. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

If $e_a = (v_{a-1}, v_a)$ is removable but none of $(v_0, v_1), (v_1, v_2), \cdots, (v_{a-2}, v_{a-1})$ is removable, then e_a is called *the first removable edge* of G. We can observe that if e_a is the first removable edge then each of $(v_0, v_1), (v_1, v_2), \cdots, (v_{a-2}, v_{a-1})$ is a bridge. (So they are not removable.)

For each triangulation G in S_m except K_2, if we remove the first removable edge then the resulting edge-induced triangulation, denoted by $P(G)$, is also a triangulation in S_m having one less edge. Thus we can define the unique triangulation $P(G)$ in S_m for each G in S_m except K_2. We say G is a *child* triangulation of $P(G)$.

Given a triangulation G in S_m, by repeatedly removing the first removable edge, we can have the unique sequence $G, P(G), P(P(G)), \cdots$ of triangulations in S_m which eventually ends with K_2. By merging those sequences we can have *the genealogical tree* T_m of S_m such that the vertices of T_m correspond to the triangulations in S_m, and each edge corresponds to each relation between some G and $P(G)$. For instance T_4 is shown in Fig. 3, in which each first removable edge is depicted by a thick black line. We call the vertex in T_m corresponding to K_2 *the root* of T_m.

4 Algorithms

Given S_m we can construct T_m by the definition, possibly with huge space and much running time. However, how can we construct T_m efficiently only given an integer m? Our idea (also used in [LN01, N01]) is by reversing the removing procedure as follows.

Given a connected rooted plane triangulation G in S_m with at most $m-1$ edges, we wish to find all child triangulations of G. Let r be the root vertex, $C_o(G) = v_0(= r), v_1, \cdots, v_{k-1}, v_0(= r)$, and (v_{k-1}, v_0) be the root edge, and (v_{a-1}, v_a) be the first removable edge of G. We denote by $G(i)$, $0 \leq i < k$, the rooted plane triangulation obtained from G by adding a new pendant at v_i, and by $G(i,j)$,

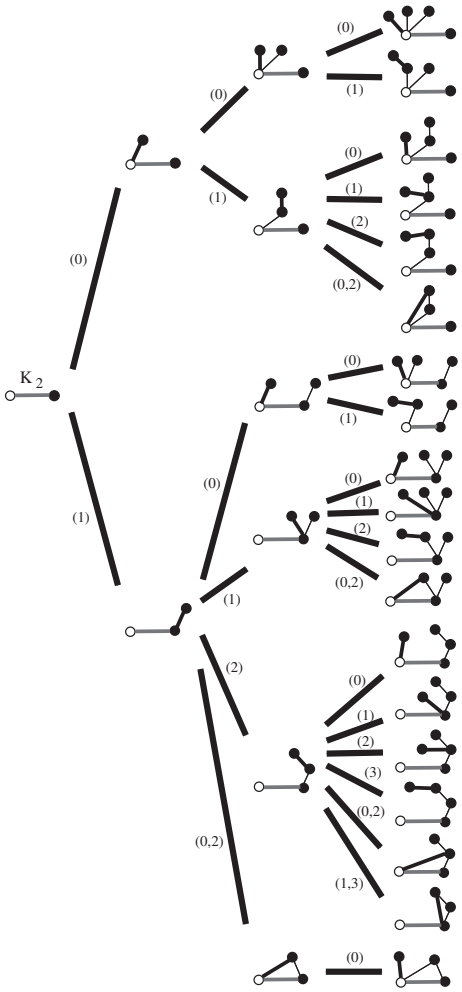

Figure 3: Genealogical tree T_4.

$0 \le i < j < k$, the rooted plane triangulation obtained from G by adding a new cycle edge connecting v_i and v_j on the outer face of G, as shown in Fig. 4. We can observe each child of G is either $G(i)$ or $G(i,j)$ for some i and j, and $G(i)$ or $G(i,j)$ is a child triangulation of G if and only if the newly added edge is the first removable edge of $G(i)$ or $G(i,j)$. Note that if $i+2 \neq j$ then $G(i,j)$ is not a plane triangulation.

Now we find all child triangulations of G. We have the following two cases to consider.

Case 1: The first removable edge (v_{a-1}, v_a) of G is a pendant.

Consider $G(i)$. For each i, $0 \le i \le a$, the newly added edge in $G(i)$ is the first removable edge of $G(i)$, and $P(G(i)) = G$. For each i, $a < i < k$, (v_{a-1}, v_a) is still the first removable edge of $G(i)$, so $P(G(i)) \neq G$.

Then consider $G(i,j)$. For each i and j such that $0 \le i \le a-2$ and $j = i+2$, the newly added edge in $G(i,j)$ is the first removable edge of $G(i,j)$, and $P(G(i,j)) = G$. For $i = a-1$, $G(i,j)$ is not a plane triangulation. For each $i \ge a$, the newly added edge in $G(i,j)$ is not the first removable edge of $G(i,j)$, since (v_{a-1}, v_a) is still removable, thus $P(G(i)) \neq G$.

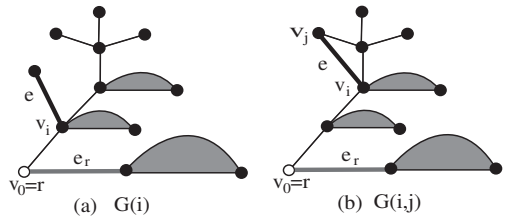

Figure 4: Illustration for (a) $G(i)$ and (b) $G(i,j)$.

Case 2: The first removable edge (v_{a-1}, v_a) of G is a cycle edge.

Consider $G(i)$. For each i, $0 \le i \le a-1$, the newly added edge in $G(i)$ is the first removable edge of $G(i)$, and $P(G(i)) = G$. For each i, $a \le i < k$, (v_{a-1}, v_a) is still the first removable edge of $G(i)$, so $P(G(i)) \neq G$.

Then consider $G(i,j)$. For each i and j such that $0 \le i \le a-2$ and $j = i+2$, the newly added edge in $G(i,j)$ is the first removable edge of $G(i,j)$, and $P(G(i,j)) = G$.

For $i = a-1$ and $j = i+2$, if (v_a, v_{a+1}) is a pendant or a bridge in G, then $G(i,j)$ is a plane triangulation and $P(G(i,j)) = G$, and if (v_a, v_{a+1}) is a cycle edge in G then we consider two cases. If G has an edge (v_{a-1}, v_{a+1}) then $G(i,j)$ is not a plane triangulation. Otherwise $G(i,j)$ is a plane triangulation, and the newly added edge in $G(i,j)$ is the first removable edge of $G(i,j)$, thus $P(G(i,j)) = G$.

Otherwise, $i \ge a$ holds. Now edge (v_{a-1}, v_a) is still removable in $G(i,j)$, so $P(G(i)) \neq G$.

Based on the case analysis above we can find all child triangulations of any given triangulation in S_m. If G has l child triangulations, then we can find them in $O(l)$ time. This is an intuitive reason why our algorithm generates triangulations in $O(1)$ time per triangulation.

Recursively repeating this process from the root of T_m corresponding to K_2 we can traverse T_m without constructing whole T_m. During the traversing of T_m, we assign a label (i) or (i,j) to each edge connecting G and either $G(i)$ or $G(i,j)$ in T_m, as shown in Fig. 3. Each label denotes how to add a new edge to G to generate a child triangulation $G(i)$ or $G(i,j)$, and each sequence of labels on a path starting from the root specifies a triangulation in S_m. For instance, the sequence $(1)(0,2)(0)$ specifies the right-bottom triangulation in Fig. 3. During our algorithm we will maintain these labels only on the path from the root to the "current" vertex of T_m responding to the "current"triangulation being generated, because those are enough information to generate the "current" triangulation. To generate next triangulation, we need to maintain some more information (the first removable edge (v_{a-1}, v_a), and $C_o(G)$, etc.) only for the triangulations on the "current" path, which has length at most m. This is an intuitive reason why our algorithm uses only $O(m)$ space, while the number of triangulations may not be bounded by a polynomial in m.

Our algorithm is as follows.

```
Procedure find-all-child-triangulations(G)
begin
01  Output G { Output the difference from the previous triangulation}
02  Assume (v_{a-1}, v_a) is the first removable edge of G
03  if G has exactly m edges then return
04  if (v_{a-1}, v_a) is a pendant then  {Case 1}
05  begin
06    for i = 0 to a
07      find-all-child-triangulations(G(i))
08    for i = 0 to a-2
09      find-all-child-triangulations(G(i, i+2))
10  end
11  else (v_{a-1}, v_a) is a cycle edge then  {Case 2}
12  begin
13    for i = 0 to a-1
14      find-all-child-triangulations(G(i))
15    for i = 0 to a-2
16      find-all-child-triangulations(G(i, i+2))
17    if (v_{a-1}, v_{a+1}) is not an edge of G then
18      find-all-child-triangulations(G(a-1, a+1))
19  end
end

Algorithm find-all-triangulations(T_m)
begin
1  output K_2
2  G = K_2
3  find-all-child-triangulations(G(0))
4  find-all-child-triangulations(G(1))
end
```

Lemma 4.1 *The algorithm uses $O(m)$ space and runs in $O(h(m))$ time, where $h(m)$ is the number of nonisomorphic connected rooted plane triangulations with at most m edges.*

Proof. Omitted.

We modify our algorithm so that it outputs all (non-rooted) plane triangulations having at most m edges, as follows. At each leaf v of the genealogical tree T_m, the triangulation G corresponding to v is checked whether the sequence of G with the root vertex and the root edge is the lexicographically first one among the k sequence of labels of G for k choices of the root vertex and the root edge on $C_o(G)$, and only if so G is output. Thus we can output only canonical representative of each isomorphism class.

Lemma 4.2 *The algorithm uses $O(m)$ space and runs in $O(m^3 \cdot g(m))$ time, where $g(m)$ is the number of nonisomorphic connected (non-rooted) plane triangulations with at most m edges.*

Proof. For each triangulation corresponding to a leaf of T_m we construct $k \leq m$ of adding sequences for each choice of the root vertex and the root edge on $C_o(G)$, and find the lexicographically first one in $O(km)$ time. And for each output triangulation our tree may contain k of isomorphic ones corresponding to at most k choices of the root vertex and the root edge. Thus the algorithm runs in $O(m^3 \cdot g(m))$ time. The algorithm clearly uses $O(m)$ space. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

5 Conclusion

In this paper we have given two simple algorithms to generate all triangulations with some property. Our algorithms first define a genealogical tree such that each vertex corresponds to each triangulation of the given property, then output each triangulation without duplications by traversing the genealogical tree.

To find other all-something-generating problems to which our method can be applied remains as an open problem. By a similar framework we have already solved listing problems for biconnected plane triangulations [LN01] and triconnected plane triangulations [N01].

References

[A96] D. Avis, *Generating rooted triangulations without repetitions*, Algorithmica, 16, (1996), pp.618-632.

[B80] T. Beyer and S. M. Hedetniemi, *Constant time generation of rooted trees*, SIAM J. Comput., 9, (1980), pp.706-712.

[FPP90] H. de Fraysseix, J. Pach and R. Pollack, *How to draw a planar graph on a grid*, Combinatorica, 10, (1990), pp.41-51.

[G93] L. A. Goldberg, *Efficient algorithms for listing combinatorial structures*, Cambridge University Press, New York, (1993).

[KS98] D. L. Kreher and D. R. Stinson, *Combinatorial algorithms*, CRC Press, Boca Raton, (1998).

[LN01] Zhangjian Li and S. Nakano, *Efficient Generation of Plane Triangulations without Repetitions*, Proc. ICALP2001, accepted.

[M98] B. D. McKay, *Isomorph-free exhaustive generation*, J. of Algorithms, 26, (1998), pp.306-324.

[N01] S. Nakano, *Efficient Generation of Triconnected Plane Triangulations*, Proc. COCOON2001, accepted.

[S90] W. Schnyder, *Embedding planar graphs on the grid*, Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms, San Francisco, (1990), pp.138-148.