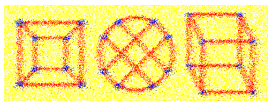
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 15, no. 3, pp. 457–482 (2011)

Generating All Triangulations of Plane Graphs

Mohammad Tanvir Parvez[1] *Md. Saidur Rahman*[1] *Shin-ichi Nakano*[2]

[1]Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka-1000, Bangladesh.
[2]Department of Computer Science, Gunma University, Gunma 376-8515, Japan.

Abstract

In this paper, we deal with the problem of generating all triangulations of plane graphs. We give an algorithm for generating all triangulations of a triconnected plane graph G of n vertices. Our algorithm establishes a tree structure among the triangulations of G, called the "tree of triangulations," and generates each triangulation of G in $O(1)$ time. The algorithm uses $O(n)$ space and generates all triangulations of G without duplications. To the best of our knowledge, our algorithm is the first algorithm for generating all triangulations of a triconnected plane graph; although there exist algorithms for generating triangulated graphs with certain properties. Our algorithm for generating all triangulations of a triconnected plane graph needs to find all triangulations of each face (a cycle) of the graph. We give an algorithm to generate all triangulations of a cycle C of n vertices in time $O(1)$ per triangulation, where the vertices of C are numbered. Finally, we give an algorithm for generating all triangulations of a cycle C of n vertices in time $O(n^2)$ per triangulation, where vertices of C are not numbered.

Key words: Triangulation; Graph; Cycle; Plane Graph; Genealogical Tree.

Submitted:	Reviewed:	Revised:	Accepted:
May 2009	October 2009	July 2010	October 2010
	Final: November 2010	Published: July 2011	
Article type: Regular paper		Communicated by: S. Das and R. Uehara	

E-mail addresses: tanvirparvez@cse.buet.ac.bd (Mohammad Tanvir Parvez) saidurrahman@cse.buet.ac.bd (Md. Saidur Rahman) nakano@cs.gunma-u.ac.jp (Shin-ichi Nakano)

1 Introduction

In this paper, we consider the problem of generating all triangulations of plane graphs. Such triangulations have many applications in Computational Geometry [5, 11], VLSI floorplanning [14], and Graph Drawing [10]. The main challenges in finding algorithms for generating all triangulations are as follows. Firstly, the number of such triangulations is exponential in general, and hence listing all of them requires huge time and computational power. Secondly, generating algorithms produce huge output, and storing these output may dominate the running time. For this reason, reducing the amount of output is essential. Thirdly, checking for any repetitions must be very efficient. Storing the entire list of objects generated so far will not be efficient, since checking each new object with the entire list to prevent repetition would require huge amount of memory and overall time complexity would be very high.

There have been a number of methods for generating combinatorial objects. Classical algorithms first generate combinatorial objects allowing duplications, but output only if the object has not been output yet. These methods require huge space to store the list and a lot of time to check duplications. Orderly algorithms [9] do not need to store the list of objects generated so far, they output an object only if it is a canonical representation of an isomorphism class. Reverse search algorithms [2] 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 the objects with the given property, and the edges of H correspond to some relation between the objects. By traversing an implicitly defined spanning tree of H, that means by checking each possible child of each vertex recursively, one can find all the vertices of H, which correspond to all the graphs with the given property.

There are some well known results for triangulating simple polygons and finding bounds on the number of operations required to transform one triangulation into another [4, 7, 15]. Researchers also have focused their attention on generating triangulated polygons and graphs with certain properties. Hurtado and Noy [6] built a tree of triangulations of convex polygons with any number of vertices. Their construction is primarily of theoretical interests; also all the triangulations of convex polygons with number of vertices up to n need to be found before finding the triangulations of a convex polygon of n vertices. Also, in [6] the authors did not discuss the time complexity of their method for generating the tree of triangulations of convex n-gons. Li and Nakano [8] gave an algorithm to generate all biconnected "based" plane triangulations with at most n vertices. Their algorithm generates all graphs with some properties without duplications. Here also, the biconnected "based" plane triangulations of n vertices are generated after the biconnected based plane triangulations of less than n vertices are generated. Hence, if we need to generate the triangulations of a convex polygon or a plane graph of exactly n vertices, existing algorithms generate all the triangulations of convex polygons or plane graphs with less than n vertices. This is not an efficient way of generation.

There are a number of works concerning enumeration and generation of pla-

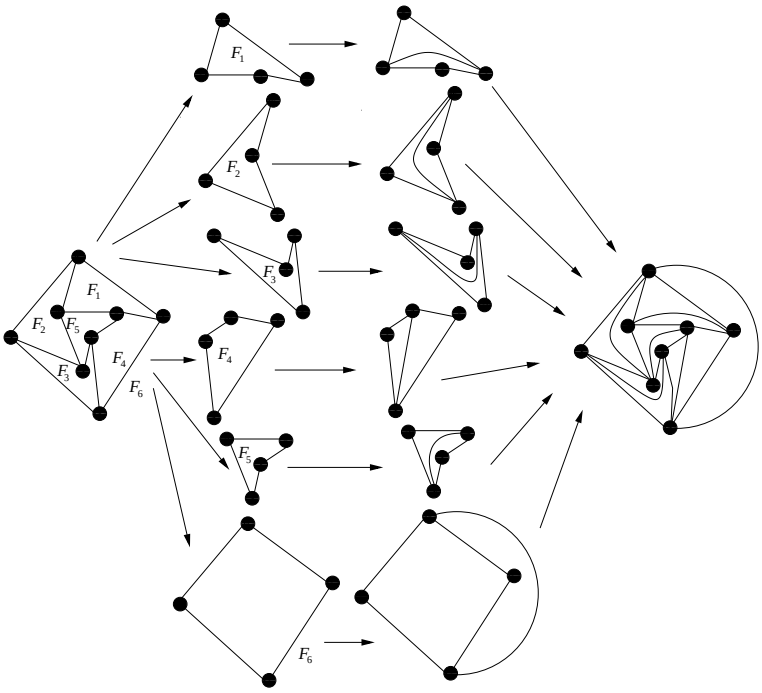

Figure 1: Illustration of the algorithm for generating the triangulations of a triconnected plane graph.

nar triangulations. Triangulations of convex polygons (with k vertices) are in bijection with binary (rooted) trees having $k-2$ inner nodes. There exist a number of efficient algorithms that generate all such trees with local perturbations, allowing to run in $O(1)$ per tree [11]. The optimal algorithm for encoding and generating planar triangulations given in [13] may work for triangulations without interior points. However, the algorithm in [13] deals with random triangulations, rather than exhaustive generations. There also exists work [3] concerning the generation of triangulations of n points in the plane based on a tree of triangulations and a lexicographic way of generating triangulations, with $O(\log\log n)$ time complexity per triangulation.

We now give the idea behind our algorithm for generating all triangulations of a triconnected plane graph G. Consider Figure 1. For a particular triconnected plane graph G, we treat each face of G as a cycle and find triangulations of those cycles. These triangulations of the cycles correspond to particular triangulations of the faces of the graph G. Combining the triangulations of the faces of G gives us a particular triangulation of G. Therefore, to generate all triangulations of G we need to triangulate those intermediate cycles in all possible ways and combine the triangulations efficiently so as to find all triangulations of G.

Therefore, in this paper, we also give an algorithm to generate all triangulations of a cycle C of n vertices in $O(1)$ time per triangulation, where the vertices of C are numbered sequentially. In our algorithm, a new triangulation of C is generated from an existing one by making a constant number of changes. The main feature of our algorithm is that we define a tree structure, based on parent-child relationships, among those triangulations. In such a "tree of triangulations," each node corresponds to a triangulation of C and each node is generated from its parent in constant time, with unique parent-child relationship to avoid duplications. In our algorithm, we construct the tree structure among the triangulations in such a way that the parent-child relationship is unique, and hence there is no chance of producing duplicate triangulations. Our algorithm also generates the triangulations *in place*, that means, the space complexity is only $O(n)$.

Our algorithm for generating all triangulations of a triconnected plane graph G generates each triangulation of G in $O(1)$ time and uses our algorithm for generating all labeled triangulations of a cycle C. We also give the idea to generate all *unlabeled triangulations* of a cycle C of n vertices in time $O(n^2)$ per triangulation, where the vertices of C are not numbered. Our algorithm for generating all triangulations of a cycle can be used for finding all triangulations of a simple polygon with "curved" diagonals and for finding all triangulations of a convex polygon with "straight" diagonals. An early version of this paper is presented at [12].

Note that, the generating algorithms proposed in this paper have some similarities with the reverse search paradigm [2]. These similarities include the use of an edge flipping operation, the definition of a spanning tree of the adjacency graph of triangulations and the efficient use of memory. However, the main feature of our algorithm for generating all triangulations of a plane graph G is that we define a tree structure *explicitly* among the triangulations of G , called "tree of triangulations". By traversing this explicitly defined spanning tree, that means by listing each exact child of each vertex recursively, our algorithm output the triangulations of G. In contrast to the reverse search paradigm, our algorithm does not need to find any "graph of triangulations" of G from which it is necessary to find a spanning tree.

However, the reverse search paradigm could be used to design enumeration algorithms for the triangulations of a plane graph and the triangulations of a cycle. Avis [1] presented two algorithms based on the reverse search paradigm [2] for enumerating all 3-connected (2-connected) r-rooted triangulations of n vertices (with r vertices on the outer face). Such an algorithm can enumerate all the triangulations of a cycle (the case where $r = n$, with no interior points). However, the time complexity of the algorithms in [1] is $O(n^2 g(n, r))$, where $g(n, r)$ is the number of objects to enumerate (e.g. $O(n^2)$ per triangulation for a cycle of n vertices). This is clearly different from the $O(1)$ time per triangulation of a cycle of n vertices achieved by our algorithm. This is due to our novel way of generating a triangulation from its parent in the tree of triangulations. Moreover, in addition to a cycle, we have achieved $O(1)$ time complexity per triangulation for generating all triangulations of a triconnected

plane graph.

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 gives the outline of the algorithm for generating all triangulations of a triconnected plane graph. Section 4 deals with generating all labeled triangulations of a cycle. Section 5 deals with generating all unlabeled triangulations of a cycle, where the vertices are not numbered. Finally, Section 6 is the conclusion.

2 Preliminaries

In this section, we define some terms used in this paper.

Let $G = (V, E)$ be a connected simple graph with vertex set V and edge set E. An edge connecting vertices v_i and v_j in V is denoted by (v_i, v_j). The *degree* of a vertex v is the number of edges incident to v in G. The *connectivity* $\kappa(G)$ of a graph G is the minimum number of vertices whose removal results in a disconnected graph or a single vertex graph. A graph is *k-connected* if $\kappa(G) \geq k$.

A graph G 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 embedding in the plane. A plane graph divides the plane into connected regions called *faces*. The unbounded face is called the *outer face* and the other faces are called *inner faces*. A plane graph is called a *plane triangulation* if each face boundary contains exactly three edges. In this paper, our notion of plane triangulation is such that the edges are not necessarily straight lines.

A cycle C in a graph G is a sequence of distinct vertices $v_1, v_2, \cdots, v_k$ such that $(v_1, v_k) \in E$ and $(v_i, v_{i+1}) \in E$ for $1 \leq i < k$. A cycle C in a plane graph G divides the plane into two regions; one region is inside of C and the other region is outside of C. We call the region which is inside of a cycle C the *inner region of* C and call the other region of C the *outer region of* C. If a graph G is a cycle then both of its inner region and outer region are faces.

Let C be a cycle corresponding to the boundary of a face F of G. Let $v_1, v_2, \cdots, v_n$ be the labels of the vertices on C in counterclockwise order. We represent C by listing its vertices as $C =< v_1, v_2, \cdots, v_n >$. We call an edge (v_i, v_j) between two vertices v_i and v_j on C a *chord* of C if (v_i, v_j) is not on C and contained in the face F of G. A chord (v_i, v_j) divides the cycle into two cycles: $v_i, v_{i+1}, \cdots, v_j$ and $v_j, v_{j+1}, \cdots, v_i$. A decomposition of a cycle into triangles by a set of non-intersecting chords is called a *triangulation of the cycle*.

Figure 2 shows two different triangulations of a cycle C. In a triangulation T of C, the set of chords is maximal, that means, every chord not in T intersects some chord in T. The sides of triangles in the triangulation are either the chords or the sides of the cycle. We say a vertex y is *visible* from a vertex x in a triangulation T of a cycle C if there exists a chord (x, y) of C in T.

In this paper, we represent each triangulation T of C by listing its chords. For example, the triangulation of Figure 2(a) is represented by $T = \{(v_4, v_6), (v_2, v_6), (v_2, v_4)\}$. Given the list of chords, we can uniquely construct the corresponding

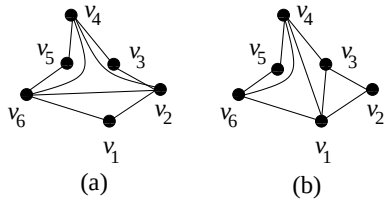

Figure 2: Two ways of triangulating a cycle of 6 vertices.

triangulation.

A triangulation of a cycle C of n vertices is called a *labeled triangulation*, if the vertices of C are numbered sequentially from v_1 to v_n. The triangulations of Figures 3(a) and 3(b) are labeled triangulations. On the other hand, if the vertices of C are not numbered, then the triangulations of C are called *unlabeled triangulations*. Both the triangulations of Figures 3(c) and 3(d) are unlabeled triangulations.

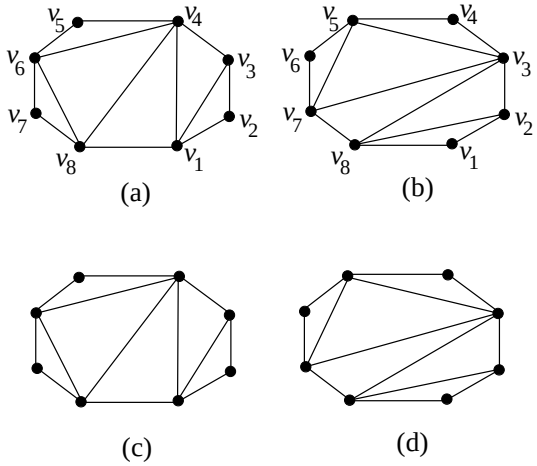

Figure 3: Illustration of labeled and unlabeled triangulations.

3 Triangulations of a Triconnected Plane Graph

In this section, we give an algorithm for generating all triangulations of a triconnected plane graph G of n vertices. Our idea is to define a parent-child relationship among the triangulations of G such that all the triangulations of G form a tree structure. Our algorithm generates the triangulations of G in the order they are present in that tree, called "tree of triangulations", without storing or building the entire tree at once in the memory.

Assume that G has k faces, arbitrarily labeled $F_1, F_2, \cdots, F_k$. For each face

F_i of G, there is a cycle C_i *associated* with F_i. Here, C_i has equal number of vertices as F_i and the labels of the vertices of C_i are similar to the vertices of F_i. A particular triangulation of F_i, denoted $T(F_i)$, corresponds to a triangulation of C_i, denoted $T(C_i)$. Therefore, triangulating the face F_i in all possible ways is equivalent to triangulating the cycle C_i in all possible ways. This is true since a triconnected plane graph has a unique embedding once the outer face of the graph is fixed. Therefore, our algorithm for generating all triangulations of a triconnected plane graph needs to find all triangulations of each cycle, the details of that is given in Section 4.

Let T be a triangulation of a cycle C of n vertices. To generate a new triangulation from T, we use the following operation. Let (v_i, v_j) be a shared chord of two adjacent triangles of T which form a quadrilateral $\langle v_q, v_i, v_r, v_j\rangle$. If we remove the chord (v_i, v_j) from T and add the chord (v_q, v_r), we get a new triangulation T'. The operation above is well known as *flipping*. Therefore, we *flip* the edge (v_i, v_j) to generate a new triangulation T', which we denote by $T(v_i, v_j)$.

For example, in Figure 4(a), the two triangles $\langle v_1, v_2, v_3\rangle$ and $\langle v_1, v_3, v_4\rangle$ form the quadrilateral $\langle v_1, v_2, v_3, v_4\rangle$ and (v_1, v_3) is the shared chord. We remove (v_1, v_3) from the triangulation of Figure 4(a) and add the chord (v_2, v_4) to generate the triangulation of Figure 4(b). Thus, we flip the chord (v_1, v_3) of the triangulation of Figure 4(a) to generate the triangulation of Figure 4(b).

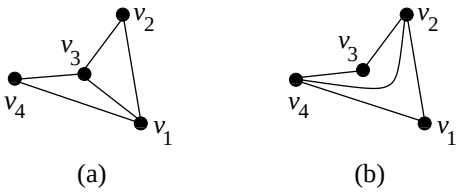

Figure 4: Illustration of flipping operation; (a) old triangulation and (b) new triangulation.

Similarly, the operation of flipping an edge of the triangulation $T(F_i)$ is defined as the flipping of the corresponding chord of the triangulation $T(C_i)$. Therefore, to generate new triangulations of the plane graph G from an existing triangulation T of G, we flip some edges of T. In our algorithm, we define the parent-child relationship among the triangulations of G in such a way that every triangulation of G, except the root triangulation, is generated from its parent by a single edge flip. Such a tree of triangulations of a triconnected plane graph G is called a *genealogical tree* and denoted by $\mathcal{T}(G)$. The genealogical tree of the triconnected plane graph G of Figure 5(a) is shown in Figure 5(b).

This definition of flipping requires G to be triconnected. This is because, if G has a cut set of two vertices, then some flip operations may introduce multi-edges. If G is triconnected then any flip operation will generate a new triangulation of G. Note that, while generating new triangulations from an existing triangulation T of G, the edges of the graph G cannot be flipped.

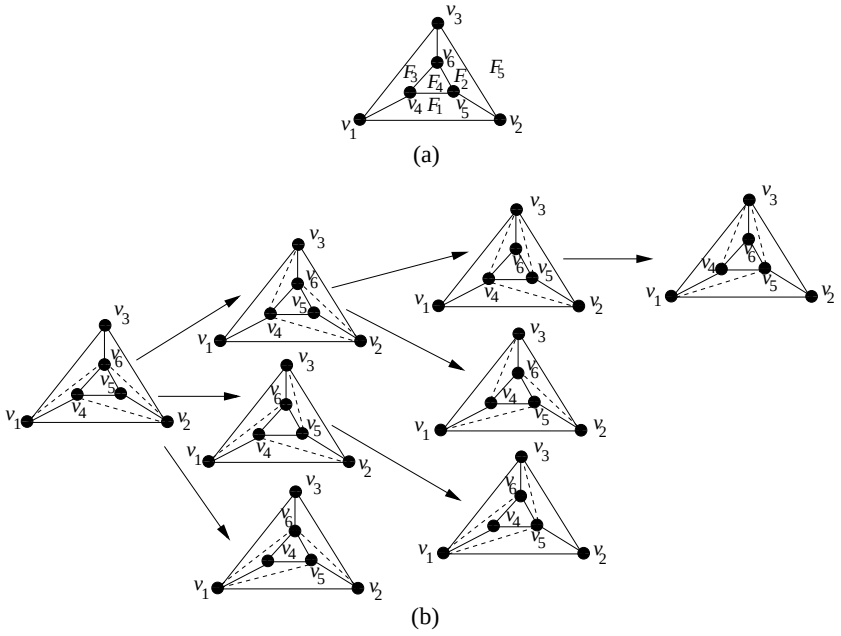

Figure 5: Illustration of (a) a triconnected plane graph G with five faces and (b) genealogical tree $\mathcal{T}(G)$ of G.

Therefore, for a triangulation T of G, we need to classify edges of T as flippable and non-flippable. We introduce the related concepts below.

Let T be a triangulation of a cycle C of n vertices. The chords of T which can be flipped to generate new triangulations of C are called *generating chords* of T. In Section 4, we describe the way to find the generating chords of T. The set of generating chords of T is called the *generating set*. The triangulation $T(F_i)$ of the face F_i of G has a generating set of edges equivalent to the generating set of the triangulation $T(C_i)$. Therefore, to generate new triangulations of G, we flip an edge from the generating set of a face F_i of G.

The rest of this section is organized as follows. In Section 3.1 we define the root triangulations of $\mathcal{T}(G)$. We give the detail algorithm for generating all triangulations of G in Section 3.2.

3.1 Finding the Root

In this section, we describe the procedure for finding the root triangulation of the genealogical tree $\mathcal{T}(G)$ of a triconnected plane graph G of n vertices.

Let F_i be a face of G. We can represent F_i as a list of vertices on the boundary of F_i. We choose a vertex v_j on the boundary of F_i arbitrarily and call it the *root vertex* of F_i. Let C_i be the cycle associated with F_i. Then v_j is also called the root vertex of C_i. Consider the triangulation $T(C_i)$ of C_i where

all the chords of $T(C_i)$ are incident to the root vertex v_j. This triangulation $T(C_i)$ of C_i gives us a triangulation of the face F_i of G. Once all the faces of G are triangulated in that way, we get a triangulation T of the graph G itself. In our algorithm, such a triangulation T of G is taken as the root triangulation T_R of the genealogical tree $\mathcal{T}(G)$. Note that, the choice of the root triangulation T_R will depend on the way the root vertices are chosen.

The procedure for finding T_R and corresponding generating sets is as follows. We traverse the face F_i of G to find the generating set of $T(F_i)$, denoted by GS_i, using the doubly connected adjacency list representation of G [10]. Face F_i can be traversed in time proportional to the number of vertices on the boundary of it. Assume that we traverse the face F_i clockwise starting at vertex v_j and take v_j as the root vertex of C_i. Let v_k, v_l and v_m be three consecutive vertices on the boundary of F_i, where $v_k \neq v_j$ and $v_m \neq v_j$. We add the edge (v_j, v_l) to the generating set GS_i of $T(F_i)$. The generating set GS^o of the root triangulation of $\mathcal{T}(G)$ can be found as follows. Let $GS_1, GS_2, \cdots, GS_k$ be the k generating sets of the triangulations $T_1, T_2, \cdots, T_k$. Initially, GS^o is empty. Once we find a generating set GS_i, we concatenate GS_i with the existing list GS^o. When all the GS_is are generated and concatenated with GS^o, we get the generating set of the root triangulation of $\mathcal{T}(G)$.

For example, Figure 6(a) shows a triconnected plane graph G of eight vertices. One possible root in $\mathcal{T}(G)$ is shown in Figure 6(b). In Figure 6(b), $GS_1 = \{(v_1, v_8)\}$, $GS_2 = \{(v_2, v_5)\}$, $GS_3 = \{(v_3, v_6)\}$, $GS_4 = \{(v_3, v_8)\}$ and $GS_5 = \{(v_6, v_8)\}$. Thus $GS^0 = \{(v_1, v_8), (v_2, v_5), (v_3, v_6), (v_3, v_8), (v_6, v_8)\}$.

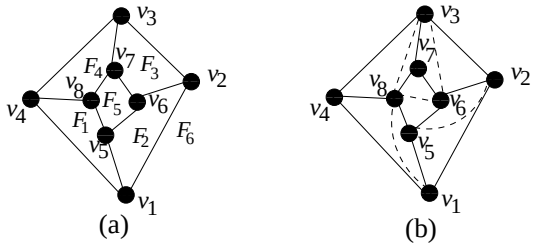

Figure 6: Illustration of (a) a triconnected plane graph G of eight vertices (b) corresponding root in $\mathcal{T}(G)$.

We now have the following lemma.

Lemma 1 *Let G be a triconnected plane graph of n vertices. Then the root triangulation T_R of the genealogical tree $\mathcal{T}(G)$ of G can be found in $O(n)$ time.*

Proof: In a triconnected plane graph G of n vertices, the number of edges is less than $3n - 6$. The number of faces of G is also bounded by a linear function of n. Since each face F_i of G can be traversed in time proportional to the number of edges on the boundary of F_i and each edge can be shared by at most two faces of G, traversing all the faces of G requires time proportional to the total number of edges of G. Thus the root of $\mathcal{T}(G)$ can be found in $O(n)$ time. □

3.2 The Algorithm

In this section, we give the details of the algorithm for generating all triangulations of a triconnected plane graph G of n vertices.

Assume that the triconnected plane graph G has k faces. For a particular triangulation T of G, we generate new triangulations of G from T as follows. If a triangulation T' of G is a child of a triangulation T of G in the genealogical tree, and T' is generated from T by flipping a generating edge in F_j, then we say that F_j is *eligible*. To generate all child triangulations of a triangulation T we need to find all eligible faces of T then flip each generating edge in each eligible face. We can observe that if a face F_j is eligible then for each i, $1 \leq i \leq j$, face F_i is also eligible. This simple condition for eligibility ensures efficient generation of the triangulations of G.

We now have the following algorithm for generating all triangulations of a triconnected plane graph G with k faces.

Procedure find-all-child-triangulations(T, i)
{T is the current triangulation and F_i is an eligible face of T}
begin
1 Let E_G be the set of generating edges of $T(F_i)$;
2 **if** E_G is empty **then return** ;
3 **for** each edge $e \in E_G$
4 Flip e to find a new triangulation T';
5 Output T';
{For T', all the faces F_j, $1 \leq j \leq i$, are eligible}
6 **for** $j = 1$ **to** i
7 **find-all-child-triangulations**(T', j);
end;
Algorithm find-all-triangulations(G, k)
{The triconnected plane graph G has k faces}
begin
Label the faces $F_1, F_2, \cdots, F_k$ arbitrarily;
Find root triangulation T_R of $\mathcal{T}(G)$;
Output *root* T_R;
{For the root T_R, all the faces of G are eligible}
for $i = 1$ **to** k
find-all-child-triangulations(T_R, i);
end.

The correctness of the algorithm **find-all-triangulations** depends on the correct finding of the generating set of the triangulation $T(F_i)$ of face F_i of G (Step 1). We also have to ensure that flipping the edges in the generating set of $T(F_i)$ generates all the children of $T(F_i)$ without duplications. Flipping an edge of $T(F_i)$ is equivalent to flipping a chord of the triangulation $T(C_i)$ of the cycle C_i associated with F_i. Therefore, we need to prove that for a triangulation T of a cycle C: (1) flipping the generating chords of T generates all the child triangulations of T without duplications and (2) the number of

generating chords in any child triangulation of T is less than in T. We prove both of these in Section 4.

The time and space complexity of the algorithm **find-all-triangulations** can be found as follows. From Lemma 1, finding the root triangulation T_R takes $O(n)$ time. To find the time required to generate each new triangulation T' from a triangulation T of G, note that the difference between the representations of T' and T can be found in the triangulation of only one face, say F_i (Steps 3 - 5). Assume that face F_i has the triangulation $T(F_i)$ in T and $T'(F_i)$ in T'. Now, to get the representation of T', all we need to do is to find the representation of $T'(F_i)$ from the representation of $T(F_i)$. Equivalently, the problem reduces to the following. Let T and T' be two triangulations of a cycle C and T' is generated from T by flipping a generating chord of T. Then, how can we find the representation and the generating set of T' from T efficiently? Section 4 shows that this can be done in $O(1)$ time.

To find the space complexity, note that, we can represent a triangulation T of G by listing its edges only. Therefore, it takes only $O(n)$ space to store a triangulation T. The height of the tree $\mathcal{T}(G)$ is bounded by the number of edges in T_R (since we may need to flip each generating edge T_R once to generate a triangulation of G), which is linear in n. The algorithm **find-all-triangulations** needs to store (1) the representation and generating set of the current triangulation T and (2) the information of the path from the root to the current node of the tree. This implies that the space complexity of the entire algorithm can be reduced to $O(n)$.

Therefore we have the following theorem.

Theorem 1 *The algorithm **find-all-triangulations** generates all the triangulations of a triconnected plane graph G of n vertices in time $O(1)$ per triangulation, with $O(n)$ space complexity.*

In the next section, we give the algorithm for generating all triangulations of a cycle C of n vertices, where the vertices of C are labeled.

4 Labeled Triangulations of a Cycle

In this section, we give an algorithm to generate all labeled triangulations of a cycle C of n vertices. Here we also define a unique parent for each triangulation of C so that it results in a tree structure among the triangulations of C, with a suitable triangulation as the root. Once such a parent-child relationship of C is established, we can generate all the triangulations of C using the relationship. We need not to build or to store the entire tree of triangulations at once, rather we generate each triangulation in the order (DFS order) it appears in the tree structure.

In our algorithm in this section, to make the data structures easier to manipulate, we write the edge (v_i, v_j) such that $i < j$. Thus the edge incident to vertex v_4 and v_1 is denoted by (v_1, v_4), and not by (v_4, v_1).

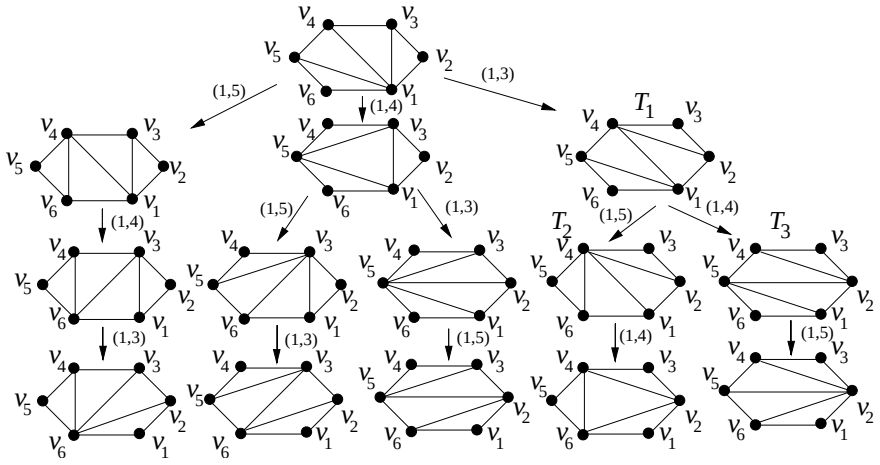

Figure 7: Genealogical tree $\mathcal{T}(C)$ of a cycle C of six vertices.

One can observe that, each triangulation of the tree of triangulations in Figure 7, except the root, is generated from its parent by flipping a single chord. Each arrow is labeled in Figure 7 to indicate which chord has been flipped to generate a particular child. We call a tree of triangulations of a cycle C of n vertices a *genealogical tree* of C and denote it by $\mathcal{T}(C)$. Figure 7 illustrates one such genealogical tree. Let T be a triangulation of C, in which all the chords of T are incident to vertex v_1. We regard T as the *root* T_r of $\mathcal{T}(C)$ and call v_1 as the root vertex of C. Note that the choice of root vertex is arbitrary in finding T_r. We can take any of the vertices of C other than v_1 as the root vertex to find a root triangulation T_r. With this definition of the root vertex of C, we describe the labeled triangulations of C as *rooted triangulations*.

Note that, in the root T_r of $\mathcal{T}(C)$, every vertex of C is visible from the root vertex v_1. We say that the root vertex v_1 has *full vision* in T_r. Obviously, in a non-root triangulation T of C, vertex v_1 does not have the full vision. The reason is that T has some "blocking chords" which are blocking some vertices of C from being visible from the root vertex v_1. A chord (v_i, v_j) of a triangulation T of C is a *blocking chord* of T if both v_i and v_j are adjacent to the root vertex of C. We say that the root vertex of C has a *blocked vision* in a non-root triangulation T of C. The following lemma characterizes the non-root triangulations of C.

Lemma 2 *Each triangulation T of a cycle $C = \langle v_1, v_2, \cdots, v_n \rangle$ has at least one blocking chord, if T is not the root of $\mathcal{T}(C)$.*

Proof: Let v_j be the vertex of C such that (v_1, v_k) is a chord of T, for all $k \geq j$. Then there exists a vertex v_i such that $i < j$ and (v_i, v_j) is a chord of T (choose i to be the minimum). Otherwise, all chords of T would be incident to v_1 and T would be the root of $\mathcal{T}(C)$. Since T is a triangulation of C, (v_1, v_i, v_j) is a triangle, and hence (v_i, v_j) is a blocking chord. □

Suppose we flip a chord (v_1, v_j) of T to generate a new triangulation T'. Let $(v_b, v_{b'})$, $b < b'$ be the newly found chord in T'. Obviously $(v_b, v_{b'})$ is a blocking chord of T'. Similarly, if we flip a blocking chord of T to generate T', the newly found chord will be non-blocking, incident to vertex v_1 in T'. For example, if we flip the chord (v_1, v_4) of the triangulation of Figure 8(a), we get the triangulation of Figure 8(b), where (v_3, v_6) is the newly found chord. This new chord (v_3, v_6) is a blocking chord of the triangulation of Figure 8(b).

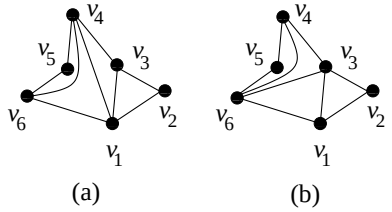

Figure 8: Illustration of the generation of a blocking chord; (a) old triangulation and (b) new triangulation.

The rest of this section is organized as follows. Section 4.1 describes child to parent relationship among the triangulations of a cycle C of n vertices. Section 4.2 deals with the generation of the children of a triangulation T in the genealogical tree $\mathcal{T}(C)$ of C. Section 4.3 describes the data structures used to represent a triangulation T of C. Finally, Section 4.4 describes the algorithm to generate all triangulations of C.

4.1 Child-Parent Relationship

It is convenient to consider the child-parent relationship before considering the parent-child relationship. Throughout the section, we denote the parent of a triangulation T by $P(T)$.

We define the child-parent relationships among the triangulations of C with two goals in mind. First, the differences between a triangulation T and its parent $P(T)$ should be minimal, so that T can be generated from $P(T)$ with the minimal effort. Second, every triangulation T of C must have a parent and only one parent in the genealogical tree $\mathcal{T}(C)$. We achieve the first goal by ensuring that the parent $P(T)$ of a triangulation T can be found by flipping a single chord of T. That means T can also be found from its parent $P(T)$ by flipping a single chord of $P(T)$. The second goal, that is the uniqueness of the parent-child relationship, can be achieved as follows.

Our idea of defining a parent-child relationship is that the parent of a triangulation T must have a "clearer vision" than T. Let T and T' be two triangulations of C. We say that T' has a *clearer vision* than T if the number of vertices visible from v_1 in T' is greater than the number of vertices visible from v_1 in T. For example, three vertices are visible from vertex v_1 in the triangulation T_2 of Figure 7, whereas four vertices are visible from vertex v_1 in the triangulation T_1 of Figure 7. Therefore T_1 has a clearer vision than T_2 in Figure 7.

We can easily get a triangulation T' from T, where T' has a clearer vision than T, by flipping a blocking chord $(v_b, v_{b'})$ of T. We say that the triangulation T' is the *parent* of T if the chord $(v_b, v_{b'})$ is the "leftmost blocking chord" of T. The chord $(v_b, v_{b'})$, $b < b'$, of T is the *leftmost blocking chord* of T if no other blocking chord of T is incident to a higher indexed vertex than $v_{b'}$ in T. For example, in Figure 7, (v_4, v_6) is the leftmost blocking chord of the triangulation T_2. Therefore we flip the chord (v_4, v_6) of T_2 to find the parent triangulation T_1 of T_2, as shown in Figure 7.

The above definition of the parent of a triangulation T of C ensures that we can always find a unique parent of a non-root triangulation T of C. From Lemma 2, a non-root triangulation T of C has at least one blocking chord. From these blocking chords of T we select the one which is the leftmost and flip that chord to find the unique parent $P(T)$ of T. Based on the above parent-child relationship, the following lemma claims that every triangulation of a cycle C of n vertices is present in the genealogical tree $\mathcal{T}(C)$.

Lemma 3 *For any triangulation T of a cycle $C = \langle v_1, v_2, \cdots, v_n \rangle$, there is a unique sequence of flipping operations that transforms T into the root T_r of $\mathcal{T}(C)$.*

Proof: Let T be a triangulation other than the root of $\mathcal{T}(C)$. Then according to Lemma 2, T has at least one blocking chord. Let $(v_b, v_{b'})$ be the leftmost blocking chord of T. We find the parent $P(T)$ of T by flipping the leftmost blocking chord of T. Since flipping a blocking chord of T results in a chord incident to vertex v_1 in the new triangulation, $P(T)$ has one more chord incident to v_1 than T. Now, if $P(T)$ is the root, then we stop. Otherwise, we apply the same procedure to $P(T)$ and find its parent $P(P(T))$. By continuously applying this process of finding the parent, we eventually generate the root triangulation T_r of $\mathcal{T}(C)$. □

Lemma 3 ensures that there can be no omission of triangulations in the genealogical tree $\mathcal{T}(C)$ of a cycle C of n vertices. Since there is a unique sequence of operations that transforms a triangulation T of C into the root T_r of $\mathcal{T}(C)$, by reversing the operations we can generate that particular triangulation, starting at the root. We give the details in the next section.

4.2 Generating the Children of a Triangulation in $\mathcal{T}(C)$

In this section, we describe the method for generating the children of a triangulation T in $\mathcal{T}(C)$.

To find the parent $P(T)$ of the triangulation T, we flip the leftmost blocking chord of T. That means $P(T)$ has fewer blocking chords than T. Therefore, the operation for generating the children of T must increase the number of blocking chords in the children of T. Intuitively if we flip a chord (v_1, v_j) of T, which is incident to vertex v_1 in T, and generate a new triangulation T', then T' contains one more blocking chord than T. We call all such chords (v_1, v_j) as the *candidate chords* of T.

Note that, flipping a candidate chord of T may not always preserve the parent-child relationship described in Section 4.1. For example, we generate the triangulation of Figure 9(b) by flipping the candidate chord (v_1, v_3) of the triangulation of Figure 9(a). The leftmost blocking chord of the triangulation of Figure 9(b) is (v_4, v_6); therefore the parent of the triangulation of Figure 9(b) is the triangulation of Figure 9(c), not the triangulation of Figure 9(a).

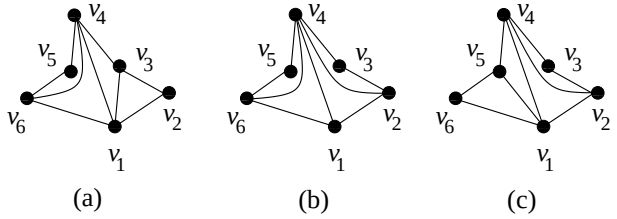

Figure 9: Illustration of a flipping that does not preserve parent-child relationship.

Therefore to keep the parent-child relationship unique, we flip a candidate chord (v_1, v_j) of T to generate a new triangulation T' if and only if flipping (v_1, v_j) of T results in the leftmost blocking chord of T'. We call such a candidate chord (v_1, v_j) of T as a *generating chord*. The generating chords of a triangulation T of C can be found as follows. Let $(v_b, v_{b'})$ be the leftmost blocking chord of a triangulation T of a cycle C of n vertices. Then (v_1, v_j) is a generating chord of T if $j \geq b$. If T has no blocking chord then all chords of T are generating chords. Thus all the chords of the root T_r of $\mathcal{T}(C)$ are generating chords. All other candidate chords of T are called *non-generating*. We call the set of generating chords of a triangulation T as the *generating set* GS of T. For example, the triangulation in Figure 10(a) is the root triangulation of $\mathcal{T}(C)$ of a cycle C of 8 vertices. Therefore, all the chords of the triangulation in Figure 10(a) are generating chords. In the triangulation of Figure 10(b), (v_1, v_4), (v_1, v_6) and (v_1, v_7) are three generating chords, whereas (v_1, v_3) is a non-generating chord.

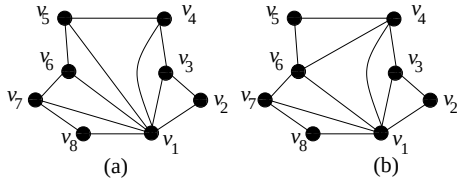

Figure 10: Illustration of generating chords.

We now have the following lemmas.

Lemma 4 *The root T_r of the genealogical tree $\mathcal{T}(C)$ of a cycle C of n vertices has $n-3$ generating chords and any other triangulations in $\mathcal{T}(C)$ has less than $n-3$ generating chords.*

Proof: The number of chords in any triangulation T of a cycle C of n vertices is $n-3$. Thus the maximum number of possible generating chords is also $n-3$. Since the root triangulation T_r has all its chords as generating, T_r contains $n-3$ generating chords. Any triangulation T other than the root T_r contains at least one blocking chord, which is not incident to vertex v_1 in T. Since generating chords must be incident to vertex v_1, any triangulation other than T_r has less than $n-3$ generating chords. □

Lemma 5 *Let (v_1, v_j) be a generating chord of a triangulation T of a cycle C of n vertices. Then flipping (v_1, v_j) in T results in the leftmost blocking chord of $T(v_1, v_j)$.*

Proof: Let $(v_r, v_{r'})$ be the leftmost blocking chord of T. We first consider the case where either $v_j = v_r$ or $v_j = v_{r'}$.

If $v_j = v_r$, then $\langle v_1, v_j, v_{r'}\rangle$ is a triangle of T (see Figure 11(a)) and after flipping (v_1, v_j) of T we get $(v_i, v_{r'})$ as a chord in $T(v_1, v_j)$, for some $i < j$ (see Figure 11(b)). Since every face of $T(v_1, v_j)$ is a triangle, $\langle v_1, v_i, v_{r'}\rangle$ is a triangle of $T(v_1, v_j)$. Therefore, $(v_i, v_{r'})$ is the blocking chord of $T(v_1, v_j)$. Since, $(v_r, v_{r'})$ is the leftmost blocking chord of T and vertex v_r is not visible from vertex v_1 in $T(v_1, v_j)$, $(v_i, v_{r'})$ is the leftmost blocking chord of $T(v_1, v_j)$

If $v_j = v_{r'}$, then $\langle v_1, v_r, v_j\rangle$ is a triangle of T (see Figure 11(c)) and after flipping (v_1, v_j) of T we get (v_i, v_r) as a chord of $T(v_1, v_j)$, for some $i > j$ (see Figure 11(d)). Since every face of $T(v_1, v_j)$ is a triangle, $\langle v_1, v_r, v_i\rangle$ is a triangle of $T(v_1, v_j)$. Therefore, (v_r, v_i) is a blocking chord of $T(v_1, v_j)$. Since, $(v_r, v_{r'})$ is a leftmost blocking chord of T and (v_r, v_i) is a blocking chord of $T(v_1, v_j)$, where $i > r'$, (v_r, v_i) is the leftmost blocking chord of $T(v_1, v_j)$

We now consider the case where $j > r'$ (see Figure 11(e)). Let $(v_q, v_{q'})$ be the chord which appears in $T(v_1, v_j)$ after flipping the chord (v_1, v_j) of T (see Figure 11(f)). Every face of $T(v_1, v_j)$ is a triangle. Thus, $\langle v_1, v_q, v_{q'}\rangle$ is a triangle of $T(v_1, v_j)$ and $(v_q, v_{q'})$ is a blocking chord of $T(v_1, v_j)$. Since, $q' > j$, we have $q' > r'$. Therefore, $(v_q, v_{q'})$ is the leftmost blocking chord of $T(v_1, v_j)$. □

Lemma 6 *Let T be a triangulation of a cycle C of n vertices. Let $T(v_1, v_j)$ be the triangulation generated by flipping the chord (v_1, v_j) of T. Then T is the parent of $T(v_1, v_j)$ in the genealogical tree $\mathcal{T}(C)$ if and only if (v_1, v_j) is a generating chord of T.*

Proof: *Necessity.* Assume that (v_1, v_j) is a non-generating chord of T. It is sufficient to show that T is not the parent of $T(v_1, v_j)$. Here, we have $j < r$ (see Figure 12(a)). Let $(v_q, v_{q'})$ be the chord which appears in $T(v_1, v_j)$ after flipping (v_1, v_j) of T (see Figure 12(b)). Since the chord (v_1, v_r) of T is also a chord of $T(v_1, v_j)$, we have $q' \leq r$. Therefore, $q < r'$. Thus, $(v_r, v_{r'})$ is the leftmost blocking chord of $T(v_1, v_j)$ and T is not the parent of $T(v_1, v_j)$.

Sufficiency. Assume that (v_1, v_j) is a generating chord of T. We show that T is the parent of $T(v_1, v_j)$ in $\mathcal{T}(C)$.

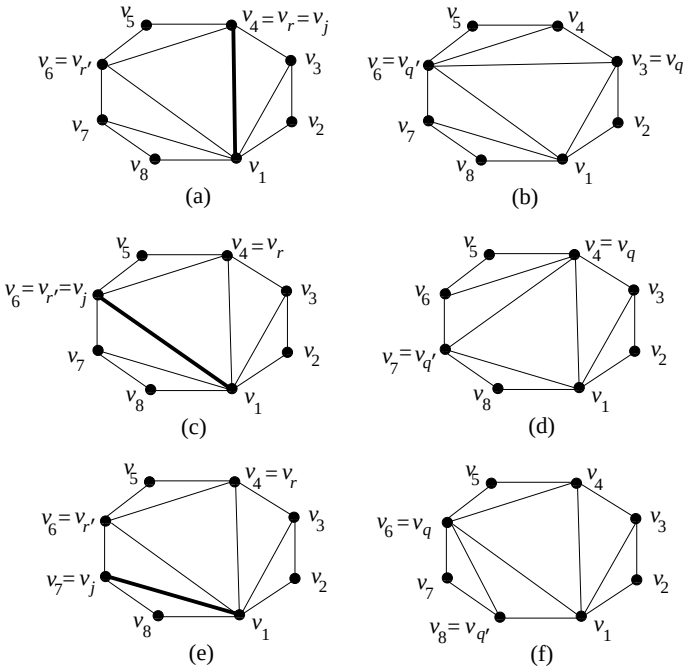

Figure 11: Illustration of Lemma 5.

Let $(v_q, v_{q'})$ be the chord which appears in $T(v_1, v_j)$ after flipping (v_1, v_j) of T. To prove that T is the parent of $T(v_1, v_j)$ in $\mathcal{T}(C)$, we must show that $(v_q, v_{q'})$ is the leftmost blocking chord of $T(v_1, v_j)$.

We first consider the case where T is the root of $\mathcal{T}(C)$. T does not have any parent and all the chords of T are incident to vertex v_1. Therefore, $(v_q, v_{q'})$ is the only chord of $T(v_1, v_j)$ which is not incident to vertex v_1. Thus, $(v_q, v_{q'})$ is the leftmost blocking chord of $T(v_1, v_j)$.

We now consider the case where T is not the root of $\mathcal{T}(C)$. Then by Lemma 5, $(v_q, v_{q'})$ is the leftmost blocking chord of $T(v_1, v_j)$. □

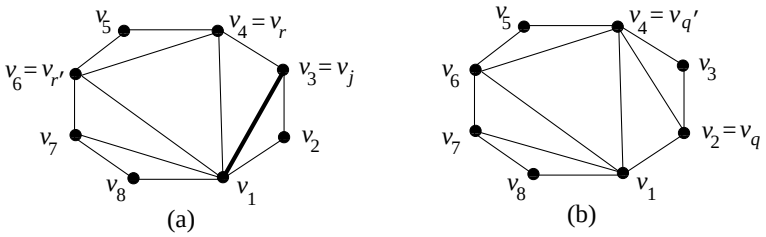

Figure 12: Illustration of Lemma 6.

According to Lemma 5 and 6, if the generating set GS of a triangulation T is

non-empty, then we can generate each of the children of T in $\mathcal{T}(C)$ by flipping a generating chord of T. Therefore, the number of children of a triangulation T in $\mathcal{T}(C)$ will be equal to the cardinality of the generating set. Thus, the following lemma holds.

Lemma 7 *The number of children of a triangulation T of a cycle C is equal to the number of chords in the generating set of T. The root of $\mathcal{T}(C)$ has the maximum number of children.*

4.3 The Representation of a Triangulation in $\mathcal{T}(C)$

In this section, we describe a data structure that we use to represent a triangulation T and that enables us to generate each child triangulation of T in constant time.

For a triangulation T of C, we maintain three lists: L, GS and O to represent T completely. Here L is the list of chords of T and GS is the generating set of T. For each chord (v_1, v_j) in the generating set GS of T, we maintain a corresponding *opposite pair* $(v_o, v_{o'})$, such that $\langle v_1, v_o, v_j, v_{o'}\rangle$ is a quadrilateral of T. Note that, $o < j$ and $o' > j$. O is the list of such opposite pairs. For example, in Figure 10(a), the generating chord (v_1, v_4) has the opposite pair (v_3, v_6).

Since we generate triangulations of C starting with the root T_r, we find the representation of T_r first. The chords of T_r are listed in L in clockwise order. That is, for T_r, $L = \{(v_1, v_{n-1}), (v_1, v_{n-2}), \cdots (v_1, v_3)\}$. The generating set GS is exactly similar to the list L of T_r: $GS = \{(v_1, v_{n-1}), (v_1, v_{n-2}), \cdots, (v_1, v_4), (v_1, v_3)\}$. The corresponding list of opposite pairs is $O = \{(v_{n-2}, v_n), (v_{n-3}, v_{n-1}), \cdots, (v_3, v_5), (v_2, v_4)\}$; that means, (v_{j-1}, v_{j+1}) is the opposite pair of (v_1, v_j) in T_r, $3 \leq j \leq n-1$.

Let $T(v_1, v_j)$ be a child triangulation of T in $\mathcal{T}(C)$ generated from T by flipping the chord (v_1, v_j) of T. Let $(v_b, v_{b'})$ be the blocking chord which appears in $T(v_1, v_j)$ after flipping (v_1, v_j) of T. The list L of $T(v_1, v_j)$ can be found easily from the representation of T by removing (v_1, v_j) from the list L of T and adding $(v_b, v_{b'})$ to it. Note that one can easily find the blocking chord $(v_b, v_{b'})$ of T', since $(v_b, v_{b'})$ is the opposite pair of (v_1, v_j) in the representation of T.

In the next section, we give the detailed algorithm for generating the triangulations of C and show that the representation of a child triangulation T' of T can be found from the representation of T in constant time.

4.4 The Algorithm

In this section, we give an algorithm to generate all triangulations of a cycle C of n vertices.

Let $v_{j_1}, v_{j_2}, \cdots, v_{j_k}$, $j_1 > j_2 > \cdots > j_k$, be the sequence of k vertices of a triangulation T of C. Here $(v_1, v_{j_1}), (v_1, v_{j_2}), \cdots, (v_1, v_{j_k})$ are the chords of T and the chords $(v_1, v_{j_i}), 1 \leq i \leq k$, are generating chords of T. Then, T has a generating set $GS = \{(v_1, v_{j_1}), (v_1, v_{j_2}), \cdots, (v_1, v_{j_k})\}$ of k generating chords,

for $0 \leq k \leq n-3$. For T_r, $GS = \{(v_1, v_{n-1}), (v_1, v_{n-2}), \cdots, (v_1, v_4), (v_1, v_3)\}$. For each chord (v_1, v_j) of T, we keep an opposite pair $(v_o, v_{o'})$ in T. O is the set of such pairs. For T_r, $O = \{(v_{n-2}, v_n), (v_{n-3}, v_{n-1}), \cdots, (v_3, v_5), (v_2, v_4)\}$ as shown in Section 4.3. We find the sets GS and O of a child T' of T by updating the lists GS and O of T while we generate T'.

We now describe a method for generating the children of a triangulation T in $\mathcal{T}(C)$. We have two cases based on whether T is the root of $\mathcal{T}(C)$ or not.

Case 1: T is the root of $\mathcal{T}(C)$.

In this case, all the chords of T are generating chords and there are a total of $n-3$ such chords in T. Any of these generating chords of T can be flipped to generate a child triangulation of T. For example, the root of the genealogical tree in Figure 7 has three generating chords; thus it has three children as shown in Figure 7.

Case 2: T is not the root of $\mathcal{T}(C)$.

Let $(v_b, v_{b'})$ be the leftmost blocking chord of T. Consider a chord (v_1, v_j) of T. If $j \geq b$, then (v_1, v_j) is a generating chord of T. Therefore, according to Lemma 6, $T(v_1, v_j)$ is a child of T in $\mathcal{T}(C)$. Thus, for all chords (v_1, v_j) of T such that $j \geq b$, a new triangulation is generated by flipping (v_1, v_j).

If $j < b$, then (v_1, v_j) is a non-generating chord of T and according to Lemma 6, we cannot flip (v_1, v_j) to generate a new triangulation from T.

Based on the case analysis above, we can generate all triangulations of C. The algorithm is as follows.

```
Procedure find-all-child-triangulations-cycle(T)
begin
  Output T; {output the difference in representation from the
        previous triangulation}
  if T has no generating chords then return ;
  Let (v_b, v_b') be the leftmost blocking chord of T;
  for all j ≥ b
    if (v_1, v_j) is a chord of T then
      find-all-child-triangulations-cycle(T(v_1, v_j));      {Case 2}
end;
Algorithm find-all-triangulations-cycle(n)
begin
  Output root T_r;
  T = T_r;
  for j = n − 1 to 3
    find-all-child-triangulations-cycle(T(v_1, v_j));      {Case 1}
end.
```

The following theorem describes the correctness and performance of the algorithm **find-all-triangulations-cycle**.

Theorem 2 *Given a cycle C of n vertices, we can generate all the triangulations of C in $O(1)$ time per triangulation, without duplications and omissions. The space complexity of the algorithm is $O(n)$.*

Proof: Let T be a triangulation of C and $T(v_1, v_j)$ be the triangulation generated from T by flipping the chord (v_1, v_j) of T. The algorithm **find-all-triangulations-cycle** generates $T(v_1, v_j)$ from T if only if (v_1, v_j) is a generating chord of T. Therefore, according to Lemma 6, T is the parent of $T(v_1, v_j)$. That means, each triangulation T of C is generated from its parent only; therefore, duplication cannot occur. To prove that no omission occurs, we use Lemma 3. Lemma 3 implies that for any triangulation T of C, there is a unique path from the root T_r to T in $\mathcal{T}(C)$. Thus, to show that the algorithm **find-all-triangulations-cycle** does not omit any triangulation, it is sufficient to prove that the algorithm **find-all-triangulations-cycle** generates all the children of a triangulation T. By Lemma 6, to generate the children of a triangulation T, only the generating chords of T need to be flipped. Since the algorithm **find-all-triangulations-cycle** flips all the generating chords of a triangulation T to generate new triangulations from T, all the children of T in $\mathcal{T}(C)$ are generated.

The complexity of the algorithm can be found as follows. We need to store the generating set GS for the current triangulation T of C. Since the maximum cardinality of GS is $n-3$, it takes $O(n)$ space to store it. Along with GS, we need to maintain for T, the set of opposite pairs O and update it while generating children. We also need to maintain another list L for listing the chords of T. To generate the triangulations of C, we start at the root of $\mathcal{T}(C)$. For the root of $\mathcal{T}(C)$, GS is identical to L. When a generating chord of T is flipped, that chord is replaced in the list L of T by its opposite pair in T to get the list L of the child. Since we use a recursive procedure to generate the triangulations without constructing the whole $\mathcal{T}(C)$, and the depth of the tree is $n-2$ (number of chords in the root plus one), the algorithm uses $O(n)$ space.

Now the question is how can we update GS and O? By implementing these two sets using linked lists and storing appropriate pointers at each node on the path from the root of $\mathcal{T}(C)$ to the current triangulation T, we can do it in constant time. Let (v_1, v_j) be the chord of T to be flipped. The updated lists GS and O correspond to the newly generated child of T.

Flipping the generating chord (v_1, v_j) of T can change the opposite pairs of maximum two other candidate chords of T in the representation of $T(v_1, v_j)$. In our algorithm, we only need to change the opposite pairs of candidate chords of $T(v_1, v_j)$. Let $(v_1, v_i), (v_1, v_j)$ and (v_1, v_k) be three candidate chords of T, $k < j < i$, such that $\langle v_1, v_k, v_j, v_i \rangle$ is a quadrilateral of T, as shown in Figure 13. We now flip (v_1, v_j) of T to generate the child $T(v_1, v_j)$ of T. Flipping the chord (v_1, v_j) of T changes the opposite pairs of the chords (v_1, v_i) and (v_1, v_k) of T in $T(v_1, v_j)$. The changes can be done as follows.

Let $(v_o, v_{o'})$ be the opposite pair of (v_1, v_j) in T. Here $o = k$ and $o' = i$, as shown in Figure 13. Let the opposite pair of (v_1, v_i) in T be $(v_l, v_{l'})$. Then $l = j$ and the opposite pair of (v_1, v_i) in $T(v_1, v_j)$ is $(v_o, v_{l'})$. Similarly, if the opposite pair of (v_1, v_k) is $(v_s, v_{s'})$ in T, then $s' = j$ and the opposite pair of (v_1, v_k) in $T(v_1, v_j)$ will be $(v_s, v_{o'})$. Figure 14 shows the update operations. Clearly, these updates can be done in $O(1)$ time.

Thus, if a triangulation T has k children, all of them can be generated in $O(k)$ time. Therefore each child of T is generated in $O(1)$ time. □

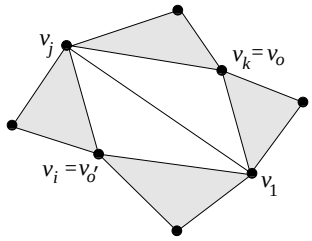

Figure 13: Flipping (v_1, v_j) can affect two candidate chords.

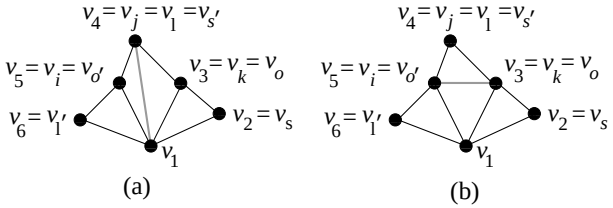

Figure 14: Illustration of the update operations of the opposite pairs of two affected edges; (a) parent and (b) child.

5 Generating Unlabeled Triangulations

In this section, we give the idea for generating all unlabeled triangulations of a cycle C of n vertices.

Generating all unlabeled triangulations of a cycle C is more difficult than generating labeled triangulations. If the vertices of C are not numbered then we need to avoid "rotational" and "mirror repetitions" among the triangulations of C. Two unlabeled triangulations of a cycle are *rotationally equivalent* to each other, if one can be found by rotating the other one. Similarly, two unlabeled triangulations of a cycle are *mirror image* of each other, if one can be found by taking the mirror image of the other one. For example, the triangulations of Figure 15(a) and (b) are rotationally similar if we remove the labels, since then both of them are similar to the triangulation of Figure 15(c). The two triangulations of Figure 16(a) and (b) are mirror images of the one another, if no labels are used, and both of them are then similar to the triangulation of Figure 16(c). In this section, we modify our algorithm for generating all triangulations of a cycle to avoid such repetitions. For this purpose, we consider each triangulation of C as belonging to a particular class, in which the triangulations of C are rotationally equivalent or mirror images of one another. We choose one representative triangulation from each class. The modified algorithm still uses the labels while generating the triangulations, but avoids any rotational or mirror repetitions by outputting a triangulation only if it is the representative of a particular class. Thus, our modified algorithm constructs the tree of triangulations T_6 of a cycle of six vertices as shown in Figure 17. Note that, only

three triangulations are there in Figure 17, while the tree of triangulations of Figure 7 contains 14 triangulations.

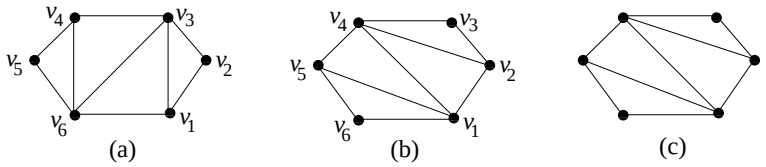

Figure 15: Two triangulations of (a) and (b) are rotationally equivalent to the triangulation of (c), when the labels are removed.

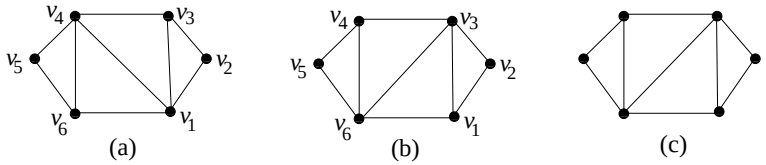

Figure 16: Two triangulations of (a) and (b) are are mirror image of each other, similar to the triangulation of (c), when the labels are removed.

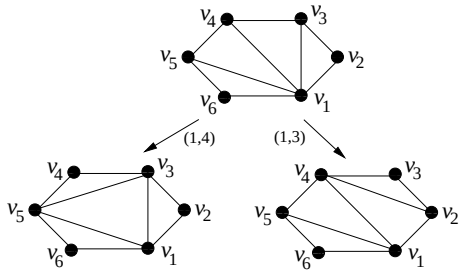

Figure 17: Illustration of $\mathcal{T}(C)$ from Figure 7 when rotational and mirror repetitions are not allowed.

We now give a new representation of each triangulation of a cycle that enables us to avoid any rotational or mirror repetitions easily. Let T be a triangulation of C where the vertices of C are labeled sequentially from v_1 to v_n. A *labeled degree sequence* $\langle d_1, d_2, \cdots, d_n \rangle$ of T is the sequence of degrees of the vertices, where d_i is the degree of v_i in the graph associated with T. A vertex with degree 2 is called an *ear* of T. We thus have the following lemma.

Lemma 8 *Let T be a labeled triangulation of a cycle C of n vertices. Then T can be represented uniquely by its labeled degree sequence.*

Proof: Let $\langle d_1, d_2, \cdots, d_n \rangle$ be the labeled degree sequence of T. We note that T has at least two ears. Let v_i is the clockwise first ear. Remove it and

decrease the degrees of its two neighboring vertices by one. Apply the procedure recursively until the vertices v_1 and v_2 are left. Thus we get a sequence of vertices $v_{i_1}, v_{i_2}, \cdots v_{i_{n-2}}$. Now adding the vertices in reverse order we can generate T. □

5.1 Removing Rotational Repetitions

In this section, we describe the procedure for avoiding rotational repetitions. The following fact is crucial for that purpose.

Fact 9 *Let T and T' be two triangulations of a cycle C of n vertices, which are rotationally equivalent to each other. Then, by rotating the labeled degree sequence of T, we get the labeled degree sequence of T'.*

As an illustration of the Fact 9, the triangulations of Figure 15(a) and (b) have the labeled degree sequences $\langle 3, 2, 4, 3, 2, 4\rangle$ and $\langle 4, 3, 2, 4, 3, 2\rangle$ respectively. By right rotating the labeled degree sequence of the triangulation of Figure 15(a) four times, we get the labeled degree sequence of the triangulation of Figure 15(b).

Let T and T' be two triangulations of a cycle C of n vertices, which are rotationally equivalent to each other. Let $\langle d_1, d_2, \cdots, d_n\rangle$ and $\langle d'_1, d'_2, \cdots, d'_n\rangle$ be the labeled degree sequences of T and T' respectively. Let $d_1 = d'_1, d_2 = d'_2, \cdots, d_{k-1} = d'_{k-1}$ and $d_k > d'_k$ for some k, $1 \leq k \leq n$. We say that the sequence $\langle d_1, d_2, \cdots, d_n\rangle$ is *greater* than the sequence $\langle d'_1, d'_2, \cdots, d'_n\rangle$ and T has a *greater* sequence than T'. For example, the triangulations of Figure 18(a) and (b) have the degree sequences $\langle 5, 2, 5, 2, 3, 4, 3, 2\rangle$ and $\langle 4, 2, 3, 4, 3, 3, 2, 5\rangle$ respectively and the first sequence is greater than the second one. Thus the triangulation of Figure 18(a) is greater than the triangulation of Figure 18(b).

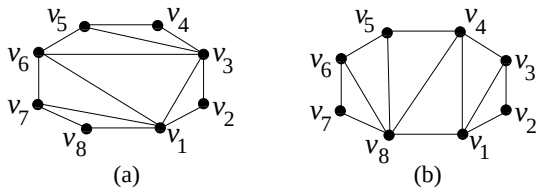

Figure 18: Illustration of two triangulations where one has greater degree sequence.

Let S be a set of triangulations of C where the triangulations are rotationally equivalent to each other. Let T be the triangulation in S whose degree sequence is greater than all other triangulations in S. Then, the labeled degree sequence of T is the *canonical representation* of S. We say that T has the *greatest* labeled degree sequence and T is the *representative* of S. We output each triangulation T of C only if T has the greatest labeled degree sequence. Let $\langle d_1, d_2 \cdots d_n\rangle$ be the degree sequence of T. If $d_1 > d_i$ for $2 \leq i \leq n$, then T has the greatest labeled degree sequence. This can be found in $O(1)$ time as explained later.

Otherwise, we generate $n-1$ other degree sequences by right rotating T's degree sequence and check whether T's sequence is greater. In this case, it takes $O(n^2)$ time to find whether T has the greatest labeled degree sequence.

For a triangulation T of C, we need to maintain an array D to store the degree sequence. It takes $O(n)$ space. Let (v_1, v_j) is a generating chord in T with opposite pair $(v_o, v_{o'})$. Flipping (v_1, v_j) changes the degrees of four vertices. The degrees of v_1 and v_j are reduced by one and the degrees of v_o and $v_{o'}$ are increased by one. All these updates can be done in $O(1)$ time. Let $\langle d'_1, d'_2, \cdots, d'_n \rangle$ be the resultant degree sequence and T' is the new triangulation. We can easily check whether $d'_1 > d'_i$ for $2 \leq i \leq n$ by storing the highest degree d_{max} among nodes other than v_1 and updating it while generating a new triangulation. Now there are three cases.

Case 1. If $d'_1 > d_{max}$, then output T'.

Case 2. If $d'_1 = d_{max}$, then check whether T' has the greatest labeled degree sequence. If YES, then output T'.

Case 3. If $d'_1 < d_{max}$, then ignore T' and prune the subtree of triangulations rooted at T'.

5.2 Avoiding Mirror Repetitions

In this section, we describe the procedure to remove mirror image repetitions while generating all triangulations of a cycle C of n vertices.

Let $\langle d_1, d_2, \cdots, d_n \rangle$ be the labeled degree sequence of a triangulation T of C. Assume that T has the greatest labeled degree sequence compared to all triangulations of C which are rotationally similar to T. Let T' be the triangulation which is the mirror image of T. Using the following fact we can find the labeled degree sequence of T'.

Fact 10 *Let T and T' be two triangulations of a cycle of n vertices, which are mirror images of each other. Let T has the labeled degree sequence $\langle d_1, d_2, \cdots, d_n \rangle$. Then the labeled degree sequence of T' is $\langle d_n, d_{n-1}, \cdots, d_2, d_1 \rangle$.*

For example, the triangulation of Figure 16(a) has the degree sequence $\langle 4, 2, 3, 4, 2, 3 \rangle$. The triangulation of Figure 16(b), which is the mirror image of the triangulation of Figure 16(a), has the *reverse* degree sequence $\langle 3, 2, 4, 3, 2, 4 \rangle$.

Now, using the labeled degree sequence of T', we can avoid mirror image repetitions as follows. We start with the sequence $\langle d_n, d_{n-1}, \cdots, d_2, d_1 \rangle$, and from it we generate $n-1$ other sequences by right rotation. These $n-1$ sequences corresponds to all the triangulations which are rotationally similar to T'. We compare the degree sequence of T with all these sequences to determine whether T's sequence is the greatest. Thus, we have to compare T's sequence with a total of n sequences. This takes $O(n^2)$ time. If T's sequence is found

greater than all these sequences, then we output T. Otherwise we discard T and prune the subtree rooted at T. Since all we need is to store the sequence of T, the space complexity is $O(n)$.

Thus we have the following theorem.

Theorem 3 *For a cycle C of n vertices, all triangulations of C can be found in time $O(n^2)$ per triangulation, where the vertices of C are not numbered. The space complexity is $O(n)$.*

6 Conclusion

In this paper we gave an algorithm to generate all triangulations of a triconnected plane graph G of n vertices in $O(1)$ time per triangulation with linear space complexity. We also gave an algorithm to generate all triangulations of a cycle of n labeled vertices in time $O(1)$ per triangulation with $O(n)$ space complexity. The performance of the algorithms can be further improved by using parallel processing. Our algorithm also works for biconnected graphs, but may produce multi-edges occasionally. Finally, we described a method to eliminate any rotational and mirror repetitions while generating all triangulations of a cycle C, when the vertices of C are not numbered.

Acknowledgements

We thank the referees for their valuable comments which helped us to improve the presentation of the paper.

References

[1] D. Avis. Generating rooted triangulations without repetitions. *Algorithmica*, 16(6):618–632, 1996.

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

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

[4] B. Chazelle. Triangulating a polygon in linear time. *Discrete Comput. Geom.*, 6:485–524, 1991.

[5] M. de Berg, M. van Krevald, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 2000.

[6] F. Hurtado and M. Noy. Graph of triangulations of a convex polygon and tree of triangulations. *Computational Geometry*, 13(3):179–188, 1999.

[7] H. Komuro, A. Nakamoto, and S. Negami. Diagonal flips in triangulations on closed surfaces with minimum degree at least 4. *Journal of Combinatorial Theory, Series B*, 76(1):68–92, 1999.

[8] Z. Li and S. Nakano. Efficient generation of plane triangulations without repetitions. In *Proc. of ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 433–443. Springer-Verlag, 2001.

[9] B. D. McKay. Isomorph-free exhaustive generation. *J. Algorithms*, 26(2):306–324, 1998.

[10] T. Nishizeki and M. S. Rahman. *Planar Graph Drawing*. World Scientific, Singapore, 2004.

[11] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, Cambridge, 1998.

[12] M. T. Parvez, M. S. Rahman, and S. Nakano. Generating all triangulations of plane graphs. In *Proc. of WALCOM 2009*, volume 5431 of *Lecture Notes in Computer Science*, pages 151–164. Springer, 2009.

[13] D. Poulalhon and G. Schaeffer. Optimal coding and sampling of triangulations. *Algorithmica*, 46(3-4):505–527, 2006.

[14] S. M. Sait and H. Youssef. *VLSI Physical Design Automation: Theory and Practice*. World Scientific, Singapore, 1999.

[15] D. D. Sleator, R. E. Tarjan, and W. P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. *J. Amer. Math. Soc*, 1:647–681, 1988.