More Efficient Generation of Plane Triangulations

Shin-ichi Nakano[1] and Takeaki Uno[2]

[1] Gunma University, Kiryu 376-8515, Japan
nakano@cs.gunma-u.ac.jp
http://www.msc.cs.gunma-u.ac.jp/~nakano/index.html
[2] National Institute of Informatics, Tokyo 101-8430, Japan
uno@nii.jp,
http://research.nii.ac.jp/~uno/

Abstract. In this paper we give an algorithm to generate all biconnected plane triangulations having exactly n vertices including exactly r vertices on the outer face. The algorithm uses $O(n)$ space in total and generates such triangulations without duplications in $O(rn)$ time per triangulation, while the previous best algorithm generates such triangulations in $O(r^2n)$ time per triangulation.

1 Introduction

Generating all graphs with some properties without duplications has many applications, including unbiased statistical analysis[M98]. Many algorithms to solve these problems are already known [A96,B80,M98,W86]. Many nice textbooks have been published on the subject[G93,KS98].

In this paper we wish to generate all biconnected plane triangulations having exactly n vertices including exactly r vertices on the outer face. Such triangulations play an important role in many algorithms, including graph drawing algorithms [CN98,FPP90,S90]. A graph G is *biconnected* if removing any vertex leaves G connected. An embedded planar graph is called *a plane triangulation* if each inner face has exactly three edges on its boundary.

Recently, we proposed an algorithm to generate all biconnected "based" plane triangulations having exactly n vertices including exactly r vertices on the outer face[LN01]. A *based* plane triangulation means a plane triangulation with one designated "base" edge on the outer face. Two based plane graphs are said to be isomorphic if they are isomorphic and the base edges correspond to each other in the bijection. For instance, four biconnected based plane triangulations are shown in Fig. 1, where the base edges are depicted by thick lines. Note that, however, those based triangulations are isomorphic as non-based plane triangulations. The algorithm for based plane triangulations uses $O(n)$ space in total and runs in constant time per output while the previous best algorithm generates such triangulations in $O(n^2)$ time per triangulation[A96]. The algorithm does not output entire triangulations but the difference from the previous triangulation.

G. Liotta (Ed.): GD 2003, LNCS 2912, pp. 273–282, 2004.

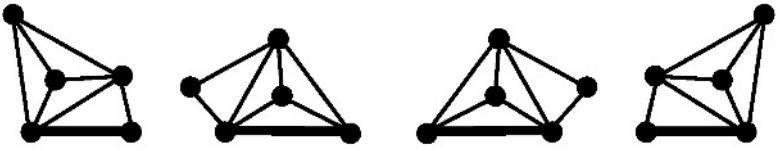

Fig. 1. Biconnected based plane triangulations.

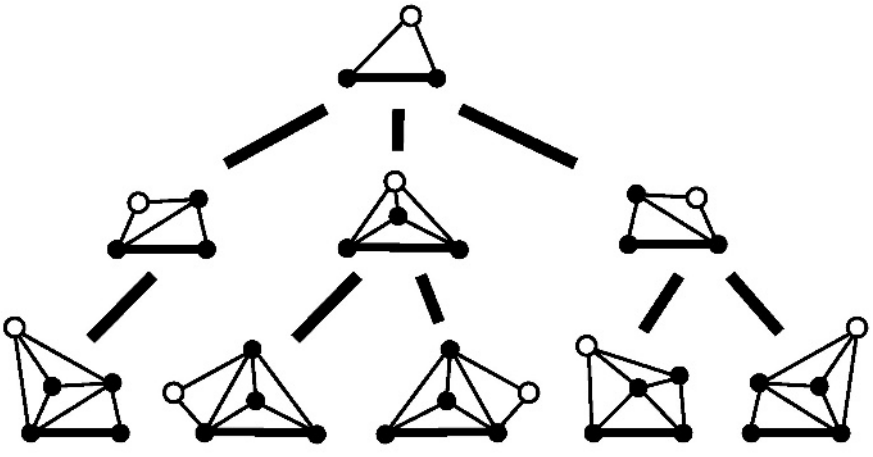

Fig. 2. The genealogical tree for $n = 5$ and $r = 4$.

The algorithm of [LN01] generates all based triangulations as follows. Given n and r, we first define a tree T such that the inner vertices of T correspond to the biconnected based plane triangulations having at most $n - 1$ vertices including at most $n - r$ inner vertices, the leaves of T correspond to the biconnected based plane triangulations having exactly n vertices including exactly r vertices on the outer face, and the edges of T correspond to some relation between the biconnected based plane triangulations. T is called the *genealogical tree,* and the genealogical tree for $n = 5$ and $r = 4$ is shown in Fig. 2. In the figure, we can observe that if we remove a vertex depicted by a white circle from a biconnected based plane triangulation, then we can have the "parent" biconnected based plane triangulation. Also we can prove the number of vertices of T is within 3 times the number of leaves of T. Since the size of T is huge in general, thus we cannot construct whole part of T at once. However, we can simply traverse T in $O(1)$ time per edge of T by partially constructing T. We need only $O(n)$ space in total. And on the traversal we can find all the vertices of T, which correspond to all the triangulations.

By modifying the algorithm, we can also generate without duplications all biconnected (non-based) plane triangulations having exactly n vertices including exactly r vertices on the outer face in $O(r^2 n)$ time per triangulation on average

[LN01]. If we maintain output triangulations in a database to avoid the duplication, then the space complexity will be exponential. So to obtain an efficient algorithm, we have to check the occurrence of duplication without any database, hence this is a non-trivial modification. Another algorithm with $O(n^2)$ time per triangulation is also claimed in [M98] without detail but using a complicated theoretical linear-time plane graph isomorphism algorithm [HW74], while the algorithm in [LN01] is simple and does not need the isomorphism algorithm.

The strategy of the algorithm in [LN01] is as follows. The genealogical tree T has many biconnected based plane triangulations which are distinct as based plane triangulations, but isomorphic as (non-based) plane triangulations. The only difference is the choice of the base edge on the outer face. See Fig. 1. On the traversal of T, we have to output exactly one biconnected (non-based) plane triangulations for each isomorphic class. By giving a unique sequence of letters for each biconnected based plane triangulations, we can define a representative triangulation among each isomorphic class as the triangulation having the lexicographically-first sequence of letters. The algorithm in [LN01] needs $O(rn)$ time computation at each leaf v of T to decide whether the sequence of letters for the based triangulation corresponding to v is the lexicographically-first one among the isomorphic class, and only in such case the based plane triangulation is output as the representative plane triangulation. Otherwise the based triangulation is not output. For each output triangulation, T may contain r isomorphic ones corresponding to the r choices of the base edge. Thus, the amortized time complexity of the algorithm is $O(r^2n)$ per output triangulation.

In this paper we improve the running time of the algorithm in [LN01] as follows. We define a new unique sequence of letters for each biconnected based plane triangulation. Given a biconnected based plane triangulation, the new sequence of letters needs less computation to decide whether the sequence of letters for the based plane triangulation is the lexicographically-first one among the isomorphic class. Our algorithm needs only $O(n)$ time computation at each leaf of T. Again, for each output triangulation, T may contain r isomorphic ones corresponding to the r choices of the base edge. Thus, the amortized time complexity of our algorithm is $O(rn)$ per triangulation.

The rest of the paper is organized as follows. Section 2 defines our new sequence of letters. Section 3 gives a new algorithm to find the lexicographically first sequence. Finally Section 4 is a conclusion.

2 Sequence of Letters

In this section, given a biconnected based plane triangulation G with the base edge e, we define a sequence of letters. Then we show that the sequence of letters has enough information to re-construct G. This means that we give a unique name to each biconnected based plane triangulation. We need some definitions first.

For a graph, a *cut* is a set of vertices whose removal results in a disconnected graph or a single-vertex graph K_1. A graph is said to be *biconnected* if no cut

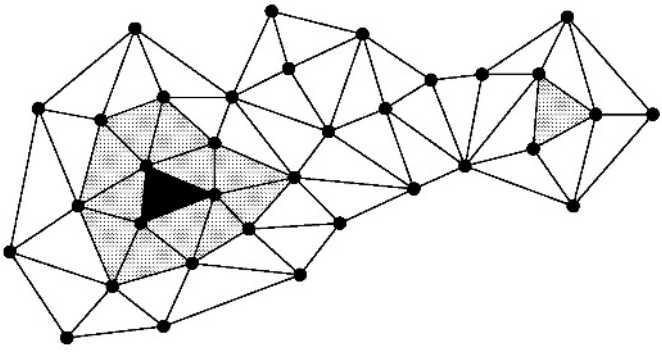

Fig. 3. Graph $G_1 = G$: the gray area is G_2, and the black area is G_3.

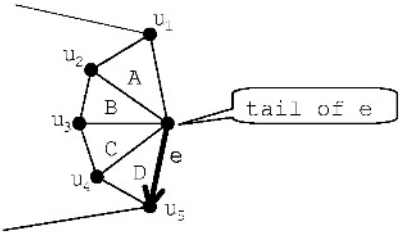

Fig. 4. An edge e and its tail: triangle sequence of e is (A, B, C, D). For triangle C, $v_1(C) = u_4$ and $v_2(C) = u_3$.

is composed of a single vertex. 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*. A plane graph is called *plane triangulation* if each inner face includes exactly three edges. Without loss of generality, we assume that the vertices are on the general position, so that no triangle is "empty." An *isomorphism* from a graph G to H is a bijection $f : V(G) \to V(H)$ such that $(u, v) \in E(G)$ if and only if $(f(u), f(v)) \in E(H)$. We say G is isomorphic to H if there is an isomorphism from G to H. Two plane graphs are said to be *isomorphic* if there is an isomorphism preserving the cyclic ordering of edges incident to each vertex.

A triangle of G is a *boundary triangle* of G if it has at least one vertex on the outer face. Let $G_1 = G$, and $G_i, i > 1$ be the graph obtained from G_{i-1} by removing all vertices included in only boundary triangles, i.e. the graph induced by non-boundary triangles of G_{i-1} (see Fig. 3). For an edge e of the outerface, the *tail* of e is the endpoint of e followed by the other endpoint of e in clockwise

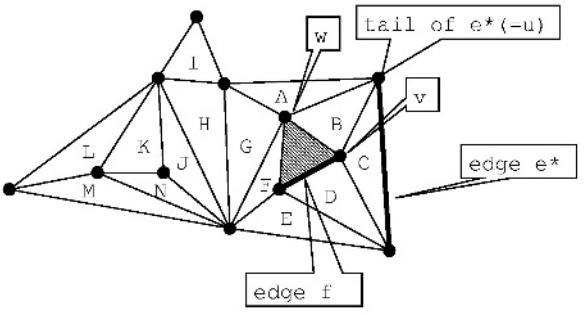

Fig. 5. seq_1 =(A,B,C, C,D,E, E,F,G,H,J,N,M M,L, L,K,J,H,I, I, I,H,G,A). By setting the triangle sequence (A, B, C) of e^* to the top, this circular sequence becomes a sequence. The outer triangle of edge f is D.

order. We define the *triangle sequence* of e by the sequence of triangles incident to the tail of e in counter-clockwise order (see Fig. 4).

Let seq_1 be the circular sequence obtained by connecting the triangle sequence of the edges of the outer face of G_1 in clockwise order (see Fig. 5). Note that this is a circular sequence so it has no end. For an edge e on the outer face of $G_i, i \geq 2$, we define the *outer triangle* of e by the boundary triangle of G_{i-1} including e (see Fig. 5). Let seq_i be the circular sequence obtained by connecting the triangle sequence of the edges on the outer face of G_i in the order of their outer triangles in seq_{i-1} (see Fig. 6). Any outer triangle appears just once in seq_{i-1}, and other boundary triangles appear in seq_i at least once and at most three times. For instance in Fig. 5, H appears in seq_1 three times and F appears once. Each appearance of a triangle in seq_i is called an *occurence*.

For two occurrence t_1 and t_2 of seq_i, we define the *distance from* t_1 *to* t_2 by the number of occurrences from t_1 to t_2 in seq_i (see Fig. 5). Note that the distance is zero if t_2 is next to t_1. If t_1 and t_2 are the same occurrence, we define the distance from t_1 to t_2 by the number of occurrences in seq_i minus one. For instance in Fig. 5, the distance from the occurrence corresponding to the first A to the occurrence corresponding to the first C is 1, and the distance from the occurrence corresponding to the last A to the occurrence corresponding to first A is 0.

For each occurrence t we assign a sequence of three integers $l(t)$ as follows. Assume that t is in the triangle sequence of edge e, u is the tail of e, and u, v, and w are the three vertices on the boundary of t, and they appear clockwise in this order. Let $d(u)$ be the distance from u' to u, where u' is the last occurence corresponding to t. Let $d(v)$ be the distance from v' to v, where v' is the last occurence corresponding to a triangle containing v. Let $d(w)$ be the distance from w' to w, where w' is the last occurence corresponding to a triangle containing v. We set $l(t) = (d(u), d(v), d(w))$.

For example, see Fig.5. The traingle sequence of e^* has an occurrence corresponding to triangle B. Now $d(u) = 24$ since seq_i has 25 occurrences, and

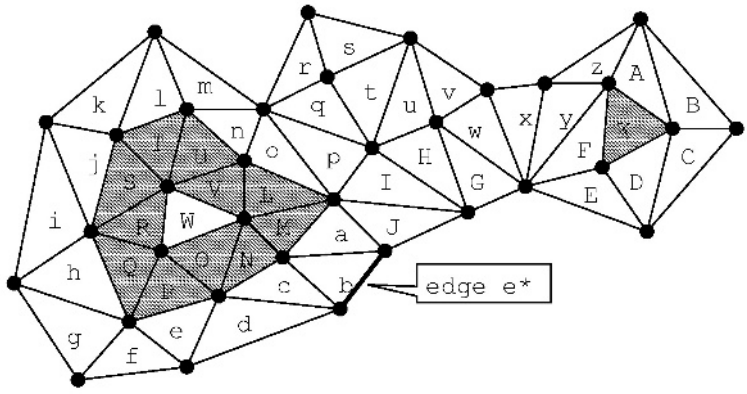

Fig. 6. seq_1 is circular sequence (b,c,d, d,e,f, f,g, g,h,i,..., G,H,I,J, J,a,b). The order of outer triangles of edges of G_1 is (a,c,e,h,j,l,n,o,A,D,F), hence seq_2 is (M,N, N,O,P, P,Q, Q,R,S, S,T, T,U, U,V,L, K, K, K, L,M). By setting the triangle sequence of e^* to the top of the sequence, seq_1 becomes a sequence (J,a,b, b,c,d, d,e,f,...,I,J), and seq_2 becomes a sequence (L,M,M,N,N,...,K,K,K).

has only one occurence corresponding to B, $d(v) = 21$ since from D to B there are 21 occurences, and $d(w) = 0$ since from A to B there are zero occurences. Note that A, B and D mean the occurrences corresponding to triangle A, B, and D, respectively. Thus $l(B) = (24, 21, 0)$. Similarly for the first occurrence C, $l(C) = (23, 20, 0)$.

For two occurences t_1 and t_2, we say $l(t_1)$ is *smaller* than $l(t_2)$ if $l(t_1)$ is lexicographically smallerthan $l(t_2)$, and we write $l(t_1) < l(t_2)$.

We can observe that the first integer of $l(t)$ is 0 iff t is the first triangle in the triangle sequence of some edge. Hence, the following property holds.

Property 1. If occurence t_1 is the first triangle of the corresponding triangle sequence and occurence t_2 is not the first triangle of the corresponding triangle sequence, then $l(t_1) < l(t_2)$. □

For any edge e on the outer face of G, we can obtain an ordinal letter sequence from the circular sequences $seq_1, seq_2, ...$ as follows. First we cut the circular sequence seq_1 to a sequence so that the triangle sequence of e appears first (see Fig. 5). By using the order of seq_{i-1}, seq_i can be considered as a sequence (see Fig. 6). We connect these sequences so that the sequence obtained from seq_i follows the sequence obtained from seq_{i-1}, and replace each occurrence t by $l(t)$. We define the *letter sequence* $L(e)$ of e by the sequence of letters. We denote the ith letter of $L(e)$ by $l^i(e)$.

Given letter sequence L, we can uniquely re-construct the biconnected plane triangulation G based at e such that e is the base edge of G, and $L(e) = L$ as follows.

First, we consider the graph G' induced by the boundary triangles of G_1. If G is not equal to G', then there exists at least one boundary triangle appearing in

seq_1 only once. From the first number of the letter of the occurrence, we can get the length of seq_1. The occurrences such that the first number of their letters are 0 are the tops of the triangle sequences. Thus, we can get the number of edges on the outer face, and the length of the triangle sequence of each edge. From the first number of the letters, we can know which triangles of distinct triangle sequences are identical. From the second and the third numbers of the letters, we can know the order of triangles incident to a vertex. Therefore, we can construct the graph G' uniquely.

Suppose that we have constructed the graph induced by the boundary triangles of $G_1, .., G_{k-1}$. In the similar way to the above, we can construct the graph induced by the boundary triangles of G_k. By connecting the graphs we constructed, we obtain the graph induced by the boundary triangles of $G_1, .., G_k$. Therefore, G can be constructed uniquely from the letter sequence.

Thus, for any biconnected based plane triangulation with base edge e, the letter sequence $L(e)$ is defined uniquely and given $L(e)$, and we can re-construct a unique based biconnected plane triangulation with base edge e. Therefore, we gave a unique name to each biconnected based plane triangulation.

3 Algorithm for Finding the Minimum Letter Sequence

We have defined a unique letter sequence for each biconnected based plane triangulation in Section 2. In this section, we define a unique letter sequence for each biconnected (non-based) plane triangulation, and show an algorithm for computing the letter sequence in short time.

A biconnected plane triangulation G corresponds to many biconnected "based" plane triangulations G_e, since we can choose any edge on the outer face as the base edge e. So intuitively each biconnected (non-based) plane triangulation G has many names which correspond to the choice of the base edge. We are going to find e^* which is the lexicographically minimum name $L(e^*)$ among all over $L(e)$, and define the name of G as $L(e^*)$.

For two sequences L_1 and L_2, we say that L_1 is smaller than L_2 iff the first $k-1$ letters of L_1 and L_2 are the same, and kth letter of L_1 is smaller than L_2. Note that k can be 1. Let e^* be the edge of the outer face of G whose letter sequence is the lexicographically minimum. In this section, we describe an algorithm for finding the edges whose letter sequences are equal to $L(e^*)$.

For an edge e on the outer face of G, we define $m(e)$ by the number such that $l^i(e) = l^i(e^*)$ if $i < m(e)$ and $l^i(e) > l^i(e^*)$ if $i = m(e)$. We define $m(e)$ by $+\infty$ if $L(e) = L(e^*)$. We are going to find the edges e with $m(e) = +\infty$ by comparing the ith letter of the letter sequences iteratively. $l^i(e^*)$ is the minimum letter among $l^i(e)$ with $m(e) \geq i$. We can see that if $m(e) \geq i$ and $l^i(e) \neq l^i(e^*)$, then $m(e) = i$, and we do not care about e in the $(i+1)$th iteration.

In this way, if we check each occurrence of seq_i to get its letter only once, the computation time is $O(n)$. However, if some occurrences are checked more than once, then the computation time may be up to $O(n^2)$. We will explain how we can save computation time in this case.

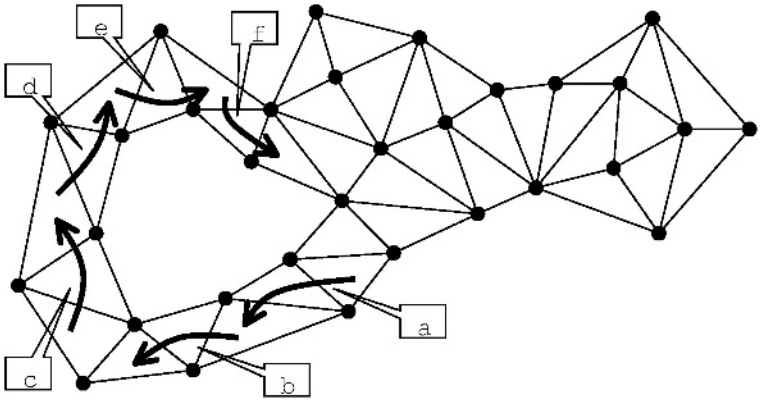

Fig. 7. a overlaps with b. Edges c, d, e, and f compose a chain, and the tail of the chain is f. The length of the chain is 4, and the length of the chain (a, b) is 2, hence only c remains.

In the ith iteration, for an edge e and e' of the outer face satisfying $m(e) \geq i$, if there exists h satisfying $h < i, h < m(e')$, and the occurrences giving $l^i(e)$ and $l^h(e')$ are on the same position in seq_j for some j, we say that e *overlaps* with e'. Let z_i be the sum of the cardinalities of $seq_1, ..., seq_i$. We define z_0 by 0. Note that $h = z_{j-1} + 1$ if e does not overlap with e' in the $(i-1)$th iteration.

Suppose that edges $e_1, ..., e_k$ overlap with $r(e_1), ..., r(e_k)$ in the ith iteration and $z_q < i \leq z_{q+1}$. From Property 1, if an edge e satisfying $m(e) \geq i$ overlaps with no edge, $m(e) = i$.

First, we consider the case that no $r(e_j)$ is in $\{e_1, ..., e_k\}$. Let $m^* = \max\{m(r(e_1)), ..., m(r(e_k))\}$. We can see that $m(e_j) = +\infty$ only if $m(r(e_j)) = m^*$, and $m(e_j) = i + m(r(e_j)) - z_q - 1$ if $m(r(e_j)) < m^*$. Hence, we can go to $(i + m^* - z_q)$th iteration. The skipped iterations are not necessary to be executed.

Second, we consider the case that some $r(e_j)$ are in $\{e_1, ..., e_k\}$. We define the *chain* of e_j by $\{e_j = e_{j_1}, e_{j_2}, ..., e_{j_h}\}$ such that $r(e_{j_p}) = e_{j_{p+1}}$, and $r(e_{j_h})$ is not in $\{e_1, ..., e_k\}$ or e_{j_h} overlaps with no edge. We denote e_{j_h} by $tail(e_j)$ (see Fig. 7). Let $d(e_j)$ be the cardinality of the chain. We consider the case later that the chain is circular, i.e., any $r(e_j)$ is in $\{e_1, ..., e_k\}$.

Let $d^* = \max\{d(e_1), ..., d(e_k)\}$. We can see that $m(e_j) = (i - z_q - 1) \times (d(e_j) - 1) + m(tail(e_j))$ if $d(e_j) < d^*$. Hence, $m(e_j) = +\infty$ only if $d(e_j) = d^*$, and we can go to $((i - z_q - 1) \times (d^* - 1) + z_q)$th iteration.

Finally, we consider the case that the chain is circular, i.e. any e_j overlaps with an edge of $e_1, ..., e_k$. In this case, a chain includes all $e_1, ..., e_k$. Hence, the first z_{q+1}th letters of $L(e_1), ..., L(e_k)$ are the same, and we can go to $(z_{q+1}+1)$th iteration.

We describe this algorithm as a procedure below.

1. Compute seq_j for each j
2. Compute the letters of all occurrences in seq_j for each j
3. $i := 1, F :=$ edges of the outer face
4. Set q so that $z_q < i \leq z_{q+1}$
5. **For each** $e \in F$ satisfying $l^i(e) < \max_{f \in F} l^i(f)$
6. $\quad m(e) := i$ and remove e from F
7. $i := i + 1$
8. **If** any edge of F overlaps with an edge of F **then**
9. $\quad i := z_{q+1} + 1$
10 **Else if** an edge of F overlaps with an edge of F **then**
11. $\quad$ **For each** $e \in F$ satisfying $d(e) < \max_{f \in F} d(f)$
$\quad\quad m(e) := (i - z_q - 1) \times (d(e) - 1) + m(tail(e)) - 1$ and remove e from F
12. $\quad i := ((i - z_q - 1) \times (\max_{f \in F} d(f) - 1) + z_q)$
13. **Else if** an edge of F overlaps with an edge not in F **then**
14. $\quad$ **For each** $e \in F$ satisfying $m(r(e)) < \max_{f \in F} m(r(f))$
$\quad\quad m(e) := i - z_q + m(r(e)) - 1$ and remove e from F
15. $\quad i := i - z_q + \max_{f \in F} m(r(f))$
16. **End if**
17. **If** i is less than the length of the letter sequences **and** $|F| > 1$ **then go to** 4.
18. **Output** all edges of F

The time complexity of this algorithm is proportional to the length of a letter sequence plus the number of overlaps. Since the length of a letter sequence is $O(n)$, we estimate the upper bound of the overlaps.

Consider the the ith iteration of the algorithm. For each $e \in F$, one of the following three cases occurs.

(1) e overlaps with no other edge.
(2) e overlaps with another edge e', and $e' = tail(f)$ for no f.
(3) e overlaps with another edge e', and $e' = tail(f)$ for some f.

If (3) occurs, $e' = tail(f)$ can never hold again for any f, hence (3) will never occur for e'. If (2) occurs, then e' can never be overlapped. Hence the number of the overlaps is at most twice the length of the letter sequence, and is $O(n)$.

By replacing the $O(rn)$ time computation at each leaf of the genealogical tree T in [LN01] to the $O(n)$ time computation above, we can improve the running time of the algorithm. We have the following theorem.

Theorem 1. *All the biconnected plane triangulations having n vertices including r vertices on the outer face can be generated in $O(rn)$ time per triangulation.* □

4 Conclusion

In this paper we have given an algorithm to generate all biconnected plane triangulations without duplication. Our idea is to define a unique sequence of letters

for each biconnected based plane triangulation, and by finding the lexicographically minimum among all the bases of a biconnected plane triangulations, we can efficiently decide for each whether the based triangulation should be output as a biconnected plane triangulation.

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.

[CN98] M. Chrobak and S. Nakano, *Minimum-width grid drawings of plane graphs*, Computational Geometry: Theory and Applications, 10, (1998), pp.29–54.

[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).

[HW74] J. E. Hopcroft and J.K. Wong, *Linear time algorithm for isomorphism of planar graphs*, Proc. of 6th STOC, (1974), pp.172–184.

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

[LN01] Z. Li and S. Nakano, *Efficient generation of plane triangulations without repetitions*, Proc. ICALP2001, LNCS 2076, (2001), pp.433–443.

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

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

[W86] R. A. Wright, B. Richmond, A. Odlyzko and B. D. McKay, *Constant time generation of free trees*, SIAM J. Comput., 15, (1986), pp.540–548.