2007－AL－114 (3)
2007／9／21

平面グラフの列挙

山中 克久†, 李 章剣†, 中野 眞一‡

概要 根つき平面グラフとは, 外周上の1辺を指定した平面グラフである. 本文では, 高々m本の辺をもつ根つき連結平面グラフを列挙する問題を考える. 我々は, 根つき連結平面グラフを1つ当たり平均$O(1)$時間で列挙するアルゴリズムを与える. 必要な記憶領域は$O(m)$である. アルゴリズムの出力は, グラフ全体ではなく, 直前に出力したグラフとの差分のみである. さらに, このアルゴリズムを拡張することにより, 高々m本の辺をもつ(根なし)連結平面グラフを1つ当たり$O(m^3)$時間で列挙できることを示す.

Listing All Plane Graphs

Katsuhisa YAMANAKA†, Zhangjian LI† and Shin-ichi NAKANO‡

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

1 Introduction

Generating all graphs with some property without duplications has many applications, including unbiased statistical analysis [M98]. A lot of algorithms to solve these problems are known [AK96, B80, LN01, M98, N04, W86, etc]. See textbooks [G93, K05, KS98, S97, S99].

In this paper we wish to generate all connected "rooted" plane graphs, which will be defined precisely in Section 2, with at most m edges. Such graphs play an important role in many algorithms, including graph drawing algorithms [CN98, FPP90, S90, etc].

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

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

†,‡ 群馬大学工学部情報工学科 〒376–8515 群馬県桐生市天神町 1–5–1.
Department of Computer Science, Gunma University, 1–5–1 Tenjin-Cho, Kiryu, 〒376–8515
†yamanaka@nakano-lab.cs.gunma-u.ac.jp,
‡nakano@cs.gunma-u.ac.jp

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

Reverse search method algorithms [AK96] also need not store the list. The idea is to implicitly define a connected graph H such that the vertices of H correspond to the graphs with the given property, and the edges of H correspond to some relation between the graphs. By traversing an implicitly defined spanning tree of H, one can find all the vertices of H, which correspond to all the graphs with the given property.

The main idea of our algorithms is that for some problems(biconnected triangulations [LN01], and triconnected triangulations [N04]) we can define a tree (not a general graph) as the graph H of the reverse search method. Thus our algorithms do not need to find a spanning tree of H, since H itself is a tree. With some other ideas we give the following two simple but efficient algorithms.

Our first algorithm generates all simple connected rooted plane graphs with at most $m(m > 0)$ edges. *Simple* means there is neither self loops nor multiple edges. A *rooted* plane graph means a plane graph with one designated "root" edge on the outer face. For instance there are nine simple connected rooted plane graphs with at most three edges, as shown in Fig. 1(a). The root edges are depicted by thick grey lines. However, there are only five simple

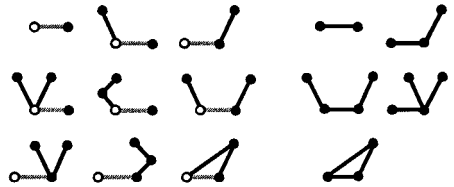


Figure 1: (a) Connected rooted plane graphs, and (b) Connected (non-rooted) plane graphs.

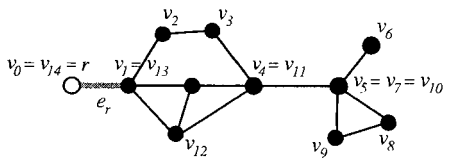


Figure 2: A connected rooted plane graph.

connected (non-rooted) plane graphs with at most three edges. See Fig. 1(b). The algorithm uses $O(m)$ space and runs in $O(g(m))$ time, where $g(m)$ is the number of nonisomorphic connected rooted plane graphs with at most m edges. The algorithm generates such graphs without duplications. So the algorithm generates each graph in $O(1)$ time on average. The algorithm does not output the entire graph but the difference from the previous graph.

By modifying the algorithm we can generate all connected (non-rooted) plane graphs with at most m edges in $O(m^3)$ time per graph.

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 shows a tree structure among connected rooted plane graphs. Section 4 presents our first algorithm to generate all connected rooted plane graphs. Then, by modifying the algorithm we give an algorithm to generate all connected (non-rooted) plane graphs. Section 5 analyzes the running time of our algorithm. Finally Section 6 is a conclusion.

2 Preliminaries

In this section we give some definitions.

Let G be a connected graph with m edges. In this paper all graphs are simple, so there is neither self loops nor multiple edges. An edge connecting vertices u and w is denoted by (u, w). The *degree* of a vertex v is the number of neighbors of v in G.

A graph is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. A *plane* graph is a planar graph with a fixed planar embedding. A plane graph divides the plane into connected regions called *faces*. The unbounded face is called *the outer face*, and other faces are called *inner faces*. We regard *the contour* of a face as the clockwise cycle formed by the vertices on the boundary of the face. We denote the contour of the outer face of plane graph G by $C_o(G)$. For instance, in Fig. 2, $C_o(G) = v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7 = v_5, v_8, v_9, v_{10} = v_5, v_{11} = v_4, v_{12}, v_{13} = v_1, v_{14} = v_0$. Note that a vertex may appear several times on $C_o(G)$. We say each v_i on $C_o(G)$ is an *appearance* of a vertex. For instance v_5, v_7 and v_{10} are the appearances of the same vertex $v_5 = v_7 = v_{10}$. A *rooted* plane graph is a plane graph with one designated edge $e_r = (v_r, v_l)$ on $C_o(G)$. We assume v_l succeeds v_r on $C_o(G)$. The designated edge is called *the root edge*, and vertex v_l is called *the root vertex*. Note that a rooted plane graph has one or more edges. From now on we write r for the root vertex.

3 The Removing Sequence and the Family Tree

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

Let G be a connected rooted plane graph with two or more edges. Let $e_r = (v_{k-1}, v_0)$ be the root edge of G and $C_o(G) = v_0(= r), v_1, v_2, \ldots, v_{k-1}, v_0(= r)$. Note that v_0 succeeds v_{k-1} on $C_o(G)$.

We classify the edges on $C_o(G)$ into three types as follows. If e on $C_o(G)$ is included in a cycle of G then e is *a cycle edge*. Otherwise, if at least one vertex of e has degree 1 then e is *a pendant*. Otherwise e is *a bridge*. We can observe if we remove a bridge from G then the resulting graph is disconnected. For instance, in Fig. 2, edge (v_2, v_3) is a cycle edge, edge (v_5, v_6) is a pendant, and edge (v_4, v_5) is a bridge.

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

Since G is a rooted plane graph, the resulting

graph after removing a removable edge is also a rooted plane graph with the same root edge.

We have the following lemma.

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

Proof. Let G be a connected rooted plane graph with two or more edges, with the root edge (v_{k-1}, v_0), and $C_o(G) = v_0(= r), v_1, v_2, \ldots, v_{k-1}, v_0$. Let $e_1 = (v_0, v_1) \neq (v_{k-1}, v_0) = e_r$ be any edge on $C_o(G)$. Now e_1 must be one of the three types, that is, a bridge, a pendant or a cycle edge. If e_1 is a pendant or a cycle edge, it is removable, and we are done. Otherwise e_1 is a bridge, then on $C_o(G)$ the next edge of e_1 is either a pendant, a bridge or a cycle edge. By repeating this procedure we can find at least one pendant or cycle edge, which is removable.

$\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

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

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

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

4 Algorithms

Given S_m we can construct T_m by the definition, possibly with huge space and much running time. However, how can we construct T_m efficiently only given an integer m? Our idea [LN01, N04] is by reversing the removing procedure as follows.

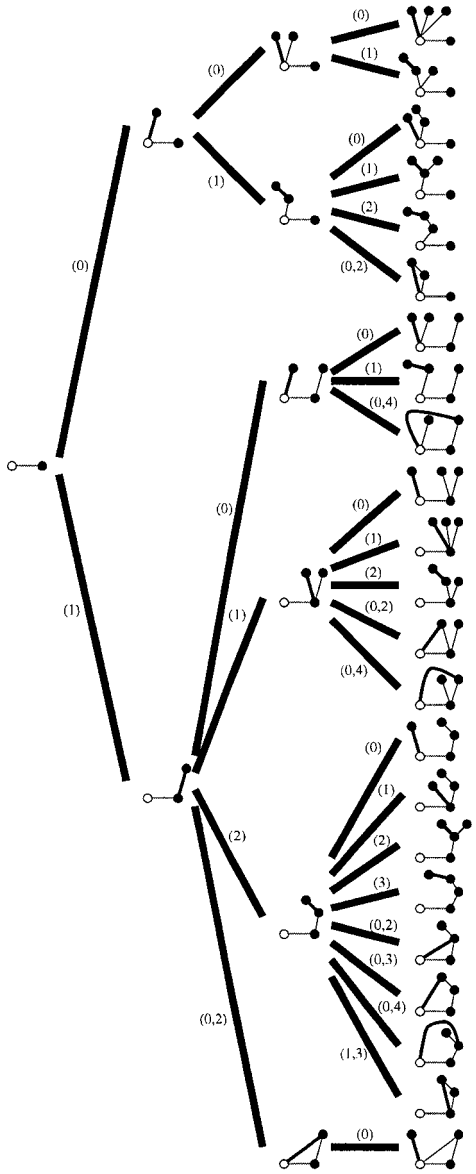
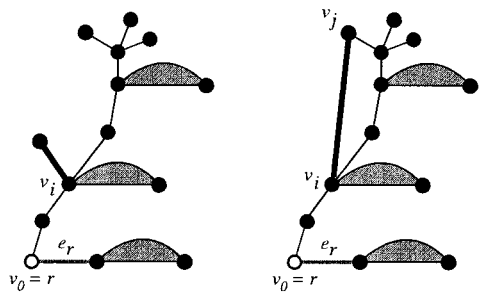


Figure 3: The family tree T_4.



(a) $G(i)$ (b) $G(i,j)$

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

Given a connected rooted plane graph G in S_m with at most $m-1$ edges, we wish to find all child graphs of G. Let e_r be the root edge. Let $C_o(G) = v_0(=r), v_1, \ldots, v_{k-1}, v_0(=r)$, and (v_{a-1}, v_a) be the first removable edge of G. Note that k is the number of appearances of the vertices on the contour of the outer face. Since K_2 has no removable edge, for convenience, we regard $e_1 = (v_0, v_1)$ as the first removable edge for K_2. We denote by $G(i)$, $0 \leq i < k$, the rooted plane graph obtained from G by adding a new pendant at v_i, and by $G(i,j)$, $0 \leq i < j < k$, the rooted plane graph obtained from G by adding a new cycle edge connecting v_i and v_j on the outer face of G, as shown in Fig. 4. We can observe that each child of G is either $G(i)$ or $G(i,j)$ for some i and j, and $G(i)$ or $G(i,j)$ is a child graph of G if and only if the newly added edge of $G(i)$ or $G(i,j)$ is the first removable edge of $G(i)$ or $G(i,j)$.

If (v_{a-1}, v_a) is the first removable edge of G, then edges $(v_0, v_1), (v_1, v_2), \ldots, (v_{a-2}, v_{a-1})$ are bridges or the root edge, and vertices $v_0, v_1, v_2, \ldots, v_a$ form a path on $C_o(G)$. We call this path *the critical path* of G and denote it $P_c(G)$. For instance, in Fig. 2, $P_c(G)$=(v_0, v_1, v_2).

Now we are going to find all child graphs of G. We have the following two cases to consider. Let $b(i)$ be the largest integer satisfying $v_i = v_{b(i)}$. Thus $v_{b(i)}$ is the last appearance of v_i on $C_o(G)$.

Case 1: The first removable edge (v_{a-1}, v_a) of G is a pendant. (including the special case when G is K_2)

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

Then consider graphs $G(i,j)$, $0 \leq i < j < k$. For each i and $j, (i < j)$ such that (1) $v_i \neq v_j$, (2) $0 \leq i \leq a-1$, (3) (v_i, v_j) is not an edge of G, and (4) $j < b(i)$, the newly added edge in $G(i,j)$ is the first removable edge of $G(i,j)$, thus $P(G(i,j)) = G$. Note that if $v_i = v_j$ edge (v_i, v_j) is a self loop, and so $G(i,j)$ is not simple. Also if G has edge (v_i, v_j) then $G(i,j)$ has a multiple edge, and so $G(i,j)$ is not simple. If $i \geq a$, then the newly added edge in $G(i,j)$ is not the first removable edge of $G(i,j)$, since (v_{a-1}, v_a) is still removable, thus $P(G(i)) \neq G$. Otherwise, $0 \leq i \leq a-1$ and $j > b(i)$ holds. Now edge (v_{i-1}, v_i) becomes removable in $G(i,j)$, so $P(G(i)) \neq G$.

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

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

Then consider graphs $G(i,j)$, $0 \leq i < j < k$. For each i and $j, (i < j)$ such that (1) $v_i \neq v_j$, (2) $0 \leq i \leq a-1$, (3) (v_i, v_j) is not an edge of G, and (4) $j < b(i)$, the newly added edge in $G(i,j)$ is the first removable edge of $G(i,j)$, thus $P(G(i,j)) = G$. If $i \geq a$, then the newly added edge in $G(i,j)$ never becomes the first removable edge of $G(i,j)$, so $P(G(i)) \neq G$. Otherwise, $0 \leq i \leq a-1$ and $j > b(i)$ holds. Now edge (v_{i-1}, v_i) becomes removable in $G(i,j)$, so $P(G(i)) \neq G$.

Based on the case analysis above we can find all child graphs of any given graph in S_m. If G has l child graphs, then we can find them in $O(l)$ time with a suitable data structure, which will be described in Section 5. This is an intuitive reason why our algorithm generates each graph in $O(1)$ time per graph on average.

And recursively repeating this process from the root of T_m corresponding to K_2 we can traverse T_m without constructing the whole part of T_m at once. During the traversal of T_m, we assign a label (i) or (i,j) to each edge connecting G and either $G(i)$ or $G(i,j)$ in T_m, as shown in Fig. 3. Each label denotes how to add a new edge to G to generate a child graph $G(i)$ or $G(i,j)$, and each sequence of labels on a path starting from the root specifies a graph in S_m. For instance, the sequence $(1)(0,2)(0)$ specifies the right-bottom graph in Fig. 3. During

our algorithm we will maintain these labels only on the path from the root to the "current" vertex of T_m, because those labels are enough information to generate the "current" graph. To generate the next graph, we need to maintain some more information only for the graphs on the "current" path, which has length at most m. This is an intuitive reason why our algorithm uses only $O(m)$ space, while the number of graphs may not be bounded by a polynomial in m.

Our algorithm is as follows.

Procedure find-all-child-graphs(G)
begin
01 Output G {Output the difference from the previous graph.}
02 Assume (v_{a-1}, v_a) is the first removable edge of G.
03 **if** G has exactly m edges **then return**
04 **for** $i = 0$ **to** $a - 1$ {Case 1 and 2}
05 **find-all-child-graphs**($G(i)$)
06 **if** (v_{a-1}, v_a) is a pendant **then** {Case 1}
07 **find-all-child-graphs**($G(a)$)
08 **for** $i = 0$ **to** $a - 1$ {Case 1 and 2}
09 **for** $j = i + 2$ **to** $b(i) - 1$
10 **if** $v_i \neq v_j$ and (v_i, v_j) is not an edge of G **then**
11 **find-all-child-graphs**($G(i, j)$)
end

Algorithm find-all-graphs(T_m)
begin
1 Output K_2
2 $G = K_2$
3 **find-all-child-graphs**($G(0)$)
4 **find-all-child-graphs**($G(1)$)
end

We have the following theorem. The proof is given in Section 5.

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

We can modify our algorithm so that it outputs all connected (non-rooted) plane graphs with at most m edges, as follows. At each vertex v of the family tree T_m, the graph G corresponding to v is checked whether the sequence of labels of G (with the root edge) is the lexicographically first one among the k sequences of labels of G for the k choices of the root edge on $C_o(G)$, and only if so G is output. Thus we can output only the canonical representative of each isomorphism class. A similar method is appeared in [LN01, N04].

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

Proof. For each graph corresponding to a vertex of T_m we construct $k \leq m$ of sequences of labels corresponding to the k choices for the root edge on $C_o(G)$ in $O(m)$ time for each sequence, and find the lexicographically first one in $O(km)$ time. And for each output graph, our tree may contain k of isomorphic ones corresponding to the k choices for the root edge. Thus the algorithm runs in $O(k^2 m \cdot h(m)) = O(m^3 \cdot h(m))$ time. The algorithm clearly uses $O(m)$ space. $\mathcal{Q.E.D.}$

5 Proof of Theorem 4.1

In this section we give a proof of Theorem 4.1, that is if G has l child graph how we can find them in $O(l)$ time.

Given a connected rooted plane graph G in S_m with at most $m - 1$ edges, we are going to find all child graphs of G by algorithm **find-all-child-graphs**. Let (v_{k-1}, v_0) be the root edge of G, $C_o(G) = v_0(= r), v_1, v_2, \ldots, v_{k-1}, v_0(= r)$, and (v_{a-1}, v_a) be the first removable edge of G.

If G has l child graphs of type $G(i)$, by only maintaining the critical path $v_0, v_1, \ldots, v_a$, we can find such child graphs in $O(l)$ time. See lines 04–07 of **find-all-child-graphs**.

On the other hand, if G has l' child graphs of type $G(i, j)$, we need to maintain a slightly complicated data structure to find all such child graphs in $O(l')$ time. Note that if either (1) $v_i = v_j$, or (2) G has an edge (v_i, v_j), then $G(i, j)$ is not simple and $G(i, j)$ is not a child graph of G, so we need to efficiently skip such j's at line 10. For each of the other j's, we need to generate $G(i, j)$, since those are child graphs of G.

Our idea is as follows. Let v_i be an appearance of a vertex on the critical path of G. We say that an appearance v_j on $C_o(G)$ is *dead* with respect to v_i if either (1) $v_i = v_j$, or (2) G has an edge (v_i, v_j). To skip dead appearances efficiently, for each vertex v_i on the critical path, we maintain a list of successive dead appearances with respect to v_i, which allow us to skip each run of successive dead appearances in $O(1)$ time. After each time skipping successive

dead appearances we can always generate a child graph of G corresponding to the next "non-dead" appearance. Thus l' child graphs of type $G(i,j)$ can be generated in $O(l')$ time. The details are as follows.

Let $v_{a(i)}$ and $v_{b(i)}$ be the first and last appearance of v_i on $C_o(G)$. Let P_i be the subpath from $v_{a(i)}$ to $v_{b(i)}$ on $C_o(G)$. A maximal subpath P_i^c of P_i is called a *dead path* of v_i if all appearances $v_c, v_{c+1}, \ldots$ on P_i^c are dead with respect to v_i. For example, the graph in Fig. 5 has 6 dead paths of v_3: $P_3^3 = (v_3, v_4, v_5, v_6)$, $P_3^8 = (v_8)$, $P_3^{10} = (v_{10})$, $P_3^{12} = (v_{12}, v_{13}, v_{14}, v_{15}, v_{16})$, $P_3^{18} = (v_{18}, v_{19}, v_{20}, v_{21})$, $P_3^{23} = (v_{23}, v_{24})$. They appear on $C_o(G)$ in this order. For each $v_i (0 \leq i \leq a-1)$, we maintain all dead paths as a list, and we call the list *the zombie list* of v_i. Using the zombie list we can skip each run of successive dead appearances in $O(1)$ time. After each time we skip a dead path, we can always generate at least one child graph. Thus, we can generate each child graph of type $G(i,j)$ in $O(1)$ time.

Now we show how to prepare those data structures for each child graph.

Given a connected rooted plane graph G and the zombie list of each vertex on the critical path, we are going to generate all child graphs, and for each child graph we prepare the zombie list of each vertex on the new critical path by modifying the list for G.

We have the following two cases.

Case1: Child graphs of type $G(i)$.

We have the following two cases.

Case1(a): $i = a$.

The first removable edge of G is a pendant, since otherwise the first removable edge of G is a cycle edge and $G(i)$ is not a child graph of G. Appending the new edge to the critical path of G generates the critical path of $G(i)$. The zombie list of each v_l, $0 \leq l \leq a-2$, in $G(i)$ is identical to the ones in G.

The zombie list of v_{a-1} in $G(i)$ is derived by dividing the first dead path P of v_{a-1} in G as follows. Let $P = (v_{a-1}, v_a, v_1', v_2', \ldots)$ then we divide P into two dead paths $P_1 = (v_{a-1}, v_a)$ and $P_2 = (v_a, v_1', v_2', \ldots)$. Note that adding the new edge generates one more appearance of v_a. See an example in Fig. 6(a). The dead path P_2^2 in Fig. 6(a) is divided into P_2^2 and P_2^3. Other dead paths of v_{a-1} in $G(i)$ are identical to the ones in G.

The zombie list of v_a consists of one dead path $P = (v_a, v_x, v_a)$, where v_x is the other end vertex of the new edge.

Thus we can modify the zombie list of each vertex on the critical path in $O(1)$ time.

Case1(b): Otherwise.

The critical path of $G(i)$ is $v_0, v_1, \ldots, v_i, v_x$, where v_x is the other end vertex of the new edge.

The zombie list of each v_l, $0 \leq l \leq i-1$ in $G(i)$ is identical to the zombie list of G.

The zombie list of v_i is derived by appending (v_i, v_x) as the prefix to the first dead path of v_i, where v_x is the other end vertex of the new edge. See an example in Fig. 6(b). By appending (v_2, v_x) into the dead path P_2^2 of v_2 in G, the dead path P_2^2 of $G(i)$ is derived. Note that the other dead path of v_i in $G(i)$ is identical to the ones in G.

Thus we can modify the zombie list of each vertex on the critical path in $O(1)$ time.

Case2: Child graphs of type $G(i,j)$.

The critical path of $G(i,j)$ is $v_0, v_1, \ldots, v_i, v_j$.

Note that $v_{i+1}, v_{i+2}, \ldots, v_{j-1}$ are not on $C_o(G(i,j))$. So we need not maintain the zombie list of those. Also each $v_{j+1}, v_{j+2}, \ldots, v_a$ are not on the critical path of $G(i,j)$. So we need not maintain the zombie lists of those.

The zombie list of each v_l, $0 \leq l \leq i-1$, is identical to the zombie list of $G(i)$.

The zombie list of v_i is derived by removing dead paths of v_i up to v_j on $C_o(G)$. If v_{j+1} is dead with respect to v_i in G, then appending (v_i, v_j) into the dead path $P_i^{j+1} = (v_{j+1}, v_{j+2}, \ldots)$ generates the zombie list of v_i in $G(i,j)$. See an example in Fig. 6(c). By appending (v_2, v_5) into the dead path P_2^6 of v_2 in G, the dead path P_2^2 of v_2 in $G(i,j)$ is derived. Otherwise if v_{j+1} is not dead then we append a new dead path $P_i^i = (v_i, v_j)$ into the zombie list of v_i. Other dead paths remain as they are.

Thus we can modify the zombie list of each vertex on the critical path in $O(1)$ time.

By the above case analysis, we can prepare the zombie list of each child graph of G in $O(1)$ time.

Next we estimate the space for zombie lists.

Since the number of dead paths of vertex v is bounded by the degree of v, the space to store the zombie lists of G is bounded by $O(m) = O(n)$.

By maintaining the zombie lists, if G has l' child graphs of type $G(i,j)$, we can find all such child graphs in $O(l')$ time. Thus, the algorithm runs in $O(g(m))$ time, where $g(m)$ is the number of nonisomorphic connected rooted plane graphs with at most m edges.

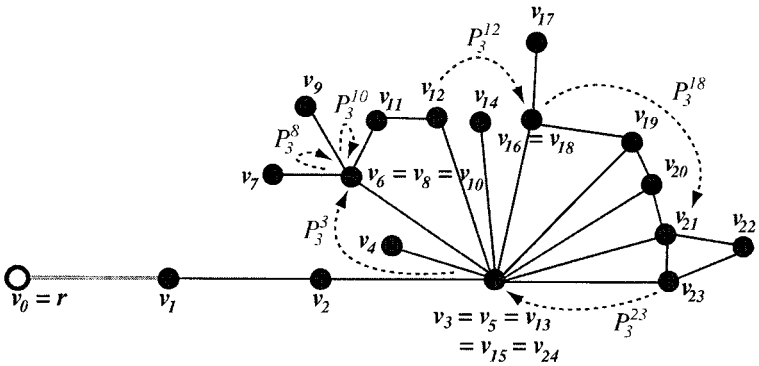
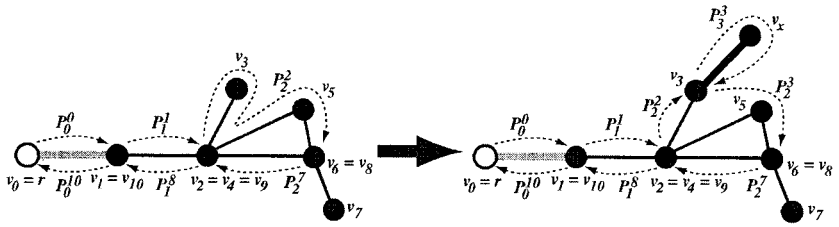
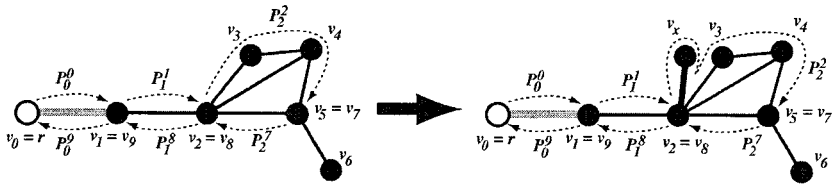


Figure 5: An illustration for the zombie list of v_3.





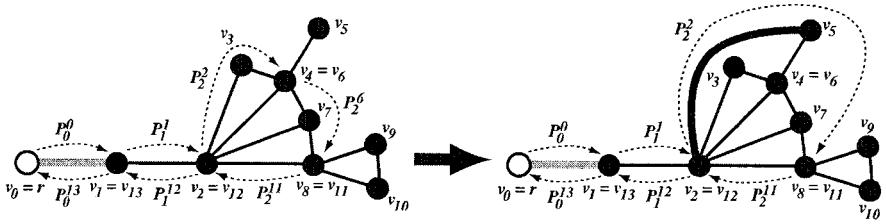


Figure 6: An update of a zombie list for (a), (b) $G(i)$ and (c) $G(i,j)$.

6 Conclusion

In this paper we have given a simple algorithm to generate all connected plane graphs with at most m edges. Our algorithm first defines a family tree whose vertices correspond to graphs, then outputs each graph without duplications by traversing the tree.

By implementing the algorithm one can compute the catalog of plane graphs.

References

[AK96] D. Avis and K. Fukuda, *"Reverse Search for Enumeration,"* Discrete Applied Mathematics, 65, pp.21–46, 1996.

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

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

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

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

[K05] D.E. Knuth, *"The art of computer programming, vol 4, fascicle 2, generating all tuples and permutations,"* Addison-Wesley, 2005.

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

[LN01] Zhangjian Li and S. Nakano, *"Efficient generation of plane triangulations without repetitions,"* Proc. ICALP2001, Lect. Notes in Comput. Sci., 2076, pp.433–443, 2001.

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

[N04] S. Nakano, *"Efficient generation of triconnected plane triangulations,"*, Computational Geometry, Theory and Applications, 27, pp.109–122, 2004.

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

[S97] R.P. Stanley, *"Enumerative combinatorics, vol.1,"* Cambridge Univ. Press, 1997.

[S99] R.P. Stanley, *"Enumerative combinatorics, vol.2,"* Cambridge Univ. Press, 1999.

[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.