A simple linear time algorithm for embedding maximal planar graphs[1]

Hermann Stamm-Wilbrandt
Institut für Informatik III
Universität Bonn
hermann@holmium.informatik.uni-bonn.de

Abstract

All existing algorithms for *planarity testing/planar embedding* can be grouped into two principal classes. Either, they run in linear time, but to the expense of *complex* algorithmic concepts or *complex* data-structures, or they are easy to understand and implement, but require more than linear time [2].

In this paper, a new linear-time algorithm for embedding maximal planar graphs is proposed. This algorithm is both easy to understand and easy to implement. The algorithm consists of three phases which use only simple, local graph-modifications.

In addition to planar embedding, the new algorithm allows to test graphs for maximal planarity. The generation of Straight Line Drawings by a technique of Read [12] can be naturally incorporated into the algorithm. We also demonstrate how to generate *random (maximal) planar graphs.*

The algorithm presented constitutes a first step towards a simple, linear-time solution for embedding general planar graphs.[2] [3]

1 Introduction

One of the main topics in graph theory is the concept of planarity. In the 18th century Euler discovered one of the first results, the well-known "Polyederformel". It was Kuratowski [9] in 1930 who gave a characterization of planar graphs in terms of *forbidden homeomorphic subgraphs*: A graph is planar if, and only if, it possesses no subgraph homeomorphic to either K_5 or $K_{3,3}$. In 1933 Whitney [15] developed the concept of *combinatorial/planar duals* of graphs and showed that a graph has a dual if, and only if, it is planar. In 1936 Wagner [14] proved the existence of *Straight Line Drawings* for planar graphs. Demoucron et al. [4] published the first algorithm with known polynomial time complexity for testing the planarity of graphs in 1964. The required running time is quadratic. In 1974 the first linear time planarity testing algorithm was developed by Hopcroft and Tarjan [8]. It is based on another algorithm by Auslander and Parter [1]. They achieved linear running time by extensive use of *depth first search* on graphs. Two years later Lueker and Booth [11] improved a planarity testing algorithm by Lempel et al. [10] of 1966. This

[1] The sources of an implementation of all algorithms and data structures of this paper can be obtained by anonymous ftp on ftp.cs.uni-bonn.de in directory /pub/paper/infIII as file IAI-TR-93-10.src.tar.Z . This report can be found in the same directory as file IAI-TR-93-10.ps.Z .

[2] This goal has been recognized as a significant open problem in [2].

[3] It is an easy matter to triangulate a planar graph in linear time if given an embedding of it. Therefore any simple linear time algorithm for triangulating a planar graph (without an embedding given) would turn our algorithm into a simple linear time algorithm for planarity testing/planar embedding.

algorithm needs only linear running time by using *PQ-trees*. Both linear time algorithms require lengthy explanation and verification; they are also difficult to implement. Many of the mentioned algorithms may be modified to embedding algorithms; in this case the goal is to reject non-planar graphs as well as to embed planar ones.

In this paper, a new linear-time algorithm for embedding maximal planar graphs is proposed. This algorithm is both easy to understand and easy to implement. It consists of three phases which use only simple, local graph-modifications. Our techniques deeply depend on an algorithm for computing *planar 3-bounded orientations* by Chrobak and Eppstein [3].

Section 2 contains the basic definitions. In Section 3 the *data structures* required for implementation are described. This is followed in Section 4 by a description of the basic operations of the algorithm, i.e. *reduction* and *inverse reduction*. In Section 5 the embedding algorithm for maximal planar graphs is presented. It is extended to additionally testing maximal planarity in Section 6. In Section 7 the generation of Straight Line Drawings for maximal planar graphs is described. Finally in Section 8 an algorithm for generating *random (maximal) planar graphs* is given.

The algorithm presented constitutes a first step towards a simple, linear-time solution for embedding general planar graphs. If a simple linear time algorithm for triangulating any planar graph without an embedding given existed then our algorithm would be extended to a simple linear time algorithm for planarity testing/planar embedding.

2 Basic definitions

The terminology used in this paper follows that of Even [5]. Let $G = (V, E)$ be a planar graph. For each $v \in V$ denote by $INC[v]$ the incidence list of v in G. We consider an *embedding* of G to be an ordering of the incidence lists of G, such that for each $v \in V$ the order of the edges in $INC[v]$ corresponds to a counter-clockwise traversal of the edges in a fixed *embedding of G in the plane*. A simple planar graph is called *maximal planar* if adding any new edge results in a non-planar graph. In this paper the complete (maximal planar) graph on 4 vertices K_4 is considered to be the smallest maximal planar graph w.r.t. the number of vertices. Maximal planar graphs possess no vertices of degree less than 3 because they are triconnected (2.1).

The next 4 lemmas deal with properties of maximal planar graphs. Their proofs can be found in [5].

Lemma 2.1 Maximal planar graphs are triconnected.

Lemma 2.2 The embedding of a planar graph in the plane possesses only triangular faces if, and only if, the graph is maximal planar.

Lemma 2.3 A simple planar graph $G = (V, E)$ is maximal planar if, and only if, $|E| = 3|V| - 6$.

Lemma 2.4 A triconnected planar graph has a unique embedding onto the sphere. After choosing the outer face the rest of the embedding in the plane is also unique. Maximal planar graphs are triconnected and thus possess also unique embeddings in the plane (w.r.t. a chosen outer face).

The next simple lemma deals with a property of *outerplanar graphs* (planar graphs with all vertices lying on a common face); the proof can be found in [5], too.

Lemma 2.5 Each outerplanar graph has at least 2 vertices of degree at most 2.

Definition 1 A vertex v of $G = (V, E)$ is called *small*, if $degree(v) < 18$, otherwise it is called *large*. A vertex $v \in V$ is called *reducible* if it satisfies one of the following conditions:

- $degree(v) \leq 3$
- $degree(v) = 4$ and v has at least 2 small neighbors
- $degree(v) = 5$ and v has at least 4 small neighbors

Lemma 2.6 Each planar graph $G = (V, E)$ with $|V| \geq 4$ has at least 4 reducible vertices.

Proof. Since the number of reducible vertices does not increase if one adds edges, it suffices to prove the lemma for maximal planar graphs. A maximal planar graph with n vertices has $m = 3n - 6$ edges (2.3). Denote by n_i the number of vertices of degree i and by $r_4(r_5)$ the number of reducible vertices of degree 4(5). Maximal planar graphs are triconnected (2.1) and thus $n_0 = n_1 = n_2 = 0$ holds for them.

$$\sum_{i\geq 3} in_i = 2m = 6n - 12 = -12 + \sum_{i\geq 3} 6n_i \Longrightarrow \sum_{i\geq 3}(i-6)n_i = -12$$

$$\Longrightarrow 2n_4 + n_5 = 12 - 3n_3 + \sum_{i\geq 7}(i-6)n_i$$

Counting the edges between non-reducible vertices of degree 4 or 5 and large vertices gives:

$$3(n_4 - r_4) + 2(n_5 - r_5) \leq \sum_{i\geq 18} in_i$$

This leads to:

$$\frac{2}{3}\sum_{i\geq 18} in_i \geq (2n_4 + n_5) - 2r_4 - \frac{4}{3}r_5 + \frac{1}{3}n_5 = 12 - 3n_3 - 2r_4 - \frac{4}{3}r_5 + \frac{1}{3}n_5 + \sum_{i\geq 7}(i-6)n_i$$

$$\Longrightarrow \sum_{i\geq 18}(2i)n_i - \sum_{i\geq 7}(3i-18)n_i \geq 36 - 9n_3 - 6r_4 - 4r_5 + n_5$$

Now we can bound the sums from above by 0:

$$\sum_{i\geq 18}(2i)n_i - \sum_{i\geq 7}(3i-18)n_i = \sum_{i\geq 18}(18-i)n_i - \sum_{7\leq i<18}(3i-18)n_i \leq 0$$

Eliminating the sums we get:

$$9n_3 + 6r_4 + 4r_5 \geq 36 + n_5$$

$$\Longrightarrow \boxed{n_3 + r_4 + r_5} \geq n_3 + \frac{2}{3}r_4 + \frac{4}{9}r_5 \geq 4 + \frac{1}{9}n_5 \geq \boxed{4}$$

The proof is completed since the number of reducible vertices is given by $n_3 + r_4 + r_5$. □

Lemma 2.7 Let v be any vertex of a maximal planar graph $G = (V, E)$ and let $N(v)$ denote the neighborhood of v. There are at least two vertices in $N(v)$ which are adjacent to exactly two other vertices of $N(v)$.

Proof. Denote by $v_1, \ldots, v_k$ the vertices of $N(v)$ in the counter-clockwise order in which they appear around v in a fixed embedding of G in the plane. Since G is maximal planar, all faces are triangles (2.2). Thus v_i and $v_{(i \bmod k)+1}$ are adjacent for $1 \leq i \leq k$ and therefore each v_i has at least two neighbors in $N(v)$. The induced graph on the vertices of $N(v)$ is outerplanar and thus there are at least two vertices of $v_1, \ldots, v_k$ with at most two neighbors in $N(v)$ (2.5). This completes the proof. □

3 Data structures

Here our special view on the data structures is presented which allows us to implement the algorithms in linear time.

For the rest of the paper we represent the vertices and edges of a graph by positive numbers. Each vertex has information on its *predecessor* and *successor* in the doubly linked list of vertices of the graph. This allows iteration over the vertices as in e.g. the statement $forall\_vertices(v, G)$. Each vertex v additionally knows about its (doubly linked) incidence list $INC[v]$. Each edge $e = \{u, v\}$ knows about its incident vertices $source(e) = u$ and $target(e) = v$ without the usual directed interpretation. Additionally each edge $e = \{u, v\}$ knows about its positions in $INC[u]$ and $INC[v]$ and its predecessor and successor in the doubly linked list of edges of the graph (e.g. for iteration purposes as in $forall\_incident\_edges(e, v)$). The identification of vertices/edges with

numbers allows in an easy way to associate information with them by using ordinary arrays[4]. The information possessed by vertices/edges is not modified after removing them, such that the reinsertion of a vertex/edge into the graph can be done in constant time by just giving its (old) number[5] . Both operations, the normal creation of a new edge e and the reinsertion of an old edge e, allow to specify the positions of e in $INC[source(e)]$ and $INC[target(e)]$; the default positions are either at the ends of the lists or the position before removal. The non-existence of a vertex/edge as a result of some function is indicated by the value 0. The *cyclic successor/cyclic predecessor* of an edge $e = \{u, v\}$ in e.g. $INC[v]$ is either given by its successor/predecessor (see above) if this is not 0, or it is given by the first/last edge in $INC[v]$.

Lemma 3.1 There is a data structure *vertex_set* N for a graph $G = (V, E)$ providing the following constant time operations: $N\_insert(v)$ $(N = N \cup \{v\})$, $N\_remove(v)$ $(N = N - \{v\})$, $N\_member(v)$ $(v \in N?)$, $N\_empty()$ $(N = \emptyset?)$, $N\_choose()$ (returns any $v \in N$ if $N \neq \emptyset$ and returns 0 otherwise) and $forall(x, N)$ (iteration: $\forall x \in N$). The time necessary to initialize N as the empty vertex_set is $O(|V|)$.

Proof. A vertex_set can be represented by a doubly linked list and an array. The actual members of the vertex_set are those of the doubly linked list. For each vertex in the vertex_set its position within the doubly linked list is stored in the array; for the vertices not in the vertex_set the value 0 is stored in the array. □

Definition 2 A *compaction* of a graph $G = (V, E)$ is a mapping $COMPACT : V \rightarrow \{ E' \mid E' \subseteq E \}$ such that $COMPACT[v] \subseteq INC[v]$ for each vertex v with the following properties:

- $\forall e = \{u, v\} \in E : e \in COMPACT[u] \cup COMPACT[v]$
- $\forall e = \{u, v\} \in E : COMPACT[u] \cap COMPACT[v] = \emptyset$

$COMPACT$ is called *k-bounded* for some constant $k > 0$ if, and only if, there are at most k edges in $COMPACT[v]$ for all vertices v of G.

Lemma 3.2 Planar graphs always possess a *5-bounded* compaction.

Proof. Let $G = (V, E)$ be a planar graph. Due to (2.3) we have $|E| \leq 3|V| - 6$ and $\sum_{v \in V} degree(v) = 2|E| \leq 2(3|V| - 6) < 6|V|$. Therefore planar graphs always possess a vertex of degree at most 5. Now set $H = G$ and repeat the following procedure until $V(H) = \emptyset$:

- find a vertex v of H with $degree(v) \leq 5$
- set $COMPACT[v] = INC[v] = \{e \in E(H) | v \in e\}$
- remove vertex v from H

This procedure defines a 5-bounded compaction of G. □

Corollary 3.3 Given a planar graph $G = (V, E)$ and a 5-bounded compaction $COMPACT$ of G it is decidable in *constant time* whether an edge $\{u, v\}$ is in E or not (simply examine $L = COMPACT[u] \cup COMPACT[v]$. If $\{u, v\} \in E$ it must be a member of L and $|L| \leq 10 = O(1)$).

Definition 3 The data type *urn* is essentially the same as the one used in mathematics for probability experiments. It is a bounded data type providing the following operations:

- urn $U(n)$ creates an empty urn capable of holding a maximum of n elements
- $U\_put(x)$ puts element x into the urn U
- $U\_draw()$ removes a randomly chosen element from U and returns it
- $U\_empty()$ returns *true* if U is empty and $false$ otherwise

[4] This is exactly the way in which *source*, *target*, ... can be realized.

[5] Newly created vertices and edges get the lowest "unused" number available in each case starting with 1. Keep in mind that removing and reinserting vertices/edges are the key operations of the algorithms.

- $U\_clear()$ removes all entries from U

Lemma 3.4 The data type *urn* can be implemented such that all operations mentioned require only constant time.

Proof. An urn can be represented by a dynamic array A and an integer $actual$. The creation of size n is done by allocating an array with n entries (range $1, \ldots, n$) to A and setting $actual = 0$. The element x is put into the urn by $\boxed{actual = actual + 1;\ A[actual] = x;}$. Testing whether the urn is empty is done by $\boxed{\text{return } (actual{==}0);}$ and clearing of the urn is done by $\boxed{actual = 0;}$. The drawing of a random element is done by

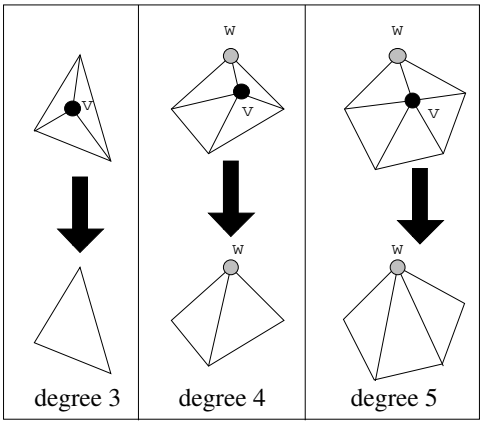
$\boxed{\text{choose randomly } k \in \{1, \ldots, actual\};\ x = A[k];\ A[k] = A[actual];\ actual = actual - 1;\ \text{return } x;}$.

This completes the proof. □

4 Reductions

Definition 4 [3] A *reduction* of a reducible vertex v in a maximal planar graph G=(V,E) is, depending on its degree, defined by:

- If $degree(v) = 3$, then simply remove v from G
- If $degree(v) \in \{4, 5\}$, then let $w \in N(v)$ be a vertex with exactly two neighbors in $N(v)$ (2.7). Now remove v and add new edges $\{w, x\}$ to G for all $x \in (N(v) - (\{w\} \cup N(w)))$.



Lemma 4.1 Let $G = (V, E)$ be a maximal planar graph with $|V| > 4$ and let v be any reducible vertex of G. Applying the reduction of v again results in a maximal planar graph.

Proof. Since G is maximal planar all faces[6] incident to v are triangles (2.2). After the removal of v these faces are joined to a face of length $degree(v)$. Inserting the new edges if $degree(v) > 3$ splits this face into 2 or 3 triangles, otherwise it is already a triangle. Thus the resulting embedding is triangular and therefore the graph is maximal planar (2.2). □

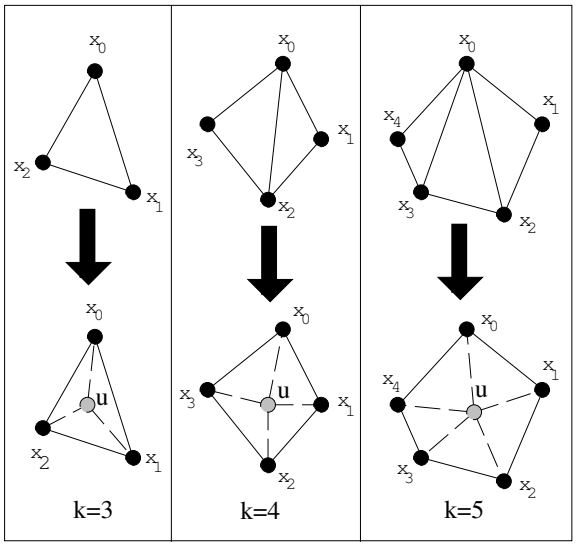
Lemma 4.2 The reduction of a reducible vertex v in a maximal planar graph $G = (V, E)$ can be done in constant time.

Proof. The need of only constant time is obvious for $degree(v) = 3$. In case of $degree(v) = 5$ we can search for the vertex w among the small vertices of $N(v)$ because there are at least 4 of them by definition, and at least two vertices of $N(v)$ have exactly two neighbors in $N(v)$ (2.7). Thus the determination of w can be done in $O(5 + 4 * 18) = O(1)$ time for $degree(v) = 5$. In case of $degree(v) = 4$ we check first if any of the (at least 2) small vertices in $N(v)$ has exactly 2 neighbors in $N(v)$. If so, we take this vertex as w and we are done. Otherwise there exist exactly 2 small neighbors of v which are adjacent to all other vertices of $N(v)$. The remaining 2 large vertices in $N(v)$ cannot be adjacent due to planarity (otherwise there is a K_5 as induced subgraph on $\{v\} \cup N(v)$). Therefore we take any of the latter as w. Thus the case $degree(v) = 4$ needs also $O(4 + 3 * 18) = O(1)$ time. □

Definition 5 Let $G' = (V', E')$ be an embedding of the graph resulting from the reduction of a reducible vertex u of a maximal planar graph $G = (V, E)$ ($V' = V - \{u\}$). The 5-bounded compaction of G' is given by $COMPACT[v]$ for all vertices v of G'. The *inverse reduction* for the vertex u and the graph G' is the inverse operation to the reduction of u in G. It is defined depending on $k = degree(u)$ in G by:

[6] (of any fixed embedding of G in the plane (2.4))

- if $k = 3$ denote by x_0, y, z the neighbors of u in G. Find edges $\{x_0, y\}$,$\{y, z\}$ and $\{z, x_0\}$ of G' in constant time (3.3). If $\{x_0, y\}$ is the cyclic successor of $\{z, x_0\}$ in $INC[x_0]$ set $x_1 = y$ and $x_2 = z$ else set $x_1 = z$ and $x_2 = y$.
- if $k = 4$ denote by $\{x_0, x_2\}$ the edge added during the reduction of u in G. Let $\{x_0, x_1\}$ be the cyclic successor of $\{x_0, x_2\}$ in $INC[x_0]$ and $\{x_0, x_3\}$ be its cyclic predecessor in $INC[x_0]$. Remove edge $\{x_0, x_2\}$ from G' and from $COMPACT[x_0]$, $COMPACT[x_2]$ respectively.
- if $k = 5$ denote by $\{x_0, y\}$ the first edge and by $\{x_0, z\}$ the second edge added during the reduction of u in G. If $\{x_0, y\}$ is the cyclic successor of $\{x_0, z\}$ in $INC[x_0]$ set $x_2 = y$ and $x_3 = z$ else set $x_2 = z$ and $x_3 = y$. Let $\{x_0, x_1\}$ be the cyclic successor of $\{x_0, x_2\}$ in $INC[x_0]$ and $\{x_0, x_4\}$ be the cyclic predecessor of $\{x_0, x_3\}$ in $INC[x_0]$. Remove edges $\{x_0, x_2\}$ and $\{x_0, x_3\}$ from G' and from either $COMPACT[x_0]$ or $COMPACT[x_2]$ or $COMPACT[x_3]$.



The other edges can be found in constant time, too. For instance $\{x_1, x_2\}$ in case $k = 4$ is the result of *G_cyclic_incident_succ*$(\{x_0, x_1\}, x_1)$. Now reinsert the vertex u in the graph G'. Reinsert edges $\{u, x_i\}$ between edges $\{x_{(i-1) \bmod k}, x_i\}$ and $\{x_i, x_{(i+1) \bmod k}\}$ in $INC[x_i]$ successively for $i = k-1, k-2, \ldots, 1, 0$ and append them at the end of $INC[u]$. Thus the order of $INC[u]$ is given by $\{\{u, x_{k-1}\}, \{u, x_{k-2}\}, \ldots, \{u, x_0\}\}$. Set $COMPACT[u] = INC[u]$.

Lemma 4.3 The *inverse reduction* for vertex u and graph G' results in an embedding of G and a 5-bounded compaction of G.

Proof. Since $COMPACT$ is a 5-bounded compaction for G' and $|COMPACT[u]| = |INC[u]| = k \leq 5$ the modified $COMPACT$ is a 5-bounded compaction for G. Since the edge $\{u, x_i\}$ is inserted in the right place in $INC[x_i]$ and the order of edges in $INC[u]$ corresponds to a counter-clockwise traversal of the edges incident to u the *inverse reduction* results in an embedding of G. □

Lemma 4.4 The inverse reduction above can be done in constant time if

- (the number representing) u is given
- – (the numbers representing) x_0, y and z are given in case $k = 3$
  – (the number representing) $\{x_0, x_2\}$ is given in case $k = 4$
  – (the numbers representing) $\{x_0, y\}$ and $\{x_0, z\}$ are given for $k = 5$
- (the numbers representing) the removed edges $\{u, x_i\}$ are known for $0 \leq i < k$.

Proof. Only constant time is necessary since the use of the compaction and examining the incidence lists takes only constant time. □

5 Embedding maximal planar graphs

The algorithm for embedding maximal planar graphs consists of 3 phases. During the first phase the graph $G = (V, E)$ is reduced to the K_4 by repeated reductions. Then in the second phase the resulting graph, the K_4, is embedded. In the third phase all reductions of the first phase are

undone by the corresponding inverse reductions in reverse order. This results in an embedding of each graph between K_4 and G and therefore in an embedding of G itself.

The set of reducible vertices is maintained in the vertex_set $REDUCIBLE$, which is initially filled by all reducible vertices $v \in V$. The next lemma shows that updating $REDUCIBLE$ after a reduction can be done in constant time.

Lemma 5.1 Let $G = (V, E)$ be a maximal planar graph, let $REDUCIBLE$ be the vertex_set of reducible vertices of G and $v \in REDUCIBLE$. Denote by $G' = (V', E')$ the graph resulting from G by the reduction of vertex v ($V' = V - \{v\}$). The necessary updates for $REDUCIBLE$ to be the set of reducible vertices of G' can be done in constant time.

Proof. [3] First we need a procedure to update the (non-)presence of a vertex x in $REDUCIBLE$ in constant time (3.1). This is simply done by:

```
Update(vertex x, vertex_set REDUCIBLE)
if (x is reducible) REDUCIBLE_insert(x);
else                REDUCIBLE_remove(x);
```

Having done the reduction of vertex v in G this changes only the degrees of vertices of $N(v)$. It can have the following effects:

- A vertex $x \in N(v)$ had $degree(x) > 5$ but has now $degree(x) \leq 5$. Therefore x may be reducible.
- A vertex $x \in N(v)$ had $degree(x) \leq 5$ but has now $degree(x) > 5$. Now x may have lost reducibility.
- A vertex $x \in N(v)$ was large, but now is small. Any neighbor z of x with $degree(z) \leq 5$ may now be reducible.
- A vertex $x \in N(v)$ was small, but now is large. Any neighbor z of x with $degree(z) \leq 5$ may now have lost its reducibility.

Thus all that needs to be done is looking at the local neighborhood $H = N(v)$ of v after the reduction of v:

```
Update_local(vertex_set H, vertex_set REDUCIBLE)
vertex x, y;
forall(x, H)
    if ((x is small) or (x was small before reducing v))
        { Update(x); forall (y, N(x)) if (degree(y) ≤ 5) Update(y, REDUCIBLE); }
```

Any reduction of vertex v changes the degree of all $x \in N(v)$ by at most one. Thus the forall–loops from above run only for a constant number of instances. This leads to constant running time of Update_local($N(v), REDUCIBLE$) as a whole. □

Now the described algorithm:

```
Maximal_planar_embedding(graph G)
stack S;                /* initially empty */
PHASE_I(G, S);
PHASE_II: rearrange the incidence lists of G = K4 to be an embedding;
PHASE_III(G, S);
```

Next the procedure PHASE_I:

```
PHASE_I(graph G, stack S)
vertex_set REDUCIBLE, H; vertex v; integer k; edge e_1, e_2;
forall_vertices(v,G) if (v is reducible) REDUCIBLE_insert(v);
while (G_number_of_vertices() > 4)
    {
    v = REDUCIBLE_choose(); H = G_adjacent_nodes(v); REDUCIBLE_remove(v);
    k = G_degree(v);
    remove all edges e incident to v and push them on S;
    remove v and push it on S;
    if (k>=4) find vertex w ∈ H with exactly 2 neighbors in H; /* according to (4.2) */
    if (k==4) { let x be the only vertex in H − (N(w) ∪ {w});
                create new edge e_1 = {w, x}; push e_1 on S; }
    if (k==5) { let x and y be the two vertices in H − (N(w) ∪ {w});
                create new edges e_1 = {w, x} and e_2 = {w, y}; push e_1 and e_2 on S; }
    Update_local(H, REDUCIBLE);
    push k on S;
    }
```

Lemma 5.2 PHASE_I runs in time $O(|V|)$ for every maximal planar graph $G = (V, E)$.

Proof. The initialization takes $O(|V|)$ time (3.1), and each of the $|V| - 4$ runs of the while-loop needs constant time (4.2,5.1). □

Now the procedure PHASE_III:

```
PHASE_III(graph G, stack S)
edge e, e_1, e_2, h_1, h_2, h_3, h_4, h_5; vertex v, y, z, x_0, x_1, x_2, x_3, x_4; integer k;
compaction COMPACT;
forall_edges(e,G) append edge e to COMPACT[G_source(e)];
while (S is not empty)
    {
    pop k from S;
    if (k==3) { pop vertex v from S; pop edges h_1, h_2, h_3 from S;
                x_0 = G_opposite(v, h_1); y = G_opposite(v, h_2); z = G_opposite(v, h_3); }
    if (k==4) { pop edge e_1 from S;
                pop vertex v from S; pop edges h_1, h_2, h_3, h_4 from S;
                x_0 = G_source(e_1); x_2 = G_target(e_1); }
    if (k==5) { pop edges e_2 and e_1 from S;
                pop vertex v from S; pop edges h_1, h_2, h_3, h_4, h_5 from S;
                denote by x_0 the vertex which e_1 and e_2 have in common;
                y = G_opposite(x_0, e_1); z = G_opposite(x_0, e_2); }
    do inverse reduction of v as in Definition 5; /* including update of COMPACT */
    }
```

Lemma 5.3 PHASE_III runs in time $O(|V|)$ for every maximal planar graph $G = (V, E)$.

Proof. The initialization takes $O(|E|) = O(|V|)$ time. Each of the $|V| - 4$ runs of the while-loop needs constant time because the necessary information for doing the inverse reduction in constant time (4.4) is provided by PHASE_III (e.g. (the numbers of) the old edges $h_1, h_2, h_3[, h_4[, h_5]]$). □

Lemma 5.4 Maximal_planar_embedding() applied to any maximal planar graph $G = (V, E)$ requires $O(|V|)$ time and space.

Proof. PHASE_I and PHASE_III need $O(|V|)$ time (5.2,5.3) and PHASE_II (rearranging the K_4) can be done in constant time. Thus Maximal_planar_embedding() needs $O(|V|)$ time. In each reduction of PHASE_I there are at most 2 new edges created, and thus at most $5|V|$ different edges are in G. Thus the algorithm works if the arrays which realize e.g. *source* have size $5|V|$. At most 7 edges, 1 vertex, and 1 integer are pushed on the stack during one reduction step of PHASE_I. Thus the stack has at most $9|V|$ entries. □

Lemma 5.5 For every maximal planar graph $G = (V, E)$ the algorithm Maximal_planar_embedding() produces an embedding in G.

Proof. Due to G being maximal planar every reduction step of PHASE_I produces again a maximal planar graph (4.1). The final reduction step results in G being the K_4. PHASE_II embeds this graph. The reconstruction of the original graph is done by inverse reductions of PHASE_III, each of them resulting in an admissible embedding of the present graph (4.3). Therefore the final graph, which is G again, is an embedding. □

6 Maximal planarity test

The embedding algorithm from the last section works well for maximal planar graphs. In this section we show which additions to Maximal_planar_embedding() enable the detection of *not maximal planar* graphs in linear time during the course of the embedding algorithm.

Assume that $G = (V, E)$ is the graph to be tested for maximal planarity and, in case of maximal planarity, the one to be embedded. There are several tests which detect not maximal planar graphs:

1. (in PHASE_I before initializing REDUCIBLE)
   (a) if ($|E| \neq 3|V| - 6$) abort with not_maximal_planar;
   (b) if (G contains parallel edges or self-loops) abort with not_maximal_planar;
2. (in PHASE_I inside the reduction loop)
   (a) if there are no reducible vertices left abort with not_maximal_planar;
   (b) if there are parallel edges in $INC[v]$ abort with not_maximal_planar;
   (c) if there is no vertex w with exactly 2 neighbors in $N(v)$ abort with not_maximal_planar;
3. (after PHASE_I)
   (a) if (G contains parallel edges) abort with not_maximal_planar;
4. (in PHASE_III during inverse reductions of vertex v, case $k == 3$)
   (a) if edges $\{x_0, y\}, \{y, z\}, \{z, x_0\}$ do not lie on a common face w.r.t. the (actual) embedding of G abort with not_maximal_planar;
5. (in PHASE_III during inverse reductions of vertex v, case $k \in \{3, 4, 5\}$)
   (a) if there is an old edge e from the stack S with $G\_oppposite(v, e) \notin \{x_0, x_1, x_2[, x_3[, x_4]]\}$ abort with not_maximal_planar.

Lemma 6.1 If the modified algorithm aborts with not_maximal_planar, then the input graph $G = (V, E)$ is not maximal planar. Tests **1a** and **1b** require time $O(|V|)$. The remaining tests can be done in constant time.

Proof. Maximal planar graphs pass test **1a** (2.3) and thus aborting is correct. If the number of edges is not known from the data structure, then start an iteration over all edges and end after having seen $3|V| - 5$ edges or completing the iteration. This leads to $O(|V|)$ running time of **1a**.

Test **1b** is passed by maximal planar graphs by definition and thus aborting here is correct, too. Testing for self-loops and parallel edges in $O(|E|) = O(|V|)$ time is easily done using stable bucket_sort by

SIMPLE_GRAPH(graph G)
array A, B; edge e; vertex v; integer n; forall_edges(e, G) if $(G\_source(e) == G\_target(e))$ abort with self_loop_present; $n = 0$; forall_vertices(v, G) { $n = n + 1$; $A[v] = n$; } forall_edges(e, G) $B[e] = MIN(A[G\_source(e)], A[G\_target(e)])$; sort the edges using bucket_sort in the range $\{1, \ldots, n\}$ with the mapping B; forall_edges(e, G) $B[e] = MAX(A[G\_source(e)], A[G\_target(e)])$; sort the edges using bucket_sort in the range $\{1, \ldots, n\}$ with the mapping B; check whether any two consecutive edges in the list of edges of G are parallel, if so abort with parallel_edges_found;

The two bucket_sorts force parallel edges to consecutive positions in the list of edges and bucket_sort of the edges of G in the range $\{1,\ldots,n\}$ needs $O(|E|+n) = O(|V|)$ time.

Planar graphs always possess reducible vertices (2.6) and thus aborting in test **2a** is correct. Test **2a** can be performed in constant time (3.1).

Reductions applied to maximal planar graphs do not create parallel edges (4.1). Therefore aborting in case **2b** is correct. Test **2b** requires constant time since $|H| \leq 5$ (parallel edges may be created **only in non-planar graphs** by a reduction of a vertex v with $degree(v) = 4$, for the non-existence of an edge between two large vertices is only deduced by planarity arguments).

Maximal planar graphs pass test **2c** (2.7) and thus aborting is correct. Test **2c** requires constant time since $|H| \leq 5$.

Test **3a** is correct for reasons similar to **2b**. It is applied to $G = (V, E)$ with $|V| = 4$ and $|E| = 6$ and therefore it can be done in constant time.

Test **4a** is passed by maximal planar graphs since the reductions of PHASE_I are only undone in PHASE_III by the corresponding inverse reductions. If adjacent edges $e = \{u, v\}$ and $f = \{v, w\}$ do not lie on a common face w.r.t. the embedding represented in graph $G = (V, E)$ then this can be tested in constant time by either $e == G\_cyclic\_incident\_succ(f, v)$ or $e == G\_cyclic\_incident\_pred(f, v)$. In case **4a** there are only 3 edges involved leading to constant time of the whole test.

Note: The graph
$G = (\{1,2,3,4,5,6\}, \{\{i,j\} | i \in \{1,2,3\}, j \in \{4,5,6\}\} \cup \{\{i, (i \bmod 3) + 1\} | i \in \{1,2,3\}\})$ is an example for a non-planar graph detected by test **4a**. It is a $K_{3,3}$ with 3 added edges in order to avoid aborting at test **1a**. If for instance the vertex 6 is chosen for the first reduction, anything works well until test **4a**, i.e. before the inverse reduction of vertex 6 takes place. The graph $G - \{6\}$ is maximal planar and thus its embedding is unique, but the edges $\{\{i, (i \bmod 3) + 1\} | i \in \{1,2,3\}\}$ between the neighbors of 6 do not lie on a common face.

Test **5a** is passed by maximal planar graphs for the same reason as in **4a**, and since at most 5 vertices and edges are concerned **5a** needs only constant time. □

Lemma 6.2 If for an input graph $G = (V, E)$ the modified algorithm does not abort with not_maximal_planar then G is maximal planar and is modified to be an embedding.

Proof. After the modified algorithm was run successfully G represents an embedding because:

- after PHASE_II the actual graph G at that moment is an embedding of the K_4
- after each inverse reduction of PHASE_III the actual graph G is an embedding (4.3)

Thus after the successful completion of PHASE_III we know that G is a maximal planar graph because we have constructed an embedding of it. □

Lemma 6.3 The modified embedding algorithm runs in linear time.

Proof. Assume first that we have a maximal planar input graph $G = (V, E)$. The procedure Maximal_planar_embedding() needs linear time (5.4). Tests **1a** and **1b** are done only once requiring linear time and all other tests need constant time (6.1). Thus the modified embedding algorithm needs linear time in this case. Now if $G = (V, E)$ is not maximal planar the algorithm terminates earlier and thus needs linear time, too. □

7 Straight Line Drawing

There are several algorithms for generating *Straight Line Drawings* of (maximal) planar graphs given their embedding. In [12] a linear time quadratic space algorithm was proposed which can easily be incorporated into PHASE_II and PHASE_III of our embedding algorithm. The benefit of doing so is that this new method requires only linear space. Further no embedding has to be given as an input (compare [12]), because the embedding is determined simultaneously during the course of Maximal_planar_embedding(). The Straight Line Drawing generated will have real-valued x- and y-coordinates in the range $[0 \ldots 1]$ for all vertices. There are other algorithms which

produce Straight Line Drawings with integer coordinates in the range $[1,\ldots,2|V|-4]\times[1,\ldots,|V|]$ for maximal planar graphs ([7],[13]). The algorithm in [13] requires only linear running time if given an embedding of the input graph. The main advantage of those algorithms is that a lower bound on the minimum angle between drawings of adjacent edges is guaranteed. Thus if needing Straight Line Drawings with integer coordinates those algorithms can be applied to the embedding generated by Maximal_planar_embedding().

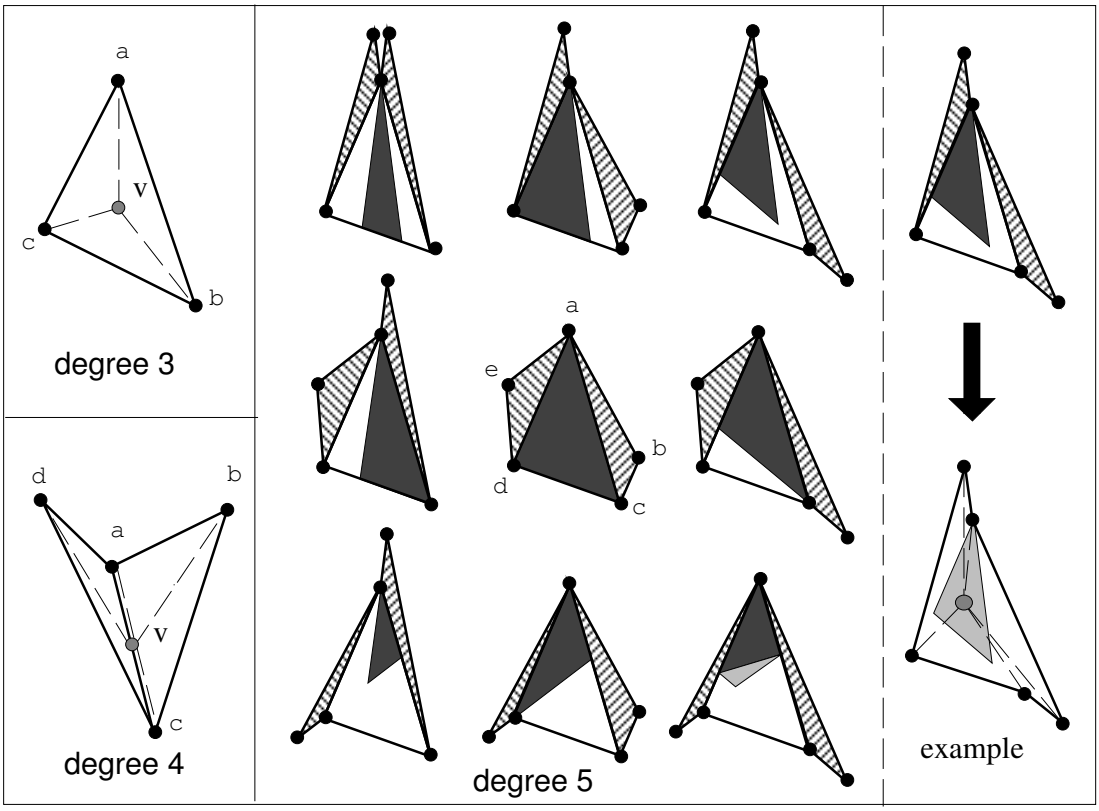
First, the new algorithm Maximal_planar_Straight_Line_Drawing() fixes an outer face of the embedding prior to execution of PHASE_I. This is maintained until PHASE_II. The following Lemma states that this may be done with only little change to the definition of *reducibility* of a vertex.

Lemma 7.1 The 3 vertices lying on the outer face of an embedding of a maximal planar graph $G=(V,E)$ can be determined in linear time and kept not reduced during PHASE_I of the algorithm.

Proof. Find any reducible vertex o_1 of G by examining all vertices. If $degree(o_1)=3$ take any two of its neighbors as vertices o_2 and o_3. Otherwise let o_2 be a vertex in $N(o_1)$ with exactly two neighbors in $N(o_1)$ (2.7). Denote by o_3 any of the two vertices in $N(o_1)\cap N(o_2)$. In both cases the three vertices o_1,o_2, and o_3 lie on a common face w.r.t. any embedding of G. We can guarantee that none of o_1,o_2, and o_3 is reduced during PHASE_I by simply modifying the definition of *reducibility*. We define that o_1,o_2, and o_3 are not reducible in the new sense. We can always find a different fourth reducible vertex (2.6) which is reducible in the new sense. □

The *middle of a segment* between (a_x,a_y) and (b_x,b_y) is given by the point $(\frac{a_x+b_x}{2},\frac{a_y+b_y}{2})$. The point $(\frac{a_x+b_x+c_x}{3},\frac{a_y+b_y+c_y}{3})$ is called the *center* of the triangle given by the points (a_x,a_y),(b_x,b_y), and (c_x,c_y). Obviously the center of a triangle lies always "inside" the triangle.

After completion of PHASE_II the vertices o_1,o_2, and o_3 get coordinates $(0,0)$,$(0,1)$, and $(\frac{1}{2},\frac{\sqrt{3}}{2})$. The only other present vertex at that moment gets the center of o_1,o_2, and o_3 as its coordinates.



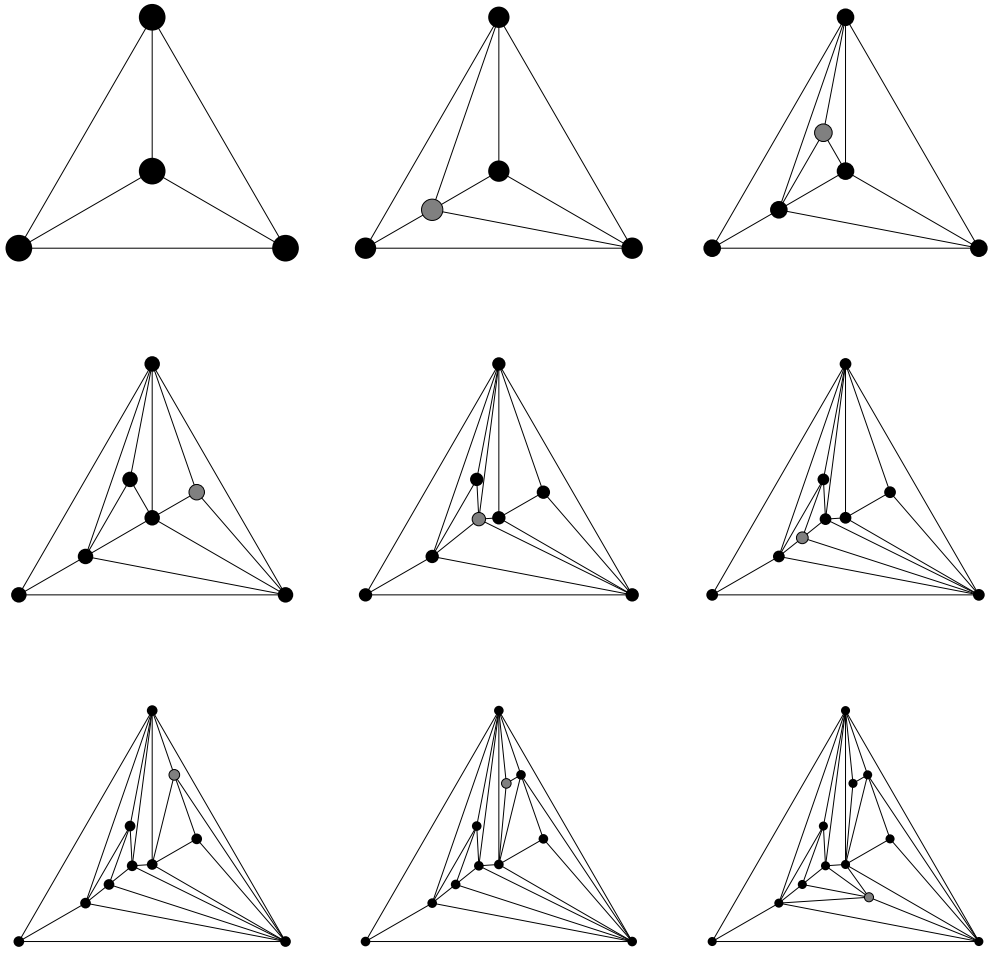
The only thing left to be specified is how the coordinates of each vertex inserted during an inverse reduction in PHASE_III are determined. A vertex v of $degree(v)=3$ gets the coordinates of the center of its neighbors. If the vertex v has $degree(v)=4$ and $\{a,c\}$ is the edge being

removed by the inverse reduction of v, then v gets the middle of the segment between a and c as its new coordinates. As shown in the figure on the previous page this enables the drawing of all edges from v to its neighbors without intersection.

The last case is the inverse reduction of a vertex v with $degree(v) = 5$. Here $\{a, c\}$ and $\{a, d\}$ are the edges having been removed by the inverse reduction of vertex v. The dark shaded area in the nine different possible drawings contains only permissible locations of v, i.e. v can be connected to all its neighbors by segments without intersection. For definiteness we place v at the center of the dark shaded triangle.

The algorithm Maximal_planar_Straight_Line_Drawing() is the algorithm Maximal_planar_embedding() (with or without the test from the previous section) with the slightly altered definition of reducibility (7.1) and the outlined determination of coordinates. The test for reducibility in this new sense requires constant time because only 3 vertices need to be checked additionally. The determination of the coordinates is done in constant time (even for $degree(v) = 5$ only 3 systems of linear equations with 2 variables and 2 equations need to be solved in order to determine the location of v). Thus Maximal_planar_Straight_Line_Drawing() has linear running time.

Below is an example showing the embeddings generated after PHASE_II (top left) and during PHASE_III of a maximal planar graph with $n = 12$ vertices. This results in a final Straight Line Drawing (bottom right). The reduction steps are for the degrees 4, 3, 3, 5, 4, 4, 3 and 5.



8 Generation of random (maximal) planar graphs

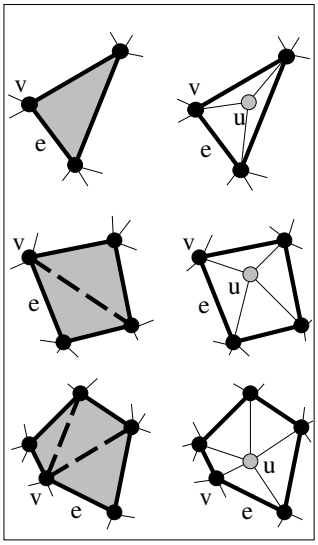
The inverse reductions performed in PHASE_III of Maximal_planar_embedding() form the base of a new algorithm for generating *random (maximal) planar graphs*. Unfortunately we cannot provide any information on the probability distribution of the generated (maximal) planar graphs.

The algorithm Random_maximal_planar_graph(n) is called with the number of vertices the generated maximal planar graph is desired to have. Every edge generated during the algorithm is put into the urn U. Since we need to know in constant time whether an edge e drawn from U at random is present in the actual graph we maintain an array *active* with $active[e] == 1$ if, and only if, e is present in the actual graph.

```
Random_maximal_planar_graph(integer n)
graph G; urn U(5n); edge e,f,g; vertex v,w; array active; integer i,type;
Initially let G be an embedding of the K_4; /* vertices 1,2,3,4, edges 1,2,3,4,5,6 */
forall i ∈ {1,...,5n} active[i] = 0;
forall_edges(e,G) { U_put(e); active[e] = 1; }
while (n > 4)
    {
    do { e = U_draw(); } while (active[e] == 0);
    U_put(e);
    choose v randomly as one of the vertices incident to e;
    if (G_degree[v] == 3) choose type randomly from {3,4};
    else                  choose type randomly from {3,4,5};
    if (type >= 4) { f = G_cyclic_incident_succ(e,v); delete f from G; active[f] = 0; }
    if (type == 5) { f = G_cyclic_incident_succ(e,v); delete f from G; active[f] = 0; }
    create a new vertex u in G;
    f = e; w = v;
    do
        {
        create a new edge g = {u,w} behind f in INC[w] and at the last position in INC[u];
        active[g]=1; U_put(g);
        w = G_opposite(w,f); f = G_cyclic_incident_pred(f,w);
        }
    while (f != e); /* f not equal to e */
    n = n - 1;
    }
```

Lemma 8.1 Random_maximal_planar_graph(n) requires $O(n)$ time and space to generate a random maximal planar graph on n vertices, which is an embedding.

Proof. Deleted edges are never inserted into G again. We charge the time for a non-active edge drawn from U until the moment of its deletion from G. Thus the amortized running time for choosing randomly an active edge at the beginning of the main-loop is constant. The edge deletions in case *type* >= 4 produce a face of length stored in *type*. The do-while-loop scans the *type* many edges of that face in counter-clockwise order starting with e. It requires constant running time as a whole since *type* <= 5. Therefore one run through the main-loop requires constant time and the main-loop is executed $n-4$ times. The initialization needs linear time and therefore *Random_maximal_planar_graph*(n) is of linear time complexity. The edges created inside the do-while-loop triangulate the face "above" e. They are inserted at the correct positions in the incidence lists of the vertices of that face. The order of edges in $INC[u]$ corresponds to a counter-clockwise traversal and thus the actual graph G is an embedding after each run of the main-loop and at the end of the algorithm. In each run of the main-loop at most 5 new edges are created such that $O(6+5(n-4)) = O(n)$ space for the arrays *active*, *source*, ... suffices. □



Given Random_maximal_planar_graph(n) it is easy to generate random planar embeddings on n vertices with m edges. Random_planar_graph(n,m) is called with the number of vertices and the number of edges the graph to generate should possess.

Random_planar_graph(integer n,integer m)

graph G; urn $U(3n-6)$; edge e;
if $(m > 3n-6)$ abort with too_many_edges;
G = Random_maximal_planar_graph(n);
forall_edges(e, G) { $U\_put(e)$; }
while ($G\_number\_of\_edges() > m$)
{
$e = U\_draw()$;
delete edge e from G;
}

It is easy to verify that Random_planar_graph(n,m) requires $O(n)$ time and space to generate a random planar graph on n vertices and m edges ($m \leq 3n-6$), which is an embedding.

References

[1] Auslander, L., Parter, S.V., 'On embedding graphs in the plane', J. Math. and Mech., **10**, 517-523 (1961).

[2] Battista, G.D., Eades, P., Tamassia, R., Tollis, I.G., 'Algorithms for Drawing Graphs: an Annotated Bibliography', /pub/gdbiblio.tex.Z from wilma.cs.brown.edu .

[3] Chrobak, M., Eppstein, D., 'Planar orientations with low out-degree and compaction of adjacency matrices', Theor. Comp. Sci., **86**, 243-266 (1991).

[4] Demoucron, G., Malgrange, Y., Pertuiset, R., 'Graphes planaires: reconnaissance et construction de représentations planaires topologiques', Rev. Française Rechereche Opérationelle, **8**, 33-47 (1964).

[5] Even, S., Graph Algorithms, Computer Science Press, 1979.

[6] Fáry, I., 'On straight line representation of planar graphs', Acta. Sci. Math. Szeged., 229-233 (1948).

[7] Fraysseix, H. de, Pach, J., Pollack, R., 'How to draw a planar graph on a grid', Combinatorica **10** (1), 41-51 (1990).

[8] Hopcroft, J., Tarjan, R.E., 'Efficient planarity testing', JACM, **21** (4), 549-568 (1974).

[9] Kuratowski, G., 'Sur le problème des courbes gauches en topologie', Fund. Math., **15**, 271-283 (1930).

[10] Lempel, A., Even, S., Cederbaum, I.,'An algorithm for planarity testing of graphs', Theory of Graphs, Int. Symp., Rome 1966, P. Rosenstiehl, ed., Gordon and Breach, NY, 215-232 (1967).

[11] Lueker, G.S., Booth, K.S., 'Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms', J. of Comp. and Sys. Sciences, **13**, 335-379 (1976).

[12] Read, R.C., 'A new method for drawing a planar graph given the cyclic order of the edges at each vertex', Congr. Numer. **56**, 31-44 (1987).

[13] Schnyder, W., 'Embedding Planar Graphs on the Grid', Proc. ACM-SIAM Symp. on Discrete Algorithms, 138-148 (1990).

[14] Wagner, K., 'Bemerkungen zum Vierfarbenproblem', Jahresber. Deutsche Math.-Verein. **46**, 26-32 (1936).

[15] Whitney, H., 'Planar graphs', Fund. Math., **21**, 73-84 (1933).