Planar Orientations with Low Out-Degree and Compaction of Adjacency Matrices

Marek Chrobak * David Eppstein †

October 4, 1996

Abstract

We consider the problem of orienting the edges of a planar graph in such a way that the out-degree of each vertex is minimized. If, for each vertex v, the out-degree is at most d, then we say that such an orientation is d-bounded. We prove the following results:

- Each planar graph has a 5-bounded acyclic orientation, which can be constructed in linear time.
- Each planar graph has a 3-bounded orientation, which can be constructed in linear time.
- A 6-bounded acyclic orientation, and a 3-bounded orientation, of each planar graph can each be constructed in parallel time $O(\log n \log^* n)$ on an EREW PRAM, using $O(n/\log n \log^* n)$ processors.

As an application of these results, we present a data structure such that each entry in the adjacency matrix of a planar graph can be looked up in constant time. The data structure uses linear storage, and can be constructed in linear time.

*Department of Mathematics and Computer Science, University of California, Riverside, CA 92521. On leave from Institute of Informatics, Warsaw University, PKIN VIIIp, 00-901 Warsaw, Poland.

†Department of Computer Science, Columbia University, New York, NY 10027. Work partially supported by NSF grants DCR-85-11713 and CCR-86-05353.

1 Introduction

There are two basic ways of representing a graph $G = (V, E)$ in a computer. The first way is to keep the list of neighbours $N(v)$ for each vertex $v \in V$. If $|V| = n$ and $|E| = m$, then this representation uses $O(n + m)$ memory, and is very useful in many graph algorithms, especially those which involve searching a graph.

The second way is the adjacency matrix: for every two vertices $u, v \in V$ we keep a boolean value $A[u, v]$ which tells us whether $(u, v) \in E$ or not. This representation uses as much as $O(n^2)$ memory, and this does not depend on the number of edges in G. The advantage of the adjacency matrix is that queries of the type:

$(*)$: Given $u, v \in V$, is $(u, v) \in E$?

can be answered in time $O(1)$, whereas this does not seem possible if we use adjacency lists.

For planar graphs, the waste of memory when the adjacency matrix is used is especially painful: of the n^2 entries in A only at most $6n$ of them are actually used. There is also a problem with initialization of this matrix—it seems at first glance that $O(n^2)$ time is necessary to initialize A. There is, however, a solution to this problem which takes only $O(n)$ time (see [1], Exercise 2.12).

In this note we show an easy way to represent a planar graph in such a way that the queries $(*)$ can be answered in time $O(1)$ and the whole data structure uses only $O(n)$ space.

An orientation ω of a graph is a function which replaces each edge $(u, v) \in E$ by an arc $u \to v$ or $v \to u$. By $\deg^+_\omega(v)$ we will denote the out-degree of v, under this orientation ω (for simplicity we will avoid the subscript ω). We say that ω is *d-bounded* if for each vertex $v \in V$ we have $\deg^+(v) \le d$.

The main results of the paper are:

- We prove that each planar graph has a 5-bounded acyclic orientation. It is easy to see that there are graphs which cannot have 4-bounded acyclic orientations, so this bound is optimal. We also present an algorithm for finding this orientation in linear time.
- We give an optimal NC parallel algorithm for finding 6-bounded acyclic orientations in planar graphs. The algorithm works in time $O(\log n \log^* n)$ on EREW PRAM and uses $O(n/\log n \log^* n)$ processors.
- We show that each planar graph has a 3-bounded orientation, and show that it can be found in linear time. Actually, we present two different linear time algorithms for this problem. The existence of a 3-bounded orientation can be also derived from the fact that planar graphs have arboricity at most 3, see [19, 12, 6].
- We give an optimal NC parallel algorithm for finding 3-bounded orientations in planar

graphs. This algorithm works in time $O(\log n \log^* n)$ on an EREW PRAM and uses $O(n/\log n \log^* n)$ processors.

Later, we also consider outerplanar graphs. We prove that each outerplanar graph has a 2-bounded acyclic orientation, and present the following algorithms:

- A linear-time sequential algorithm for finding a 2-bounded acyclic orientation.
- An optimal parallel EREW algorithm for finding a 2-bounded orientation which works in time $O(\log n \log^* n)$ on $O(n/\log n \log^* n)$ processors.
- A parallel CRCW algorithm for finding an acyclic 2-bounded orientation which works in time $O(\log n)$ on $O(n)$ processors.

We also show that some of those results for outerplanar graphs can be extended to series-parallel graphs.

Most of our algorithms do not use an embedding of the input graph. For sequential algorithms this leads to simpler algorithms. More importantly, the best known parallel algorithm for planar embedding takes time $O(\log^2 n)$ [14], so the use of an embedding would considerably slow down our parallel algorithms.

These results give immediately algorithms for constructing compacted adjacency matrices: given a d-bounded orientation ω of G, it is sufficient to store, for each v, only these neighbours x of v such that $\omega(v, x) = v \rightarrow x$.

We also show that this new way of representing planar graphs is very useful in some algorithms on graphs. The problems we concentrate on are the subgraph listing problems. We show how this data structure yields linear-time algorithms for listing triangles and 4-cliques in planar graphs. It has been known before that these two problems can be solved in linear time [20, 6]. However, using our compacted adjacency matrix, both problems become trivial.

Let us also point out the connection between our work and the recent paper of Kannan, Naor and Rudich [16]. They investigate the problem of labelling the vertices of a graph G in such a way, that given the labels of u and v it is possible to tell whether u and v are adjacent. Their solution for graphs, in our terminology, is to label u with the four-tuple (u, x, y, z), where x, y, z are neighbours of v, such that the edges (v, x), (v, y) and (v, z) are directed outwards from v, in some fixed 3-bounded orientation of G. They call it a 4-*labelling*. It is obvious that these labels indeed determine the adjacency relation. The algorithms from our paper can be applied also to yield a linear-time algorithm for finding such a 4-labelling of planar graphs.

2 Acyclic orientations

Theorem 1 *Each planar graph $G = (V, E)$ has a 5-bounded acyclic orientation; such an orientation can be constructed in linear time.*

Proof: A 5-bounded orientation of G can be computed by the following algorithm:

for $i \leftarrow 1$ **to** n **do begin**

choose $v \in V$ with $\deg(v) \leq 5$;

$f(v) \leftarrow i$;

remove v from G

end

By Euler's formula G has always a vertex of degree at most 5, so we can always find v in the **for** loop. The function f computed above determines the acyclic orientation in the following way: given $(u, v) \in E$, let $\omega(u, v) = u \rightarrow v$ if $f(u) < f(v)$, and $v \rightarrow u$ otherwise. At the moment we remove v, it has at most 5 neighbours, and only they will be assigned numbers greater than $f(v)$; therefore the resulting orientation is 5-bounded.

It remains to show that the algorithm above can be implemented in time $O(n)$. To do this, we use a queue $\mathcal{Q}$ in which we keep all vertices in G of degree at most 5. The vertex v can be found in time $O(1)$ by taking the first vertex from $\mathcal{Q}$. When we remove v we have to update $\mathcal{Q}$: look at all the neighbours of v, and if any of them has degree at most 5 after removing v and does not belong to $\mathcal{Q}$, then add it to $\mathcal{Q}$. This takes time $O(1)$. Therefore the whole computation takes time $O(n)$. $\Box$

Using Theorem 1 we can represent a planar graph in the following way. Compute the function f as in the proof of Theorem 1, and let $N^+(v)$ denote the set of descendants of v under the orientation determined by f. Represent G by an $n \times 5$ array B such that $B[v, 1 \ldots 5]$ contains the list of the vertices in $N^+(v)$. This representation can be computed in linear time, as in the proof of Theorem 1.

In order to answer query $(*)$ we check first which of $f(u)$, $f(v)$ is smaller. Suppose that $f(u) < f(v)$. Then we check whether $v \in N^+(u)$ by scanning the entries $B[u, i]$ for i from 1 to 5. Thus in at most 5 steps we can answer our query. Alternatively, we can sort the entries $B[u, i]$ for each u, and perform binary search. This solution requires $5n$ memory locations, so only $2n$ of them are not used. It may be actually more space- and time-efficient than lists, because we do not need pointers.

It is easy to see that there are planar graphs which do not have acyclic 4-bounded orientations. Take, for example, any planar graph with minimum degree 5. Then for each acyclic orientation there is a vertex v which has in-degree 0, so $\deg^+(v) \geq 5$.

Theorem 2 *There is a parallel EREW PRAM algorithm for computing a 6-bounded acyclic orientation of planar graphs, which runs in time $O(\log n \log^* n)$ with $O(n/\log n \log^* n)$ processors.*

Proof: The algorithm is very similar to parallel 5-coloring algorithms for planar graphs (see, for example [11]), so we only sketch it here. The computation is divided into $O(\log n)$ phases. In phase i we find a set R of vertices of degree at most 6. Now we construct a graph $H = (R, F)$, where $(u, v) \in F$ if either $(u, v) \in E$ or u and v have a common neighbour x such that the edges (u, x) and (v, x) are consecutive in the adjacency list of x. In the next step we compute a maximal independent set I in H. Since the maximum degree in H is $O(1)$, $|I| = \Omega(n)$. Finally, we remove all vertices in I, and each $v \in I$ is assigned the number $f(v) = i$. The orientation is determined from f as in the sequential case.

The time for each phase is dominated by the computation of the maximal independent set I. This can be done in either of two similar ways, one taking time $O(\log n)$ with $O(n/\log n)$ processors, and the other taking time $O(\log^* n)$ with $O(n)$ processors (see [10]). We use the first method for the first $O(\log^* n)$ phases, after which we use the second method. Because at each step the number of operations to be performed decreases as the size of the graph decreases, the total number of operations is $O(n)$. By a theorem of Brent [3], if these operations can be scheduled among p processors, the total parallel time will be $O(n/p + \log n \log^* n)$.

We perform this scheduling by keeping the names of remaining vertices and edges in an array, and periodically compacting the array to remove positions no longer holding an edge or vertex. The compaction is performed with a prefix computation [17], each iteration of which takes time $O(\log n)$ and uses $O(n)$ operations. Again the total number of operations is $O(n)$. If we perform these compactions at appropriately chosen intervals, the compactions will also take a total time of $O(\log n \log^* n)$, and we can use Brent's theorem to perform the algorithm with only $O(n/\log n \log^* n)$ processors. □

An interesting problem, whether finding a 5-bounded acyclic orientation is in NC, remains open. This question is related to a problem if a $p(G)$-coloring can be computed fast in parallel ($p(G)$ is the maximum over all subgraphs G' of G, of the minimum degree of G'). This other problem was shown recently to be P-complete (see [24]).

3 3-bounded orientations

In this section we show that planar graphs have a 3-bounded orientation. This result can be derived independently from a general fact about arboricity of planar graphs. The *arboricity* of a graph G, denoted by $a(G)$ is the smallest number of edge-disjoint spanning forests, whose union is G. Nash-Williams [19] proved a general fact that

$$a(G) = \max_H \frac{q}{p-1},$$

where the maximum is over all non-trivial subgraphs of G, p is the number of vertices and q is the number of edges in G. From this formula it is easy to derive that if G is planar then $a(G) \leq 3$. Since we can orient every forest such that the out-degree of every vertex is at most 1, this shows that every planar graph has a 3-bounded orientation. However, the proof of [19] does not seem to yield a linear time algorithm. We present below another proof of this fact, and two linear-time algorithms for this problem.

Theorem 3 *Each planar graph G has a 3-bounded orientation, and it can be found in linear time.*

Proof: Assume we are given an embedding of G in the plane, and let one face of the embedding be specially marked (we call this the *unbounded face*). Call each vertex v of G either *exterior* if v is on the unbounded face, or *interior* otherwise. We prove a stronger version of the theorem:

Claim 1: Every planar graph G has a 3-bounded orientation such that for each exterior vertex v, $\deg^+(v) \leq 2$.

Claim 1 is proved by induction on n. There must be some exterior vertex v adjacent to at most 2 other exterior vertices. (Actually, there must be at least two such vertices. This is obvious when one realizes that the subgraph of G induced by the external vertices is outerplanar). Let the graph G' be the subgraph of G formed by removing v, and having as its exterior vertices the remaining exterior vertices of G together with the neighbors of v. Then by induction G' has a 3-bounded orientation such that, for each exterior vertex w of G', $\deg^+(w) \leq 2$. Now we may orient the edges (v, x) between v and each of its neighbors. If x is exterior in G, we orient (v, x) from v to x; otherwise we orient it from x to v. It can be seen that the resulting orientation satisfies the properties of the theorem.

We explain now how to implement the method from the proof in linear time. The algorithm consists of two phases. In the first phase we remove vertices from the graph. In the second phase we return these vertices in reverse order, and orient the edges adjacent to them.

In the first phase we keep for each vertex an information whether it is or is not exterior, and how many exterior neighbors it has. We also have a queue $\mathcal{Q}$ of the exterior vertices which have at most 2 exterior neighbours. A vertex v to be removed is chosen in time $O(1)$ by taking the first vertex from $\mathcal{Q}$. When we remove v, we have to update the information stored in other vertices. Let $e(v)$ be a boolean variable which tells whether v is exterior or not, and let also $\deg^e(v)$ be the number of exterior neighbours of v. The steps taken to remove a vertex v are as follows.

$G \leftarrow G \setminus \{v\}$;

for each $w \in N(v)$ **do begin**

  $\deg^e(w) \leftarrow \deg^e(w) - 1$;

  if $e(w) = \mathit{false}$ **do begin**

    $e(w) \leftarrow \mathit{true}$;

    for each $x \in N(w)$ **do begin**

      $\deg^e(x) \leftarrow \deg^e(w) + 1$;

      if $\deg^e(x) \geq 3$ **and** $x \in \mathcal{Q}$ **then**

$\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{x\}$

end;

end;

if $\deg^e(w) \leq 2$ **and** $w \notin \mathcal{Q}$

then $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{w\}$

end

Let us analyze now the complexity of this algorithm. Charge the time for each iteration of the inner loop to the edge (w, x), and charge the remaining time in each iteration of the outer loop to the edge (v, w). Then each edge (s, t) in the graph is charged at most three times: once when each of $e(s)$ and $e(t)$ become true, and once when one of s or t is removed from the graph. The total number of edges is at most $3n$, so at most $9n$ charges are made, and the total time of the algorithm is $O(n)$.

The second phase is very easy to implement in time $O(n)$ by following the method from the proof. □

Note that in fact the algorithm above can be used to give a linear time decomposition of a planar graph into three forests, giving another proof that the arboricity of a planar graph is at most 3.

Note also that we use an embedding of G when we construct our 3-bounded orientation. The question arises whether it is possible to find such an orientation without using an embedding. There are two reasons for considering this question. First, it is not clear whether the existence of a linear-time algorithm for constructing a 3-bounded orientation is a *topological* property of planar graphs, or whether it follows simply from their low density. Observe that the proof of Theorem 3 does not work for toroidal graphs, because if we take a face of a toroidal graph then each vertex on this face may have three neighbours on this face. But toroidal graphs have, asymptotically, the same density as planar graphs. Second, from the point of view of the application to compressing adjacency matrices, it would be optimal to use a 3-bounded orientation, because then we would need only $3n$ entries in the adjacency matrix. It seems to us, however, that the need to find an embedding before actually computing the representation, would limit possible applications of our method.

Therefore, we present now also another proof of Theorem 3, and an algorithm which does not need the embedding. We need some more definitions. By n_d we will denote the number of vertices of degree d. A vertex $v \in V$ will be called *small* if $\deg(v) \leq 18$, otherwise it will be called *large*. A vertex $v \in V$ is *reducible* if it satisifes one of the following conditions:

(r1) $\deg(v) \leq 3$.

(r2) $\deg(v) = 4$ and v has at least 2 small neighbours.

(r3) $\deg(v) = 5$ and v has at least 4 small neighbours.

Before describing the algorithm, we prove the lemma below.

Lemma 1 *Each planar graph G has a reducible vertex.*

Proof: The proof is by contradiction. Suppose that G does not have reducible vertices. Therefore, $n_d = 0$ for $d = 0, \ldots, 3$. From Euler's formula we have $m < 3n$, which after substituting $n = \sum_{d \geq 4} n_d$ and $m = \frac{1}{2} \sum_{d \geq 4} d n_d$, and some simple rearranging yields

$$2n_4 + n_5 > \sum_{d \geq 7} (d-6) n_d. \tag{1}$$

From the assumption that G does not have reducible vertices we obtain that each vertex of degree 4 has at least 3 large neighbours, and each vertex of degree 5 has at least 2 large neighbours. By counting the edges between vertices of degree 4, 5 and large vertices we obtain that

$$3n_4 + 2n_5 \leq \sum_{d \geq 18} d n_d. \tag{2}$$

Then, from (1) and (2) we have

$$\begin{aligned} \frac{2}{3} \sum_{d \geq 18} d n_d &\geq \frac{2}{3}(3n_4 + 2n_5) \\ &\geq 2n_4 + n_5 \\ &> \sum_{d \geq 7} (d-6) n_d. \end{aligned} \tag{3}$$

This gives the contradiction by rearranging the inequality above, as follows:

$$\sum_{d \geq 7} (3d - 18) n_d + \sum_{d \geq 18} (d - 18) n_d \quad < \quad 0.$$

This completes the proof. □

The idea of the algorithm is as follows: we choose a reducible vertex v, and perform an appropriate reduction. A reduction consists of removing v and possibly adding some edges between its neighbours. We orient the obtained graph, and then we extend the orientation to the edges incident to v. The extension method will depend on the orientation of the edges added during the reduction.

Let us note first the following, easy lemma.

Lemma 2 *Let $v \in V$. Then there are at least two vertices $x, y \in N(v)$ such that $|N(x) \cap N(v)| \leq 2$ and $|N(y) \cap N(v)| \leq 2$.*

Proof: This follows from the observation that the subgraph of G induced by the vertices in $N(v)$ is outerplanar. □

Now we describe the reduction and extension methods. We have three cases, depending on the degree of v.

1. $\deg(v) \leq 3$.

   Reduction: Remove v from G.

   Extension: Add v to G. For each edge (v, x), set $\omega(v, x) := v \to x$.

2. $\deg(v) = 4$.

   Reduction: Find a vertex $t_v \in N(v)$ such that $|N(t_v) \cap N(v)| \leq 2$. Remove v, and add an edge (t_v, x) for some $x \in N(v) \setminus N(t_v)$.

   Extension: Add v to G. Suppose that $\omega(t_v, x) = t_v \to x$ (the other case is symmetric). Set $\omega(t_v, v) := t_v \to v$. For all $s \in N(v) \setminus \{t_v\}$, set $\omega(s, v) := v \to s$.

3. $\deg(v) = 5$.

   Reduction: Find a vertex t_v such that $N(t_v) \cap N(v)| \leq 2$. Remove v, and add edges (t_v, x), for all $x \in N(v) \setminus N(t_v)$.

   Extension: Add v to G. Let $T = N(v) \setminus N(t_v)$. Therefore T contains t_v and all vertices x joined to t_v by added edges. We have some cases now.

   (a) Two of the vertices in T have an added edge directed outwards from it. Let x and y be these vertices. Then, set $\omega(x, v) := x \to v$, $\omega(y, v) := y \to v$, and for all $s \in N(v) \setminus \{x, y\}$ do $\omega(s, v) := v \to s$.

   (b) (a) is false. This means, that $\omega(t_v, x) = t_v \to x$, for all $x \in T \setminus \{t_v\}$. That is, all added edges are directed outwards from t_v.
   Since ω is 3-bounded, we must have that $|T \setminus \{t_v\}| \leq 3$, so $|N(t_v) \cap N(v)| =$ 1 or 2. In both cases there is a non-added edge (t_v, z) such that $z \in N(t_v) \cap N(v)$ and $\omega(t_v, z) = z \to t_v$. Therefore we can proceed as follows: $\omega(z, t_v) := t_v \to z$ (that is, we reorient (z, t_v)), $\omega(t_v, v) := t_v \to v$, $\omega(z, v) := z \to v$ and for all $x \in N(v) \setminus \{t_v, z\}$ do $\omega(x, v) := v \to x$.

An example of a reduction and an extension is shown in Fig.1. We note now the following lemma.

Lemma 3 *Let G' be a graph obtained from G by applying one of the reductions above. Then,*

(i) If G is planar then so is G'.

(ii) If ω is 3-bounded on G', then it is also 3-bounded on G.

Proof: The first part of the lemma is obvious, since the edges we add can be drawn through the region we obtain after removing v, and they do not cross because they all have a common endpoint t_v.

So let us concentrate now on *(ii)*. We have three cases, corresponding to the three reductions above.

The case when $\deg(v) \leq 3$ is obvious. The case when $\deg(v) = 4$ is also easy. No matter what the orientation of (t_v, x) is, we will have at least one edge oriented into v, so $\deg^+(v) \leq 3$. And for each neighbour of v, the number of edges oriented outwards does not increase.

So let us consider now the case $\deg(v) = 5$. Suppose first that the first subcase *(a)* in the extension procedure holds. Then we will have at least two edges directed into v, so $\deg^+(v) \leq 3$. If *(b)* holds, then we will also have at least two edges directed into v, namely (t_v, v) and (z, v). The value of $\deg^+(z)$ does not change, because we reorient (t_v, z). Also, the value of $\deg^+(t_v)$ does not increase. This is because we remove at least one outwards oriented edge, so even though we reorient (t_v, z), the total change of $\deg^+(t_v)$ cannot be positive. □

Note that the reductions above can be applied to any vertex of degree at most 5. This gives an easy $O(n^2)$ algorithm for finding a 3-bounded orientation, because each reduction and extension can be implemented in time $O(n)$. The difficulty is in finding the vertex t_v, because we need to choose t_v such that it has at most 2 common neighbours with v. It does not seem possible to do it faster than in time $O(n)$, unless one uses an adjacency matrix, but this leads to a vicious circle, considering the applications we have in mind.

A way around this is to put some restrictions on the vertices we reduce, as shown in the lemma below.

Lemma 4 *If v is a reducible vertex, then the reduction of v, and the extension of ω can be done in time $O(1)$.*

Proof: Consider first a reducible vertex v of degree 4. The vertex t_v can be found as follows: let $x, y \in N(v)$ be small. If one of them has at most two common neighbours with v, then let t_v be this vertex. Otherwise, take t_v to be any vertex in $N(v) \setminus \{x, y\}$. By planarity, if z is the fourth neighbour of v, t_v and z cannot be adjacent.

If $\deg(v) = 5$, look at the four small neighbours of v. By Lemma 2, at least one of them must have at most two common neighbours with v. □

Now we can describe the algorithm. We maintain a queue $\mathcal{Q}$ which contains all reducible vertices in G. The algorithm consists of two phases. In the first phase we perform the reductions on G, until G is empty. We have also a stack $\mathcal{S}$, on which we store the information about the applied reductions, sufficient to undo them in the second phase. Clearly, $O(1)$ space per reduction suffices. In the second phase we take the reductions from $\mathcal{S}$, so that they will be considered in reverse order, undo them and extend gradually the current orientation.

The vertex to be reduced can be found in time $O(1)$ by taking the first vertex from $\mathcal{Q}$. The reductions and extensions cost time $O(1)$ each. Therefore, to prove that the algorithm can be implemented in time $O(n)$ we need to show that the total time of updating $\mathcal{Q}$ is also $O(n)$.

Let *Update*(x) be the following procedure:

if x is reducible **then**

  if $x \notin \mathcal{Q}$ **then** $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x\}$

else

  if $x \in \mathcal{Q}$ **then** $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{x\}$

While executing the reduction at v we update the information about $\mathcal{Q}$, by looking at all vertices which have to be put into, or removed from $\mathcal{Q}$. Note that the only vertices whose degree changes during the reduction are the neighbours of v. Several things can happen:

- A vertex $x \in N(v)$ had $\deg(x) > 5$ but now $\deg(x) \leq 5$. Clearly, x may be reducible now.
- A vertex $x \in N(v)$ had $\deg(x) \leq 5$, but now $\deg(x) > 5$. Then, we may have to remove x from $\mathcal{Q}$.
- A vertex $x \in N(v)$ was large, but now it is small. If x has a neighbour z of degree at most five, then z may have become reducible, and we have to put it in $\mathcal{Q}$.
- A vertex $x \in N(v)$ was small, but now it is large. Then we may have to remove one of its neighbours from $\mathcal{Q}$.

Therefore we need to look at small neighbours of v, and their neighbours of degree at most five. We do the following:

for each $x \in N(v)$ **do**

  if x is small **or** x was small before reducing v **then**

    begin

      Update(x);

      for each $y \in N(x)$ **with** $\deg(y) \leq 5$ **do** *Update*(y)

    end

By the consideration above, this will ensure that $\mathcal{Q}$ contains exactly those vertices which are reducible. It is easy to see that the procedure above takes only $O(1)$ time. So, finally, we obtain

Theorem 4 *There is an $O(n)$-time algorithm which finds a 3-bounded orientation in a planar graph, and does not use an embedding of this graph.*

4 A parallel algorithm for 3-bounded orientations

Before we present the algorithm, we need to prove a combinatorial result about the distribution of degrees in planar graphs. The idea of the algorithm is similar to that of the second sequential algorithm from the preceding section. This time, however, we must reduce a linear number of vertices simultaneously. Therefore we need a more relaxed notion of reducibility, and an appropriate stronger version of Lemma 1.

We will redefine now slightly the notions of small and reducible vertices. A vertex $v \in V$ will be called *small* if $\deg(v) \leq 25$. Also, let us call a vertex *reducible* if it satisfies one of the conditions *(r1)*, *(r2)*, *(r3)*, or the additional condition below:

(r4) $\deg(v) \leq 6$ and all neighbours of v are small.

We will use the following theorem.

Lemma 5 *If R is the set of reducible vertices in G, then $|R| \geq n/21$.*

Proof: The proof is similar to the one for Lemma 1. It is sufficient to consider only connected graphs, so $n_0 = 0$. Let us denote by $\bar{n}_d$ the number of reducible vertices of degree d, and by $\tilde{n}_d$ the number of non-reducible vertices of degree d. Clearly, $\bar{n}_d = n_d$ for $d = 1, 2, 3$, and $n_d = \bar{n}_d + \tilde{n}_d$ for $d = 4, 5, 6$.

Using Euler's formula, and some simple rearrangements similar to those in the proof of Lemma 1, we obtain

$$\sum_{d=1}^{5}(6-d)n_d \geq \sum_{d\geq 7}(d-6)n_d. \tag{4}$$

By counting the edges between non-reducible vertices of degree 4,5,6, and large vertices, we obtain

$$3\tilde{n}_4 + 2\tilde{n}_5 + \tilde{n}_6 \leq \sum_{d\geq 25} dn_d. \tag{5}$$

Using (1) and (2), we proceed as follows:

$$\begin{aligned}
21|R| &\geq 21n_1 + 17n_2 + 13n_3 + 9\bar{n}_4 + 5\bar{n}_5 + 3\bar{n}_6 \\
&= 21n_1 + 17n_2 + 13n_3 + 9n_4 + 5n_5 + 3n_6 - (9\tilde{n}_4 + 5\tilde{n}_5 + 3\tilde{n}_6) \\
&\geq \sum_{d=1}^{6} n_d + 4\sum_{d=1}^{5}(6-d)n_d - 3(3\tilde{n}_4 + 2\tilde{n}_5 + \tilde{n}_6) \\
&\geq \sum_{d=1}^{6} n_d + 4\sum_{d\geq 7}(d-6)n_d - 3\sum_{d>24} dn_d \\
&= \sum_{d=1}^{6} n_d + 4\sum_{d=7}^{24}(d-6)n_d + \sum_{d>24}(d-24)n_d \\
&\geq \sum_{d\geq 1} n_d = n.
\end{aligned}$$

The above inequality directly implies the lemma. $\Box$

The general idea of the algorithm is to perform some reductions on the graph such that it will be eventually become empty. Reductions consist of removing some vertices, and possibly adding some edges between the neighbours of removed vertices. In the second phase, the removed vertices are returned to the graph in reverse order, and the current orientation is extended to new edges. The method of extending the orientation depends on the applied reduction. The reduction and extension methods for vertices of degree at most 5 are the same as in the second algorithm from the preceding section. We show only how to reduce vertices of degree 6.

Reduction: Remove v from the graph. Find a small vertex $t_v \in N(v)$ such that $|N(v) \cap N(t_v)| \leq 2$. Add all edges (t_v, x) for $x \in N(v) \setminus N(t_v)$.

Extension: There are four cases. In each case we find three edges (v, x) to direct as $x \to v$; we direct the remaining three edges (v, y) as $v \to y$.

(a) Three of the edges (t_v, x) added in the reduction are directed $x \to t_v$. Then for each such x we direct the edge (v, x) as $x \to v$.

(b) Two added edges (t_v, x) are directed $x \to t_v$. For both such x we direct the edge (v, x) as $x \to v$, and we direct the edge (v, t_v) as $t_v \to v$.

(c) One added edge (t_v, x) is directed $x \to t_v$. We direct the edge (x, v) as $x \to v$, and the edge (v, t_v) as $t_v \to v$. In addition, since t_v is adjacent in the reduced graph to all 5 neighbors of v, it can only have 3 out-edges among these adjacencies, so one edge (t_v, z) that was not added in the reduction must also be directed as $z \to t_v$. We reverse the orientation of this edge and direct the edge (v, z) as $z \to v$.

(d) All added edges (t_v, x) are directed $t_v \to x$. Then t_v must be adjacent in the unreduced graph to exactly two neighbours z of v, and both edges (t_v, z) must be directed (z, t_v). We reverse the orientation of both edges and direct both edges (v, z) as $z \to v$. In addition we direct the edge (t_v, v) as $t_v \to v$.

We note first the following lemma.

Lemma 6 *Let G' be a graph obtained from G by applying the reduction above. Then,*

(a) G' is planar,

(b) If ω is a 3-bounded orientation of G', then after applying the extension, ω is a 3-bounded orientation of G.

Proof: Part (a) follows as in the proof of Lemma 3.

For part (b), we need only consider the new extensions for degree-six vertices. First note that each neighbour of v other than t_v has its out-degree unchanged by the extension. Also, v itself is given in-degree 3, and therefore out-degree 3. Finally, each edge for which we change the orientation to be away from t_v is balanced by an edge directed from t_v in the reduced graph which no longer exists in the unreduced graph. Therefore the out-degree at t_v also remains at most three, and the lemma follows. $\Box$

Before presenting the whole algorithm we observe the following.

Lemma 7 *If v is a reducible vertex, then the vertex t_v can be found in time $O(1)$.*

In the algorithm we will attempt to reduce many reducible vertices. The problem is that some of reductions might conflict one with another. For example, we cannot, in general, apply a reduction to adjacent vertices. A more subtle problem arises when we execute reductions of vertices u and v such that $t_u = t_v$. Then it may happen that u and v have another common neighbour x, and in the extension procedure, u will try to reorient (t_v, x), while the extension at v does not need it. Another possibility of a conflict arises when we remove vertices which have a common neighbour, not necessarily small. Say, u and v are reducible and $x \in N(u) \cap N(v)$. When we remove u and v we need to update the adjacency list at x. More specifically, we need to remove the entries corresponding to u and v. But this obviously causes a problem when these entries are consecutive. Finally, two different reductions might attempt to add the same edge to the graph. Therefore we need to choose the set of vertices to be reduced very carefully.

By a *conflict graph* we will mean a graph $\mathcal{H} = (R, F)$, where $(u, v) \in F$ if one of the following conditions holds:

(c1) $(u, v) \in E$,

(c2) there is a small vertex $x \in N(u) \cap N(v)$,

(c3) there is a vertex $x \in N(u) \cap N(v)$ such that the entries in the adjacency list of x corresponding to edges (x, u) and (x, v) are consecutive.

Intuitively, the edges of $\mathcal{H}$ correspond to possible conflicts in the reductions in G. So to avoid conflicts, we must execute only reductions belonging to an independent set in $\mathcal{H}$. We now show that, if we do this, the resulting algorithm will be correct.

Lemma 8 *Let I be an independent set in $\mathcal{H}$. If we execute the reductions from I in parallel, they remain correct and no memory conflict occurs.*

Proof: Correctness could only be violated if two reductions or extensions attempted to perform an operation on the same edge. All edges involved in extensions and reductions have both endpoints either v itself or a neighbour of v, so if an edge has a small endpoint then *(c1)* or *(c2)* will prevent any conflict in this case. The only remaining possibility is that two degree-four vertices attempt to add the same edge, in each case between two large vertices. But this would mean that the two vertices are non-adjacent, each has two adjacent (small) neighbors, and both are adjacent to the same pair of (large) vertices. But this is not possible in a planar graph.

Finally, condition *(c3)* ensures that no memory conflict can occur in updating the adjacency lists of each vertex. $\Box$

The following lemma ensures that restricting our reductions to an independent subset of $\mathcal{H}$ still allows us to perform many reductions at once.

Lemma 9 *Let I be a maximal independent set in $\mathcal{H}$. Then $|I| = \Theta(n)$.*

Proof: Since $|R| = \Theta(n)$, it is sufficient to show that $|I| = \Theta(|R|)$. This fact follows easily from the observation that the maximum degree in $\mathcal{H}$ is at most $6(1+24) = 150$. $\square$

Now we are ready to present the algorithm.

Reduction phase:

for $k \leftarrow 1$ **to** $c \log n$ **do begin**

identify the set R of reducible vertices;

construct the conflict graph $\mathcal{H} = (R, F)$;

find a MIS I in $\mathcal{H}$;

for each $v \in I$ **do parallel begin**

execute the reduction at v;

$\mathrm{nr}(v) \leftarrow k$

end;

end;

Extension phase:

for $k \leftarrow c \log n$ **downto** 1 **do begin**

$I \leftarrow \{v \mid \mathrm{nr}(v) = k\}$;

for each $v \in I$ **do parallel**

extend the orientation ω at v

end

Theorem 5 *There is a parallel EREW PRAM algorithm, which computes a 3-bounded orientation of a planar graph G, and it runs in time $O(\log n \log^* n)$ with $O(n / \log n \log^* n)$ processors.*

Proof: By Lemma 5, if c is a large enough constant, the graph will be exhausted after $c \log n$ steps. As in the algorithm for acyclic 6-orientation, the computation of maximal independent sets can be performed in time $O(\log^* n)$. So it is sufficient to show that the remaining steps in each iteration can be performed in time $O(1)$.

During the algorithm we have processors assigned to each vertex, and to each entry of adjacency lists in the representation of G. Consider one iteration of the reduction phase. We need $O(1)$ time for finding R. The construction of $\mathcal{H}$ can be also done in time $O(1)$ by the processors assigned to the reducible vertices. This can be achieved by coordinating their scanning the adjacency lists of their small neighbours. After computing the independent set I, each vertex $v \in I$ can execute safely its reduction independently of the others, in constant time.

A similar analysis shows that each iteration in the second phase costs time $O(1)$. □

5 Outerplanar graphs

In this section we consider the outerplanar graphs. An *outerplanar* graph is a planar graph with the additional requirement that it has an embedding such that all vertices are on the same face. We prove first the following.

Theorem 6 *Each outerplanar graph has a 2-bounded acyclic orientation, which can be found in linear time.*

Proof: The proof is very similar to the proof of Theorem 1, so we only sketch it here. We use the fact that each outerplanar graph has a vertex of degree at most two (actually, it must have at least two such vertices). Let v be such a vertex. Remove v from G, find a 2-bounded acyclic orientation of the resulting graph, and add v back to G. Orient the edges incident to v outwards. It is obvious that this gives a 2-bounded acyclic orientation.

It is easy to implement this algorithm in time $O(n)$, by maintaining a queue containing vertices of degree 2, and updating it each time a vertex is removed. □

Now we will present a parallel algorithm for finding a 2-bounded orientation, not necessarily acyclic. The algorithm is very similar to the 3-orientation algorithm for planar graphs.

We redefine first the notion of reducibility. A vertex $v \in V$ is called now *small* if $\deg(v) \leq 7$, otherwise it is called *large*. A vertex v is called *reducible* if one of the following conditions holds:

(o1) $\deg(v) \leq 2$,

(o2) $\deg(v) = 3$ and v has a small neighbour,

(o3) $\deg(v) = 4$ and v has a small neighbour.

We will use the following lemma.

Lemma 10 *Let R be the set of reducible vertices in an outerplanar graph G. Then $|R| \geq \frac{n}{10}$.*

Proof: Let $\bar{n}_d$ and $\tilde{n}_d$ denote, respectively, the number of reducible and non-reducible vertices of degree d. It is well-known that for an outerplanar graph G we have $m \leq 2n$. This implies, after substituting $n = \sum_{d \geq 1} n_d$ and $m = \frac{1}{2} \sum_{d \geq 1} d n_d$, that

$$3n_1 + 2n_2 + n_3 \geq \sum_{d \geq 5} (d-4) n_d. \tag{6}$$

Consider a bipartite graph induced by edges between non-reducible vertices of degree 3 or 4, and large vertices. By counting the edges in this graph we obtain

$$\begin{aligned} 3\tilde{n}_3 + 4\tilde{n}_4 &\leq 2(\tilde{n}_3 + \tilde{n}_4 + \sum_{d \geq 7} n_d), \\ \tilde{n}_3 + 2\tilde{n}_4 &\leq 2 \sum_{d \geq 7} n_d. \end{aligned} \tag{7}$$

Using (6) and (7), we proceed as follows:

$$\begin{aligned} 10|R| &\geq 10n_1 + 7n_2 + 4\bar{n}_3 + 8\bar{n}_4 \\ &\geq \sum_{d=1}^{4} n_d + 3(3n_1 + 2n_2 + n_3) - 4(\tilde{n}_3 + 2\tilde{n}_4) \\ &\geq \sum_{d=1}^{4} n_d + 3 \sum_{d \geq 5} (d-4) n_d - 8 \sum_{d > 7} n_d \\ &= \sum_{d=1}^{4} n_d + 3 \sum_{d=5}^{7} (d-4) n_d + \sum_{d > 7} (3d - 20) n_d \\ &\geq \sum_{d \geq 1} n_d = n. \end{aligned}$$

This completes the proof. □

The algorithm is almost identical to the one from the preceding section, so we only sketch it here. As before, we reduce only reducible vertices. We define a conflict graph $\mathcal{H}$, find a maximal independent set I in $\mathcal{H}$, and execute only the reduction for vertices in I. It is not hard to see that the lemmas corresponding to Lemmas 6, 7, and 9 are also true. This yields the following theorem.

Theorem 7 *A 2-bounded orientation in an outerplanar graph G can be found in time $O(\log n \log^* n)$ on an EREW PRAM with $O(n/\log n \log^* n)$ processors.*

Let us consider now acyclic orientations. It is easy to see that each outerplanar graph has a linear number of vertices of degree at most 4. Following the idea of the algorithm for the 6-bounded acyclic orientations in planar graphs, we obtain the following result.

Theorem 8 *An acyclic 4-bounded orientation of an outerplanar graph can be found in time $O(\log n \log^* n)$ on an EREW PRAM with $O(n/\log n \log^* n)$ processors.*

Now we will show that we can construct even an acyclic 2-bounded orientation in an outerplanar graph. This algorithm is very different from the other algorithms presented in this paper. Unlike the others, it is not based on reduction techniques.

We assume the graph to be already embedded in the plane. This costs time $O(\log n)$ with $O(n)$ processors, if we use Diks' algorithm from [7]. The first phase is to reduce the problem to orienting 2-connected components of G, as follows:

find a tree $\mathcal{T}$ of 2-connected components of G;

for each 2-connected component C **do begin**

let z_C denote the vertex which

attaches C to its father in $\mathcal{T}$;

$Orient(C, z_C)$

end

The procedure $Orient(C, z)$ finds a 2-bounded orientation ω in C such that $\deg^+(z) = 0$. To find 2-connected components, and construct $\mathcal{T}$ we can use the Tarjan-Vishkin algorithm from [22], which works in time $O(\log n)$ and uses $O(n)$ processors. It is clear that, after $Orient(C, z_C)$ is completed for each C, then the obtained orientation will be 2-bounded and acyclic.

So it is sufficient to describe the procedure $Orient(C, z)$. Before we do this, let us introduce a notion of a dual. A *dual* of an outerplanar graph G is a graph $D(G) = (U, F)$, whose vertices are the regions of G, except the external one, and for two such regions p, q, we have $(p, q) \in F$ iff the regions p and q have a common edge. This edge will be called the edge *dual* to (p, q), and denoted by $e_{p,q}$. The relationship between an outerplanar graph and its dual was studied in [9]. The following fact is well-known.

Lemma 11 *Let G be outerplanar. Then $D(G)$ is a tree.*

Now we can describe the procedure for orienting 2-connected components of G.

$Orient(C, z)$**:**

construct $D(C)$;

transform $D(C)$ into a directed in-tree $\vec{D} = (U, \vec{F})$,

rooted at a region containing z;

for each $p \in U$ **do parallel begin**

if p does not contain z **then begin**

$q \leftarrow$ the father of p in $\vec{D}$;

$(x, y) \leftarrow e_{p,q}$

end else

$(x, y) \leftarrow (z, z')$, where (z, z') is any

edge in the external region;

$t \leftarrow$ any vertex on p other than x and y;

let X and Y be, respectively, the paths

from t to x and y, along p;

orient the edges in X from t to x;

orient the edges in Y from t to y;

if p contains z **then**

orient (z, z') from z' to z

end

Note that each region p orients all edges on this region, except the edge dual to (p, q), where q is the father of p. This edge $e_{p,q}$ will be oriented by q. In Fig.2, the reader can find an example of an outerplanar graph G, its dual $\vec{D}$, and a relationship between $\vec{D}$ and the orientation of G.

Theorem 9 *There is a parallel CRCW PRAM algorithm which finds an acyclic 2-bounded orientation in an outerplanar graph, and which runs in time $O(\log n)$ on $O(n)$ processors.*

Proof: First we prove the correctness. As it was already mentioned above, it is sufficient to prove the correctness of the procedure $Orient(C, z)$. Consider the regions around a given vertex u. It is easy to see that there is at most one region around u, such that its father in $\vec{D}$ is not a region around u, because otherwise we would have a contradiction with the fact that $\vec{D}$ is an in-tree. Therefore, in $\vec{D}$ the regions around u correspond to two paths meeting at p (one of these paths may be empty). Let us look at some region q around u. Suppose that $q \neq p$. This means that the successor of q, say r, is also a region around u. Let $(u, y) = e_{q,r}$. If (u, s) is the other edge from q incident to u, then, according to the algorithm, we will have $\omega(u, s) = s \rightarrow u$. Therefore all edges incident to u, except these which are on region p, will be directed into u. This implies that $\deg^+(u) \leq 2$. A similar argument shows that $\deg^+(z) = 0$.

Consider now the complexity of this algorithm. Finding the embedding and the tree $\mathcal{T}$ of 2-connected components costs time $O(\log n)$, using the algorithm from [7, 22]. In $Orient(C, z)$, construction of $D(G)$ and $\vec{D}$ can be done in $O(\log n)$ time, if the embedding is already computed. Orienting the edges around a region also can be done in time $O(\log n)$. Therefore the total time complexity is $O(\log n)$. □

6 Series-Parallel Graphs

In this section we show that the results from the previous section can be generalized to series-parallel graphs. The class of *series-parallel* graphs contains graphs G in which two vertices s_G and t_G are distinguished, and called often *source* and *sink*. We define such graphs inductively as follows.

(a) A single edge (s,t) is a series-parallel graph.

(b) Suppose that H_1 and H_2 are series-parallel. Then

  (i) Let $H = H_1 \otimes H_2$ be the graph obtained from H_1 and H_2, by identifying t_{H_1} with s_{H_2}, and taking $s_H := s_{H_1}$, $t_H := t_{H_2}$. Then also H is series-parallel. The operation $\otimes$ is called the *series composition.*

  (ii) Let $H = H_1 \oplus H_2$ be the graph obtained from H_1 and H_2 by identifying s_{H_1} with s_{H_2} and t_{H_1} with t_{H_2}, and taking s_H, t_H to be the vertices obtained by this identification. Then, H is also series-parallel. The operation $\oplus$ will be called the *parallel composition.*

Both series and parallel compositions can be extended in an obvious way to have more than two arguments. We prove first the following theorem.

Theorem 10 *Each series-parallel graph has an acyclic 2-bounded orientation, and this orientation can be found in linear time.*

Proof: We will prove a slightly stronger fact.

($*$) Every series composition H has an acyclic 2-bounded orientation such that $\deg^+(s_H) = \deg^+(t_H) = 0$.

($**$) Every parallel composition H has an acyclic 2-bounded orientation with $\deg^+(t_H) = 0$ and $\deg^+(s_H) \le 1$.

Note that an isolated edge satisfies ($**$). The proof is by induction. Suppose first that H is a series composition, that is $H = H_1 \otimes H_2$. We know that H_2 has an acyclic orientation such that $\deg^+(s_{H_2}) \le 1$ and $\deg^+(t_{H_2}) = 0$. By symmetry, H_1 has an acyclic orientation such that $\deg^+(t_{H_1}) \le 1$ and $\deg^+(s_{H_1}) = 0$. After identifying s_{H_2} with t_{H_1} we obtain a desired orientation of H.

Suppose now that H is a parallel composition. Then we can represent H as $H = H_1 \oplus \ldots \oplus H_k$, where the graphs H_i are series compositions, except possibly of one of them, say H_k, which is a single edge. If $H_k = (s_H, t_H)$, then we set $\omega(s_H, t_H) = s_H \to t_H$. The orientation of the other graphs H_i remains unchanged. Them H clearly satisfies ($*$).

It is very easy to implement this method in linear time, given a series-parallel representation of a given graph. Such a representation can be also found in linear time (see [23]). □

Theorem 11 *The 2-bounded acyclic permutation from Theorem 10 can be constructed in time $O(\log n)$ with $O(n)$ processors on an CREW PRAM, or in time $O(\log^2 n)$ with $O(n/\log n)$ processors on an EREW PRAM.*

Proof: We only sketch the algorithm here. We use the algorithm from [8], where a recognition algorithm for series-parallel graphs is presented, which works in time $O(\log n)$ and uses $O(n)$ processors on an CREW PRAM, or in time $O(\log^2 n)$ with $O(n/\log n)$ processors on an EREW PRAM. A less efficient algorithm can be found also in [13]. These algorithms actually construct the series-parallel representation of a given graph G. Such a representation can be visualized as a tree T in which each node corresponds to some component H of G. We can transform this tree in such a way that if H is a parallel decomposition, and $H = H_1 \oplus \ldots \oplus H_k$, where the H_i are series compositions, except possibly of H_k which can be a single edge, then the graphs H_i are the sons of H. This tree will correspond to the method from the proof of Theorem 10.

Suppose that H is a parallel composition, $H = H_1 \oplus \ldots \oplus H_k$, as in the proof of Theorem 10. If H_k is an edge (s_{H_k}, t_{H_k}), then we call this edge an *st*-edge for H. Each edge is and *st*-edge for some parallel component of G (we treat edges in series compositions as parallel components with $k = 1$). Thus in the algorithm we have to decide for each edge $e = (s_H, t_H)$, whether to orient it from s_H to t_H, or vice versa.

This can be easily determined by looking at the father H' of H. H' is a series composition of H and some other component H_1. If $H' = H \otimes H_2$, then $\omega(e) := t_H \to s_H$. Otherwise, if $H' = H_2 \otimes H$, then $\omega(e) := s_H \to t_H$.

The total time complexity is dominated by the construction of the representation of G, the rest is easy to do within the complexity bounds stated in the theorem. □

7 Applications

In this section we show how our compacted adjacency matrix can be applied in some algorithms for planar graphs. Consider the two following problems:

- Given a planar graph G, list all triangles in G.
- Given a planar graph G, list all 4-cliques in G.

There are several algorithm for these problems which use a linear time [2, 15, 20, 6]. However, some of them tend to be rather complicated, especially the algorithm of Papadimitriou and Yannakakis for listing all 4-cliques. In this section we show how we can use a compacted adjacency matrix, together with adjacency lists, for this purpose.

We consider only the problem of listing 4-cliques; listing all triangles is even simpler. As in some previous algorithms we mantain a queue $\mathcal{Q}$ on which we keep all vertices of degree at most 5. The algorithm is as follows:

construct a compacted adjacency matrix;

$\mathcal{Q} \leftarrow \{v \in E | \deg(v) \leq 5\}$;

while $\mathcal{Q} \neq \emptyset$ **do begin**

$v \leftarrow$ the first vertex from $\mathcal{Q}$;

for every triple $x, y, z \in N(v)$ **do**

  if $(x, y), (x, z), (y, z) \in E$ **then**

    print the 4-clique $\{v, x, y, z\}$;

$N \leftarrow N(v)$;

$G \leftarrow G - \{v\}$;

for each $x \in N$ **do**

  if $\deg(x) \leq 5$ **and** $x \notin \mathcal{Q}$

    then $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x\}$

end

Theorem 12 *The algorithm above lists all 4-cliques in a planar graph, and it works in time $O(n)$.*

Proof: Let v be a vertex chosen at some iteration from $\mathcal{Q}$. We have $O(1)$ triples of neighbours of v to check. Using our compacted adjacency matrix, each test costs time $O(1)$. So in time $O(1)$ we list all 4-cliques containing v at this phase. (Some of the cliques containing v might have been already listed before, though.)

To complete the proof we need to make yet two observations. First, all 4-cliques will indeed be listed. Consider some 4-clique $\{v, x, y, z\}$, and let v be the vertex which was put first into $\mathcal{Q}$. Then this clique will be listed when v is removed from $\mathcal{Q}$. This also shows that this clique will be listed only once.

Second, observe that we have to update also our compacted adjacency matrix when we remove v. This is easy to do, if we have also adjacency lists: we remove all entries at the row v, and for all neighbours $x \in N(v)$ we remove v from the row of x. $\square$

8 Final Remarks

In this paper we have presented sequential and parallel algorithms for orienting edges in a planar graph in such a way that the out-degree of each vertex is bounded by a constant: 5 or 6 in case of acyclic orientations, and 3 in case of arbitrary orientations. Our sequential algorithms are optimal; they run in linear time. The parallel algorithms are also optimal, and they do not need the input graph to be given with an embedding.

For outerplanar graphs, the most interesting fact, in our opinion, is that it is possible to compute in NC an optimal, that is 2-bounded, acyclic orientation. Actually, the algorithm we presented runs in time $O(\log n)$ with $O(n)$ processors, so it is almost optimal. It would be very interesting to find fast parallel algorithms for better acyclic orientations of planar graphs, that is at least 5-bounded. Unfortunately, the technique we use for outerplanar graphs does not seem to apply in more general cases.

As it was already noted, these bounded orientations can be applied to compact the adjacency matrix of planar graphs. This work was motivated by the paper of Chiba, Nishizeki and Saito [4], who present an $O(n \log n)$-time algorithm for finding large independent sets in planar graphs. The main drawback of this algorithm is that it uses an adjacency matrix, so it requires $O(n^2)$ space. Unfortunately, the way of compacting adjacency matrix we present is not yet sufficient to reduce the space requirements in their algorithm, because their algorithm performs vertex contractions during its execution, and it is not clear how these contractions can be done with such a compacted adjacency matrix. This problem was solved in [6] by a different method, which also reduces the total time of the algorithm to $O(n)$.

Another possibility of applying our methods may be to find grid embeddings of planar graphs. In a recent paper on this topic, W. Schnyder [21] presented a very elegant method for finding such embeddings by using a decomposition of planar graphs into three trees. The decomposition he uses must have certain acyclity properties, but we suspect that suitable modifications of our algorithms may give an appropriate decomposition.

As we have shown in the preceding section, our method turns out to be useful in other algorithms on planar graphs. Having both the adjacency lists and our compacted adjacency matrix, it is possible both to search a graph quickly, and to answer queries $(*)$ in constant time.

Acknowledgements. We would like to thank the referee for pointing out several references, and suggesting that our results for outerplanar graphs can be extended to series-parallel graphs.

References

[1] A. Aho, J. Hopcroft, and J. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, 1974.

[2] R. Bar-Yehuda, S. Even, On approximating a vertex cover for planar graphs, Proc. 14th Annual ACM Symposium on Theory of Computing, San Francisco, May 5-7, 1982, pp. 303–309.

[3] R.P. Brent, The parallel evaluation of general arithmetic expressions, J. Assoc. Comput. Mach. 21 (1974), pp. 201–206.

[4] N. Chiba, T. Nishizeki, and N. Saito, An approximation algorithm for the maximum independent set problem on planar graphs, SIAM J. Comput. (1982), pp. 663–675.

[5] N. Chiba and T.Nishizeki, Arboricity and subgraph listing algorithms, SIAM J. Comput. 14 (1985), pp. 210–223.

[6] M. Chrobak and J. Naor, An efficient parallel algorithm for computing a large independent set in planar graphs, submitted for publication.

[7] K. Diks, Parallel recognition of outer-planar graphs, submitted for publication.

[8] D. Eppstein, Parallel recognition of series-parallel graphs, submitted for publication.

[9] H.J.Fleishner, D.P.Geller, F.Harary, Outerplanar graphs and weak duals, J. Indian Math. Soc. 38 (1974) 215-219.

[10] A. Goldberg, S. Plotkin, and G. Shannon, Parallel symmetry breaking in sparse graphs, Proc. 19th Annual Symposium on Theory of Computing (1987), pp. 315–324.

[11] T. Hagerup, M. Chrobak, and K. Diks, Optimal parallel 5-coloring of planar graphs, to appear in SIAM J. Comput.

[12] F. Harary, Graph Theory, Academic Press, 1972.

[13] X. He and Y. Yesha, Parallel recognition and decomposition of two terminal series parallel graphs, Info. and Comput. 75 (1987) pp. 15–38.

[14] J.E. Hopcroft and R.E. Tarjan, Efficient planarity testing, J. Assoc. Comput. Mach. 21 (1974), pp. 549–568.

[15] A. Itai and M. Rodeh, Finding a minimum circuit in a graph, SIAM J. Comput. 7 (1978), pp. 413–423.

[16] S.Kannan, M.Naor and S.Rudich, Implicit representations of graphs, Proc. 20th Annual Symposium on Theory of Computing, 1988, pp. 334–343.

[17] R.E. Ladner and M.J. Fischer, Parallel prefix computation, J. Assoc. Comput. Mach. 27 (1980), pp. 831–838.

[18] C. Nash-Williams, On orientations, connectivity and odd-vertex pairings in finite graphs, Canad. J. Math. 12 (1960), pp. 555–567.

[19] C. Nash-Williams, Edge-disjoint spanning trees of finite graphs, J. London Math. Soc. 36 (1961), pp. 445–450.

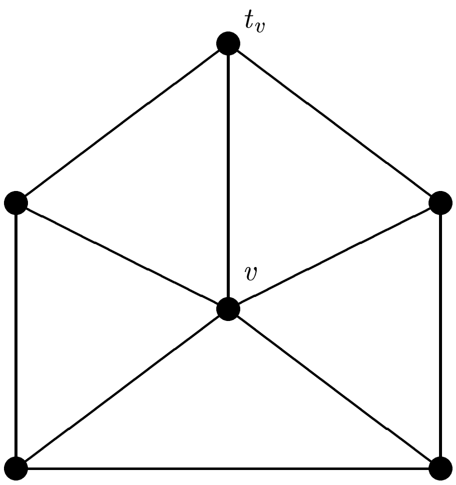
[20] C. Papadimitriou and M. Yannakakis, The clique problem for planar graphs, Inform. Proc. Letters 13 (1981), pp. 131–133.

[21] W.Schnyder, Embedding planar graphs on the grids, a manuscript.

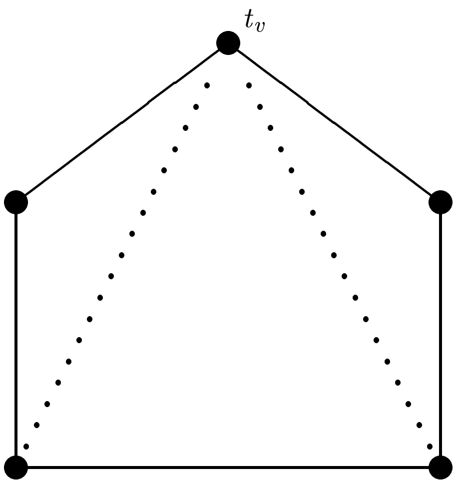
[22] R. Tarjan and U. Vishkin, An efficient parallel biconnectivity algorithm, SIAM. J. Comput. 14 (1985), pp. 862–874.

[23] J. Valdes, R.E. Tarjan, and E.L. Lawler, The recognition of series parallel digraphs, Proc. 11th Annual ACM Symposium on Theory of Computing, 1979, pp. 1–12.

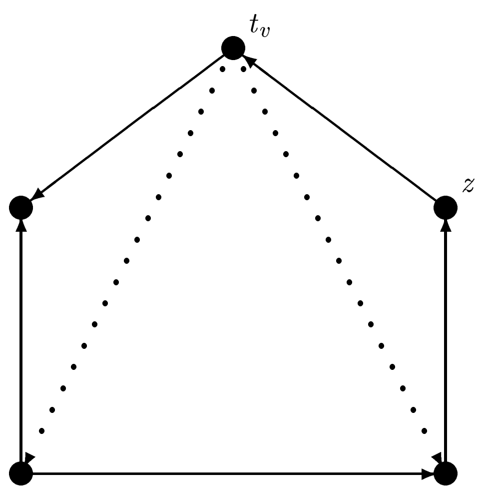
[24] S. Vishwanathan and M.A. Sridhar, Some results on graph coloring in parallel, manuscript.



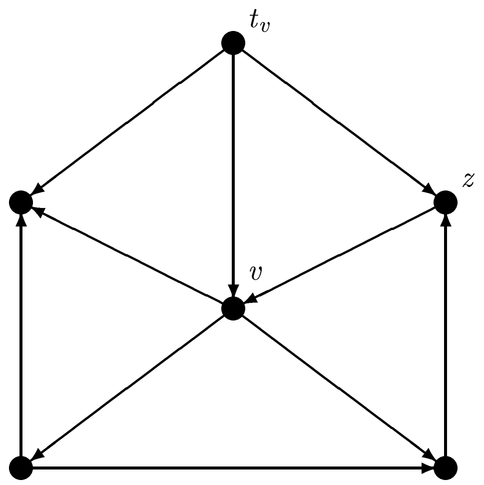
(a) A reducible vertex v before a reduction.



(b) After the reduction.



(c) After orienting G'.



(d) After extending the orientation.

Fig.1. An example of a reduction and extension.

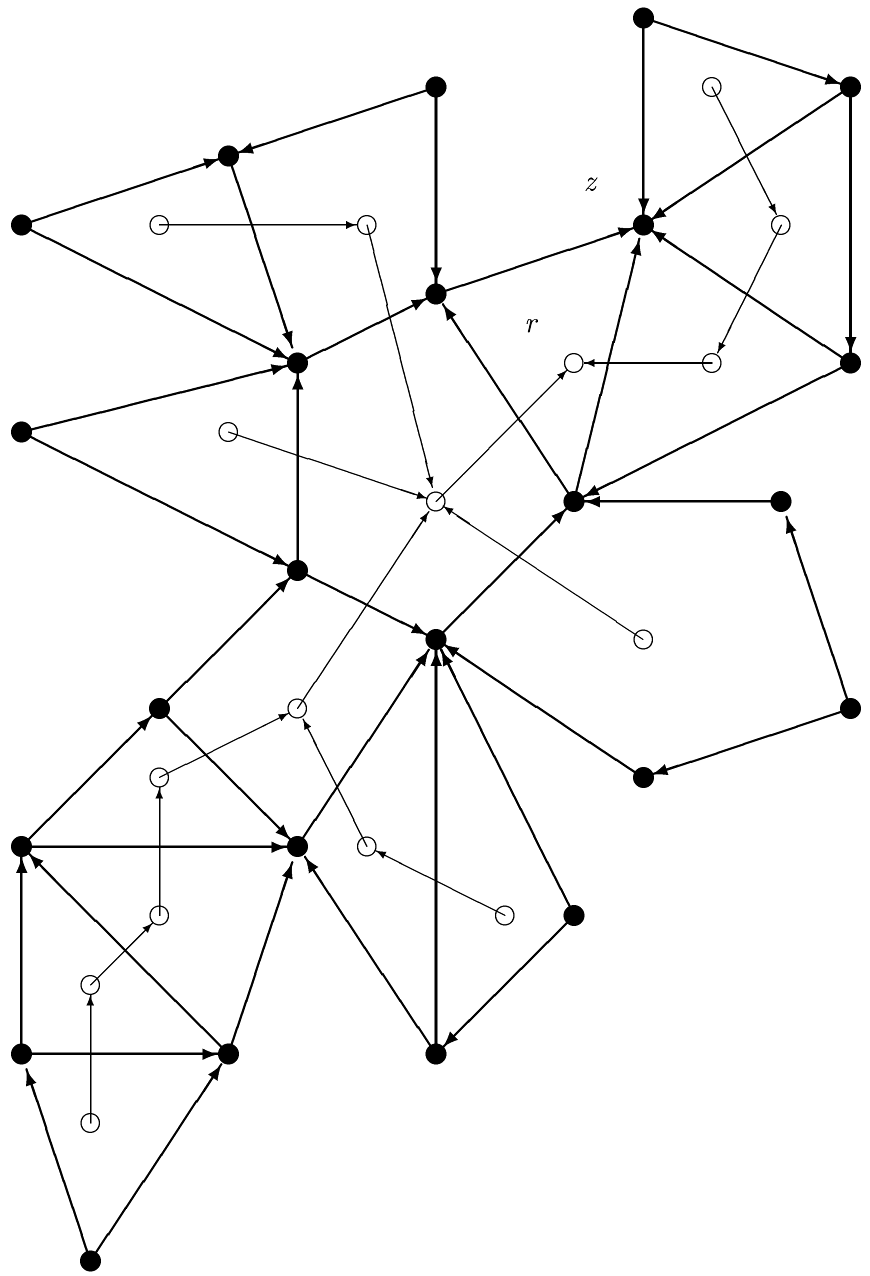


Fig. 2. An example of an outerplanar graph G (thick lines) and its dual $\vec{D}$ (thin lines) after orientation. The root of $\vec{D}$ is denoted by r.