References

[1] Auslander, L., and S.V. Parter, On embedding graphs in the plane, *J. Math. Mech.* 11 (1961), pp. 517–523.

[2] Booth, K.S., and G.S. Lueker, Testing for the consecutive ones property, interval graphs and graph planarity testing using PQ-tree algorithms, *J. of Computer and System Sciences* 13 (1976), pp. 335–379.

[3] Cai, J., X. Han and R.E. Tarjan, *New solutions to four planar graph problems*, Technical Report, Dept. of Computer Science, New York University/Courant Institute, 1989.

[4] Chiba, N., T. Nishizeki, S. Abe and T. Ozawa, A linear algorithm for embedding planar graphs using PQ-trees, *J. of Computer and System Sciences*, Vol. 30 (1985), pp. 54–76.

[5] Chiba, T., I. Nishicka and I. Shirakawa, An algorithm of maximal planarization of graphs, *Proc. 1979 IEEE Intern. Symp. on Circuits and Systems*, 1979, pp. 649–652.

[6] Di Battista, G., and R. Tamassia, Incremental planarity testing, *Proc. 30th Annual IEEE Symp. on Found. on Comp. Science*, North Carolina, 1989, pp. 436–441.

[7] Garey, M.R., and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman & Co., San Francisco, 1979.

[8] Hopcroft, J., and R.E. Tarjan, Efficient planarity testing, *J. ACM* 21 (1974), pp. 549–568.

[9] Jayakumar, R., K. Thulasiraman and M.N.S. Swamy, $O(n^2)$ algorithms for graph planarization, *IEEE Trans. on Computer-aided Design*, Vol. 8 (1989), pp. 257–267.

[10] Jayakumar, R., K. Thulasiraman and M.N.S. Swamy, On Maximal Planarization of non-planar graphs, *IEEE Trans. on Circuits and System Sciences*, Vol. CAS-23 (1986), pp. 843–844.

[11] Lempel, A., S. Even and I. Cederbaum, An algorithm for planarity testing of graphs, *Theory of Graphs, Int. Symp. Rome* (1966), pp. 215–232.

[12] Ozawa, T., and H. Takahashi, A graph-planarization algorithm and its applications to random graphs, in: *Graph Theory and Algorithms*, Springer-Verlag Lecture Notes in Computer Science 108 (1981), pp. 95–107.

nodes	edges	v-TEST	e-TEST
100	50	50	50
100	100	98	92
100	150	114	114
100	200	129	131
100	250	134	152
200	100	100	100
200	200	180	168
200	300	216	210
200	400	238	247
200	500	247	287
300	150	150	150
300	300	277	260
300	450	322	304
300	600	350	372
300	750	363	434
400	200	200	200
400	400	362	328
400	600	414	379
400	800	441	476
400	1000	458	555
500	250	250	250
500	500	438	401
500	750	514	504
500	1000	546	580
500	1250	559	682

Figure 8: Comparison between v-TEST and e-TEST, when applied to random graphs.

k	100 nodes		200 nodes		300 nodes		400 nodes		500 nodes	
	v-TEST	e-TEST	v-TEST	e-TEST	v-TEST	e-TEST	v-TEST	e-TEST	v-TEST	e-TEST
35	189	187	504	325	731	480	730	656	1189	804
75	207	293	438	326	683	474	917	637	1147	738
115	203	238	428	318	644	482	894	746	1107	763
155	205	239	421	324	630	487	829	635	1073	800
190	212	161	396	319	668	472	820	638	1037	791
230	222	163	387	318	624	471	823	636	1020	778
265	226	162	412	319	606	476	815	632	1048	769
305	232	162	405	325	606	467	805	633	1011	750
345	245	162	409	319	615	470	789	634	1008	786
375	234	166	398	317	567	480	795	617	965	804

Figure 9: Comparison between v-TEST and e-TEST on "planar-like" graphs.

However, the best time bound for maximal planarization is still $O(m \log n)$ by algorithms of Cai et al. [3] and Di Battista & Tamassia [6]. It is an open problem whether there exists a maximal planarization algorithm based on PQ-trees, which also works in $O(m \log n)$ time, or whether there is a general maximal planarization algorithm that is faster than $O(m \log n)$ time. From first experimental results we learned that in the maximal planarization phase of our algorithm, only a very few edges were added. Therefore, we only implemented the planarization algorithm described in section 3 in more detail (originally due to Jayakumar et al. [9]). We call this algorithm v-TEST, because it tests for every vertex, which set of incident edges must be deleted to preserve the planarity. We compared the results with the algorithm of Di Battista & Tamassia (here called e-TEST, because this algorithm tests for every edge whether it can be added to G_p, the planar subgraph). We applied this approach to two kinds of graphs: (i) random graphs and (ii) triangulated planar graphs augmented with a number k of random edges. The results can be found in figure 8 and 9, respectively.

From figure 8 we conclude that for random graphs with $m \leq \frac{3}{2}n$ the planarization algorithm of section 3 (v-TEST) is better than the algorithm of Di Battista & Tamassia (e-TEST). For more dense graphs ($m > \frac{3}{2}n$) the approach of e-TEST seems to be better. Further experiments on this and larger graphs confirm this. However, for "planar-like" graphs, which are graphs that are almost planar, the approach of v-TEST is by far better than e-TEST. In figure 9 a clear difference is given. A triangulated planar graph with n nodes has $3n-6$ edges, hence for $n = 400$ and $k = 190$ we have 1484 edges, of which 820 remain by v-TEST, compared with 638 of e-TEST. Notice also that when k increases, v-TEST deletes more edges in general, while the number of edges of e-TEST remains constant. Similar results are suspected when applying the algorithm of Cai, Han & Tarjan [3]. We conclude that for planar-like and sparse (which are essentially planar-like) graphs the planarization algorithm based on the PQ-tree algorithm of Booth & Lueker [2] is preferred above the algorithm of Di Battista & Tamassia [6], when we search for a minimal number of deleted edges.

The PQ-trees are also used for checking whether a given inputgraph is an interval graph [2]. Hence it seems that our maximal planarization algorithm can be used in this context as well, to compute a subgraph G' of G such that G' is an interval graph. This area is a nice topic for further research on algorithms, based on PQ-trees.

Acknowledgements

The author wishes to thank Jan van Leeuwen for some useful suggestions and reading earlier drafts.

Proof: By lemma 5.1 it follows that G'_p is planar. For suppose that G'_p is not maximal planar, i.e., an edge $e \in G - G_p$ could be added to G'_p without detroying the planarity. But now the corresponding potential leaf l must form a near pair with its sequence indicator by lemma 5.1, and l and $si(l)$ are always once part of a maximal pertinent sequence. But now the algorithms P-NEAR and Q-NEAR are applied and by lemma 5.3 and 5.4, this pair will be detected and reduced, if no intersecting pair of $l, si(l)$ is reduced. Hence e will be added to G'_p, which yields a contradiction. □

Theorem 5.6 MAX_PLANARIZE2 *can be implemented to run in* $O(n^2)$ *time and space.*

Proof: From lemma 1 of [9] it follows that the number of non-empty Q-nodes is at most n and the number of non-empty P-nodes is at most i in every PQ-tree T_i. Therefore we initialize $O(n)$ arrays SI and PL.

Both in Q-NEAR and P-NEAR, we have to compute PL_X and SI_X for every preferred node X. But using the information of its children Y, we can update these arrays and test for near pairs in $O(|PLSI(Y)|)$ time. When X is a pertinent P-node, then we also have to compute PL_X and SI_X from the arrays of the empty children Y. But when Y is not removed after the reduction, then we have to calculate this information for X every time when it becomes pertinent. Instead of doing this, we keep the information of PL_Y and SI_Y stored in PL_X and SI_X after the reduction.

By using the pointers $PLSI(X)$ it follows that visiting a pertinent node X requires $O(|PLSI(X)|)$ time, for testing for near pairs. If we reduce a near pair $l, si(l) =< i >$, then this costs $O(|PL_Y[i]|)$ time for every node Y on the path between l and $si(l)$. After the reduction $|PL_Y[i]| = 0$, because the corresponding children are removed from the PQ-tree.

Hence the total time for MAX_PLANARIZE2 is $O(n^2 + m + \sum_X |PLSI(X)|)$ time, for all pertinent nodes X, visited in all steps of MAX_PLANARIZE2. $|PLSI(X)|$ is equal to the sum of all sizes of all pertinent children in PLANARIZE. Since by lemma 1 of [9] the sum of all pertinent nodes in $T_1, \ldots, T_{n-1}$ is $O(n^2)$, this completes the proof. □

6 Concluding Remarks

In this paper we showed that the maximal planarization algorithm of Jayakumar et al [9] is not correct. We corrected and generalized the algorithm by describing a maximal planarization algorithm MAX_PLANARIZE2, based on the "vertex addition" planarity-testing algorithm of Booth & Lueker [2]. Our algorithm MAX_PLANARIZE2 is the first maximal planarization algorithm based on PQ-trees that is working faster than $O(mn)$.

that descendants of empty children of X also may form a near pair with descendants of pertinent children of X. Therefore we fill PL_X and SI_X in the following order of the children of X: first with PL_Y, SI_Y of Y, Y an empty child of X, then with PL_Y, SI_Y, Y a full child of X, then with PL_Y, SI_Y, Y a partial child of X. When updating PL_X and SI_X, we of course test for near pairs. When descendants of Y_i and Y_j form a near pair, then we make Y_i and Y_j the two children of a new Q-node and do a REDUCE. If Y_i (Y_j) was a Q-node, then we make Y_j (Y_i) endmost child of Y_i (Y_j), at this side where the near pair is. After reducing the near pairs we update PL_X and SI_X. The algorithm can now be described as follows:

```
P-NEAR(X);
  for all empty children Y of X do COMPUTE_ARRAY(X, Y);
  let Y_1, ..., Y_p be the pertinent children of X, with full children first;
  for i := 1 to p do
    for all non-empty entries PL_{Y_i}[k] do
      add Y_i to PL_X[k];
      if SI_X[k] points to a child, say Y_j of X then
        make Y_i and Y_j siblings of a Q-node X';
        REDUCE(k, Y_i, Y_j, X');
    od;
    for all non-empty entries SI_{Y_i}[k] do
      add Y_i to SI_X[k];
      if PL_X[k] is not nil then
        make children Y_j and Y_i siblings of a Q-node X';
        REDUCE(k, Y_j, Y_i, X');
    od;
```

Lemma 5.4 *When there is a near pair $l, si(l)$ with least common parent X, then the edge e of l is added to G_p and the PQ-tree is correctly updated, when no intersecting near pair of $l, si(l)$ is reduced by* P-NEAR.

Proof: By testing if both $PL_X[k]$ and $SI_X[k]$ are not empty, we detect the near pairs. By the ordering of updating PL_X and SI_X, we first reduce the near pairs between the empty and full children, then between full children. Finally the partial children are added to this ordering of the maximal pertinent sequence. By the proof of lemma 5.3, REDUCE gives a correct reduction of the PQ-tree of a near pair. □

This leads to the following theorem:

Theorem 5.5 *Given a spanning planar subgraph G_p of G* MAX_PLANARIZE2 *computes a maximal planar subgraph G'_p with $G_p \subseteq G'_p \subseteq G$.*

3. Do step 2. for $Y_1, \ldots, Y_k$ the path of nodes between $si(l)$ and X ($k \geq 1$).

4. For all empty leaves between the rightmost incoming edge of i and sequence indicator $si(l) =< i >$, check whether there exist other near pairs, by checking SI_Y, PL_Z, PL_Y and SI_Z accordingly. If this is the case, reduce correspondingly.

5. Delete all (empty) nodes between the leftmost incoming edge (j', i) and $< i >$ from the tree. If a deleted leaf is an edge (j', i) then add it to G_p.

6. for all deleted nodes Y' of the tree, remove their presence in the arrays PL_Y and SI_Y of their parent Y of Y'.

Notice that empty leaves left from a pertinent node Y_i can never form a near pair, since Y_j should be deleted by the corresponding reduction. Therefore the entries SI_X and PL_X are initialized when visiting a pertinent node. The following lemma shows that Q-NEAR reduces the near pairs in a correct way.

Lemma 5.3 *When there is a near pair $l, si(l)$ with least common parent X, then the edge e of l is added to G_p and the PQ-tree is correctly updated, when no intersecting near pair of $l, si(l)$ is reduced by* Q-NEAR.

Proof: Assume w.l.o.g. that l is a descendant leaf of Y_j and $si(l)$ is a descendant leaf of Y_i, $i > j$. When there is an intersecting near pair of $l, si(l)$ already reduced by Q-NEAR, then Y_j is already deleted, hence we would not detect the near pair $l, si(l)$. Otherwise we detect the near pair when visiting Y_i. We delete the other children Y_r ($j < r < i$) from the PQ-tree. When Y_i (or Y_j) is of Type U, then we update Y_i (or Y_j) as defined for U-Types. If Y_j is pertinent, then it has some pertinent children. These children may not come between the near pair, hence come left from l. We add the other (empty) children of Y_j between the near pair (if Y_j is a P-node), because there may be other potential leaves, which can form a near pair with $si(l)$. If Y_j is a Q-node, then the order is already fixed, so we only have to test for the empty leaves in between, whether they form a near pair with $si(l)$.

Finally, if Y_j is empty, then we introduce the U-Type nodes, as defined before, because Y_j may have several children, which can form a near pair with $si(l)$. Only one of these children can be leftmost; all other children must be removed. This is exactly done by REDUCE.

When l and $si(l)$ are changed, then a similar proof follows. After defining the ordering we delete the leaves between the near pair, and the elements of the near pair become adjacent siblings of a common parent. This completes the proof. □

When the current node X for the reduction-step in MAX_PLANARIZE2 is a P-node, then we apply a similar approach. We have to find an ordering of the full pertinent children between the partial pertinent children of X. Moreover we notice

computing the near pairs is described by Q-NEAR(X) and the reduction of near pairs is formally given by REDUCE.

Q-NEAR(X);
  initialize PL_X and SI_X to be empty;
  for $i := 1$ **to** p **do**
    for all non-empty entries $PL_{Y_i}[k]$ **do**
      add Y_i to $PL_X[k]$;
      if $SI_X[k]$ points to a child, say Y_j of X **then** REDUCE(k, Y_i, Y_j, X);
    od;
    for all non-empty entries $SI_{Y_i}[k]$ **do**
      add Y_i to $SI_X[k]$;
      if $PL_X[k]$ is not nil **then** REDUCE(k, Y_j, Y_i, X), $Y_j \in PL_X[k]$ with smallest j;
    od;
    if Y_i is pertinent **then**
      initialize PL_X and SI_X to be empty;
      COMPUTE_ARRAY(X, Y_i);
  od;

REDUCE(i, Y, Z, X);
{ Y and Z are children of Q-node X and ancestors of a near pair $l, si(l)$, respectively, with $si(l) =< i >$. }

1. Delete all children and their descendants of X between Y and Z from the tree.
   if a deleted leaf is an edge (j', i) **then** add it to G_p;

2. Let $Y_1, \ldots, Y_k$ be the path of nodes between l and X ($k \geq 1$);
   for $i := 1$ **to** k **do** case Y_i of the following Types:
   Type U : save this side, where Y_{i-1} belongs to (Y_i a Q-node)
   or child Y_{i-1} (Y_i a P-node);
   delete all other children and remove Type U;
   pertinent: **if** Y_i is a P-node **then**
   include all empty children between near pair;
   empty: **if** Y_i is a P-node **then**
   let $PL_{Y_i}[k] = \{Y'_1, \ldots, Y'_p\}$; make $Y'_1, \ldots, Y'_p$
   children of a new P-node Y' of Type U;
   make Y' and Y_i child of a new Q-node Z, child of Y_{i+1}.
   else
   let $Y'_1, \ldots, Y'_p$ be the smallest consecutive sequence
   of children of Y_i, containing incoming edges (j', i)
   and the pertinent children in their frontier;
   replace this sequence $Y'_1, \ldots, Y'_p$ by a special
   marked child Y'; make Y_i Type U;

Similar we have the situation that in T_i R has no partial children, then two full children of R can be made Type H. We introduce two P-nodes X_1, X_2 of Type U and make all full children $P_1, \ldots, P_k$ children of X_1 and X_2. We introduce a Q-node Y with endmost children X_1 and X_2. In the vertex addition step, the P-node with outgoing edges comes between X_1 and X_2 as child of Y. As soon when one descendant leaf of $P_1, \ldots P_{k-1}$ or P_k, say P_i, forms a near pair with another leaf not in the frontier of Y, then we replace X_1 by P_i and remove P_i as child from X_2. This means that we should make node P_i Type H in T_i. Therefore X_1 is changed by P_i and the reduction of the near pair can be done accordingly. The other node of Type H is one of the sons of X_2 of Type U, similar to case (b) of section 4, where R had only one partial child.

A similar argument can be applied when P_i is a full Q-node, with at both sides from the pertinent sequence to the endmost children only empty leaves, say the sets L_1 and L_2. When we make P_i Type H, one (arbitrary) set L_1 or L_2 must be deleted, but (as above) this may not always lead to a maximal planar subgraph. This question of deleting L_1 or L_2 can be delayed by making P_i Type U and replacing the pertinent sequence of children by one special marked child. When in a later step a descendant leaf of P_i, say a leaf of L_1, forms a near pair with an empty leaf, not a descendant of P_i, we remove the set L_2 of empty leaves and Y from the tree. P_i is not of Type U anymore.

We introduce two arrays PL_X and SI_X for every node X in the maximal pertinent sequence. $PL_X[i]$ will contain those children of X which are ancestors of some potential leaves of edges (j, i), and similar for SI with respect to the sequence indicators. These arrays can easily be filled as follows:

COMPUTE_ARRAY(X, Y)
for every non-empty entry $SI_Y[i]$ (or $\mathrm{PL}_Y[i]$) **do** add Y to $SI_X[i]$ (or $PL_X[i]$);
if Y is a potential leaf (j, i) **then** add Y to $PL_X[i]$
if Y is a sequence indicator $< i >$ **then** add Y to $SI_X[i]$

Using a linked list $PLSI(X)$ for every node X, with pointers to the non-empty entries of node X, we can find these entries in $|PLSI(X)|$ time.

The idea to recognize and reduce near pairs is the following when X is a Q-node. We first apply the replacement step Q2 or Q3 of [2], which gives us exactly one sequence of pertinent children $Y_1, \ldots, Y_p$ of X, which are all full. We visit Y_1, which is not endmost, and walk to the endmost child Y_p (say right of Y_1) of X, thereby updating PL_X, SI_X and testing for near pairs. If two leaves of two nodes Y_i and Y_j form a near pair (e.g., $Y_i \in PL_X[k]$ and $SI_X[k] = \{Y_j\}, i < j$), then no node Y_r can be pertinent of course, but also descendants of Y_r cannot form a near pair with descendants of Y_s $(s < r)$, because then we would detect and reduce this before visiting Y_j. To reduce Y_i and Y_j, we have to delete all nodes Y_r, $(i < r < j)$, and we have to reduce the children of Y_i and Y_j such that the near pair become adjacent siblings of a common parent. The algorithm for computing SI_X, PL_X and

and making an arbitrary choice for fixing one node Type H will not necessarily lead to a maximal planar subgraph. But here again we use the principle of "delaying the questions" by introducing a special Type U for P- and Q-nodes (U stands for unknown), defined as follows:

P-node: Except one child (unknown yet) with its descendants, all other children with its descendants must be removed from the PQ-tree in a later step.

Q-node: It has one special marked child Y, and from Y to one of its endmost children (unknown yet), all children of the Q-node including descendants must be removed from the PQ-tree in a later step.

For a P-node only one child can be "saved" and for a Q-node only one side of descendants from the special marked child to one endmost child can be "saved"; all other children, including their descendants must be removed from the tree in a later step (see figure 7).

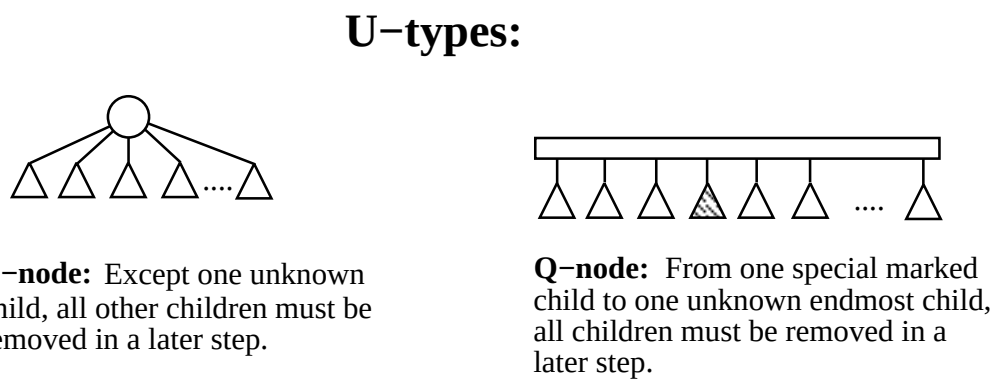


Figure 7: The U-Types for a P- and Q-node.

Using a P-node of Type U, case (b) of section 4 is solved as follows: Let $P_1, \ldots, P_k$ be the new full children of R, the root of the pertinent subtree which has exactly one partial pertinent child. Hence exactly one set L_i of empty leaves, unknown yet, need not be destroyed from the PQ-tree. Delaying the question which set L_i can be saved is solved by introducing a P-node X of Type U, which we make child of R. We make all full nodes $P_1, \ldots, P_k$ child of X and in the vertex addition step, only the pertinent children of $P_1, \ldots, P_k$ are removed from the PQ-tree. This implies that all the sets $L_1, \ldots, L_k$ of empty leaves are descendants of X. As soon when one empty leaf of $L_1, \ldots, L_{k-1}$ or L_k, say L_j, forms a near pair with another leaf, not a descendant of X, then the corresponding child P_j of X is saved, i.e., should be made Type H, and all other children of X are removed. We then replace node X by P_j.

thus during the algorithm we can inspect for near pairs. The maximal planarization algorithm MAX_PLANARIZE2 can now be described at a high level as follows:

```
MAX_PLANARIZE2
  assign st-numbers to all the vertices of G;
  PLANARIZE(G);
  construct the PQ-tree T_1 corresponding to G'_1;
  for i := 1 to n − 1 do
    {compute step}
      compute the maximal pertinent sequence in tree T_i of incoming
      edges of node i + 1 in G_p;
    {reduction step}
      apply the template matchings in the PQ-tree,
      and apply an additional reduce step to reduce near pairs
      in the maximal pertinent sequence;
    {vertex addition step}
      for all deleted sequence indicators < j >,
      remove the corresponding potential leaves from T_i;
      replace all the full nodes in T_i by a P-node X with all
      outgoing edges of node i + 1 appearing as children of X;
      add the sequence indicator < i + 1 > as a sibling of X in T_i;
  od;
```

In the compute step in MAX_PLANARIZE2, potential leaves and sequence indicators can be included in the maximal pertinent sequence in T_i, but a suitable version of BUBBLE-UP [2] can easily take care of this. In the vertex addition step, we additionally have to add a sequence indicator in the PQ-tree, as a sibling of the new P-node, and remove potential leaves of deleted sequence indicators, but its implementation is straightforward and omitted here.

5.2 Reducing the near pairs

We now focus our attention on the reduction step of MAX_PLANARIZE2, to obtain a maximal planar subgraph G_p. In this reduction step we reduce T_i by using template replacements (see Booth & Lueker [2]), but when applying this for a current node X, we now have to apply an additional reduction operation on near pairs, with least common parent X. Reducing a near pair means that we reduce the PQ-tree T_i such that the elements $l, si(l)$ of the near pair are adjacent siblings of a common parent, not binding partial nodes to new places, and deleting only empty nodes. Then the corresponding edge $e \in G - G_p$ is added to G_p, and the leaf l is removed from T_i.

However, as explained in more detail in case (b) of section 4, there occur instances for which every full child of the root of a pertinent subtree can be made Type H,

$i >$ in the PQ-tree. This means that in every tree $T_j, j > i$, there is always at least one partial leaf l_j between l and $si(l)$. If (k, i) could be added to G_p, then it must be adjacent to its maximal pertinent sequence in some step, by the algorithm PLANARITY_TEST of Booth & Lueker [2]. Hence l and $si(l)$ must be adjacent in some step. But to obtain this, we have to delete at least one partial leaf l_j of T_j, which means that when adding e to G_p, at least one other edge, corresponding with l_j, must be removed from G_p, to preserve the planarity. This contradicts with the fact that e could be added to G_p without destroying planarity.

Finally we notice that we test for near pairs $l, si(l)$, when they are part of another maximal pertinent sequence. By definition, there will be no partial nodes inside the pertinent sequence, because otherwise we had to remove edges of G_p to admit planarity.

This completes the proof. □

Between two elements of a near pair several potential leaves and sequence indicators may occur. Two near pairs $l, si(l)$ and $l', si(l')$, are said to be *intersecting* in T_i, if either l' or $si(l')$ is between l and $si(l)$ in all equivalent PQ-trees of T_i. In figure 6, an example of two intersecting near pairs is given.

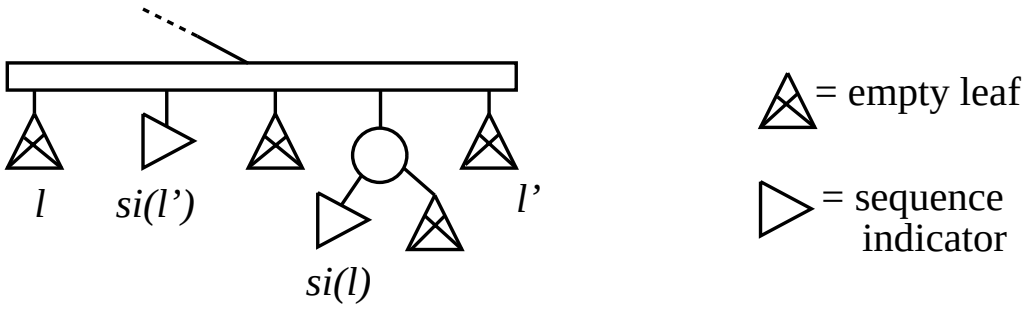


Figure 6: Two intersecting near pairs $l, si(l)$ and $l', si(l')$.

Lemma 5.2 *If two near pairs $l, si(l)$ and $l', si(l')$ are intersecting, then only one corresponding edge of l or l' can be added to G_p, without destroying the planarity.*

Proof: Suppose l occurs between the elements l' and $si(l')$ of a near pair. But when reducing near pair $l', si(l')$, l will be removed from the PQ-tree, hence does not form a near pair with $si(l)$ after reducing near pair $l', si(l')$. By lemma 5.1, l cannot be added to G_p without destroying planarity. □

Hence the order of inspecting the near pairs and applying the necessary reductions is essential for constructing the augmented planar graph G_p. All potential leaves and sequence indicators are once element of a maximal pertinent sequence,

planarity. We call leaves of edges $(k, i+1) \in G - G_p$ *potential leaves* in the PQ-tree $T_j, j > i$. As in algorithm PLANARITY_TEST, we reduce the tree T_i such that the pertinent nodes form a sequence of adjacent siblings of a common parent by applying the templates and replacement patterns (see Booth & Lueker [2]). This sequence is replaced by a P-node X with a set of leaves, representing the outgoing edges of $i+1$. Furthermore, to store the place of maximal pertinent sequence in T_i, we add adjacent to X a *sequence indicator*. This special node, denoted by $< i+1 >$, marks the place of the sequence in the PQ-tree. In [4], a related special node, a *direction indicator*, is introduced to store the place of the maximal pertinent sequence of vertex $i+1$ in the tree, with the direction of enumeration (from left to right or vice versa). This is used in [4] to compute a planar embedding of the graph using PQ-trees. We treat sequence indicators and potential leaves as *empty* leaves, as well as leaves of edges $\in G - G_p$, which are not potential yet.

In MAX_PLANARIZE1, a new pertinent leaf l could be included in the sequence of pertinent nodes, if only empty nodes are between l and the maximal pertinent sequence. Following this principle in our algorithm, a potential leaf l can only be added to the maximal pertinent sequence in T_i, if only empty leaves are between l and its sequence indicator, denoted by $si(l)$, in some tree $T_j, j > i$. Moreover, when we reduce the tree such that all elements of the maximal pertinent sequence are adjacent siblings of a common parent ([2]), we have to take care that we do not bind partial nodes to new places (recall the definition of binding to a new place in section 4). This observation can formally be described as follows

Definition 5.1 *A potential leaf l is* near *its sequence indicator $si(l)$, if the PQ-tree T_i can be reduced such that they are adjacent siblings, by deleting only empty nodes and not binding partial nodes to new places.*

When a potential leaf l and its sequence indicator $si(l)$ are near, then this is called a *near pair*, and we can reduce the PQ-tree such that they are adjacent siblings, and add the edge $e \in G - G_p$ to G_p, without binding partial nodes and leaves to new places. The following lemma is crucial for our algorithm.

Lemma 5.1 *An edge $e \in G - G_p$ can only be added to G_p without destroying planarity, if and only if at some step the corresponding potential leaf l is near its sequence indicator $si(l)$ in the PQ-tree.*

Proof: Assume first that a potential leaf l is near its sequence indicator $si(l)$ in the PQ-tree T_j. This means that by deleting empty nodes between l and $si(l)$ and reducing T_j without binding partial nodes to new places, l and $si(l)$ are adjacent siblings. Hence the corresponding edge of l can be added to G_p without destroying planarity and without removing edges of G_p.

Suppose now an edge $e = (k, i) \in G - G_p$ can be added to G_p without destroying planarity, while the corresponding leaf l is never near its sequence indicator $si(l) =<$

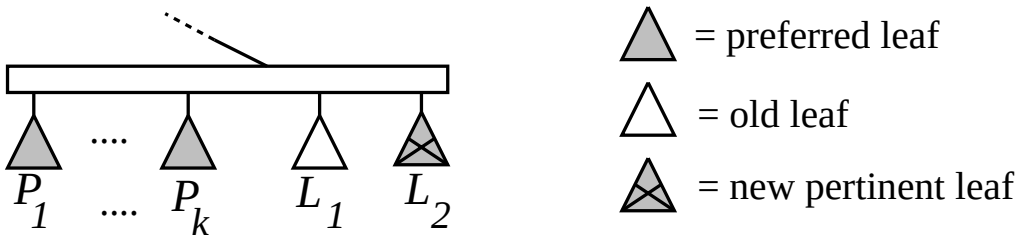


Figure 5: MAX_PLANARIZE1 does not work when G_p is not biconnected.

$P_1, \ldots, P_k$, because there is an partial leaf L_1 in between. Assume L_1 is the only incoming edge of a node i, which has no outgoing edges in G_p. This implies that in step i of MAX_PLANARIZE1, L_1 will be changed into an empty P-node with the outgoing edges. But if we apply step i first then L_2 could be added to the maximal preferred sequence $P_1, \ldots, P_k$. Since L_2 is not added to the maximal preferred sequence by MAX_PLANARIZE1, the resulting planar subgraph will not be maximal planar.

In section 5 we present the algorithm MAX_PLANARIZE2 for the maximal planarization of graphs, which will deal with these problems.

5 A New Maximal Planarization Algorithm

5.1 Outline of the algorithm

In section 4 we have shown that the algorithm MAX_PLANARIZE1 of [9] for maximal planarization of graphs is not correct. In this section we present a corrected version of MAX_PLANARIZE1, that even works when G_p, the planar spanning subgraph computed by PLANARIZE, is not biconnected. Our algorithm, MAX_PLANARIZE2, is based on the idea of *direction indicators* in PQ-trees, as introduced by Chiba et al. [4].

In MAX_PLANARIZE1, insufficient information is available to decide whether an edge $e \in G - G_p$ can be added to G_p when the corresponding leaf l becomes pertinent in T_i. Our approach is to "delay these questions" by keeping l in T_i, until l is part of a maximal pertinent sequence in T_j, $j > i$. Then we decide whether e can be added to G_p. In every tree T_i we again compute the maximal pertinent sequence of incoming edges of node $i+1$ in G_p, hence we call only these leaves pertinent. (Recall the definitions of section 3 for the pertinent, full, partial and empty nodes.) The leaves of edges $(k, i+1) \in G - G_p$ become not pertinent in T_i and will not be added in this step. We don't remove $(k, i+1) \in G - G_p$ from T_i, because maybe in some later PQ-tree $T_j, j > i$, $(k, i+1)$ can be added to G_p while preserving

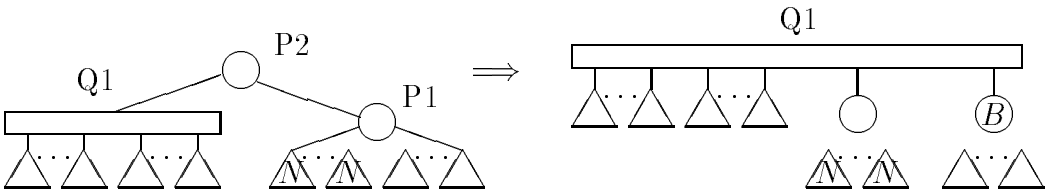


Figure 4: Partial nodes are bound to new places. N denotes a new pertinent node and B denotes a bound node.

problem of MAX_PLANARIZE1 is that it can bind partial nodes to new places in the PQ-tree, since some new pertinent leaves may be included in the preferred sequence. Inspect for this the instance, as given in figure 4.

There is a partial P-node P1, whose new pertinent descendant leaves become part of the preferred sequence, which consists of the preferred children of Q-node Q1. In PLANARIZE the pertinent children of P1 are removed from the PQ-tree. But adding P1 to the preferred sequence implies that P1 becomes a child of Q1, while in PLANARIZE P1 still was a child of P2. So T_i of PLANARIZE and T_i of MAX_PLANARIZE1 are not equivalent. Hence we cannot always form the maximal preferred sequence in some later step $j > i$. In some instances, however, the new pertinent children of P1 must be added to the preferred sequence to obtain a maximal planar subgraph.

case (b) Assume new full nodes $P_1, \ldots, P_k$ can be added to the maximal preferred sequence, without binding partial nodes to new places in the PQ-tree. Let $P_1, \ldots, P_k$ all have a set of empty leaves $L_1, \ldots, L_k$ in their frontier, which are not pertinent. Let the father R of $P_1, \ldots, P_k$ be the root of the pertinent subtree. When R has at most one partial preferred child, then at least one new full node, say P_i, can be made Type H. This implies that the corresponding set of descendant leaves L_i need not be removed from the PQ-tree. But when all leaves of L_i are removed from T_k in later steps in MAX_PLANARIZE1, while leaves of some other set L_j could be included in a maximal preferred sequence, we should make P_j Type H instead of making P_i Type H, to obtain a maximal planar subgraph G_p. Since in MAX_PLANARIZE1 this node P_i, with $|L_i|$ minimal is made Type H, this will not always lead to a maximal planar subgraph G_p.

case (c) Finally, as also mentioned in [9], MAX_PLANARIZE1 may not find a maximal planar subgraph when G_p is not biconnected. Inspect figure 5 for this problem.

L_2 is a new pertinent leaf and cannot be added to the maximal planar subgraph

maximal planar, when G_p is not biconnected. Moreover we show that Jayakumar's algorithm, which is a modification of an incorrect maximal planarization algorithm of Ozawa & Takahashi [12] is not correct, even when G_p is biconnected. We explain here in detail the problems of this algorithm to get some insight in the problems in terms of PQ-trees. We use the same examples to show the behaviour of our algorithm in section 5. We first give a brief overview of MAX_PLANARIZE1.

Let G_p be the biconnected planar subgraph of G obtained by the algorithm PLANARIZE. The idea to augment G_p to a maximal planar subgraph G'_p is as follows. Start with G and construct its chain of PQ-trees. After constructing a PQ-tree T_i, reducibility is obtained by deleting a minimum number of leaves representing the edges in E'_{i+1}. (Note that T_i will become reducible if all the leaves from the set E'_{i+1} are deleted from T_i.) This can be done by computing the $[w, h, a]$ number of the pertinent nodes in T_i. Let $T_i(G_p)$ denote the smallest subtree of T_i whose frontier contains all the pertinent leaves from G_p. Since we would like to include G_p in the final maximal planar subgraph, we take care that, during the reduction of T_i, no node in $T_i(G_p)$ is made Type A except its root. This ensures that the bottom-up reduction process proceeds at least up to the root of $T_i(G_p)$ and possibly beyond. While computing the $[w, h, a]$ numbers we ignore the presence of leaves from $G - G_p$, which we will call *empty* leaves. In the following, the empty leaves in T_i corresponding to the edges in E'_{i+1} will be called the *new pertinent leaves* of T_i and the other pertinent leaves of T_i (corresponding to the edges entering vertex $i+1$ in G_p) will be called *preferred leaves*.

Again a node is called full if its frontier has no empty leaf from G_p; it is empty if its frontier has only empty leaves from G_p; otherwise it is partial. We call node X a *preferred node* if it has some of the preferred leaves in its frontier. This procedure leads to a construction of a sequence, here called the *preferred sequence*, containing all preferred nodes and further only empty nodes and as much as possible new pertinent nodes. Hence if X is new pertinent, then then X may either be retained in the reducible T_i or X may be deleted along with all its descendants to make T_i reducible. The formulas for computing the $[w, h, a]$ numbers of the pertinent nodes are similar as in COMPUTE(T_i).

Having computed the $[w, h, a]$ numbers for the pertinent nodes in T_i, we can obtain a reducible T_i by traversing the pertinent subtree top-down from the pertinent root using the procedure DELETE_NODES. Processing the PQ-trees $T_2, T_3, \ldots, T_{n-2}$ using the different procedures described above, a maximal planar subgraph of the nonplanar graph G is obtained, according to theorem 5 of Jayakumar et al. [9], when G_p is biconnected. However, as we will show in the next three cases (a), (b) and (c), the algorithm is not correct. These examples are explained here, to use them in section 5 to describe the behaviour of our maximal planarization algorithm.

case (a) Call a node X in T_i *bound to a new place*, when in MAX_PLANARIZE1 the tree T_i is not equivalent to T_i of PLANARIZE, due to the node X, e.g., it has a different parent in MAX_PLANARIZE1 as in PLANARIZE. One important

Theorem 3.3 *The planar subgraph G_p produced by applying* PLANARIZE *on inputgraph $K_{m,n}$ is maximal planar.*

Proof: First we note that a planar graph with only faces with 4 sides has $2n-4$ edges, hence a maximal planar bipartite graph with $m+n$ vertices has $2(m+n)-4$ edges, because it only has faces with 4 sides. For $K_{m,n}$ we assume that $m \geq n$ and we number the vertices as follows: for $0 \leq i \leq n-1$ we give vertices on the side of the n vertices, number $2i+1$ and one vertex gets number $m+n$. The vertices on the other side obtain the remaining numbers. This is a legal st-numbering and when we run our algorithm, the PQ-tree in each step $2i, 1 < i < n$ is as shown in figure 3.

From this PQ-tree it follows that after DELETE_NODES in step $2i$ vertex $2i$ has only one lower-numbered neighbor and in step $2i+1$, vertex $2i+1$ has two lower-numbered neighbors. In step $i, i > 2n-2$, vertex i has only one lower-numbered neighbor, because the maximal consecutive sequences consist of only one element. Finally, vertex $m+n$ is connected with m lower-numbered vertices after the algorithm DELETE_NODES. Counting all this with the observation that vertex 1 has no lower-numbered neighbors, and vertex 2 just one, we obtain $0+1+3(n-2)+(m+n-(2n-1))+m = 2(m+n)-4$ edges in G_p. □

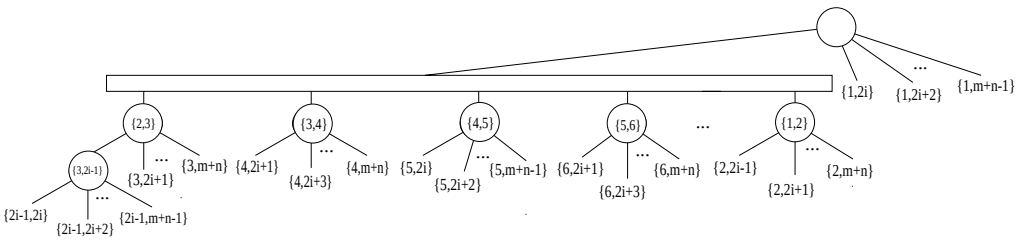


Figure 3: PQ-tree in step $2i$ with $K_{m,n}$ as input.

We will use this algorithm PLANARIZE as a basis for obtaining a maximal planar subgraph by the algorithm MAX_PLANARIZE2, described in section 5.

4 Jayakumar's Maximal Planarization Algorithm is not correct

For a given nonplanar graph G, let G_p be a planar spanning subgraph of G obtained by the algorithm PLANARIZE. In [9], Jayakumar et al. gave an $O(n^2)$ algorithm, here called MAX_PLANARIZE1, to add edges to G_p such that the augmented graph G'_p, $G'_p \supseteq G_p$, is maximal planar with respect to the inputgraph G, when G_p is biconnected. We will show indeed that the augmented planar subgraph is not necessarily

the $[w, h, a]$ numbers of all its pertinent children are computed and we can compute the $[w, h, a]$ number for X, using this information.

After computing the $[w, h, a]$ number for the pertinent root R of T_i, we can determine whether T_i is reducible or not. If the minimum of h and a is zero for R, then T_i is reducible, otherwise we make R Type H or A depending on which one of h and a is minimum, and make T_i reducible by deleting the corresponding pertinent leaves from T_i. The procedure which traverses the tree top-down and determines the Type for each pertinent node in T_i to obtain a reducible T_i will be denoted by DELETE_NODES(T_i). E'_{i+1} denotes the corresponding set of removed incoming edges of $i + 1$.

The algorithm PLANARIZE can now be described as follows:

```
PLANARIZE
  Construct the initial tree T_1 = T_1^*;
  for i := 1 to n - 1 do
    BUBBLE_UP(T_i);
    COMPUTE(T_i);
    if min{h, a} for the pertinent root R is not zero then
      make R Type H or A corresponding to the minimum of h and a;
      DELETE_NODES(T_i);
    replace all full nodes of T_i by a P-node X with all
    outgoing edges of node i + 1 appearing as children of X.
  od
```

Theorem 3.2 ([9]) *Algorithm* PLANARIZE *determines a planar spanning subgraph G_p of the nonplanar graph G in $O(n^2)$ time.*

Proof: For the proof that PLANARIZE determines a planar spanning subgraph G_p of G, the reader is referred to [9].

For the complexity bound it can be shown that the number of children of all the Q-nodes in T_i is at most n [9]. Since there are at most i P-nodes and n Q-nodes in every T_i, it follows that the amount of work in COMPUTE(T_i) for all the P- and Q-nodes is $O(n + i + indeg(i + 1))$, where $indeg(i + 1)$ is the number of edges entering vertex $i + 1$ in G. Summing up the work for all T_i's, we get the complexity of computing the $[w, h, a]$ numbers as $O(m + n^2) = O(n^2)$.

By a similar proof it follows that the complexity of determining and removing the sets E'_{i+1}, $2 \leq i \leq n - 1$ is $O(n^2)$. □

However, the algorithm PLANARIZE does **not** compute a maximal planar subgraph, and in [9] an example is given. In the case that G is a complete graph, it is shown in [10] that PLANARIZE results a maximal planar subgraph with $3n - 6$ edges. We are able to prove that the algorithm also works for the case of complete bipartite graphs.

Theorem 2.1 *The sum of the sizes of all the pertinent nodes in the PQ-trees* $T_1, T_2, \ldots, T_{n-1}$ *of a planar graph is* $O(m+n)$.

3 The Planarization Algorithm

In this section we discuss the basic principle of an approach for planarization, due to Ozawa & Takahashi [12] and also studied by Jayakumar et al. [9]. Following these papers, we classify the nodes of a PQ-tree according to their frontier as follows:

Type W: A node is said to by Type W, if its frontier consists of only empty leaves.

Type B: A node is said to be Type B, if its frontier consists of only full leaves.

Type H: A node X is said to be Type H if the subtree rooted at X can be arranged such that all the descendant pertinent leaves of X appear consecutively at either the left end or at the right end of the frontier.

Type A: A node X is said to be Type A if the subtree rooted at X can be arranged such that all the descendant pertinent leaves of X appear consecutively in the middle of the frontier with at least one non-pertinent leaf appearing at each end of the frontier.

The central concept of the planarization algorithm is stated in the following theorem of [9] which is essentially a reiteration of the principle on which PLANARITY_TEST is based.

Theorem 3.1 *A graph* G *is planar if and only if the pertinent roots of all subtrees in* $T_2, T_3, \ldots, T_{n-1}$ *of* G *are Type B, H or A.*

We call a PQ-tree *reducible* if its pertinent root is Type B, H or A; otherwise it is called *irreducible*. A graph G is planar iff all the $T_i's$ are reducible. If any T_i is irreducible, we can make it reducible by appropriately deleting some of the leaves in it. For a node X in an irreducible PQ-tree T_i, let the w-, h- and a-number be the minimum number of descendant leaves of X, which should be deleted from T_i such that X becomes Type W, H and A, respectively. We denote the tuple of numbers thus associated to a node by $[w, h, a]$. (Note that a partial node can not be made Type B, because we do not delete empty children.) When we have calculated these numbers for the root of the subtree of the pertinent nodes, we set the Type of the root according to the minimum of these numbers. If this minimum is not zero, we traverse the tree top-down and determine the Type of each pertinent node. Using this information, a decision about pertinent leaves can be made, which must be thrown away to make the tree reducible.

For this, we process T_i bottom-up by the algorithm COMPUTE(T_i) from the pertinent leaves to the pertinent root. When a pertinent node X is processed,

ponents in B_k. G, G_k, B_k and the corresponding PQ-tree are illustrated in figure 2.

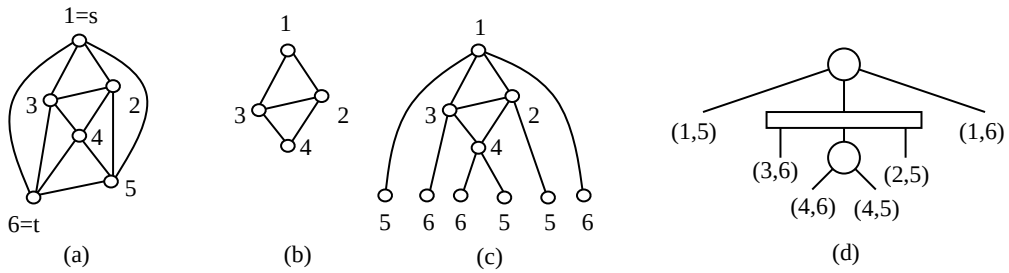


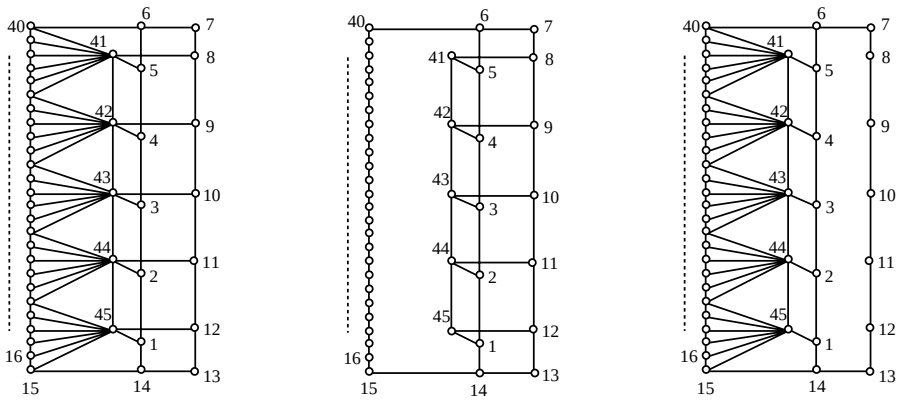
Figure 2: Example of G, G_k, B_k and corresponding PQ-tree (from [4]).

A few definitions are now in order. Let E_{k+1} denote the set of leaves in T_k which corresponds to the virtual vertex $k+1$. A node X in T_k is said to be *full* if all its descendant leaves are in E_{k+1}; X is said to be *empty* if none of its descendant leaves are in E_{k+1}; otherwise X is *partial*. If X is full or partial, then it is called a *pertinent node*. The frontier of T_k is the sequence of all the descendant leaves of T_k read from left to right. The *pertinent subtree* of T_k is the smallest connected subtree which contains all the leaves in E_{k+1}. The root of the pertinent subtree is called the *pertinent root*. Two PQ-trees are considered *equivalent* if one can be obtained from the other by performing one or more of the following type of operations.

- Reversing the order of the children of a Q-node.
- Permuting the children of a P-node.

It can be shown [2] that B'_k exists if and only if T_k can be converted into an equivalent PQ-tree T'_k such that all the pertinent leaves appear consecutively in the frontier of T'_k. Booth & Lueker have defined a set of patterns and replacements using which T'_k can be reduced into a PQ-tree T^*_k in which all the pertinent leaves appear as children of a single node. The reduction process consists of two phases. In the first phase, called the BUBBLE_UP phase, the pertinent subtree is identified. In the second phase, called the REDUCTION phase, pattern matching and corresponding replacements are carried out using the reversing and permutation operations.

To construct T_{k+1} from T_k, we first reduce T_k to T^*_k and then replace all the leaves corresponding to virtual edges of vertex $k+1$ by a P-node whose children are all leaves, corresponding to outgoing edges of vertex $k+1$ in G. The algorithm of Booth & Lueker, which we will refer to as PLANARITY_TEST, starts with T_1 and constructs the sequence of PQ-trees $T_1, T_2, \ldots$. If the graph G is planar, then the algorithm terminates after constructing T_{n-1}; otherwise, it terminates after detecting the impossibility of reducing some T_k into T^*_k. The crucial result in the complexity analysis of PLANARITY_TEST is stated in the following theorem [2]:



(a) A nonplanar graph with given st−numbering or dfs−numbering

(b) The resulting maximal planar subgraph when applying Cai's or Di Battista's algorithm.

(c) The resulting maximal planar subgraph when using PQ−trees.

Figure 1: Example to show that our algorithm deletes only $O(n^{0.5})$ edges and the other algorithms delete $O(n)$ edges.

2 Planarity Testing Using PQ-trees

Let $G = (V, E)$ be a graph with n nodes and m edges. We assume that G is simple, that is, has no multiple edges or loops. A graph is *planar* if it can be embedded in the plane without any crossing edges. A graph G is planar if and only if the biconnected components of G are planar [11]. We henceforth assume that G is biconnected. The planarity testing algorithm of Lempel, Even & Cederbaum first labels in linear time the vertices of G as 1,2,..., n, using what is called an $st - numbering$. An st-numbering is a numbering of the vertices of G by 1, 2, ..., n such that the vertices 1 and n are necessarily adjacent and each j of the other vertices is adjacent to two vertices i and k satisfying $i < j < k$. Let $G_k = (V_k, E_k)$ be the subgraph of G induced on the vertices 1, 2, ..., k. If $k < n$ then there must exist an edge of G with one end in V_k and the other in $V - V_k$. Let G'_k be the graph formed by adding to G_k all these edges. These edges are called *virtual edges*, and their ends in $V - V_k$ are called *virtual vertices* and labelled as their counterparts in G, but they are kept separate; i.e., there may be several virtual vertices with the same label, each with exactly one entering edge. Let B_k (the *bush form*) be an embedding of G'_k such that all the virtual vertices are placed on the outer face. It can be shown [11] that the st-graph G is planar if and only if for every B_k, $2 \leq k \leq n-2$, there exists a planar drawing B'_k isomorphic to B_k such that in B'_k all the virtual vertices labeled $k+1$ appear consecutively.

The *PQ-tree* T_k corresponding to the bush form B_k consists of three types of vertices: (i) *Leaves* in T_k represent virtual edges in B_k, (ii) *P-nodes* in T_k represent cutvertices in B_k, and (iii) *Q-nodes* of T_k represent the maximal biconnected com-

edges whose deletion from a nonplanar graph gives a planar subgraph, is an NP-complete problem [7]. Therefore, we restrict our attention to computing a *maximal planar subgraph* G' of G, that is, a subgraph G' such that for all edges $e \in G - G'$ the addition of e to G' destroys the planarity. The first algorithms for this problem work with an $O(mn)$ worst-case time bound [5, 12], which can also be achieved trivially by starting with one edge and checking for every subsequent edge, whether its addition to the graph preserves the planarity (by employing a linear-time planarity tester). Recently Cai, Han & Tarjan [3] described an $O(m \log n)$ maximal planarization algorithm, based on the Hopcroft-Tarjan planarity testing algorithm. Di Battista & Tamassia described an algorithm to check in $O(\log n)$ amortized time whether adding an edge to the graph preserves the planarity, which yields an $O(m \log n)$ time maximal planarization algorithm as well.

Up to now, no general maximal planarization algorithm based on PQ-trees was known, having a complexity better than $O(mn)$. Jayakumar, Thulasiraman & Swamy [9] presented an $O(n^2)$ planarization algorithm for a special class of graphs based on PQ-trees but, as will be shown in this paper, this algorithm is not correct. In this paper we correct this algorithm to a maximal planarization algorithm, which works for all instances. The algorithm can be implemented to work in $O(n^2)$ time, which is better than the time bound of $O(m \log n)$ for dense graphs. Moreover, instead of testing for every edge whether or not it can be added without destroying the planarity, it calculates for every vertex the minimum number of edges which must be deleted to preserve planarity. This is the main reason why we employ the PQ-tree approach.

This algorithm not only constructs a maximal planar subgraph in an efficient way, but it really tries to minimize the number of deleted edges. In figure 1 an example is given where from a nonplanar graph with $k^2 + 3k + 5$ vertices and $2k^2 + 6k + 5$ edges, $k(k+1)$ edges are deleted by the algorithms of Cai et al. [3] and Di Battista & Tamassia [6], while only k edges are deleted by our algorithm (in figure 1, $k = 5$). Due to the computation of the minimum number of deleted edges per added vertex, our algorithm seems to have such a better behaviour than the other planarization algorithms in general. In section 6 some experimental results are given.

The paper is organized as follows. In section 2 we describe the PQ-tree data-structure and its implementation for testing planarity of graphs. In section 3 we revise this algorithm to compute a particular planar subgraph G' of a nonplanar graph G. In section 4 we give three totally different counterexamples to show that the maximal planarization algorithm of Jayakumar et al. [9] is not correct. In section 5 we present our maximal planarization algorithm. Section 6 contains some experimental results and some concluding remarks.

An $O(n^2)$ Maximal Planarization Algorithm based on PQ-trees*

Goos Kant
Dept. of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands

Abstract

In this paper we investigate the problem how to delete a number of edges from a nonplanar graph G such that the resulting graph G' is maximal planar, i.e., such that we cannot add an edge $e \in G - G'$ to G' without destroying planarity. Actually, our algorithm is a corrected and more generalized version of the maximal planarization algorithm of Jayakumar et al., based on the planarity testing algorithm using PQ-trees of Booth & Lueker. Our algorithm can be implemented to run in $O(n^2)$ time.

1 Introduction

Planarity testing is the problem of determining whether a given graph G with n vertices and m edges is planar or not. It has many applications, e.g. in the design of VLSI circuits, determining the isomorphism of chemical structures and in various problems dealing with the readability of diagrams. Two planarity testing algorithms of different types are known, both having a linear time complexity. One is called a "path addition" algorithm and the other is called a "vertex addition" algorithm. These terms refer to the principles used in the algorithms. The path addition algorithm is originally due to Auslander & Parter [1] and a linear time implementation was developed by Hopcroft & Tarjan [8]. In this paper, we investigate the vertex addition algorithm, which was presented first by Lempel, Even & Cederbaum [11], and improved later into a linear algorithm by Booth & Lueker [2], using a novel data structure called the *PQ-tree*.

If a graph is not planar, then we may want to delete some edges to obtain a planar subgraph (e.g. for embedding purposes). Finding the minimum number of

*This work was supported by the ESPRIT Basic Research Actions of the EC under contract No. 3075 (project ALCOM).

ISSN: 0924–3275

An $O(n^2)$ Maximal Planarization Algorithm based on PQ-trees

Goos Kant
Dept. of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands

Technical Report RUU-CS-92-03
January 1992

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

An $O(n^2)$ Maximal Planarization Algorithm based on PQ-trees

Goos Kant
Dept. of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands

RUU-CS-92-03
January 1992

Utrecht University
Department of Computer Science
Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : + 31 - 30 - 531454