Pitfalls of Using PQ-Trees in Automatic Graph Drawing[*]

Michael Jünger[1], Sebastian Leipert[2] and Petra Mutzel[3]

[1] Institut für Informatik, Universität zu Köln, mjuenger@informatik.uni-koeln.de
[2] Institut für Informatik, Universität zu Köln, leipert@informatik.uni-koeln.de
[3] Max-Planck-Institut für Informatik, Saarbrücken, mutzel@mpi-sb.mpg.de

Abstract. A number of erroneous attempts involving PQ-trees in the context of automatic graph drawing algorithms have been presented in the literature in recent years. In order to prevent future research from constructing algorithms with similar errors we point out some of the major mistakes.
In particular, we examine erroneous usage of the PQ-tree data structure in algorithms for computing maximal planar subgraphs and an algorithm for testing leveled planarity of leveled directed acyclic graphs with several sources and sinks.

1 Introduction

A PQ-tree is a powerful data structure that represents the permutations of a finite set in which the members of specified subsets occur consecutively, and in which updates require linear time. This data structure has been introduced by Booth and Lueker (1976) to solve the problem of testing for the consecutive ones property. The most well known applications of PQ-trees in automatic graph drawing are planarity testing (see Lempel *et al.* (1967), Booth and Lueker (1976)) and embedding (see Chiba *et al.* (1985)). Therefore PQ-trees have become standard tools in automatic graph drawing systems.

Other attempts to use algorithms based on PQ-trees for automatic graph drawing problems have not been successful. One well known example is the computation of maximal planar subgraphs. Given a simple, connected graph $G = (V, E)$ with n vertices and m edges, a planar subgraph G' of G is a *maximal planar* subgraph, if for all edges $e \in G - G'$ the addition of e to G' destroys planarity. Several efforts have been made in the literature to solve the problem with PQ-trees following a certain strategy that was first presented by Ozawa and Takahashi (1981). They describe an $O(nm)$ algorithm using PQ-tree techniques based on the vertex addition algorithm of Lempel *et al.* (1967). Jayakumar *et al.* (1986) show that in general this algorithm does not determine a maximal planar subgraph. Moreover, the resulting planar subgraph may not

[*] Partially supported by DFG-Grant Ju204/7-2, Forschungsschwerpunkt "Effiziente Algorithmen für diskrete Probleme und ihre Anwendungen" and ESPRIT Long Term Research Project Nr. 20244 (ALCOM-IT)

even contain all vertices. Jayakumar *et al.* (1989) present an algorithm called PLANARIZE that computes a spanning planar subgraph G_p of G in $O(n^2)$ time. Furthermore, they present an algorithm called MAX-PLANARIZE that augments G_p to a subgraph G' of G by adding additional edges in $O(n^2)$ time. They claim that G' is a maximal planar subgraph of G if G_p (the result of phase 1 of the two phase algorithm) turns out to be biconnected. Kant (1992) shows that this algorithm is incorrect, and suggests a modification of the second phase of the algorithm that augments G_p to a maximal planar subgraph of G, even if G_p is not biconnected, maintaining $O(n^2)$ time requirement. In Jünger *et al.* (1996) we show that this modification is not correct either. Here we point out a substantial flaw in both the original and the modified two phase algorithm that was not detected previously. This is the subject of section 2.

PQ-trees have also been proposed by Heath and Pemmaraju (1996a, 1996b) to test leveled planarity of leveled directed acyclic graphs with several sources and sinks. In section 3 we show why this application leads to an incorrect algorithm as well. Since this "algorithm" is the only attempt to prove the polynomial time complexity in the literature, the complexity status of leveled planarity testing is still open. Only in the special case in which there is only one source (or only one sink) the algorithm is correct and implies linear time solvability, but this has already been shown previously by Di Battista and Nardelli (1988).

2 Case study: maximal planar subgraphs

2.1 PQ-trees for planarity testing

Let $G = (V, E)$ be a simple graph with n vertices and m edges. A graph is *planar*, if it can be embedded in the plane without any edge crossings. A graph is obviously planar, if and only if its biconnected components are planar. We therefore assume that G is biconnected. The planarity testing algorithm of Lempel *et al.* (1967) first labels the vertices of G as $1, 2. \dots, n$ using an st-numbering (see Even and Tarjan (1976)). A numbering of the vertices of G by $1, 2. \dots, n$ is an *st-numbering*, if the vertices "1" and "n" are adjacent and each other vertex j is adjacent to two vertices i and k such that $i < j < k$. The vertex 1 is denoted by s and the vertex n is denoted by t. The st-numbering induces an orientation of the graph, in which every edge is directed from the incident vertex with the higher st-number towards the incident vertex with the lower st-number. From now on we refer to the vertices of G by their st-numbers and call an edge (u, v), with $v < u$, *incoming* edge of v and *outgoing* edge of u.

For $1 \leq k \leq n$, let G_k denote the subgraph of G induced by the vertex set $V_k := \{1, 2, \dots, k\}$. The graph G'_k arises from G_k as follows: For each edge $e = (u, v)$, where $v \in V_k$ and $u \in V \setminus V_k$, we introduce a virtual vertex u_e with label u and a virtual edge (u_e, v). Let B_k be a planar embedding of G'_k such that all virtual vertices are placed on the outer face. Then, B_k is called a *bush form*. It has been shown by Lempel *et al.* (1967) that G is planar, if and only if for every B_k, $k = 1, 2, \dots, n-1$, there exists a bush form B'_k isomorphic to B_k, such that all virtual vertices in B'_k labeled $k + 1$ appear consecutively.

The *PQ-tree* T_k corresponding to the bush form B_k is a rooted ordered tree that consists of three types of nodes:

1. Leaves in T_k represent virtual edges in B_k.
2. *P-nodes* in T_k represent cutvertices in B_k.
3. *Q-nodes* represent maximal biconnected components in B_k.

The *frontier* of a *PQ*-tree is the sequence of all leaves of T_k read from left to right. The frontier of a node X, denoted by frontier(X), is the sequence of its descendant leaves read from left to right.

Let E_{k+1} denote the set of leaves in T_k that correspond to the virtual vertices labeled $k+1$. A node X is called *full*, if all leaves in its frontier are in E_{k+1}. A node X is *empty*, if its frontier does not contain any leaf of E_{k+1}. Otherwise, X is called *partial*. A node is called *pertinent*, if it is full or partial. The *pertinent subtree* is the smallest connected subtree that contains all leaves of E_{k+1} in its frontier. The root of the pertinent subtree is called *pertinent root*. Two *PQ*-trees are *equivalent*, if one can be obtained from the other by one or more of the following operations:

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

These operations are called *equivalence transformations* and describe *equivalence classes* on the set of all *PQ*-trees. An equivalence class of *PQ*-trees corresponds to a class of permutations called the *permissible permutations.*

It has been shown by Booth and Lueker (1976) that B'_k exists if and only if T_k can be converted into an equivalent *PQ*-tree T'_k such that all pertinent leaves appear consecutively in the frontier of T'_k. Booth and Lueker (1976) have defined a set of patterns and replacements called *templates* that can be used to reduce the *PQ*-tree such that the leaves corresponding to edges of the set E_{k+1} appear consecutively in all permissible permutations. To construct T_{k+1} from T_k they first reduce T_k by use of the templates and then replace all leaves corresponding to virtual edges incident to vertices labeled $k+1$ by a P-node, whose children are the leaves corresponding to the incoming edges of the vertex $k+1$ in G.

The planarity testing algorithm now starts with T_1 and constructs a sequence of *PQ*-trees $T_1, T_2, \ldots$. If the graph is planar, the algorithm terminates after constructing T_{n-1}. Otherwise it terminates after detecting the impossibility of reducing some T_k, $1 \leq k < n$.

2.2 Principle of an approach for planarization

The basic idea of a planarization algorithm using *PQ*-trees presented by Jayakumar *et al.* (1989) is to construct the sequence of *PQ*-trees $T_1, T_2, \ldots, T_{n-1}$ by deleting an appropriate number of pertinent leaves every time the reduction fails, such that the resulting *PQ*-tree becomes reducible. In every step of the algorithm PLANARIZE, a maximal consecutive sequence of pertinent leaves is computed by using a $[w, h, a]$-*numbering* (see Jayakumar *et al.* (1989)). All pertinent leaves that are not adjacent to the maximal pertinent sequence are removed from the PQ-tree in order to make it reducible. Hence the edges corresponding to the leaves are removed from G and the resulting graph G_p is planar.

It has been shown by Jayakumar *et al.* (1989) that the graph G_p computed by PLANARIZE is not necessarily maximal planar. The authors therefore suggest to apply a second phase MAX-PLANARIZE, also based on PQ-trees. Knowing which edges have been removed from G to construct G_p, edges from $G - G_p$ are added back to G_p in the second phase without destroying planarity.

During the reduction of a vertex v, there may exist nonpertinent leaves that are in all permissible permutations of the PQ-tree T_{v-1} between a pertinent leaf l_v and its maximal pertinent sequence. This maximal pertinent sequence has been determined with the help of the $[w, h, a]$-numbering. In order to make the tree T_{v-1} reducible, the leaf l_v is removed from the tree and the corresponding edge is removed from the graph G, guaranteeing that the subgraph G_p will be planar. However, it may occur that the nonpertinent leaves that are positioned between l_v and its maximal pertinent sequence in T_{v-1}, are removed as well from a tree T_k, $v \leq k < n$, in order to obtain reducibility. Therefore, there is no need to remove the edge corresponding to l_v from the graph G.

In order to find leaves such as l_v, Jayakumar *et al.* (1989) use the algorithm MAX-PLANARIZE. In step i, both PLANARIZE as well as MAX-PLANARIZE reduce the same vertex i. The difference between the PQ-trees in the two algorithms is, according to the authors, that all leaves that have been deleted in PLANARIZE are ignored in MAX-PLANARIZE from the moment they are introduced into the tree until they get pertinent. This causes the nonpertinent leaves between the pertinent leaf l_v and its maximal pertinent sequence to be ignored. Hence l_v is adjacent to its maximal pertinent sequence and the corresponding edge can be added back to G_p, while the leaves between l_v and the maximal pertinent sequence are removed from the PQ-tree.

2.3 On the incorrectness of the algorithm

While some incorrect details of the approach of Jayakumar *et al.* have been described in a technical report by Kant (1992), who attempted to correct the algorithm, a major problem has not been detected.

Jayakumar *et al.* assume that the maximal planar subgraph G_p is biconnected for the correct application of the Lempel-Even-Cederbaum algorithm. Furthermore, as they have stated correctly, this is necessary in order to have an st-numbering. Nevertheless, the PQ-trees in MAX-PLANARIZE are constructed according to the st-numbering that was computed for the graph G.

As a matter of fact, the st-numbering of G does not imply an st-numbering of any subgraph G_p even if the subgraph G_p is biconnected. This results in two problems, of which one is crucial and cannot be dealt with even by the ideas described by Kant (1992).

Both problems are based on the fact that during the application of PLANARIZE for some vertices of V all incoming edges may be deleted from the graph while the resulting graph G_p stays biconnected. In this abstract, we consider only the crucial problem. The other problem is described in detail by Jünger *et al.* (1996).

The planarization algorithm of Jayakumar *et al.* (1989) does not obey an important invariant implied by the following lemma, shown by Even (1979).

Lemma 2.1 *Let $G = (V, E)$ be a planar graph with an st-numbering and let $1 \leq k \leq n$. If the edge (t, s) is drawn on the boundary of the outer face in an embedding of G, then all vertices and edges of $G - G_k$ are drawn in the outer face of the plane subgraph G_k of G.*

This result allowed Lempel *et al.* (1967) to transform the problem of planarity testing to the construction of a sequence of bush forms B_k, $1 \leq k \leq n$. For a planar graph G, edges and vertices that have not been introduced into the current subgraph G_k are always embedded into the outer face of G_k.

The approach of Jayakumar *et al.* (1989) does not obey this invariant in the second phase. There exist edges that have to be embedded into an inner face of some G_k, even if (t, s) is drawn on the outer face. Due to the above lemma, the correction step MAX-PLANARIZE only considers edges for reintroduction into the planar subgraph G_p that are on the outer face of the current graph G_k. Since the numbering that is used to determine the order in which the vertices are reduced does not correspond to an st-numbering of G_p in general, the algorithm of Jayakumar *et al.* (1989) ignores edges that have to be added into an inner face of the embedding of a current graph G_k. This fact is fatal, as we are about to show now.

In Fig. 1, a part of a bush form B_{k-1}, $1 < k \leq n$ of a graph G is shown. The virtual vertices corresponding to the vertex k are labeled $k_1, k_2, \ldots, k_5$ and all other virtual vertices are left unlabeled. The corresponding part of the PQ-tree is shown in Fig. 2. Obviously, there do not exist any reversions or permutations such that the virtual vertices of k occupy consecutive positions. Hence, the graph G is not planar. Applying the $[w, h, a]$-numbering of Jayakumar *et al.* (1989) allows us to delete the virtual vertex k_5 and to reduce the other four vertices k_1, k_2, k_3, k_4. The resulting bush form B_k is planar and the relevant part is shown in Fig. 3. Figure 4 shows the corresponding part of the PQ-tree. Assume now that all descendants of k have to be removed from the PQ-tree in a later step. Hence all incoming edges incident on k are removed from the tree. Now assume further that there exists a path $v_1, v_2, \ldots, v_l$ in G_p such that

- for all i, j, $1 \leq i < j \leq l$ the inequality $v_i < v_j$ holds,
- the edge (v_2, v_1) corresponds to one of the virtual edges that are between the leaf k_5 and the maximal pertinent sequence k_1, k_2, k_3, k_4 in all PQ-trees equivalent to T_{k-1},
- $v_l = t$.

This path guarantees that all outgoing edges of the vertex k cannot be embedded into the outer face of the embedding of B_{k-1} without crossing an edge on this path. Hence the edge e_{k_5} corresponding to the leaf k_5 is not considered

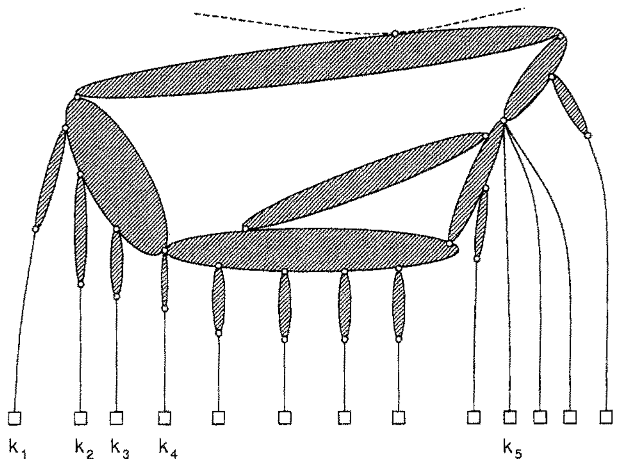


Fig. 1. Part of a bush form B_{k-1}

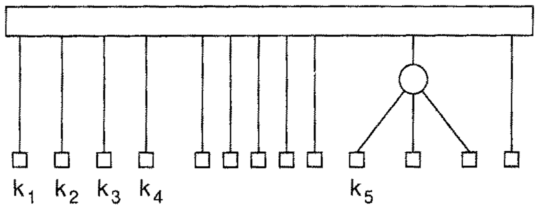


Fig. 2. Part of a PQ-tree corresponding to bush form B_{k-1}

by the algorithm MAX-PLANARIZE as being an edge that does not destroy planarity. Therefore, e_{k_5} is not added back to the planar subgraph G_p.

Nevertheless adding the edge e_{k_5} to G_p may not destroy planarity of G_p as is shown in our example in Fig. 5. Since all incoming edges of the vertex k have been deleted by PLANARIZE and are not added back by MAX-PLANARIZE, it may be possible to swap the vertex k into an inner face of the embedding of B_k such that the virtual vertex k_5 can be identified with k and the edge e_{k_5} is embedded into the bush form B_k without destroying planarity.

Therefore, the strategy of using PQ-trees presented by Jayakumar *et al.* (1989) does not compute a maximal planar subgraph in general. Furthermore, we point out that the same problem holds for the modified version of this algorithm, presented by Kant (1992). This version follows a similar strategy of computing a spanning planar subgraph G_p using PLANARIZE and then adding edges that do not destroy planarity in a second phase. The order of reductions that is used to insert vertices into existing bush forms is the same as the one implied by the st-numbering on G. Hence this approach is not able to compute a maximal planar subgraph for the same reason.

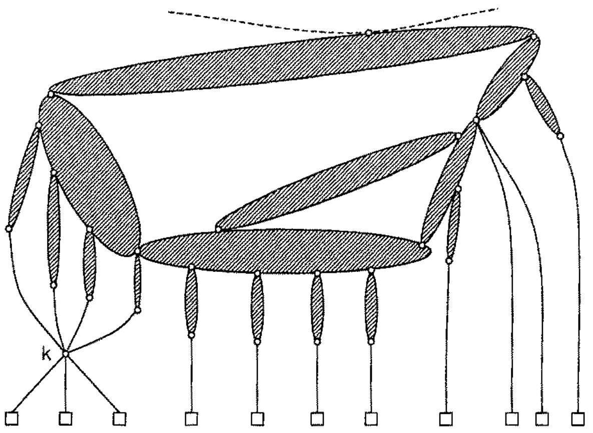


Fig. 3. Part of a bush form B_k

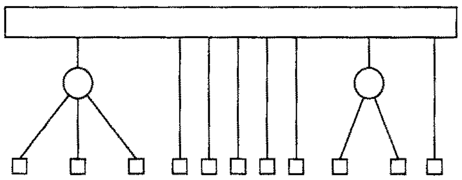


Fig. 4. Part of a PQ-tree corresponding to bush form B_k

Summarizing, we state the following lemma that has been shown in the discussion above.

Lemma 2.2 *Let $G = (V, E)$ be a nonplanar graph. Let $G_p = (V, E_p)$, $E_p \subseteq E$, be a planar subgraph of G, such that G_p was obtained from G by*

1. *computing an st-numbering for all vertices and*
2. *applying the algorithm of Lempel* et al. *(1967) constructing a sequence of bush forms B_k, $1 \leq k \leq n$, by embedding a maximal number of outgoing edges of a vertex k, $1 < k \leq n$, in the outer face of B_{k-1} without crossings, deleting all other outgoing edges of k.*

Let $G'_p = (V, E'_p)$, be a planar subgraph of G such that

1. *$E_p \subseteq E'_p \subseteq E$,*
2. *the graph G'_p is computed by constructing a sequence of bush forms B'_k, $1 \leq k \leq n$, based on the st-numbering used for determining G_p, and possibly embedding outgoing edges $e \in E \setminus E_p$ of every vertex k, $1 < k \leq n$, without crossings in the outer face of B_{k-1}.*

Then the subgraph G'_p is not necessarily maximal planar.

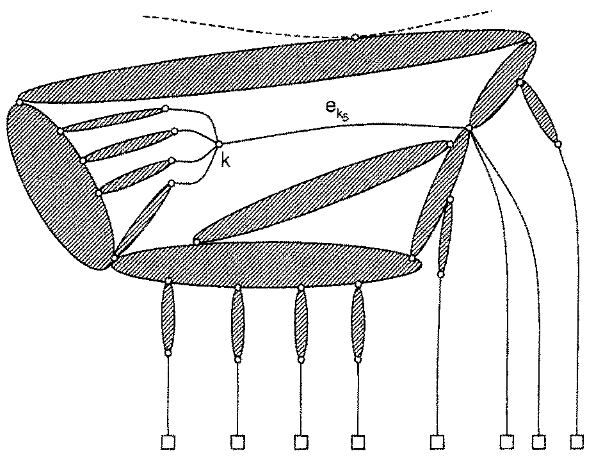


Fig. 5. Part of a bush form B_k with e_{k_5} embedded

Considering a computation of an *st*-numbering for the planar subgraph G_p in order to augment G_p to a maximal planar subgraph of G and then construct a sequence of bush forms B'_k, $1 \leq k \leq n$, is aggravated by the fact that the graph G_p is not biconnected in general. Furthermore, the difference between the bush forms of the first phase and the second phase may result in the deletion of the edges of G_p as soon as edges of $E \setminus E_p$ are added to G_p.

3 Case study: leveled planarity testing

3.1 Principle of an approach for recognizing leveled planar dags

Let $G = (V, E)$ be a directed acyclic graph. A leveling of G is a function $lev : V \to \mathbb{Z}$ mapping the nodes of G to integers such that $lev(v) = lev(u) + 1$ for all $(u, v) \in E$. G is called a *leveled dag* if it has a leveling. If $lev(v) = j$, then v is a *level-j vertex.* Let $V_j = lev^{-1}(j)$ denote the set of level-j vertices. Each V_j is a *level* of G.

For the rest of this section, we consider G to be a leveled dag with $m \in \mathbb{N}$ levels. An embedding of G in the plane is called *leveled* if the vertices of every V_j, $1 \leq j \leq m$, are placed on a horizontal line $l_j = \{(x, m - j) \mid x \in \mathbb{R}\}$, and every edge $(u, v) \in E$, $u \in V_j$, $v \in V_{j+1}$ is drawn as straight line segment between the lines l_j and l_{j+1}. A leveled embedding of G is called *leveled planar* if no two edges cross except at common endpoints. A leveled dag is leveled planar, if it has a leveled planar embedding. The dag G is obviously leveled planar, if all its components are leveled planar. We therefore assume that G is connected.

A leveled embedding of G determines for every V_j, $1 \leq j \leq m$, a total order $\leq_j$ of the vertices of V_j, given by the left to right order of the nodes on l_j. In order to test whether a leveled embedding of G is leveled planar, it is sufficient to find an order of the vertices of every set V_j, $1 \leq j < m$, such that for every pair of edges $(u_1, v_1), (u_2, v_2) \in E$ with $lev(u_1) = lev(u_2) = j$ and $u_1 \leq_j u_2$ it follows that $v_1 \leq_{j+1} v_2$. Apparently, the ordering $\leq_j$, $1 \leq j \leq m$, describes a permutation of the vertices of V_j. Let G_j denote the subgraph of G, induced by $V_1 \cup V_2 \cup \ldots \cup V_j$. Unlike G, G_j is not necessarily connected.

The basic idea of the leveled planarity testing algorithm presented by Heath and Pemmaraju Heath and Pemmaraju (1996a, 1996b) is to perform a top-down sweep processing the levels in the order $V_1, V_2, \ldots, V_m$ computing for every level V_j, $1 \leq j \leq m$, a set of permutations of the vertices of V_j that appear in some leveled planar embedding of G_j. In case that the set of permutations for G_m is not empty, the graph $G = G_m$ is obviously leveled planar.

As long as the graph G_j is connected for some $j \in \{1, 2, 3, \ldots, m\}$ standard PQ-tree techniques similar to the ones used in the planarity test can be applied in order to determine the required set of permutations (see Di Battista and Nardelli (1988)). In case that G_j, $1 \leq j < m$, consists of more than one connected component, Heath and Pemmaraju suggest to use a PQ-tree for every component and formulate a set of rules of how to merge components F_1 and F_2, respectively their corresponding PQ-trees T_1 and T_2, if F_1 and F_2 both are adjacent to some vertex $v \in V_{j+1}$.

The authors first reduce the pertinent leaves of T_1 and T_2 corresponding to the vertex v. After successfully performing the reduction, the consecutive sequence of pertinent leaves is replaced by a single pertinent representative in both T_1 and T_2. Going up one of the trees T_i, $i \in \{1, 2\}$, from its pertinent representative, an appropriate position is searched, allowing the tree T_j, $j \neq i$ to be placed into T_i. After successfully performing this step the resulting tree T' has two pertinent leaves corresponding to the vertex v, which again are reduced. If any of the steps fails, Heath and Pemmaraju state that the graph G is not leveled planar.

Merging two PQ-trees T_1 and T_2 corresponds to merging the two components F_1 and F_2 and is accomplished using certain informations that are stored at the nodes of the PQ-trees. For any subset S of the set of vertices in V_j, $1 \leq j \leq m$, that belong to a component F, define $\mathrm{ML}(S)$ to be the greatest $d \leq j$ such that $V_d, V_{d+1}, \ldots, V_j$ induces a dag in which all nodes of S occur in the same connected component. For a Q-node q in the corresponding PQ-tree T_F with ordered children $r_1, r_2, \ldots, r_t$ maintain in node q integers denoted $\mathrm{ML}(r_i, r_{i+1})$, where $1 \leq i < t$, satisfying $\mathrm{ML}(r_i, r_{i+1}) = \mathrm{ML}(\mathrm{frontier}(r_i) \cup \mathrm{frontier}(r_{i+1}))$. For a P-node p maintain in p a single integer denoted $\mathrm{ML}(p)$ that satisfies $\mathrm{ML}(p) = \mathrm{ML}(\mathrm{frontier}(p))$. Furthermore define $\mathrm{LL}(F)$ to be the smallest d such that F contains a vertex in V_d and maintain this integer at the root of the corresponding PQ-tree.

Using these LL- and ML-values, Heath and Pemmaraju (1996a, 1996b) describe a set of rules how to connect two PQ-trees claiming that the pertinent leaves of the new tree T' are reducible if and only if the corresponding component F' is leveled planar.

3.2 On the incorrectness of the algorithm

Within the merge phase, pertinent leaves are reduced pairwise in any given order. This includes the pairwise reduction of pertinent leaves of different components as well. Hence, components that have pertinent leaves of the same vertex in their frontier, are merged in an arbitrary order.

Consider four different components F_1, F_2, F_3, F_4 and their corresponding PQ-trees T_1, T_2, T_3, T_4 each having at least one pertinent leaf corresponding to some level-j vertex k. For simplicity, assume that the pertinent leaves of every component appear consecutively in all permutations on one side of their PQ-trees and assume further that the smallest common ancestor of the pertinent leaves and some other leaves is a Q-node. In Fig. 6 such a component F_i, $i \in \{1,2,3,4\}$, and its corresponding PQ-tree T_i, $i \in \{1,2,3,4\}$, is shown. The number c_i, $i \in \{1,2,3,4\}$, depicts the ML-value between the leftmost pertinent leaf and the frontier of its left neighbor. We have marked all pertinent leaves with a k for simplicity.

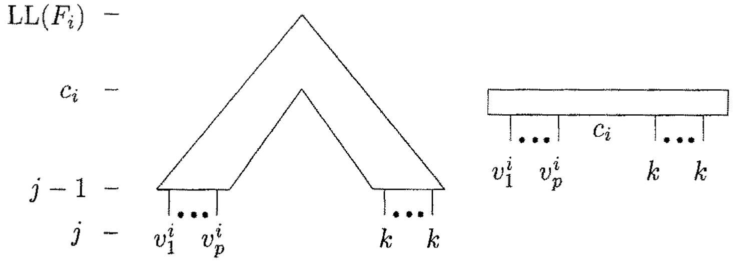


Fig. 6. Component F_i and its corresponding PQ-tree T_i. On the left side of F_i, some levels of F_i are indicated. The value c_i is equal to $\mathrm{ML}(\{v_p^i, k\})$.

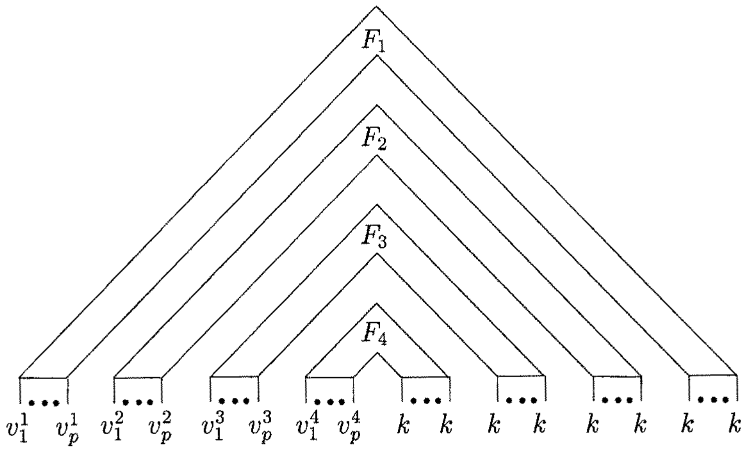
Assuming that the following condition,

$$LL(F_1) \leq c_1 < LL(F_2) \leq c_2 < LL(F_3) \leq c_3 < LL(F_4) \leq c_4$$

on the ML- and LL-values of the components holds, it is possible to merge all four components into one component such that the pertinent leaves form a consecutive sequence. Figure 7 shows the four components, indicating how the components can be merged so that a reduction of the pertinent leaves becomes possible.

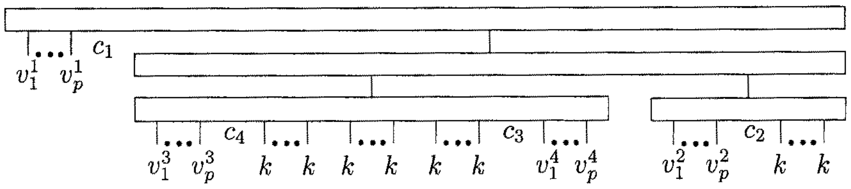
Consider the following merge operations on the components F_1, F_2, F_3, F_4 and their corresponding PQ-trees:

1. merge F_1 and F_4 into component F',
2. merge F' and F_3 into component F'',
3. merge F'' and F_2 into component F'''.



F_1
F_2
F_3
F_4
v_1^1 v_p^1 v_1^2 v_p^2 v_1^3 v_p^3 v_1^4 v_p^4 k k k k k k k k

Fig. 7. Possible leveled planar arrangement of the components F_1, F_2, F_3, F_4.



c_1
v_1^1 v_p^1
c_4
c_3
c_2
v_1^3 v_p^3 k k k k k k v_1^4 v_p^4 v_1^2 v_p^2 k k

Fig. 8. *PQ*-tree T''' whose pertinent leaves depicted by k are not reducible.

The resulting PQ-tree T''' corresponding to F''' is shown in Fig. 8. Obviously, the pertinent leaves do not form a consecutive sequence in any permissible permutation of the PQ-tree. Hence the algorithm presented by Heath and Pemmaraju (1996a, 1996b) states leveled nonplanarity although the graph may be leveled planar.

As a matter of fact, the order of merging the components is important for testing a leveled dag. Moreover it is easy to see, that using different orderings while merging three or more components results in different equivalence classes of PQ-trees. So even if every order of merging PQ-trees with pertinent leaves results in a reducible PQ-tree, a PQ-tree may be constructed such that the leaves of some vertex l, $lev(l) > j$ are not reducible, although the graph G is leveled planar. Hence the algorithm presented by Heath and Pemmaraju (1996a, 1996b) may state incorrectly the leveled nonplanarity of a leveled planar graph.

References

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

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

Di Battista, G. and Nardelli, E. (1988). Hierarchies and planarity theory. *IEEE Transactions on systems, man, and cybernetics*, **18**(6), 1035–1046.

Even, S. (1979). *Graph Algorithms*. Computer Science Press, Potomac, Maryland.

Even, S. and Tarjan, R. E. (1976). Computing an st-numbering. *Theoretical Computer Science*, **2**, 339–344.

Heath, L. and Pemmaraju, S. (1996a). Recognizing leveled-planar dags in linear time. In F. J. Brandenburg, editor, *Proc. Graph Drawing '95*, volume 1027 of *Lecture Notes in Computer Science*, pages 300–311. Springer Verlag.

Heath, L. and Pemmaraju, S. (1996b). Stack and queue layouts of directed acyclic graphs: Part II. Technical report, Department of Computer Science, Virginia Polytechnic Institute & State University.

Jayakumar, R., Thulasiraman, K., and Swamy, M. (1986). On maximal planarization of non-planar graphs. *IEEE Transactions on Circuits Systems*, **33**(8), 843–844.

Jayakumar, R., Thulasiraman, K., and Swamy, M. (1989). On $O(n^2)$ algorithms for graph planarization. *IEEE Transactions on Computer-Aided Design*, **8**(3), 257–267.

Jünger, M., Leipert, S., and Mutzel, P. (1996). On computing a maximal planar subgraph using PQ-trees. Technical Report 96.227, Institut für Informatik der Universität zu Köln.

Kant, G. (1992). An $O(n^2)$ maximal planarization algorithm based on PQ-trees. Technical Report RUU-CS-92-03, Department of Computer Science, Utrecht University.

Lempel, A., Even, S., and Cederbaum, I. (1967). An algorithm for planarity testing of graphs. In *Theory of Graphs: International Symposium: Rome, July 1966*, pages 215–232. Gordon and Breach, New York.

Ozawa, T. and Takahashi, H. (1981). A graph-planarization algorithm and its application to random graphs. In *Graph Theory and Algorithms*, volume 108 of *Lecture Notes in Computer Science*, pages 95–107. Springer Verlag.