
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 9, no. 1, pp. 53–97 (2005)

Radial Level Planarity Testing and Embedding in Linear Time

Christian Bachmaier *Franz J. Brandenburg*

University of Passau,
94030 Passau, Germany
http://www.infosun.fmi.uni-passau.de/br/lehrstuhl/
{bachmaier,brandenb}@fmi.uni-passau.de

Michael Forster

IMAGEN Program, National ICT Australia,
Eveleigh, NSW 1430, Australia
http://nicta.com.au/director/research/programs/imagen/people/michael_forster.cfm
michael.forster@nicta.com.au

Abstract

A graph with an ordered k-partition of the vertices is radial level planar if there is a strictly outward drawing on k concentric levels without crossings. Radial level planarity extends level planarity, where the vertices are placed on k horizontal lines and the edges are routed strictly downwards without crossings. The extension is characterised by rings, which are certain level non-planar biconnected components.

Our main results are linear time algorithms for radial level planarity testing and for computing a radial level planar embedding. We introduce PQR-trees as a new data structure where R-nodes and associated templates for their manipulation are introduced to deal with rings. Our algorithms extend level planarity testing and embedding algorithms, which use PQ-trees.

Article Type	Communicated by	Submitted	Revised
regular paper	G. Liotta	February 2004	June 2005

1 Introduction

We consider the problem of drawing a graph with a given ordered vertex partition. Such a partition may be due to application-specific attributes of the graph (e. g. hierarchies as in [23]) or it may be introduced by structural evaluation (e. g. centrality measures as in [6, 7]) or by the drawing algorithm (e. g. the Sugiyama framework [12, 33]).

Formally, we are given a *k-level graph* $G = (V, E, \phi)$ with a level assignment $\phi\colon V \to \{1, 2, \ldots, k\}$ with $k \leq |V|$ that partitions the vertex set into k pairwise disjoint subsets $V = V^1 \dot\cup V^2 \dot\cup \cdots \dot\cup V^k$, $V^j = \phi^{-1}(j)$, $1 \leq j \leq k$, such that $\phi(u) \neq \phi(v)$ for each edge $(u, v) \in E$. A k-level graph is *proper* if $|\phi(u) - \phi(v)| = 1$ for each edge $(u, v) \in E$. Typically the vertices are constrained to lie on horizontal or radial levels to make the partition visible. As is the case for arbitrary drawings empirical evidence suggests that the number of crossings is a major factor for readability of levelled drawings [31]. The (horizontal) level planarity problem [13, 20, 26] is the question whether or not a level graph G can be drawn in the plane such that all vertices of the j-th level V^j are placed on the j-th horizontal line $l_j = \{\, (x, j) \mid x \in \mathbb{R} \,\}$ and the edges are drawn as strictly y-monotone curves without crossings. The topological structure of such a drawing is given by a level planar embedding, which is characterised by a linear ordering of the vertices on each level V^j. Level planarity has been intensively investigated in literature. The main achievements are linear time algorithms for the test of level planarity and for the computation of an embedding [13, 20, 21, 24–26, 28].

For k-level graphs the partition of the set of vertices into levels is given. Assigning vertices to levels (*levelling*) is a different problem: Heath and Rosenberg [22] have shown that it is $\mathcal{NP}$-hard deciding whether a planar graph has a levelling into a proper level planar graph. In the non-proper case every planar graph has a level planar partitioning of its vertices, but with up to $\mathcal{O}(|V|)$ levels and many long edges spanning several levels. This follows for example from planar straight-line grid drawings or from visibility representations [12]. The number of levels spanned by long edges may be linear in the size of the graph, as a nested sequence of triangles shows [10,11]. However, every planar graph has a concentric representation [34] based on a breadth first search (BFS) traversal, if in addition to the levelling the monotonicity of the edges is discarded. There the vertices are placed on concentric circles corresponding to BFS-levels and the edges are routed as curves without crossings.

Our contribution is a generalisation of level planarity to radial level planarity. Now the vertices are placed on k concentric circles $l_j = \{\, (j \cos\theta, j \sin\theta) \mid \theta \in [0, 2\pi) \,\}$, $1 \leq j \leq k$. A k-level graph is *radial k-level planar* if there are permutations of the vertices on each radial level such that the edges can be drawn as strictly monotone curves from inner to outer levels without crossings. Such drawings [3] extend the radial tree drawings of Eades [16], where the levels of the vertices are given by their depth, i. e., BFS-level. Figure 1(b) shows a radial level planar drawing of the graph in Figure 1(a) which is not level planar. Another simple example is a levelled $K_{2,2}$ which is proper radial 2-level planar but not 2-level planar.

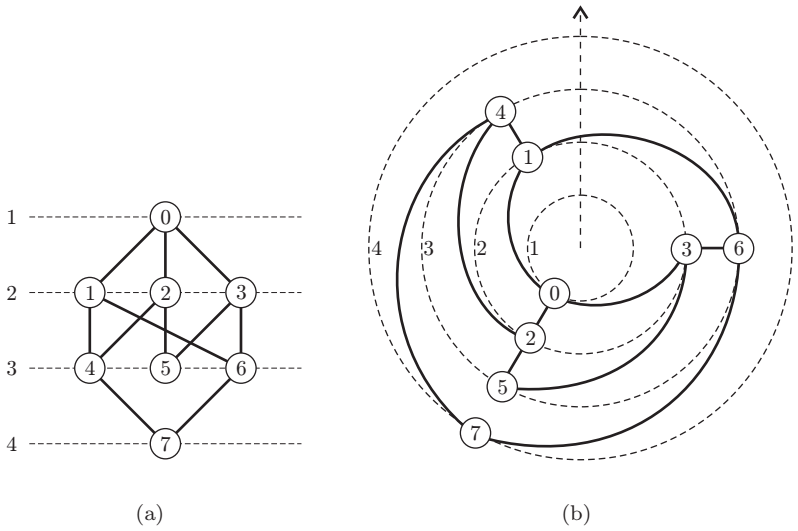


Figure 1: A radial level planar graph with a radial level planar drawing.

Every level planar embedding can be transformed into a radial level planar embedding by connecting the ends of each level to concentric circles. This introduces new possibilities to add edges as monotone curves from the end of one level to the beginning of another or vice versa. These *cut edges* cross a ray from the *centre* of the concentric levels to infinity through the gluing points of the level lines exactly once. There are two directions for routing cut edges around the centre. Hence, as an extension of level planar embeddings, *radial level planar embeddings* need additional information about cut edges and their direction. Consider the graph in Figure 1(b). The edge $(1, 6)$ crosses the ray and thus is a clockwise cut edge, following its implicit direction from the lower to the higher level. Obviously, a radial level planar graph without cut-edges is level planar. Thus the cut-edges make up the major difference between radial level planarity and level planarity.

In the following let $V(G)$ denote the set of vertices of a graph G and $E(G)$ its set of edges. Without loss of generality, we only consider simple graphs without self loops and parallel edges. Obviously, a graph with $|E| > 3|V| - 6$ can be rejected as not (radial) level planar.

Lemma 1 *For a k-level graph G:*

$$G \text{ is level planar } \Rightarrow G \text{ is radial level planar } \Rightarrow G \text{ is planar}$$

This paper is organised as follows: In the next section we survey previous results related to radial level planarity. We recall the linear time level planarity testing and embedding algorithm of Jünger, Leipert, and Mutzel [24–26, 28],

which is a basis for our algorithms. Section 3 introduces the concepts of our linear time approach to decide whether a graph is radial level planar in Section 4. The computation of an embedding is described in Section 5. We conclude with a summary.

2 Level Planarity

2.1 Foundations

Dujmović et al. [15] have applied the concept of fixed parameter tractability to graph drawing. It can be shown that k-level planarity and radial k-level planarity are fixed parameter tractable. However, k must be bounded by a constant. As a consequence an $\mathcal{O}(|V|)$ running time is obtained for a fixed number of levels, but the $\mathcal{O}$-notation hides large constants. We pursue a direct approach and improve the result of Dujmović et al. to linear time for an arbitrary number of levels. Our algorithms are practical and have been realised in a prototypical implementation in C++ using the Graph Template Library (GTL) with improved symmetric lists [4]. They are based on the level planarity testing algorithm of Jünger, Leipert, and Mutzel [24–26, 28], in the following called the JLM algorithm, which in turn is based on the approach of Heath and Pemmaraju [20, 21]. All these algorithms extend the level planarity testing algorithm of Di Battista and Nardelli [13] to arbitrary level graphs. Previously only *hierarchies* could be treated, which are level graphs with a single source. A *source* is a vertex with edges only to vertices on higher levels, whereas a *sink* is a vertex with edges only from vertices on lower levels. The linear time algorithm of Chandramouli and Diwan [8] determines whether a triconnected DAG is level planar. Because the JLM algorithm is rather involved and difficult to implement, Healy and Kuusik [18] have presented a simpler approach for the detection of level planarity. Their algorithm runs in $\mathcal{O}(|V|^2)$ time for proper level graphs and $\mathcal{O}(|V|^4)$ time in the general case. If an embedding is needed, the time complexity raises to $\mathcal{O}(|V|^3)$ and $\mathcal{O}(|V|^6)$, respectively. Finally, Randerath et al. [32] have presented a quadratic time reduction of level planarity of proper level graphs to the satisfiability problem of Boolean formulas in 2CNF, which is solvable in linear time.

Our algorithm is based on the JLM algorithm which must be extended in various directions. Familiar readers may proceed to Section 3.

2.2 Level Planarity Testing

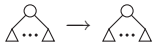
Let G be a k-level graph. The algorithm performs a top down sweep, processing the levels in ascending order. Let G^j be the subgraph induced by the vertices of the first j levels $V^1 \cup V^2 \cup \cdots \cup V^j$. For every G^j a set of *admissible* permutations of V^{j+1} is computed, which are the permutations of level planar embeddings of G^{j+1}. The input graph G is level planar if and only if the set of permutations of $G^k = G$ is non-empty.

The sets of admissible vertex permutations can efficiently be stored and manipulated using PQ-trees. This data structure has been introduced by Booth and Lueker [5] for the consecutive ones property in matrices. A PQ-tree represents the set of permutations of the elements of a finite set S, where the members of specified subsets of S occur consecutively. It is a rooted, ordered tree with leaves and two types of internal nodes, P-nodes and Q-nodes. In the context of this paper the term vertex always denotes an element of a graph and the term node denotes an element of a PQ-tree. It is common to draw P-nodes as circles and Q-nodes as rectangles. The leaves correspond to the elements of S. The set of permutations is encoded by the combination of the internal nodes. The children of a P-node can be permuted arbitrarily, whereas the children of a Q-node are ordered and only reversion is allowed. If PQ-trees are used in planarity tests, a P-node represents a cut vertex and a Q-node represents a biconnected component of the visited part of the graph. The leaves represent edges to the unvisited part. Restrictions on the set of permutations are introduced by edges towards the same vertex. If there are no permutations with the given restrictions, the PQ-tree is empty.

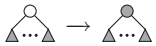
The most important operation on PQ-trees is REDUCE. It restricts the set of permutations such that all elements of a set $S' \subseteq S$ are consecutive in all remaining permutations. In a bottom up strategy REDUCE uses eleven template matching patterns to realise local changes within the tree. These are given in Figure 2 and Figure 3, see [5]. PQ-leaves representing elements of S' are called *pertinent*. The *pertinent subtree* is the subtree of minimum height containing all pertinent PQ-leaves. Its root is called the *pertinent root*. A PQ-node with at least one pertinent child different from the pertinent root is pertinent, too. A PQ-node is *full* if it has only pertinent children, *partial* if it has at least one pertinent and at least one non-pertinent child, and *empty* if it has no pertinent children.

For an application of the templates P2, P4, P6, or Q3, the root of the left side (*pattern*) of the template must be the pertinent root, whereas P3 and P5 cannot be applied to the pertinent root. There are no such restrictions for the applicability of the remaining templates. The application of a template may reverse the order of the children of some Q-nodes and insert the children of a Q-node somewhere in between the children of another Q-node. In order to achieve the linear time complexity as in [5], both reversing a list of children and inserting a list of children into another must be done in constant time. The complexity of REDUCE is crucial for level planarity testing and embedding in linear time.

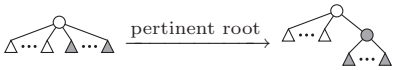
The subgraph G^j induced by the first j levels is not necessarily connected. Thus a separate PQ-tree $T(F_i^j)$ is introduced for every component F_i^j of G^j with m_j such components and $1 \leq i \leq m_j$. $T(F_i^j)$ represents the set of admissible permutations of the vertices of F_i^j in V^j that appear in some level planar embedding of G^j. If two different components are adjacent to a common vertex v, their corresponding PQ-trees must be merged. $\mathcal{T}(G^j)$ denotes the set of all $T(F_i^j)$.



(a) Template P0.

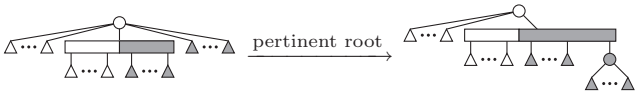


(b) Template P1.



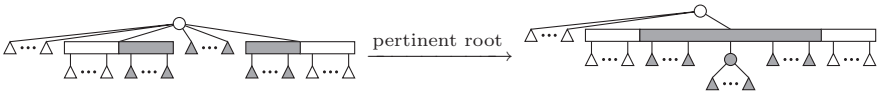
(c) Template P2.

(d) Template P3.



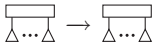
(e) Template P4.

(f) Template P5.

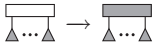


(g) Template P6.

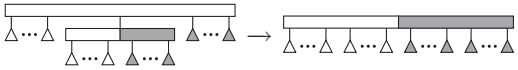
Figure 2: Templates for testing (level) planarity. The grey shading indicates pertinent nodes.



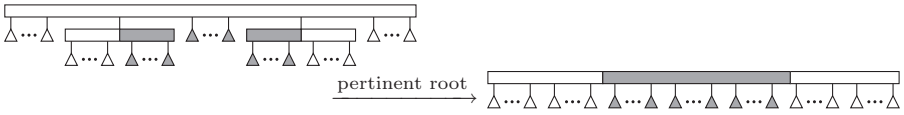
(a) Template Q0.



(b) Template Q1.



(c) Template Q2.



(d) Template Q3.

Figure 3: Templates, Part 2.

A formal description of the LEVEL-PLANARITY-TEST algorithm is given by Algorithm 1. All operations are applied directly to $T(F_i^j)$ and not to the graph. The procedure CHECK-LEVEL in Algorithm 2 is a sweep over a single level j, divided into a first and a second reduction phase.

Algorithm 1: LEVEL-PLANARITY-TEST

Input: A level graph $G = (V^1 \dot\cup V^2 \dot\cup \ldots \dot\cup V^k, E, \phi)$
Output: A Boolean value indicating whether G is level planar

Initialise $\mathcal{T}(G^1)$
for $j \leftarrow 1$ ***to*** $k-1$ **do**
  $\mathcal{T}(G^{j+1}) \leftarrow$ CHECK-LEVEL($\mathcal{T}(G^j)$, V^{j+1})
  if $\mathcal{T}(G^{j+1}) = \emptyset$ **then return false**
end

return true

Algorithm 2: CHECK-LEVEL

Input: The set of PQ-trees $\mathcal{T}(G^j)$ and the vertices V^{j+1} of the next level
Output: The PQ-trees $\mathcal{T}(G^{j+1})$ of the next level

$\mathcal{T}(G^j) \leftarrow$ FIRST-REDUCTION-PHASE($\mathcal{T}(G^j)$, V^{j+1})
if $\mathcal{T}(G^j) = \emptyset$ **then return** $\emptyset$
$\mathcal{T}(G^j) \leftarrow$ SECOND-REDUCTION-PHASE($\mathcal{T}(G^j)$, V^{j+1})
if $\mathcal{T}(G^j) = \emptyset$ **then return** $\emptyset$
$\mathcal{T}(G^j) \leftarrow$ FINAL-UPDATES($\mathcal{T}(G^j)$, V^{j+1})
return $\mathcal{T}(G^{j+1}) \leftarrow \mathcal{T}(G^j)$

Algorithm 3 describes the *first reduction phase*. Define H_i^j to be the *extended form* of F_i^j, which consists of F_i^j and some new *virtual vertices* and *virtual edges*. For every edge (u, v) with $u \in V(F_i^j) \cap V^j$ and $\phi(v) > j$, a new virtual vertex v' with label v and a virtual edge (u, v') are introduced in H_i^j. The set of all virtual vertices of H_i^j with label v is denoted by S_i^v. Note that there may be several virtual vertices with the same label, possibly adjacent to different components of G^j and each with exactly one entering edge.

The extension of $T(F_i^j)$ to $T(H_i^j)$ is called the *vertex addition step* and is accomplished by the REPLACE PQ-tree operation. After REDUCE all PQ-leaves with the same label v appear consecutively in every admissible permutation. REPLACE replaces every such consecutive set with a P-node labelled v. This is the parent of some new leaves representing the adjacent vertices of v in $V^{j+1} \cup V^{j+2} \cup \cdots \cup V^k$. Thereafter all PQ-leaves representing vertices in V^{j+1} with the same label are reduced to appear as a consecutive sequence in any permutation stored by the PQ-trees. Then REPLACE-SINGLE replaces them with a single representative PQ-leaf with the same label. This *reduced extended form* of H_i^j is denoted by R_i^j. If the graph is not a hierarchy, the replacement

with a single representative is necessary for the correctness of the algorithm, as JLM [28, p. 71ff] have discovered.

Algorithm 3: FIRST-REDUCTION-PHASE

Input: The set of PQ-trees $\mathcal{T}(G^j)$ and the vertices V^{j+1} of the next level
Output: The reduced PQ-trees $\mathcal{T}(G^j)$

```
foreach component F_i^j in G^j do
    construct T(H_i^j) from T(F_j^i)
end
foreach v ∈ V^{j+1} do
    foreach extended form H_i^j do
        if |S_i^v| ≥ 2 then
            T(R_i^j) ← REDUCE(T(H_i^j), S_i^v)
            if T(R_i^j) = ∅ then return ∅
            let ṽ be a single representative of S_i^v
            T(R_i^j) ← REPLACE-SINGLE(T(R_i^j), S_i^v, ṽ)
        end
    end
end
return T(G^j)
```

Different PQ-trees may contain PQ-leaves with the same label. Thus a *second reduction phase* is needed to merge these trees. A reduced extended form R_i^j is called *v-singular* if all its virtual vertices have the same label, i. e., $\bigcup_{w \in V, \phi(w) > j} S_i^w = \{v\}$. Whenever new inner faces are created by replacing all leaves labelled v with a single representative, a value PML or QML, which stores the lowest level of this faces, is maintained in the PQ-leaf representing v. Using this information it is possible to decide whether or not a v-singular component fits into an inner face above v. Otherwise, it is checked whether it can be placed into the outer face with the same mechanism as for non-singular forms.

Next we briefly describe the pairwise merge operations. Define the *low indexed level* $\mathrm{LL}(F_i^j)$ of F_i^j to be the least d such that F_i^j contains a vertex in V^d. This value is maintained as an attribute of the corresponding PQ-tree $T(F_i^j)$. The *height* of a component F_i^j is $j - \mathrm{LL}(F_i^j)$. A merge operation is accomplished by using information that is stored at the nodes of the PQ-trees. For any set of virtual vertices $S \subseteq V^{j+1} \cup V^{j+2} \cup \cdots \cup V^k$ of a form H_i^j or R_i^j, define the *meet level* $\mathrm{ML}(S)$ of S to be the largest $d \leq j$ such that $V^d \cup V^{d+1} \cup \cdots \cup V^j$ induces a subgraph of G where all $s \in S$ occur in the same connected component. For every P-node X a single value $\mathrm{ML}(X) = \mathrm{ML}(\mathrm{frontier}(X))$ is maintained, where $\mathrm{frontier}(X)$ is the sequence of its descendent leaves from left to right. For every Q-node Y with ordered children $Y_1, Y_2, \ldots, Y_t$ the values $\mathrm{ML}(Y_i, Y_{i+1}) = \mathrm{ML}(\mathrm{frontier}(Y_i) \cup \mathrm{frontier}(Y_{i+1}))$, $1 \leq i < t$, are stored. These indicators tell whether a PQ-tree with a given low indexed level fits into the indentations below a P-node or between two sons of a Q-node. The mainte-

Algorithm 4: SECOND-REDUCTION-PHASE

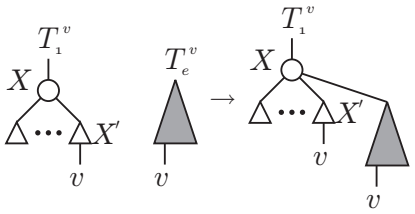
Input: The set of PQ-trees $\mathcal{T}(G^j)$ and the vertices V^{j+1} of the next level
Output: The merged PQ-trees $\mathcal{T}(G^j)$

foreach $v \in V^{j+1}$ **do**
  foreach *PQ-tree* $T(R_i^j)$ *containing a leaf labelled with* v **do**
    // lazy reduce
    if $|S_i^v| \geq 2$ **then**
      $T(R_i^j) \leftarrow \text{REDUCE}(T(R_i^j), S_i^v)$
      if $T(R_i^j) = \emptyset$ **then return** $\emptyset$
      let $\tilde{v}$ be a single representative of S_i^v
      $T(R_i^j) \leftarrow \text{REPLACE-SINGLE}(T(R_i^j), S_i^v, \tilde{v})$
    end
  end
  eliminate all v-singular R_i^j except for the one with the lowest LL-value
  reorder indices such that $S_1^v, S_2^v, \ldots, S_p^v \neq \emptyset$, $S_q^v = \emptyset$ for $q > p$,
    and $\text{LL}(R_1^j) \leq \text{LL}(R_2^j) \leq \ldots \leq \text{LL}(R_p^j)$
  for $i \leftarrow 1$ ***to*** p **do**
    $T(R_1^j) \leftarrow \text{INSERT}(T(R_1^j), T(R_i^j), v)$
    $T(R_1^j) \leftarrow \text{REDUCE}(T(R_1^j), S_1^v)$
    if $T(R_1^j) = \emptyset$ **then return** $\emptyset$
    let $\tilde{v}$ be a single representative of S_1^v
    $T(R_1^j) \leftarrow \text{REPLACE-SINGLE}(T(R_1^j), S_1^v, \tilde{v})$
  end
end
return $\mathcal{T}(G^j)$

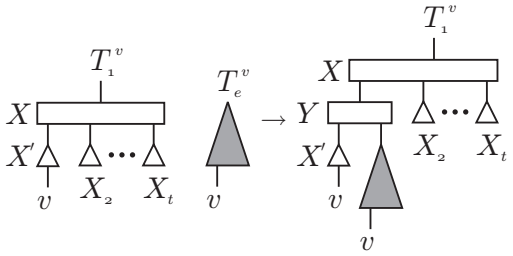
nance of the ML-values during template reductions and insertions in PQ-trees is straightforward.

Let $T_1^v, T_2^v, \ldots, T_f^v$ be all PQ-trees containing a leaf $v \in V^{j+1}$ sorted according to descending height. All PQ-trees T_e^v, $2 \leq e \leq f$, are merged sequentially into T_1^v. This corresponds to adding the root of T_e^v as a child to a PQ-node of T_1^v. In order to find an appropriate location to insert T_e^v, the method starts with the leaf in T_1^v labelled with v and proceeds upwards in T_1^v until a node X' and its parent X are encountered which satisfy one of the following merge conditions. These are checked in the order A to E.

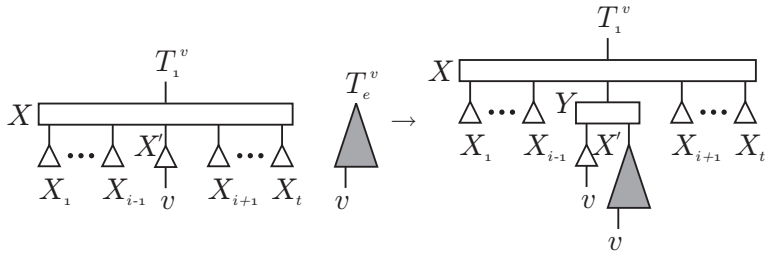
Merge Condition A The node X is a P-node with $\mathrm{ML}(X) < \mathrm{LL}(T_e^v)$. Then attach T_e^v as a child of X in T_1^v.



Merge Condition B The node X is a Q-node with ordered children X_1, X_2, $\ldots$, X_t, $X' = X_1$, and $\mathrm{ML}(X_1, X_2) < \mathrm{LL}(T_e^v)$. Then replace X' in T_1^v with a new Q-node Y with X' and T_e^v as children. The case were $X' = X_t$ and $\mathrm{ML}(X_{t-1}, X_t) < \mathrm{LL}(T_e^v)$ is symmetric.



Merge Condition C The node X is a Q-node with ordered children X_1, X_2, $\ldots$, X_t, $X' = X_i$, $1 < i < t$, and $\mathrm{ML}(X_{i-1}, X_i) < \mathrm{LL}(T_e^v)$ and $\mathrm{ML}(X_i, X_{i+1}) < \mathrm{LL}(T_e^v)$. Then replace X' with a new Q-node Y with X' and T_e^v as children.



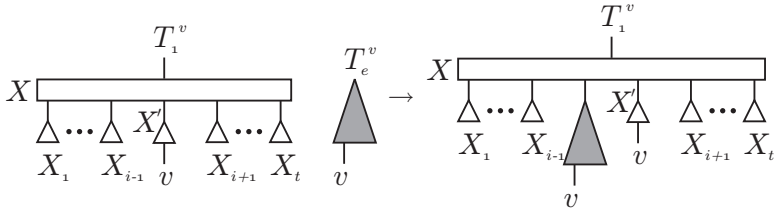
Merge Condition D The node X is a Q-node with ordered children X_1, X_2, $\ldots$, X_t, $X' = X_i$, $1 < i < t$, and

$$\mathrm{ML}(X_{i-1}, X_i) < \mathrm{LL}(T_e^v) \leq \mathrm{ML}(X_i, X_{i+1}).$$

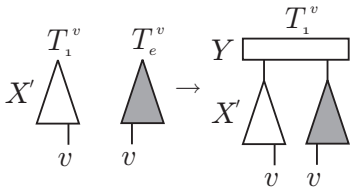
Then attach T_e^v as a child of X between X_{i-1} and X_i. If

$$\mathrm{ML}(X_i, X_{i+1}) < \mathrm{LL}(T_e^v) \leq \mathrm{ML}(X_{i-1}, X_i)$$

then attach T_e^v as a child of X between X_i and X_{i+1}.



Merge Condition E The node X' is the root of T_1^v. Then reconstruct T_1^v by inserting a new Q-node Y as the new root with X' and T_e^v as its children.



After each merge operation, REDUCE and REPLACE are called again to make all v-leaves consecutive and then to replace them with a single representative PQ-leaf. Afterwards T_e^v is deleted from $\mathcal{T}(G^{j+1})$. In order to achieve linear running time, there is no scan for other leaves with the same label after v-merging several reduced extended forms. However, this strategy results in improper reduced extended forms possibly with several virtual vertices with the same label. These are called *partially reduced extended forms* and are reduced on demand.

Finally, in a new sweep over the current level Algorithm 5 deletes all PQ-leaves representing sinks v in V^{j+1} from their corresponding PQ-tree and reconstructs the tree such that it obeys the properties of a valid PQ-tree.

Algorithm 5: FINAL-UPDATES

Input: The set of PQ-trees $\mathcal{T}(G^j)$ and the vertices V^{j+1} of the next level
Output: The PQ-trees $\mathcal{T}(G^j)$

delete leaves representing sink vertices of V^{j+1} from the PQ-trees
update the pointers of the leaves to their PQ-tree
add for every source in V^{j+1} a new PQ-tree to $\mathcal{T}(G^j)$
return $\mathcal{T}(G^j)$

Note that the LEVEL-PLANARITY-TEST also works on non-proper level graphs within $\mathcal{O}(|V|)$ time and without inserting up to $\mathcal{O}(|V|^2)$ dummy vertices for long edges by adding all children on higher levels and not only those on the next level.

2.3 Level Planar Embedding

For a witness of the level planarity of a graph after a positive level planarity test and for a level planar drawing the algorithm computes a level embedding in two passes. It is outlined in Algorithm 6. First G is augmented to a planar st-graph [17,29]. This is a biconnected graph with two adjacent vertices s and t and a bijective numbering $st : V \to \{1, \ldots, |V|\}$ of the vertices such that $st(s) = 1$, $st(t) = |V|$, and that for every vertex v with $1 < st(v) < |V|$ there are adjacent vertices u and w with $st(u) < st(v) < st(w)$. An st-numbering for G can be computed by topologically sorting the vertices using implicit edge directions from lower to higher levels. This corresponds to numbering the vertices level by level in ascending order. Then a planar st-embedding can be obtained by the algorithm of Chiba et al. [9], from which a level planar embedding can directly be computed.

Augmenting a level graph G to an st-graph G_{st} is divided into two phases. After adding a new source s and a new target t, in the first phase an outgoing edge is added to every old sink of G by the application of a modified LEVEL-PLANARITY-TEST algorithm from level 1 to k. Using the same algorithmic concept bottom up from level k to 1, an incoming edge is added to every old source of G in the second phase. To add these edges without violating level planarity, every PQ-leaf representing a sink in G is replaced with a *sink indicator* as a leaf in its corresponding PQ-tree. This indicator is ignored throughout the application of the algorithm. If all siblings of a node are ignored, its parent is ignored, too. Thus whole PQ-trees can be ignored. Sink indicators are removed either together with the leaves representing the incoming edges of some vertex $w \in V^l$, $l > j$, or they are still left in the final PQ-trees. In the first case vertices which are represented by sink indicators are connected to w after its reduction by the subsequent REPLACE on w. In the second case they are connected to t

Algorithm 6: LEVEL-PLANAR-EMBED

Input: A level graph $G = (V^1 \dot\cup V^2 \dot\cup \ldots \dot\cup V^k, E, \phi)$
Output: A level embedding $\mathcal{E}_l$ of G if it is level planar, $\emptyset$ otherwise

expand G to G_{st} by adding $V^0 \leftarrow \{s\}$ and $V^{k+1} \leftarrow \{t\}$

AUGMENT(G_{st})
if *AUGMENT fails* **then return** $\mathcal{E}_l \leftarrow \emptyset$

// G_{st} is now a hierarchy

reverse the level numbering G_{st} from bottom to top
AUGMENT(G_{st}) *// cannot fail*
restore the original level numbering G_{st}
$E_{st} \leftarrow E_{st} \cup (s,t)$

// G_{st} is now an st-graph

TOPSORT(V_{st})
compute a planar embedding $\mathcal{E}_{st}$ according to Chiba et al. [9]
using the topological sorting as an st-ordering

$\mathcal{E}_l \leftarrow$ CONSTRUCT-LEVEL-EMBED($\mathcal{E}_{st}$, G_{st})

return $\mathcal{E}_l$

at the end of the augmentation phase. Sink indicators in PQ-trees representing a v-singular form are connected to v if they are inserted into an inner face above v.

Algorithm 6 computes an *st*-embedding $\mathcal{E}_{st}$ by the technique of Chiba et al. [9] using a topological sorting of the augmented graph as the *st*-numbering. The algorithm CONSTRUCT-LEVEL-EMBED computes a level planar embedding $\mathcal{E}_l$ of G from the planar embedding $\mathcal{E}_{st}$. It traverses the graph in depth first search (DFS) order from t and proceeds from every visited vertex v to the unvisited neighbour w on a smaller level that appears first in the clockwise ordering of v's adjacency list in $\mathcal{E}_{st}$. Initially, all levels in $\mathcal{E}_l$ are empty. If a vertex $w \notin \{s,t\}$ is visited, it is appended at the end of the ordered list of the vertices assigned to $\phi(w)$. Since the DFS starts at t and uses only edges to vertices with a smaller *st*-number, the DFS in Chiba's method ENTIRE-EMBED [9, p. 62] extending the obtained directed upward embedding $\mathcal{E}_u$ into a complete and undirected *st*-embedding $\mathcal{E}_{st}$ can be omitted.

In order to achieve linear running time, it must be avoided to search for sink indicators which can be considered for an augmentation. But sink indicators must be treated correctly by merge operations. Therefore, a new node type called *contact* is introduced in the PQ-trees during the merge operations B–D. The contacts store which sinks have to be augmented if the new introduced Q-node is inserted into its parent Q-node by an application of a template later in the algorithm. For details see [24, 25, 28].

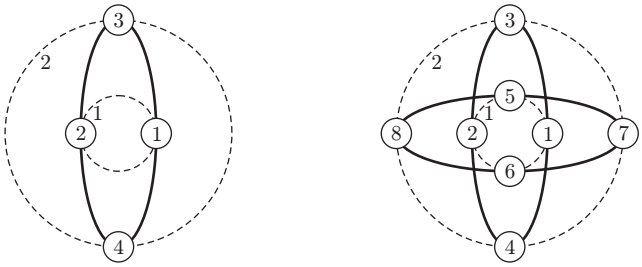


Figure 4: A radial level planar connected component and a radial level non-planar graph which is a combination of two such components.

3 Fundamental Properties

In this section we establish some fundamental properties of radial level planar graphs and elaborate distinctions between level and radial level planar graphs. Our first result is obvious.

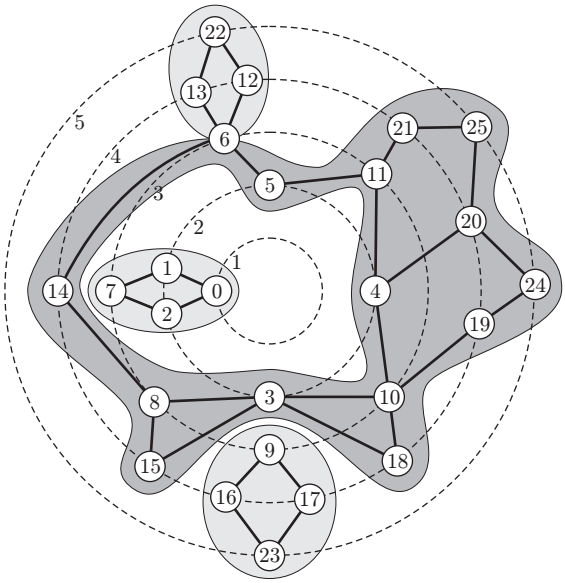
Lemma 2 *A radial level planar graph is level planar if there are no cut edges.*

Next consider connectivity. The JLM algorithm relies on the fact that a level graph is level planar if and only if each connected component is level planar. Therefore, it tests each connected component separately for level planarity, what is no restriction since separate components can be placed next to each other. This is no more true for radial level planarity as Figure 4(b) illustrates. There two disjoint ovals interleave.

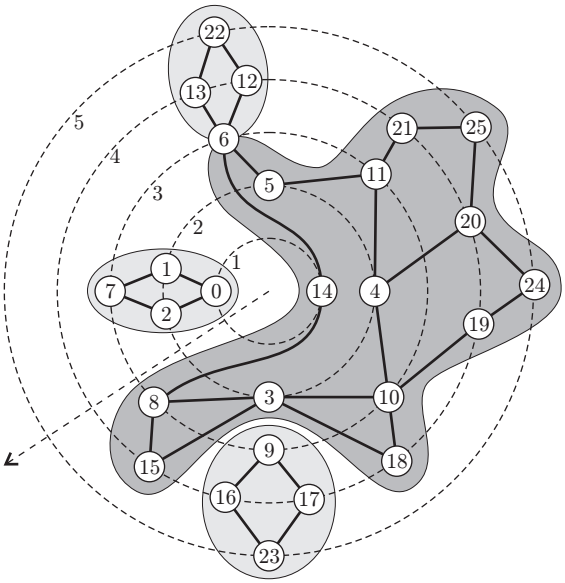
Obviously, a graph is radial level planar if it consists of level planar components only. Hence, we must consider those components of a level graph that are radial level planar and level non-planar.

Definition 1 *A* ring *is a biconnected component of a level graph which is radial level planar and not level planar. A level graph containing a ring is called a* ring graph*.*

It is not immediate whether a biconnected component is a ring. We will see later how rings are detected. Nevertheless we investigate some interesting properties of rings first. The graph in Figure 5(a) consists of four biconnected components with a darker shaded ring. Observe that a component can and sometimes must be nested in another one. This may occur if the outer component is a ring. Clearly, a ring must contain a cycle, but a cycle does not necessarily induce a ring. Being a ring also depends on the levelling. If vertex 14 was on level 1, this graph would not contain a ring, because according to the ray in Figure 5(b) there are no cut edges and every component is level planar. The definitions imply the following fact:



(a) A ring graph.



(b) Not a ring graph.

Figure 5: Rings depend on the levelling.

Lemma 3 *If a level graph G does not contain a ring, the following are equivalent:*

1. *G is radial level planar.*
2. *G is level planar.*
3. *Each connected component of G is level planar.*
4. *Each connected component of G is radial level planar.*

Hence, if a graph does not contain a ring, we can use the JLM level planarity testing algorithm to test for radial level planarity. For ring graphs the algorithm needs an extension. Before we describe how our algorithm stores the admissible permutations of the vertices on each concentric circle in the next section we discuss some more properties of rings.

Lemma 4 *In every radial level planar embedding of a ring graph the centre of the concentric levels lies in an inner face. This face is called the* centre face.

Proof: Suppose there exists a ring graph G that has a radial level planar embedding with the centre lying in the outer face. Then there is a ray from the centre to infinity which crosses no edges. Hence, there are no cut edges and every biconnected component of G is level planar. Thus G does not contain a ring in contradiction to the assumption. □

Lemma 5 *A ring contains at least four vertices and four edges, and there is a ring of that size.*

Proof: A ring is not level planar. Thus every level embedding contains at least two crossing edges (u, v) and (w, x) with mutually different vertices u, v, w, and x. To ensure biconnectivity at least four edges are needed. The $K_{2,2}$ on two levels is a ring, cf. Figure 4(a). □

Another important property of rings is the nesting, which is determined by some characterising parameters:

Definition 2 *For a k-level graph G containing a ring R let α_R and δ_R be the minimum and maximum levels of G containing a vertex of R, respectively. Let the* inner radius *β_R of R be the maximum level with a vertex of the centre face of R in any radial level planar embedding, and let the* outer radius *γ_R of R be the minimum level with a vertex of the outer face of R in any radial level planar embedding.*

These parameters are illustrated in Figure 6. The minimum level α_R and the maximum level δ_R of a ring are independent of the embedding because they are given by the levelling of the graph. This is not necessarily true for the inner radius β_R and the outer radius γ_R.

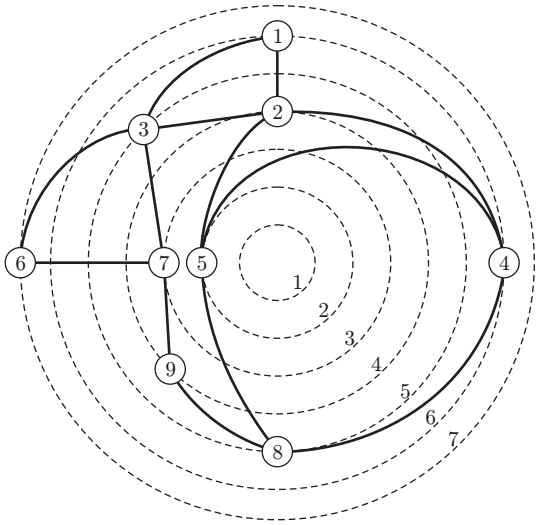


Figure 6: Extreme levels of a ring. $\alpha_R = 2$, $\beta_R = 6$, $\gamma_R = 3$, $\delta_R = 7$.

Definition 3 *A radial level planar embedding of a ring R is* level optimal *if β_R and γ_R are the extreme levels of the centre face and of the outer face, respectively.*

Lemma 6 *Every level planar graph has an embedding that is level optimal for each contained ring.*

Proof: Our algorithm always constructs such an embedding, as will be shown in Lemma 12. □

Lemma 7 *Every ring R spans at least two levels and its characterising parameters relate by $\delta_R > \gamma_R \geq \alpha_R$ and $\delta_R \geq \beta_R > \alpha_R$.*

Proof: The source and the target vertices of an edge always lie on different levels since horizontal edges are not allowed. By Lemma 5 a ring always contains edges and thus has vertices on at least two levels. Therefore, the four relations follow directly from the definitions. □

Lemma 8 *Let G be a level graph consisting of two disjoint rings R and S. G is radial level planar if and only if R and S are radial level planar and R fits in the centre face of S or vice versa, i. e.,*

$$(\alpha_S > \gamma_R \text{ and } \beta_S > \delta_R) \quad \text{or} \quad (\alpha_R > \gamma_S \text{ and } \beta_R > \delta_S).$$

Proof: For the only if direction let G be a radial level planar graph consisting of two disjoint rings R and S. Since subgraphs of radial level planar graphs are always radial level planar, R and S are radial level planar. Each ring is biconnected and encloses the centre according to Lemma 4. Thus in any planar

embedding one ring is completely contained within the centre face of the other. Let w. l. o. g. R be contained in S. If $\alpha_S \leq \gamma_R$ or $\beta_S \leq \delta_R$ then the border of the centre face of S intersects the border of the outer face of R, which contradicts the radial level planarity of G.

For the if direction let G be a level graph consisting of two disjoint rings R and S with $\alpha_S > \gamma_R$ and $\beta_S > \delta_R$. The case $\alpha_R > \gamma_S$ and $\beta_R > \delta_S$ is symmetric. We show the radial level planarity of G by an embedding of G which combines level optimal radial level planar embeddings $\mathcal{E}_l^R$ and $\mathcal{E}_l^S$ of R and S. These embeddings have only the levels between α_S and δ_R in common, whereas the others remain unchanged. Since $\alpha_S > \gamma_R$ all vertices of S are on higher levels than γ_R and thus can be placed beyond the border of the outer face of R. Since $\beta_S > \delta_R$ all vertices of R fit inside the border of the centre face of S. Note that it may be necessary to rotate and squeeze R, so that all vertices fit inside the largest cavity of S and vice versa. □

The nesting of disjoint rings is illustrated in Figure 7. This is an essential difference to disjoint components of level planar graphs, which are usually placed side by side and which can be treated separately.

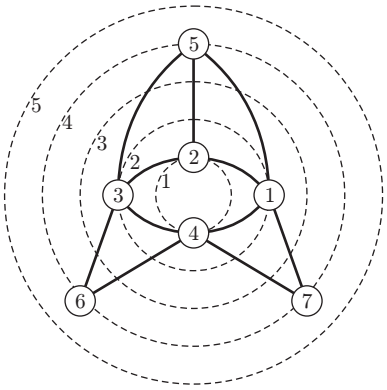
4 Radial Level Planarity Testing

We now come to the main results of this paper and extend the JLM algorithm for radial level planarity testing, see Algorithm 1. Our extensions are PQR-trees as a new data structure, an advanced merging of them, and the detection of nested rings.

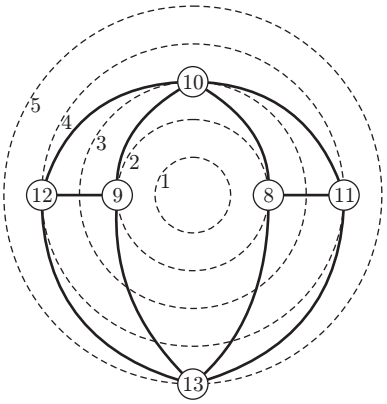
The input graph is traversed in a top down sweep, which now becomes a wavefront sweep from the centre. The processed part of the graph is represented by a collection of trees, which is denoted by $\mathcal{T}$. We need a new data structure, *PQR-trees*, to deal with rings. PQR-trees store the admissible edge permutations of radial level planar graphs. They are based on PQ-trees and contain a new "R" node type for the rings. PQR-trees are not related to SPQR-trees that are used for example in incremental planarity testing [14].

4.1 R-Nodes

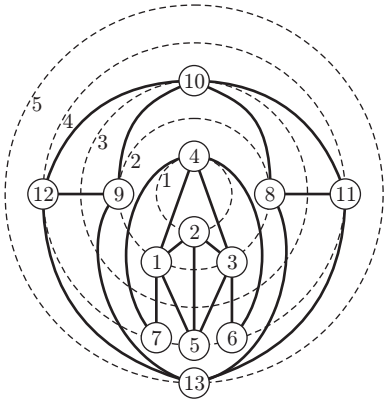
R-nodes are similar to Q-nodes. Their new properties express the differences between rings and other biconnected components. An R-node is drawn as an elliptical ring. The admissible operations on an R-node are reversion, i. e., inverting the iteration direction of its children in the same way as for Q-nodes, and a new one, *rotation*. A rotation corresponds to rotating the graph around the centre and moves a subsequence of the children of an R-node from the beginning of the children list to its end, or vice versa, while maintaining their relative order. See Figure 8 for an example. This happens implicitly by using circular lists. Therefore, R-nodes (as well as Q-nodes) can be implemented with the *improved symmetric list* data structure [4]. This is an encapsulated data



(a) The inner ring R, where $\gamma_R = 1$ and $\delta_R = 4$.

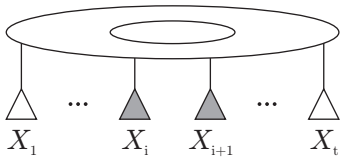


(b) The outer ring S, where $\alpha_S = 2$ and $\beta_S = 5$.

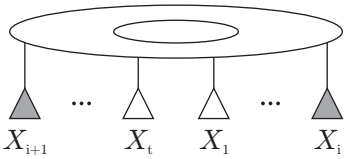


(c) R nested in the centre face of S.

Figure 7: Nesting of rings.



(a) An R-node with children $X_1, \ldots, X_t$.



(b) Figure 8(a) after rotation.

Figure 8: Rotation of an R-node.

structure where insertions, reversions and rotations can be done in constant time. This is crucial for the linear running time of the test.

Lemma 9 *During the radial planarity test the admissible edge permutations can be stored such that R-nodes only occur as the root of PQR-trees.*

Proof: At any time in the wavefront sweep the leaves of a PQR-tree represent edges to vertices of the unvisited part of the graph. When a ring R is encountered, radial planarity implies that there are no such edges left originating from a component nested within the centre face of R. Otherwise, an edge would cross a cycle of R which encloses the centre. Hence, it is sufficient that R-nodes never have siblings and thus they only occur at the root of PQR-trees. This follows from the definition of PQ-trees, since P-nodes or Q-nodes must have at least two children, see [5, p. 339]. As we see later, R-nodes can have a single child, and thus chains of R-nodes representing nested rings would be possible. This is unnecessary since it suffices to keep only the outermost ring. The embedding of the inner components can be left unchanged. □

4.2 New Templates

For the R-nodes twelve new templates are needed to implement REDUCE on PQR-trees, some of them being analogous to the Q-node templates adopted from PQ-trees. They are given in Figure 9 and Figure 10. A new R-node is generated only by the templates P8, Q4, and Q6. The displayed children are optional, as long as the child sequence of the resulting R-node starts and

ends with pertinent children and has at least one empty child. Obviously, in these cases it is impossible to apply any of the standard templates, i. e., the graph is no more level planar. An R-node is created only when needed, i. e., if newly encountered edges transfer a represented biconnected component from level planar into a ring. By Lemma 9 templates P8, Q4, and Q6 may only be applied to the root of a PQR-tree. This is different from the restriction that some PQ-tree templates may only be applied to the pertinent root.

The meet level between two children of an R-node which are direct siblings or are both endmost is defined and maintained analogously to the meet levels between children of a Q-node, cf. p. 61. In order to know what fits below a ring component we define the following:

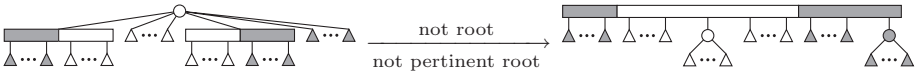
Definition 4 *For an R-node X with children X_1, X_2, $\dots X_t$ let*

$$\text{minML} = \min\{\, \text{ML}(X_i, X_{i+1}) \mid 1 \leq i \leq t, X_{t+1} = X_1 \,\}.$$

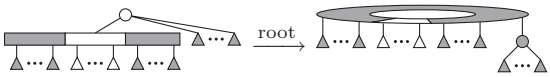
In analogy to the templates Q0–Q3 and Q6 it is necessary to provide the new templates R0–R4 from Figure 10 to treat patterns with an R-node as root. Before an R-template can be applied it may be necessary to rotate the R-node. R0, R2, R3, and R4 are the straightforward transformations of Q0, Q2, Q3, and Q6, respectively. For technical reasons we introduce a pseudo Q-node X' in R1 as a parent of all full children. Its R-parent preserves the information that the PQR-tree represents a ring component and allows the computation of a value minML in order to know what fits below this ring component. The single meet level at the root is set to $\text{ML}(X', X') = \text{minML}$.

In P8 and Q6 a Q-node may be *boundary partial*, i. e., it may have pertinent children at the boundaries, enclosing some empty children in the middle. In radial level planar graphs this can occur if the root of the PQR-tree is already an R-node or becomes an R-node during the current reduction step and thus if a rotation is possible thereafter, see Figure 11. Then the front and the back can be connected by cut edges. Of course, every child of every ancestor of the boundary partial Q-node Z which is not on the path from Z to the root must be full. All pertinent children become children of the R-node and can be made consecutive by a rotation. If no template matches for a boundary partial Q-node during REDUCE, the graph is not radial level planar because its PQR-tree contains non-consecutive pertinent nodes. Observe that the templates prohibit that a boundary partial Q-node is created at the pertinent root because this always results in a non consecutive pertinent sequence, except in one special case: Because of R1, an R-root is the only internal node which may have a single child in a valid PQR-tree. If this single child later becomes boundary partial and would be the pertinent root during REDUCE, we must explicitly set the pertinent root to its father R-node to allow the application of R4 and a rotation thereafter.

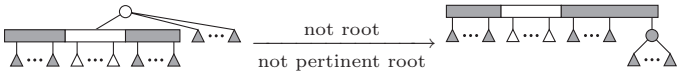
For the boundary partial Q-nodes we must provide the additional templates P7–P9 and Q5–Q7. P7 is the straightforward transformation of P6 if P6 is not applied to the pertinent root. The full children are grouped by a new P-node which is inserted into the Q-node. It is admissible to place it at either boundary



(a) Template P7.

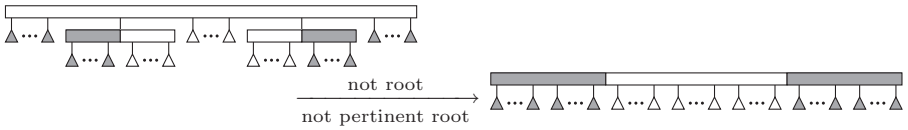


(b) Template P8.



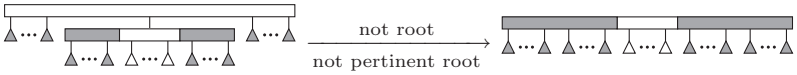
(c) Template P9.

(d) Template Q4.



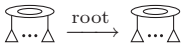
(e) Template Q5.

(f) Template Q6.



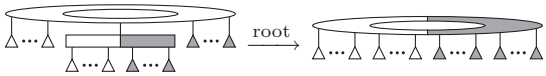
(g) Template Q7.

Figure 9: Templates P7–P9 and Q4–Q7 for radial level planarity testing. The grey shading indicates pertinent subtrees.

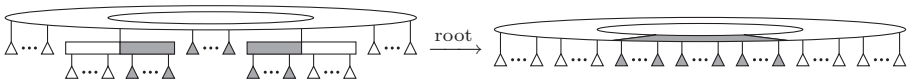


(a) Template R0.

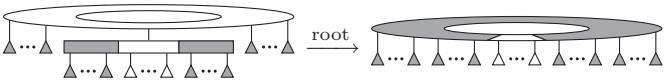
(b) Template R1.



(c) Template R2.



(d) Template R3.



(e) Template R4.

Figure 10: Templates R0–R4 for radial level planarity testing.

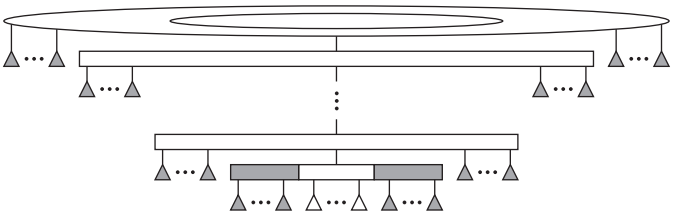


Figure 11: Iterative merges of boundary partial Q-nodes.

of the Q-node. The only difference between these two positions is whether or not the edges represented by the descendant leaves later become cut edges. This holds accordingly for the new P-node created in P8 or P9. P8 can only be applied to the root, otherwise P9 is applied. Template Q5 is basically the same as Q4, but it treats non-roots. Q4 and Q5 are the inversion of Q3. Templates Q6 and Q7 are used for Q-nodes with only full children except for one boundary partial child. The former is used for the root and the latter for a non-root. Now we are ready to establish another important property of R-nodes.

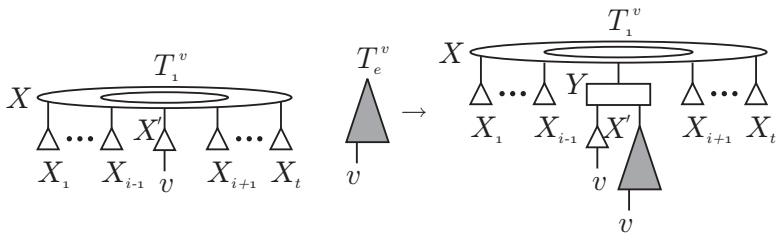
Lemma 10 *If an R-node is created, it is preserved until its host PQR-tree is deleted.*

Proof: There is no template which destroys or replaces an R-node. Furthermore, R1 ensures that an R-node never becomes full, which means that it is never replaced by an application of REPLACE. □

4.3 Merge Operations on PQR-Trees

Since radial level planarity works on graphs which are not necessarily hierarchies, merges of PQR-trees are needed by the same reason as for PQ-trees. If there is no R-node, the merge conditions for PQR-trees are the same as for PQ-trees described in Section 2.2. Because of Lemma 8 merge condition E cannot be applied if any of the trees has an R-root. As a consequence a merge operation may fail in contrast to the non-radial case, where condition E always is admissible if no other condition applies. For PQR-trees with an R-node as its root we have to provide two additional merge conditions. If the root of T_e^v is an R-node then the merge operation fails and the input graph is rejected as radial level non-planar, see proof of Lemma 9. For an R-root X of the host PQR-tree T_1^v, condition B and C collapse to the new condition C^{R}. This is because R-nodes can be rotated such that the merge can be done on its interior children. Similarly, if X is the root of the pattern of condition D and X is an R-node we obtain D^{R}.

Merge Condition $\mathbf{C^R}$ The root of T_e^v is not an R-node. The node X is an R-node with ordered children $X_1, X_2, \ldots, X_t$, $X' = X_i$, $1 < i < t$, and $\mathrm{ML}(X_{i-1}, X_i) < \mathrm{LL}(T_e^v)$ and $\mathrm{ML}(X_i, X_{i+1}) < \mathrm{LL}(T_e^v)$. Then replace X' with a new Q-node Y with X' and T_e^v as children.



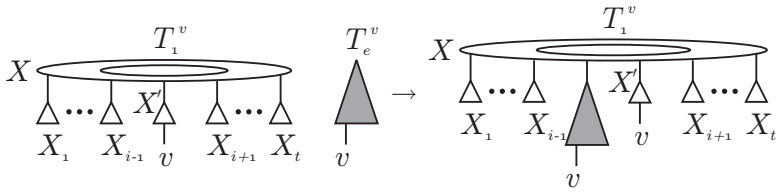
Merge Condition D$^{\mathbf{R}}$ The root of T_e^v is not an R-node. The node X is an R-node with ordered children $X_1, X_2, \ldots, X_t$, $X' = X_i$, $1 < i < t$, and

$$\mathrm{ML}(X_{i-1}, X_i) < \mathrm{LL}(T_e^v) \leq \mathrm{ML}(X_i, X_{i+1}).$$

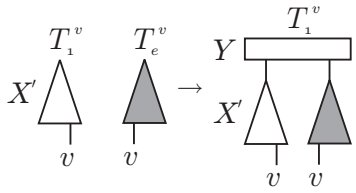
Then attach T_e^v as a child of X between X_{i-1} and X_i. If

$$\mathrm{ML}(X_i, X_{i+1}) < \mathrm{LL}(T_e^v) \leq \mathrm{ML}(X_{i-1}, X_i)$$

then attach T_e^v as a child of X between X_i and X_{i+1}.



Merge Condition E The node X' is the root of T_1^v. X' and the root of T_e^v are not R-nodes. Reconstruct T_1^v by inserting a new Q-node Y as the new root with X' and T_e^v as children.



4.4 Nesting of Processed Non-Rings

In level planar graphs separate components can always be placed next to each other without violating planarity. This is not necessarily true for radial level planar graphs. If a component of the input graph G contains a ring, it must be checked that each other component detected so far fits into an inner face of the ring or into its outer face. First we consider the case that the other components do not contain a ring. For the efficient execution of the additional checks the algorithm maintains the lowest level minLL where an insertion of such a component is necessary.

Definition 5 *A* completely processed PQR-tree *is a PQR-tree representing a component of the graph not having any vertices on the current or on higher levels.* $\mathrm{minLL} = \min(\{\,\mathrm{LL}(T) \mid T$ is a completely processed PQR-tree without an R-root $\}\cup\{\infty\})$. *If there is no completely processed tree T then* $\mathrm{minLL} = \infty$.

The detection of a processed PQR-tree T works as follows: After every call of REPLACE-SINGLE we check whether T consists of a single leaf (or an R-node with one leaf) and whether the vertex represented by this leaf is a sink of

the graph. As soon as a PQR-tree T is classified as completely processed after REPLACE-SINGLE, minLL is updated by $\min\{\text{minLL}, \text{LL}(T)\}$. All processed PQR-trees are discarded as in the JLM testing algorithm. It suffices to check whether the component C of the completely processed PQR-tree starting at the lowest level fits into an internal face. For all other processed (non-ring) components there is enough space to embed them in the same face as C. Inner faces are always closed by a call of REPLACE-SINGLE for a vertex v. If there is a processed PQR-tree without an R-root, i. e., if minLL $< \infty$, we check if the newly created inner face starting at the lowest level can include C. Here we use the same mechanism as JLM do for v-singular forms and compare minML with the new PML/QML value, see p. 61. If minLL $>$ PML or minLL $>$ QML, we set minLL $= \infty$. Otherwise, we need not care whether another processed component smaller than C and whose PQR-tree has already been discarded can be nested inside a face without violating planarity. These will fit later when a face for C is found. If no such face can be found, the graph is not radial level planar. Recall that a processed PQR-tree with an R-root cannot be included in this way. Their nesting is described in the next section.

4.5 Nesting of Processed Rings

Our algorithm maintains the invariant that at any time while testing a radial level graph there is at most one PQR-tree T^R with an R-root. T^R may be processed. A *link vertex* v denotes the vertex for which the reduction of all leaves labelled with v makes a ring out of the component represented by the PQR-tree. At the start of the algorithm T^R is undefined and the invariant is obviously true. In the further process it is maintained as follows: If the algorithm detects a ring for the very first time, T^R is defined and remains defined until the end of the algorithm, although the tree for T^R may change. If another PQR-tree T gets an R-root by the application of template P8, Q4 or Q6 during the reduction of a link vertex v, we proceed as described by Algorithm 7.

Algorithm 7: TREAT-NEW-RING

Input: The PQR-tree T of a newly encountered ring, the link vertex v, T^R, and minLL

Output: A boolean value indicating whether radial level planarity is preserved

if $T^R \neq NULL$ **then**
  if T^R *is not processed and* T^R *is not* v*-singular* **then return false**
  minML $\leftarrow \min\{$ML between the children of the root of $T^R\}$
  if minML $\geq$ LL(T) *or* minML $\geq$ minLL **then return false**
  delete T^R
end
$T^R \leftarrow T$

return true

If there is a PQR-tree T^R with an R-node as root, it must be either completely processed or v-singular. Otherwise, G is not radial level planar as we have seen in Section 4.3. The algorithm checks whether minML is small enough for T to fit below T^R. Moreover, the tree with the smallest low indexed level minLL and thus all other trees must fit between T and T^R. Recall that before the nesting T^R must be rotated and squeezed such that all its jags are embedded into the space above v and that the indentation of T^R with the minML meet level encloses all inner jags of T. See Figure 12 for an illustration. The rotation of T^R is not done explicitly because T^R is discarded anyway.[1] If any of the checks fails then by Lemma 8 G is not radial level planar. Finally T^R is updated. The algorithm is constructed to preserve the following invariant:

Lemma 11 *At any time while testing a radial level graph, the collection of trees $\mathcal{T}$ contains at most one PQR-tree with an R-node as its root.*

4.6 Completion

Finally, if there is no PQR-tree T^R representing a ring graph, the graph is level planar. Otherwise, if no other trees have occurred after T^R has been detected, the graph is radial level planar. This is the case if $\text{minLL} = \infty$. If $\text{minLL} < \infty$ it remains to check whether the other PQR-trees fit below T^R, i. e., $\text{minML} < \text{minLL}$. Otherwise, G is not radial level planar.

4.7 Correctness

For the correctness of the algorithm every computed embedding of a ring must be level optimal, and this property is granted by our algorithm.

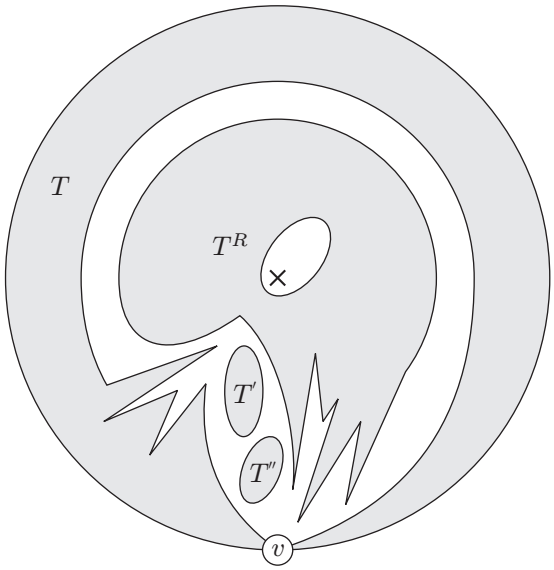
Lemma 12 *Our algorithm for testing radial level planarity induces a level optimal embedding for every ring.*

Proof: Let R be a ring of the given graph. As long as the corresponding PQR-tree does not contain an R-node, the centre of the concentric levels lies in the outer face. Only the templates P8, Q4, and Q6 introduce a new R-node which closes the centre face. This does not cover the case shown in Figure 13, where two nested rings share a common vertex on a lower level than the link vertex of the outer ring. Then the centre face of the outer ring is closed by the application of template R4.

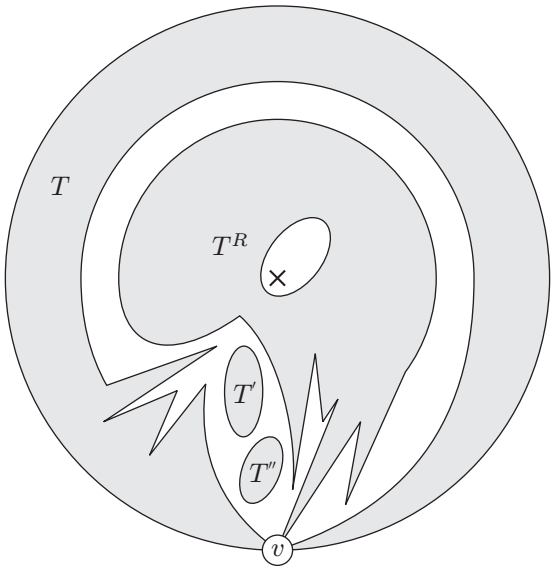
As these four templates are only applied if no other template matches, there is no admissible permutation which allows to close the centre face on a higher level. Hence, the centre face ends on level β_R. Note that inserting v-singular forms into the centre face of R does not influence β_R.

Each PQR-tree representing a ring R has an R-node as its root. At any time during the application of the algorithm the indentations of the outer face

[1] When computing an embedding this has already be done by an earlier application of the embedding variant of template R1 as we will see later in Section 5.2.

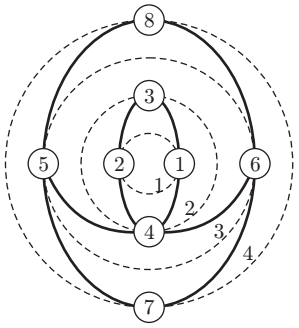


(a) T^R is completely processed.

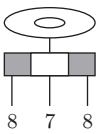


(b) T^R is v-singular.

Figure 12: Schematic nesting of rings. T' and T'' correspond to two other completely processed components.



(a) Vertex 4 links two rings.



(b) Before closing the outer ring at link vertex 8.

Figure 13: Linked and nested rings.

are represented by the ML-values between two siblings. The meet levels stored between a node and its siblings are always less or equal to those between its children. Thus the least values are stored between children of the root and minML represents the highest indentation of the outer border of R, i. e., the border of the outer face of R in the embedding. The value of minML can only change if an inner face is closed by REPLACE-SINGLE. Then there may be several faces which can be closed due to the freedom of rotation. Which one is taken only depends on the templates applied in REDUCE. Only template R1 has multiple options. Since R1 always preserves the minimum meet level as shown in Section 4.2, it is guaranteed that the highest possible indentation is preserved whenever this is possible. Thus the outer radius $\gamma_R = \text{minML}$ is level optimal in the induced embedding. □

Lemma 13 *The REDUCE operation, extended by the new templates from Figure 9 and Figure 10, correctly computes the new set of admissible permutations for radial level planarity.*

Proof: We follow the corresponding arguments for PQ-trees in [5, p. 348f]. It must be shown that no template violates radial level planarity. This is obvious because the templates are constructed exactly that way. Further, it must be shown that any radial level planar graph can be processed successfully, i. e., no further templates are necessary. This is true because in all cases where no template can be applied, the graph is not radial level planar. This is shown easily by considering all possible constellations of node types and the order of empty, full, and (boundary) partial children. □

In analogy to Jünger et al. [24–26, 28] this implies our first main theorem:

Theorem 1 *There is an $\mathcal{O}(|V|)$ time algorithm for testing radial k-level planarity.*

5 Radial Level Planar Embedding

Algorithm 6 describes the algorithm of Jünger et al. [24, 25, 28] for computing level planar embeddings of level planar graphs. This algorithm can be extended to compute radial level planar embeddings of radial level planar graphs. In addition to an ordering of the vertices our algorithm determines clockwise and counterclockwise cut edges. We extend the upward embedding algorithm of Chiba et al. [9] to work with PQR-trees instead of PQ-trees and present a new algorithm for generating a radial level planar embedding from the upward embedding.

5.1 Meet Levels between Ignored Siblings

When computing an embedding, PQ-trees can contain ignored nodes, see Section 2.3. Since we use the same strategy for computing radial embeddings, we have to treat ignored nodes. This is particularly important when minML is computed because we have to consider ML-values between any pair of adjacent children of the R-node. This includes ignored children. Therefore, we have to ensure that the ML-values of ignored nodes are computed correctly. For example, the outer ML-values have to be initialised when a Q-node with outermost ignored children is inserted into another one. This is straightforward and can be done in constant time.

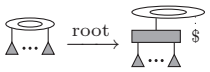
5.2 Embedding the Edges

We not only have to compute a vertex ordering on each level j but also the edge routing. It is not necessary to sort the adjacent edges of each vertex as it has been done in [9], but it suffices to determine cut edges. Cut edges are detected by the *st*-embedding creation step described in Section 5.4 and not during the augmentation phase.

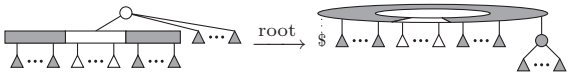
Initially there are no edges marked as cut edges. They are recognised as follows: For an R-node we introduce a new child denoted by *ray indicator* and labelled with \$ which marks where the ray splits the children. Like the sink indicators the ray indicator is ignored throughout the algorithm and it always remains a child of the R-node. It is created with every R-node by modified templates P8, Q4, and Q6, see Figure 14.

R1 has to be modified, too. Recall that R1 creates a pseudo Q-node X'. Before this is done the R-node is rotated such that the two siblings with minML in between become the end vertices of X'. Otherwise, level optimality can get lost. See Figure 15 for an illustration.

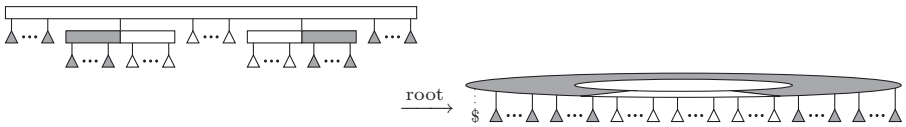
The ray indicator \$ may divide the children of X' into two parts. Thus before X' is created it is necessary to drag one part over \$ because it must remain a child of the R-node. The leaves of all pertinent subtrees that are dragged over the ray indicator represent cut edges. They can be computed by DFS without violating the $\mathcal{O}(|V|)$ time bound since REPLACE removes the subtrees from the PQR-tree after each drag operation. Accordingly, if the ray indicator in



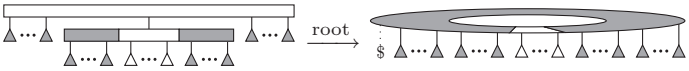
(a) Template R1.



(b) Template P8.

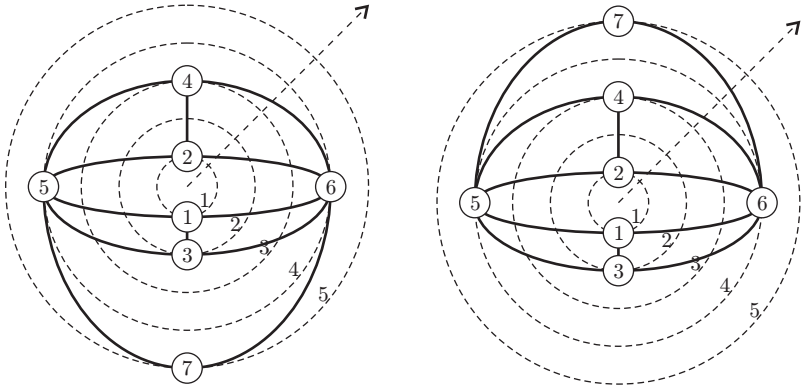


(c) Template Q4.



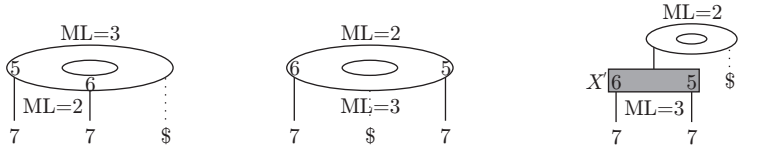
(d) Template Q6.

Figure 14: Radial level planarity with level embedding templates.



(a) Not level optimal.

(b) Level optimal.

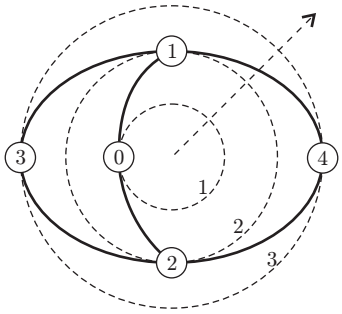


(c) Initial PQR-tree on level 4.

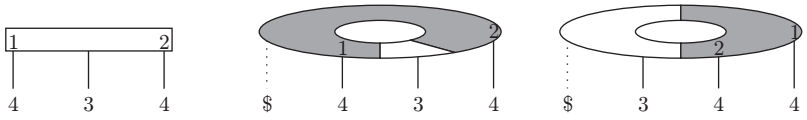
(d) After rotation.

(e) Final PQR-tree.

Figure 15: Preserving level optimality.



(a) The input graph.



(b) The PQR-tree before Q4.

(c) Figure 16(b) after REDUCE.

(d) $(1, 4)$ is dragged over \$.

Figure 16: Detection of a cut edge while reducing the leaves of vertex 4.

REPLACE lies within the pertinent sequence, one part of the pertinent sequence is dragged over the ray indicator before the pertinent sequence is replaced. For an example consider the graph shown in Figure 16(a). Figure 16(b) shows its corresponding PQR-tree before the reduction of all leaves with label 4, whereas Figure 16(c) shows the resulting PQR-tree after the reduction by template Q4. As shown in Figure 16(d) the leaf representing the edge $(1, 4)$ is dragged over \$ in REPLACE and thus the edge $(1, 4)$ becomes a cut edge.

5.3 Augmentation to an *st*-Graph

The processed PQR-trees from Section 4 are now called *ignored PQR-trees* because they consist of ignored nodes only, cf. Section 2.3. However, the LL-value of the highest ignored PQR-tree minLL is not sufficient here. We also must store the ignored PQR-trees, because their sinks must later be augmented with edges if a ring is closed by template P8, Q4, or Q6. Then all sinks are connected to the link vertex w on which REDUCE was called and which closes the ring. The embedding of ignored components within a newly encountered ring never introduces crossings because $\phi(w) > \phi(u)$ for each ignored sink u. Further, it can be the case that connecting to a vertex v is necessary if a face is closed by REPLACE-SINGLE for v. Hence, we maintain a *collection* $\mathcal{T}^*$ which stores all ignored PQR-trees in addition to the *active collection* $\mathcal{T}$.

When an R-node is created, an existing PQR-tree with an R-root is nested into the centre face. This includes an ignored PQR-tree with an R-root. Only a single PQR-tree T^R is left. In analogy to Lemma 11, this leads directly to the following property:

Lemma 14 *$\mathcal{T} \cup \mathcal{T}^*$ contains at most one R-rooted PQR-tree T^R.*

If a vertex v closes a face, it does not suffice to test whether the highest PQR-tree fits into this face after REPLACE-SINGLE. If it fits, additionally all sinks in $\mathcal{T}^*$ are connected to v and $\mathcal{T}^*$ is emptied. Similar to the radial level planarity test, if there exists an ignored R-rooted PQR-tree T^R, this step is omitted for a face different from the centre face. Rings cannot be embedded within faces not containing the centre. The other PQR-trees in $\mathcal{T}^*$ are embedded later in the same face as T^R. If they do not fit in this face, the graph is not radial level planar because previous tests have shown that they do not fit in an inner face of the ring represented by T^R, too.

The tests whether minLL and the LL-value of a newly detected ring are greater than the minML-value of an enclosed ignored ring can be omitted as an optimisation. These checks are done in the bottom up phase with the single hierarchy rooted at t. However, the sinks have to be connected to the link vertex.

If a PQR-tree contains ignored nodes, the templates P8, Q4, and Q6 can be applied to nodes other than the root of a PQR-tree. There may be a path from the PQ-node X to the root which is the only non-ignored path from the root downwards, i. e., all predecessors of X have only one non-ignored child. Then all vertices represented by nodes that are not descendants of X can be embedded within the ring represented by the new R-root. Therefore, these nodes are removed and the corresponding sinks are connected to the link vertex. The $\mathcal{O}(|V|)$ time bound is preserved. If the test on the above situation fails either the input graph is not radial level planar and the algorithm rejects or there is a similar situation to the one shown in Figure 11 and other templates fit. This case can be checked in $\mathcal{O}(1)$ time since there is no node chain in a PQR-tree and thus the parent Q-node of X has at least one other non-ignored child. If the test does not fail, the traversed nodes are removed. Hence, the total computation time remains linear.

5.4 Computation of an *st*-Embedding

To compute an *st*-embedding $\mathcal{E}_{st}$ of the graph G_{st} (see Algorithm 6) the algorithm of Chiba et al. [9] is used. It is based on the vertex addition method of [17, 29] and needs an *st*-graph. But in our case G_{st} has no *st*-edge (s, t). If G is a ring graph, s and t are not in the same face of any planar level embedding $\mathcal{E}_l$ of G, i. e., s does not lie in the outer face as t does, cf. Lemma 4. Therefore, the introduction of a new edge (s, t) as in the JLM algorithm is not possible since it may destroy radial level planarity and the *st*-embedding algorithm would fail. Thus we omit introducing the edge (s, t) and obtain only

an induced st-numbering by numbering the vertices level by level in ascending order. After augmentation each vertex except s and t has at least one incoming and at least one outgoing edge. There are no other sources than s and no other sinks than t. Without the st-edge, G_{st} may be not biconnected.

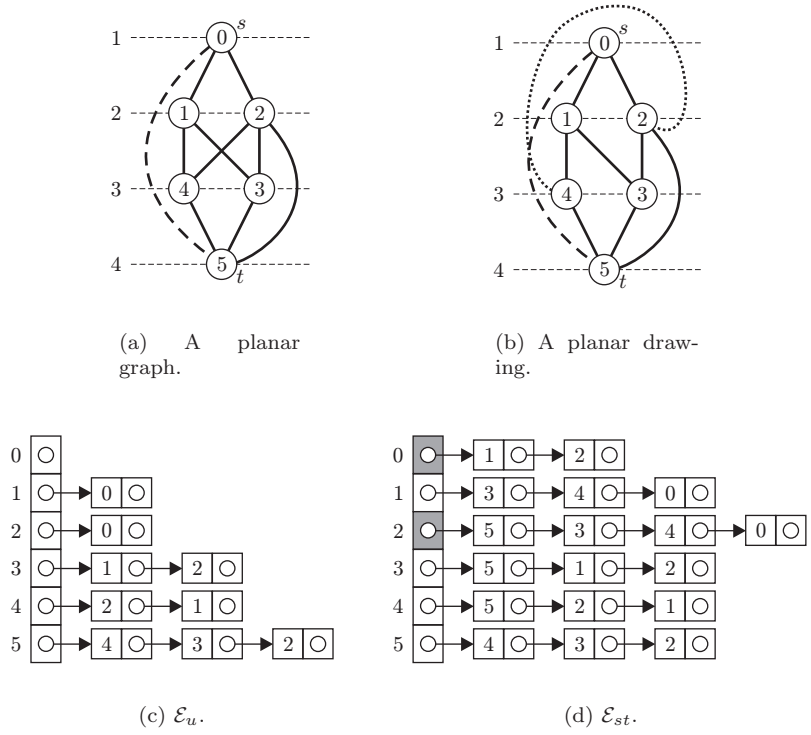
If an embedding is computed by the standard vertex addition method [9,17, 29], the edge (s,t) is similar to the ray in the radial level planarity test. The st-edge is real, however, and therefore no other edge is allowed to cross it. Thus cyclic reductions, i. e., cut edges, are not allowed and need not be considered. Without (s,t) cyclic reductions are admissible. We adopt our ideas from extending the level planarity test to the radial case. The standard planar embedding algorithm is updated with the PQR-tree data structure to realise cyclic reductions. Again, we omit ENTIRE-EMBED of Chiba's algorithm for computing an st-embedding $\mathcal{E}_{st}$ from the upward st-embedding $\mathcal{E}_u$, cf. Section 2.3. Here the reason is both efficiency and correctness. In the radial case the upward embedding $\mathcal{E}_u$ can be seen as an *inward embedding*. In our approach it is possible to route edges around s. The routing around t is not allowed because we consider only monotone level planar graphs. Figure 17(b) without the dashed st-edge is a radial level planar drawing of the graph shown in Figure 17(a). If cut edges exist, Chiba's DFS may provide an invalid edge ordering around each vertex. The adjacency lists of vertices 0 and 2 in Figure 17(d) are incorrect, while in $\mathcal{E}_u$ the orderings of the incoming edges are correct, see Figure 17(c). Thus, we use $\mathcal{E}_u$ instead of $\mathcal{E}_{st}$ to compute a radial level planar embedding $\mathcal{E}_l$ in Section 5.5.

The procedure UPWARD-EMBED of [9, p. 67f] relies on the fact that the leaves which are removed from the PQR-tree by REPLACE for storing the represented edges in $\mathcal{E}_u$ are in an admissible order except for reversion. Possible subsequent reversions of a parent Q-node are handled by direction indicators. Reversions of a parent R-node X are accomplished accordingly. However, if the ray indicator occurs within the pertinent sequence of X, we have to drag a part of the sequence over it. This is done before the removal of the pertinent sequence. Later in the algorithm there is the possibility of a rotation of X and thus of an implicit rotation of its children. However, this only means a rotation of the whole graph including the ray. Hence, the ordering of the stored sequence remains valid. If an R-node has only pertinent children then it is admissible to move the ray indicator arbitrarily, leading to different cut edges and thus to different embeddings. This is not significant because we are interested in a single admissible embedding. Thus analogously to UPWARD-EMBED of Chiba et al. we obtain a valid inward embedding.

5.5 Computation of a Radial Level Embedding

In this section we assume that in the upward embedding $\mathcal{E}_u$ the incoming edges of every vertex are sorted in clockwise order. Before we present our algorithm for computing a radial level embedding $\mathcal{E}_l$ we establish further properties.

Lemma 15 *Let $G_{st} = (V_{st}, E_{st})$ be the augmented st-graph. Then every vertex $v \in V_{st} - \{s\}$ has at least one incoming non-cut edge.*



(a) A planar graph.

(b) A planar drawing.

(c) $\mathcal{E}_u$.

(d) $\mathcal{E}_{st}$.

Figure 17: Embedding an edge around s without the (dashed) st-edge. The numbers in the vertices not only show their label but also represent their induced st-numbers.

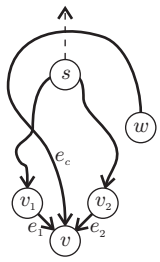


Figure 18: No cut edge can be between two non-cut edges.

Proof: G_{st} is an induced st-graph without an st-edge. Thus every vertex $v \in V_{st} - \{s\}$ has at least one incoming edge. An edge is only marked as a cut edge in REPLACE or if in template R1 the ray indicator lies within the pertinent sequence. In both cases there are PQ-leaves representing edges on both sides of the ray indicator. They must be placed on the same side of the ray indicator. Thus the edges which are not dragged over \$ are non-cut edges. If \$ already is at the beginning or at the end of the pertinent sequence, there are no cut edges. □

Corollary 1 *To any vertex $v \in G_{st}$ there exists a path from s not containing a cut edge.*

Lemma 16 *In any upward embedding $\mathcal{E}_u$ the ordered adjacency list of a vertex v never contains a cut edge between two non-cut edges.*

Proof: Assume that v has adjacent incoming edges in the ordering e_1, e_c, e_2, where e_1 and e_2 are non-cut edges and e_c is a cut edge, see Figure 18. Let v_1 be the source vertex of e_1 and v_2 the source vertex of e_2. Then $v_1 \neq v_2$. Thus there exist two paths p_1 and p_2 from s to v_1 and from s to v_2, respectively, which according to Corollary 1 differ in at least one edge. The cut edge e_c violates planarity by crossing the boundaries of the face between p_1 and p_2, which is a contradiction. □

This leads to two different types of cut edges according to their position in the adjacency list. We call them clockwise or counterclockwise according to the implicit direction from lower to higher levels.

Definition 6 *A cut edge is called* clockwise *with respect to $\mathcal{E}_u$ if it occurs at the right end of the incoming adjacency list of its target vertex. Otherwise, it is called* counterclockwise.

Lemma 17 *All cut edges of a radial level planar embedding that end on the same level have the same direction (clockwise or counterclockwise) and the same target vertex.*

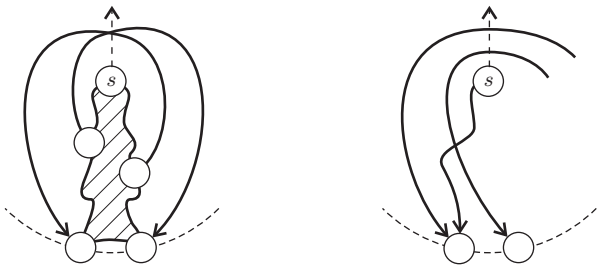


Figure 19: In a radial level planar graph all cut edges ending on the same level have the same direction and the same target vertex.

Proof: First we show that all cut edges with their target vertex on the same level have the same direction. Assume two cut edges with different directions ending on the same level. Their target vertices need not be different. The obtained crossing, see Figure 19(a) for an illustration, contradicts radial level planarity. This crossing cannot be avoided because there are paths from s to the source vertices of the cut edges according to Corollary 1.

It remains to show that there is at most one vertex with incoming cut edges on a level. Assume a level with two vertices which both have an incoming cut edge. We have already shown that they have the same direction. Then the inner cut edge crosses a path from s to the target of the other cut edge, see Figure 19(b) for an illustration. Such a path always exists because of Corollary 1. This contradicts radial level planarity. □

Because of the above lemmata we introduce Algorithm 8, CONSTRUCT-LEVEL-EMBED. $\mathcal{E}_l[j]$ denotes the ordered vertex list of the radial level j. The algorithm is a sequence of ordered backward DFS traversals in $\mathcal{E}_u$ which use no cut edges. The first of these traversals starts at the sink vertex t and inserts every visited vertex v at the right end of $\mathcal{E}_l[\phi(v)]$. The part of the graph visited in this first step is called *trunk*, see Figure 20. Source vertices of discovered cut edges are placed into a queue together with the information on which side of the trunk they have to be placed later. It is important that these vertices are inserted into the queue in the correct order, from right to left for incoming clockwise cut edges and from left to right for incoming counterclockwise cut edges. The subsequent DFS traversals start at a vertex from the queue and insert visited vertices at the respective side of $\mathcal{E}_l$. Source vertices of newly detected cut edges are again inserted into the queue. The algorithm terminates when the queue is empty and thus all vertices have been visited. For proving the correctness of the algorithm we introduce another lemma.

Algorithm 8: CONSTRUCT-LEVEL-EMBED

Input: The upward embedding $\mathcal{E}_u$ and the st-graph $G_{st} = (V_{st}, E_{st})$
Output: A level embedding $\mathcal{E}_l$

```
procedure DFS(v, dir)
    if visited[v] = false then
        visited[v] ← true
        if dir = left then insert v at the left end of E_l[φ(v)]
        else insert v at the right end of E_l[φ(v)]
        foreach incoming non-cut edge e of v scanned in direction dir do
            DFS(source(e), dir)
        end
        if v has incoming clockwise cut edges then
            foreach incoming cut edge e of v scanned from right to left do
            insert(Q, source(e), left)
        else
            foreach incoming cut edge e of v scanned from left to right do
            insert(Q, source(e), right)
        end
    end
end
foreach v ∈ V_st do visited[v] ← false
Queue Q                                   // stores pairs
insert(Q, t, right)
while Q not empty do DFS(delete_first(Q))
return E_l
```

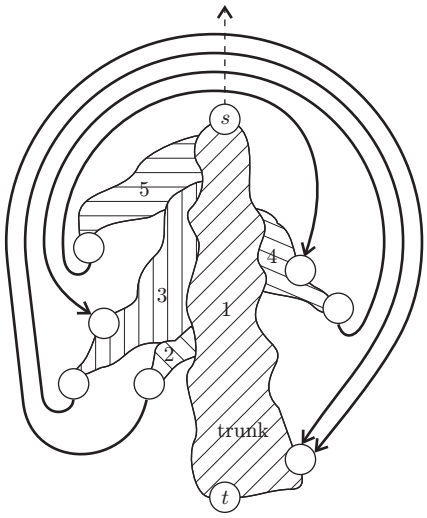


Figure 20: Successive and ordered attachments of faces to the sides of the trunk.

Lemma 18 *Let G be a level graph with a single sink assuming edge directions from lower to higher levels. Then an upward embedding of G induces a unique level embedding.*

Proof: Assume two different level embeddings of G. Then there are two vertices u and v on the same level whose relative positions differ in the two embeddings. From both vertices there exists a path to the sink. Let w be the first common vertex on these paths and let u' and v' be the direct predecessors on the respective paths, see Figure 21. Since the paths from u to u' and from v to v' are disjoint and do not cross, the edges (u', w) and (v', w) have different relative positions in the incoming adjacency list of w and thus contradict the common upward embedding. □

Theorem 2 *Algorithm 8 constructs a valid radial level planar embedding of the given upward embedding $\mathcal{E}_u$ in $\mathcal{O}(|V|)$ time.*

Proof: Since the algorithms performs DFS only with different parts of the graph one after the other, the $\mathcal{O}(|V|)$ running time is obvious. To see the correctness the algorithm starts at t and first traverses the trunk. This is the same mechanism as in Algorithm 6. A *branch* is a subgraph that is traversed with a single invocation of DFS. Since each branch is level planar and meets the requirements of Lemma 18, there is a unique level embedding for it. Because the side of the trunk on which the branches are placed is determined by the cut edges that led to them, it only remains to be shown that they are attached in the correct order. This ensures the processing order of the cut edges, which are

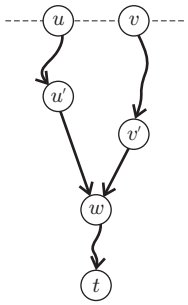


Figure 21: An upward embedding induces an unique level embedding.

attached on the left side of the trunk from right to the left, on the right side of the trunk from left to right, and in each case from bottom to the top. □

6 Conclusion

We have presented new linear time algorithms for testing radial level planarity and computing radial level planar embeddings. They can easily be extended to *circle planarity* testing and embedding [2], where edges having both vertices on the same level are allowed.

Further investigations are required to expand the test algorithms for level planarity for detecting the so called *minimum level non-planar subgraph patterns (MLNP-patterns)* if the tested graph is level non-planar. MLNP-patterns for level graphs are characterised in [19] and are the counterparts of the Kuratowski Graphs $K_{3,3}$ and K_5 for graph planarity which can efficiently be computed [27, 30, 35]. Similar patterns for the radial case are desirable, see [1] for first steps in that direction. As already mentioned in the conclusion of [28, p. 211] the detection of MLNP-patterns can also be used to verify the results of a (radial) level planarity test. Since such a test is a non-trivial algorithm and thus it is not unlikely that an implementation is faulty, it is desirable to not only prove planarity by an embedding or by a drawing but also to show non-planarity on the basis of MLNP-patterns.

Another interesting topic is the generalisation to non-monotonic edges while the levels of the vertices remain fixed. How can a graph be tested and embedded efficiently for non-monotone variations of (radial) level planarity?

References

[1] C. Bachmaier. *Circle Planarity of Level Graphs*. Dissertation, University of Passau, 2004.

[2] C. Bachmaier, F. J. Brandenburg, and M. Forster. Track planarity testing and embedding. In P. Van Emde Boas, J. Pokorný, M. Bieliková, and J. Štuller, editors, *Proc. Software Seminar: Theory and Practice of Informatics, SOFSEM 2004*, volume 2, pages 9–17. MatFyzPres, 2004.

[3] C. Bachmaier, F. Fischer, and M. Forster. Radial coordinate assignment for level graphs. In L. Wang, editor, *Proc. Computing and Combinatorics, COCOON 2005*, volume 3595 of *LNCS*, pages 401–410. Springer, 2005.

[4] C. Bachmaier and M. Raitner. Improved symmetric lists. Technical Report MIP-0409, University of Passau, October 2004.

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

[6] U. Brandes, P. Kenis, and D. Wagner. Centrality in policy network drawings. In J. Kratochvíl, editor, *Proc. Graph Drawing 1999*, volume 1731 of *LNCS*, pages 250–258. Springer, 1999.

[7] U. Brandes, P. Kenis, and D. Wagner. Communicating centrality in policy network drawings. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):241–253, 2003.

[8] M. Chandramouli and A. A. Diwan. Upward numbering testing for triconnected graphs. In F. J. Brandenburg, editor, *Proc. Graph Drawing 1995*, volume 1027 of *LNCS*, pages 140–151. Springer, 1996.

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

[10] H. de Fraysseix, J. Pach, and R. Pollack. Small sets supporting Fáry embeddings of planar graphs. In *Proc. ACM Symposium on Theory of Computing, STOC 1988*, pages 426–433. ACM Press, 1988.

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

[12] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[13] G. Di Battista and E. Nardelli. Hierarchies and planarity theory. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(6):1035–1046, 1988.

[14] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996.

[15] V. Dujmović, M. Fellows, M. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. Rosamond, M. Suderman, S. Whitesides, and D. R. Wood. On the parameterized complexity of layered graph drawing. In F. Meyer auf der Heide, editor, *Proc. European Symposium on Algorithms 2001*, volume 2161 of *LNCS*, pages 488–499. Springer, 2001.

[16] P. Eades. Drawing free trees. *Bulletin of the Institute of Combinatorics and its Applications*, 5:10–36, 1992.

[17] S. Even. *Algorithms*, chapter 7, pages 148–191. Computer Science Press, 1979.

[18] P. Healy and A. Kuusik. The vertex-exchange graph: A new concept for multi-level crossing minimisation. In J. Kratochvíl, editor, *Proc. Graph Drawing 1999*, volume 1731 of *LNCS*, pages 205–216. Springer, 1999.

[19] P. Healy, A. Kuusik, and S. Leipert. Characterization of level non-planar graphs by minimal patterns. In D.-Z. Du, P. Eades, V. Estivill-Castro, X. Lin, and A. Sharma, editors, *Proc. Computing and Combinatorics, COCOON 2000*, volume 1858 of *LNCS*, pages 74–84. Springer, 2000.

[20] L. S. Heath and S. V. Pemmaraju. Recognizing leveled-planar dags in linear time. In F. J. Brandenburg, editor, *Proc. Graph Drawing 1995*, volume 1027 of *LNCS*, pages 300–311. Springer, 1996.

[21] L. S. Heath and S. V. Pemmaraju. Stack and queue layouts of directed acyclic graphs: Part II. *SIAM Journal on Computing*, 28(5):1588–1626, 1999.

[22] L. S. Heath and A. L. Rosenberg. Laying out graphs using queues. *SIAM Journal on Computing*, 21(5):927–958, 1992.

[23] C. Hundack, P. Mutzel, I. Pouchkarev, and S. Thome. ArchE: A graph drawing system for archeology. In G. Di Battista, editor, *Proc. Graph Drawing 1997*, volume 1353 of *LNCS*, pages 297–302. Springer, 1998.

[24] M. Jünger and S. Leipert. Level planar embedding in linear time. In J. Kratochvíl, editor, *Proc. Graph Drawing 1999*, volume 1731 of *LNCS*, pages 72–81. Springer, 1999.

[25] M. Jünger and S. Leipert. Level planar embedding in linear time. *Journal of Graph Algorithms and Applications*, 6(1):67–113, 2002.

[26] M. Jünger, S. Leipert, and P. Mutzel. Level planarity testing in linear time. In S. H. Whitesides, editor, *Proc. Graph Drawing 1998*, volume 1547 of *LNCS*, pages 224–237. Springer, 1998.

[27] A. Karaberg. Classification and detection of obstructions to planarity. *Linear and Multilinear Algebra*, 26:15–38, 1990.

[28] S. Leipert. *Level Planarity Testing and Embedding in Linear Time*. Dissertation, Mathematisch-Naturwissenschaftliche Fakultät der Universität zu Köln, 1998.

[29] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In P. Rosenstiehl, editor, *Proc. of Theory of Graphs, International Symposium, Rome*, pages 215–232. Gordon and Breach, 1967.

[30] K. Mehlhorn and S. Näher. *LEDA, A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[31] H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In G. Di Battista, editor, *Proc. Graph Drawing 1997*, volume 1353 of *LNCS*, pages 248–261. Springer, 1997.

[32] B. Randerath, E. Speckenmeyer, E. Boros, P. Hammer, A. Kogan, K. Makino, B. Simeone, and O. Cepek. A satisfiability formulation of problems on level graphs. Rutcor Research Report RRR 40-2001, Rutgers Center for Operations Research, Rutgers University, 2001.

[33] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.

[34] J. D. Ullman. *Computational Aspects of VLSI*, chapter 3.5, pages 111–114. Computer Science Press, 1984.

[35] S. G. Williamson. Depth-first search and Kuratowski subgraphs. *Journal of the ACM*, 31(4):681–693, 1984.