Radial Level Planarity Testing and Embedding in Linear Time*

Technical Report MIP-0303
June, 2003

Christian Bachmaier, Franz J. Brandenburg, and Michael Forster

University of Passau,
94030 Passau, Germany
Fax: +49 851 509 3032
{bachmaier,brandenb,forster}@fmi.uni-passau.de

Abstract. Every planar graph has a concentric representation based on a breadth first search, see [24]. The vertices are placed on concentric circles and the edges are routed as curves without crossings. Here we take the opposite view. A graph with a given partitioning of its vertices onto k concentric circles is k-radial planar, if the edges can be routed monotonic between the circles without crossings. Radial planarity is a generalisation of level planarity, where the vertices are placed on k horizontal lines. We extend the technique for level planarity testing of [13, 14, 16–18, 20] and show that radial planarity is decidable in linear time, and that a radial planar embedding can be computed in linear time.

1 Introduction

The display of hierarchical structures is an important issue in automatic graph drawing. Directed acyclic graphs (DAGs) and ordered trees are usually drawn such that the vertices are placed on horizontal levels, and the edges are drawn as straight lines or as y-monotone polylines. This technique is used by the Sugiyama algorithm, the most common algorithm for drawing DAGs [6, 23]. After the calculation of a level assignment, the algorithm tries to minimise the number of edge crossings. In the best case there are no crossings at all, and the graph is level planar. Level planarity has been investigated intensively, and there are linear time algorithms both for the test of k-level planarity and for the computation of an embedding.

A *k-level graph* $G = (V, E, \phi)$ is an undirected graph with a level assignment $\phi\colon V \to \{1, 2, \ldots, k\}$, $1 \leq k \leq |V|$, that partitions the vertex set into $V = V^1 \dot\cup V^2 \dot\cup \cdots \dot\cup V^k$, $V^j = \phi^{-1}(j)$, $1 \leq j \leq k$, 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 $(u, v) \in E$.

* This research has been supported in part by the Deutsche Forschungsgemeinschaft, grant BR 835/9-1.

The level planarity problem [7,13,18] can be formulated as follows: Is it possible to draw a level graph G in the Cartesian plane such that all vertices $v \in V^j$ of the j-th level are placed on a single horizontal line $l_j = \{ (x, j) \mid x \in \mathbb{R} \}$ and the edges are drawn as strictly y-monotone curves without crossings. For a planar k-level drawing with the above restrictions an embedding of the graph has to be computed. Level planar embeddings are characterised by linear orderings $\leq_j$ of the vertices in each V^j, $1 \leq j \leq k$, which is the order of the vertices from left to right. See Fig. 1 for an example of a level planar graph.

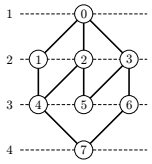
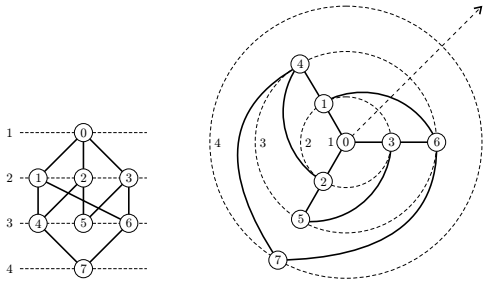


Fig. 1. A level planar graph

Note that we do not consider the problem of finding a level planar or radial planar embedding for graphs without a given levelling. Heath and Rosenberg [15] have shown that the levelling problem is NP-hard for proper graphs. In the non-proper case, i. e., with arbitrarily long edges, each planar graph has a k-level planar levelling. This follows for example from the planar grid drawings of de Fraysseix et al. [5]. There neither the number of levels nor the length of the edges is taken into account, e. g., for a minimisation.

Level planarity can be generalised to radial level planarity or short radial planarity. In contrast to the above the vertices are not drawn on k horizontal lines but on k concentric circle lines $l_j = \{ (j \cos\theta, j \sin\theta) \mid \theta \in [0, 2\pi) \}$, $1 \leq j \leq k$. A k-level graph is *radial k-level planar* if there are orderings $\leq_j$ of the vertices on each radial level such that edges are drawn as strictly monotone curves from inner to outer levels without crossings. The transformation of a level planar embedding to a radial planar embedding can be obtained by connecting the ends of each level and thus forming concentric level circles. This allows the insertion of some additional edges connecting the end of one level with the beginning of another. These *cut edges* cross an imaginary ray from the centre of the concentric levels to infinity through the points where the connected fronts and backs of the levels meet. There are two directions for routing cut edges around the centre, clockwise and counterclockwise. As an extension to level planar embeddings, *radial planar embeddings* need additional information about cut edges and their direction. Figure 2(b) shows a radial planar drawing of the graph in Fig. 2(a) which is not level planar. The edge $(1, 6)$ crosses the imaginary ray and thus is a clockwise cut edge, following its implicit direction from lower to higher levels. Obviously, a radial planar graph is level planar, if there are no cut edges.



(a) (b)

Fig. 2. A radial planar graph with a radial planar drawing

Lemma 1. *Let G be a k-level graph. Then the following holds:*

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

Proof. The correctness follows directly from the definitions. □

The more general approach of Dujmović et al. [9] leads to a linear time fixed parameter tractable algorithm for detecting radial planar graphs for a fixed number of levels. We give a practical algorithm based on the level planarity test of Leipert et al. [16–18,20] that improves this result to $\mathcal{O}(|V|)$ time for an arbitrary number of (non empty) levels.

Without loss of generality, we only consider simple graphs without self loops and parallel edges. Because of Lemma 1 a input graph with $|E| > 3|V| - 6$ is rejected as not radial planar. In the following $V(G)$ denotes the vertex set of a graph G and $E(G)$ its edge set respectively.

This paper is organised as follows: After a brief outline of the linear level planarity testing and embedding algorithm of Leipert et al. in the next section, we present our linear time approach to decide whether a graph is radial planar in Sect. 3. Section 4 describes how a radial level planar embedding can be found with our algorithm within the same linear time bounds. We close with a summary in Sect. 5.

2 Related Work

The basis of our algorithm is the linear time algorithm of Leipert et al. [16–18,20] for level planarity testing and embedding which is in turn based on previous work of Heath and Pemmaraju [13,14]. These algorithms extend the planarity test for *hierarchies*[1] of Di Battista and Nardelli [7] to arbitrary level graphs. Chan-

[1] A hierarchy is a level graph with a single *source*, where a source is a vertex having only edges to a higher level. A *sink* is defined analogously.

dramouli and Diwan [3] present a linear time algorithm to determine whether a triconnected DAG is level planar.

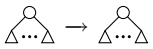
2.1 Linear Time Level Planarity Test

The basic idea of the algorithm is to perform a top down sweep, processing the levels in order $1, 2, \ldots, k$ and to compute for every level V^j, $1 \leq j \leq k$, a set of permutations of V^j that appear in some level planar embedding of G^j. G^j is the subgraph induced by $V^1 \cup V^2 \cup \cdots \cup V^j$. G is level planar if and only if the set of permutations of $G^k = G$ is not empty.

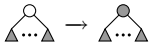
In order to represent and store sets of vertex permutations efficiently a data structure called PQ-tree, introduced by Booth and Lueker [2], is used. A PQ-tree represents the permutations of a finite set S in which the members of specified subsets of S occur consecutively. It is a rooted and ordered tree with leaves and two types of inner nodes, P- and Q-nodes. In the context of this paper the term vertex denotes an element of a graph and the term node denotes an element of a PQ-tree. Usually P-nodes are drawn as circles while Q-nodes are drawn as rectangles. The leaves correspond to the elements of S and the possible permutations are encoded by the combination of the two types of inner nodes. The children of a P-node can be permuted arbitrarily, whereas the children of a Q-node are ordered and this order can only be reversed. If PQ-trees are used in planarity tests, a P-node always represents a cut vertex and a Q-node represents a biconnected component of a graph. The leaves represent edges to the not yet processed part of the graph. 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 stored set of permutations such that all elements of a given set $S' \subseteq S$ are consecutive in all remaining permutations. In a bottom up strategy it uses eleven template matching patterns, P0–P6 and Q0–Q3, to realise local changes within a tree, see Fig. 3 and Fig. 4. During the reduce phase PQ-leaves representing elements of S' are called *pertinent*. A PQ-node having at least one pertinent child is called pertinent, too. A PQ-node having only pertinent children is called *full*, one having at least one pertinent and at least one non-pertinent is called *partial* and one having no pertinent children is called *empty*. The *pertinent subtree* is the subtree of minimum height that contains all pertinent PQ-leaves. Its root is called *pertinent root* or short *pert-root*.

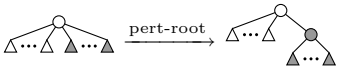
For an application of P2, P4, P6, and Q3 the root of the left side (*source pattern*) of the template must be pert-root. For an application of P3 and P5 this must not be the case, whereas for the remaining templates it may be the case. Some template applications may reverse the order of the children 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 complexity of REDUCE proved in [2], 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 linear time level planarity testing and embedding. For further reading on PQ-trees see [2].



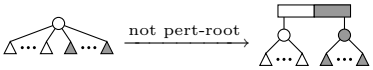
(a) Template P0



(b) Template P1

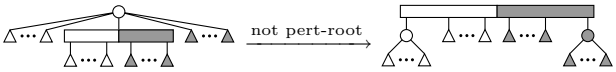


(c) Template P2



(d) Template P3

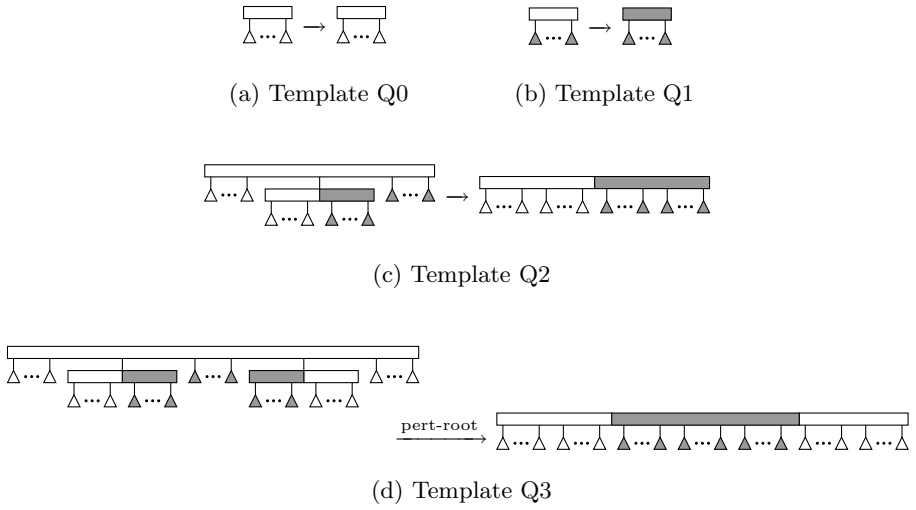
(e) Template P4



(f) Template P5

(g) Template P6

Fig. 3. Templates for testing (level) planarity. The grey shading indicates that subtrees are pertinent



(a) Template Q0

(b) Template Q1

(c) Template Q2

(d) Template Q3

Fig. 4. Templates, Part 2

Let G^j, $1 \leq j \leq k$, be the induced subgraph of G from level 1 to j. G^j is not necessarily connected, so a separate PQ-tree $T(F_i^j)$ is introduced for every component F_i^j of G^j with m_j denoting the number of 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 have to be merged. $\mathcal{T}(G^j)$ denotes the set of all $T(F_i^j)$.

Algorithm 1: LEVEL-PLANARITY-TEST

Input: $G = (V^1 \dot\cup V^2 \dot\cup \ldots \dot\cup V^k, E, \phi)$
Output: 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

See Algorithm 1 for a formal description of the LEVEL-PLANARITY-TEST. The procedure CHECK-LEVEL, as a sweep over a single level j, is divided into two phases. All operations are not done on the graph but in $T(F_i^j)$. The following

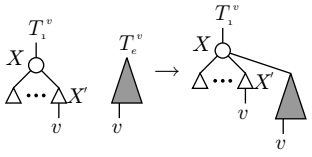
describes the *first reduction phase*. Define H_i^j to be the *extended form* of F_i^j. It consists of F_i^j plus some new *virtual vertices* and *virtual edges*. For every edge (u, v) with $u \in V(F_i^j) \cap V^j$ and $\phi(v) > j$, a new virtual vertex with label v and a virtual edge connecting it to u is introduced in H_i^j. The set of all virtual vertices of H_i^j that are labelled with 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 $T(H_i^j)$ which is called the *vertex addition step* is accomplished by a PQ-tree operation called REPLACE_PERT. Since all PQ-leaves with the same label appear consecutively after the PQ-tree operation REDUCE in every admissible permutation, each such consecutive set of PQ-leaves is replaced by REPLACE_PERT with a P-node. This is the parent of new leaves representing the adjacent vertices of v in $V^{j+1} \cup V^{j+2} \cup \cdots \cup V^k$. Afterwards 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 in the PQ-trees. Then REPLACE-SINGLE replaces them with a single representative PQ-leaf with the same label. R_i^j denotes this *reduced extended form* of H_i^j. If the graph is not a hierarchy, the replacement with a single representative is necessary for the correctness of the algorithm, see [20, p. 71ff].

Now it is possible that there are PQ-leaves with the same label in different PQ-trees, so the *second reduction phase* merges these trees. A reduced extended form R_i^j is called *v-singular* if all virtual vertices have the same label, i. e., $\bigcup_{w \in V, \phi(w) > j} S_i^w = \{v\}$. Singular components are handled by examining space information (values called PML and QML) on interior faces above v which are created when all leaves labelled with v are replaced with a single representative. See [20, p. 71ff] for details. If there is no interior face having enough space to contain the singular component, it is checked if it can be placed on the outer face with the same mechanism as for non-singular forms.

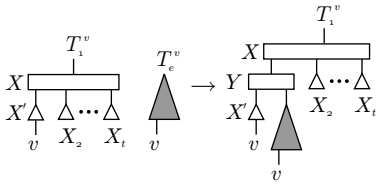
The following is a short description of these pairwise merge operations. Define the *low indexed level* of F_i^j, $\mathrm{LL}(F_i^j)$, to be the smallest d such that F_i^j contains a vertex in V^d. This value is maintained as an attribute of the corresponding PQ-tree $T(F_i^j)$. The *height* of a component F_i^j is $j - \mathrm{LL}(F_i^j)$. A merge operation is accomplished using information that is stored at the nodes of the PQ-trees. For any set of virtual vertices $S \subseteq V^{j+1} \cup V^{j+2} \cup \cdots \cup V^k$ that belongs to a form H_i^j or R_i^j, define the *meet level* of S, $\mathrm{ML}(S)$, to be the greatest $d \leq j$ such that $V^d \cup V^{d+1} \cup \cdots \cup V^j$ induces a subgraph of G in which 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 with $\mathrm{frontier}(X)$ being the sequence of its descendent leaves from left to right. For every Q-node Y with ordered children $Y_1, Y_2, \ldots, Y_t$ 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. The maintenance of the ML-values during template reductions and insertions in the PQ-trees is straightforward. They are indicators if a PQ-tree with a given low indexed level fits into the indentations under a P-node or between two sons of a Q-node.

Let $T_1^v, T_2^v, \ldots, T_f^v$ be all PQ-trees with $v \in V^{j+1}$ in their frontier sorted according to their descending height. All PQ-trees T_e^v, $2 \leq e \leq f$, are merged sequentially into the highest one, 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 labelled with v in T_1^v and proceeds upwards in T_1^v until a node X' and its parent X are encountered satisfying one of the following ordered conditions A–E.

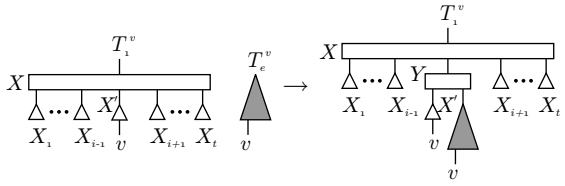
Merge Condition A The node X is a P-node with $\mathrm{ML}(X) < \mathrm{LL}(T_e^v)$. Attach T_e^v as child of X in T_1^v.



Merge Condition B The node X is a Q-node with ordered children X_1, X_2, $\ldots$, X_t, $X' = X_1$, and $\mathrm{ML}(X_1, X_2) < \mathrm{LL}(T_e^v)$. Replace X' in T_1^v with a Q-node Y having 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 $\mathrm{ML}(X_i, X_{i+1}) < \mathrm{LL}(T_e^v)$. Replace X' with a Q-node Y having X' and T_e^v as children.



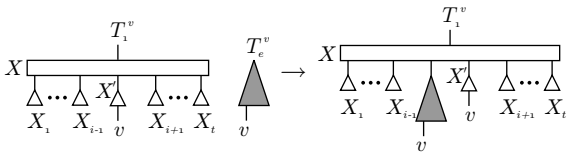
Merge Condition D The node X is a Q-node with ordered children X_1, X_2, $\ldots$, X_t, $X' = X_i$, $1 < i < t$, and

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

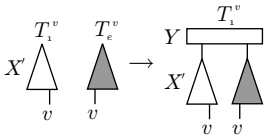
Attach T_e^v as a child of X between X_{i-1} and X_i. In case that

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

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



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



After each merge operation, REDUCE and REPLACE_PERT are called to make all v-leaves consecutive and then to replace them with one single representative PQ-leaf. Afterwards the entry for T_e^v is deleted from the set $\mathcal{T}(G^{j+1})$. Scanning for other leaves with the same label after v-merging several reduced extended forms is omitted in order to achieve linear running time. However, this strategy results in improper reduced extended forms, called *partially reduced extended forms*, possibly having several virtual vertices with the same label. They are reduced lazily on demand.

Finally, in a new sweep over this level all PQ-leaves representing sinks v in V^{j+1} are deleted from their corresponding PQ-tree reconstructing this tree such that it obeys the properties of a valid PQ-tree again.

Note that this mechanism also works on non-proper level graphs without any extra effort (adding all children on higher levels not only the ones from the next level), within $\mathcal{O}(|V|)$ time and without inserting up to $\mathcal{O}(|V|^2)$ dummy vertices for long and thus multi level spanning edges.

2.2 Linear Time Level Planar Embedding

In order to get a witness for the result of the level planarity test or to draw the graph nicely, i. e., planar, this algorithm can compute a level embedding in two passes. See Algorithm 2 for an outline. After G is augmented to a planar st-graph, a planar st-embedding can be obtained via the algorithm of Chiba et al. [4]. With this a level planar embedding can be computed easily.

An st-graph, as defined in [10,21], is a biconnected digraph with two adjacent vertices s and t and a numbering $st : V \to \{1, \ldots, |V|\}$ of the vertices such that $st(s) = 1$, $st(t) = |V|$, and for any vertex v with $1 < st(v) < |V|$ two adjacent

Algorithm 2: LEVEL-PLANAR-EMBED

Input: $G = (V^1 \dot\cup V^2 \dot\cup \ldots \dot\cup V^k, E, \phi)$
Output: 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 failed* **then return** $\mathcal{E}_l \leftarrow \emptyset$
// G_{st} is now a hierarchy
reverse level numbering G_{st} from the bottom to the top
AUGMENT(G_{st}) *// cannot fail*
reverse level numbering G_{st} from the top to the bottom
$E_{st} \leftarrow E_{st} \cup (s, t)$
// G_{st} is now an st-graph
TOPSORT(V_{st})
compute a planar embedding $\mathcal{E}_{st}$ according to Chiba et al. [4]
using the topological sorting as an st-ordering
$\mathcal{E}_l \leftarrow$ CONSTRUCT-LEVEL-EMBED($\mathcal{E}_{st}$, G_{st})
return $\mathcal{E}_l$

vertices u and w exist with $st(u) < st(v) < st(w)$. Augmenting a level graph G to an st-graph G_{st} is divided into two phases. After adding s as a single source and t as a single target, in the first phase an outgoing edge is added to every sink of G by application of a modified LEVEL-PLANARITY-TEST algorithm from level 1 to k. Using the same algorithmic concept bottom up, i. e., the modified LEVEL-PLANARITY-TEST from level k to 1, an incoming edge is added to every source of G in the second phase. To add these edges without destroying 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 and is removed either with the leaves representing the incoming edges of some vertex $w \in V^l$, $l > j$, or is still left in the final PQ-trees. In the first case the node which is represented by the sink indicator is connected to w after the reduction of w by REPLACE_PERT called on w. In the second case it is connected to t at the end of the algorithm. Sink indicators in PQ-trees which represent a v-singular form are connected to v if they are inserted in a inner face above v. Note that if all siblings of a node are ignored, its parent becomes recursively ignored, too. Thus it can happen that whole PQ-trees get ignored.

An st-numbering is naturally defined on digraphs. Therefore the implicit edge direction given by the levelling is used and edges are directed from the lower to the higher level. The numbering of the vertices by topological sorting results in a valid st-numbering. Now Algorithm 2 computes an st-embedding $\mathcal{E}_{st}$ with the technique of Chiba et. al. [4].

The function CONSTRUCT-LEVEL-EMBED derives a level planar embedding $\mathcal{E}_l$ of G from the planar embedding $\mathcal{E}_{st}$. It traverses the graph in depth first search order (DFS) starting at vertex t and proceeding from every visited vertex v to the unvisited neighbour w on a smaller level that appears first in clockwise ordering of v's adjacency in $\mathcal{E}_{st}$. Initially, all levels in $\mathcal{E}_l$ are empty. When a vertex $w \notin \{s, t\}$ is visited, it is appended at the end of the ordered list of the vertices assigned to $\phi(w)$. Because this DFS starts at t and uses only edges to vertices with a smaller st-number, the DFS in Chiba's method ENTIRE-EMBED [4, p. 62] to extend the obtained directed upward embedding $\mathcal{E}_u$ into a complete and undirected st-embedding $\mathcal{E}_{st}$ can be omitted for the sake of efficiency.

In order to achieve linear running time, searching for sink indicators that can be considered for augmentation must be avoided. 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. These contacts store which sinks have to be augmented if the new introduced Q-node is spliced by application of a template matching with its parent Q-node in the further progress of the algorithm. For further reading on this topic see [16, 17, 20].

3 Radial Level Planarity Test

3.1 Concepts

Level planarity and radial planarity look similar, but there are some essential differences. Leipert's algorithm heavily depends on the fact that a level graph is level planar if and only if each connected component is level planar. Therefore it suffices to test each connected component for level planarity. This is no more true for radial planarity, as Fig. 5 explains.

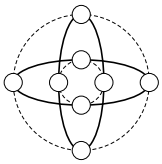
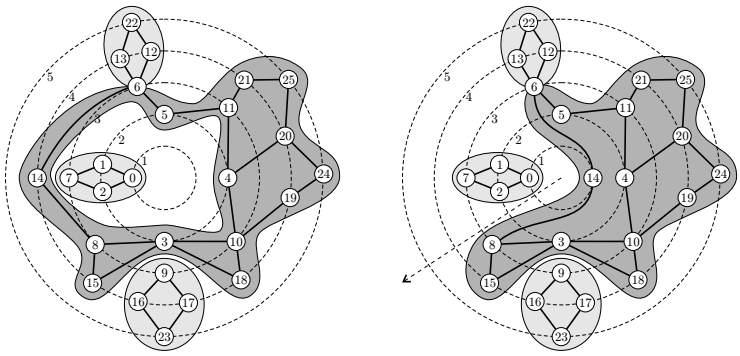


Fig. 5. A non-radial planar graph consisting only of radial planar connected components

Clearly a graph is radial planar, if it consists only of level planar components, because it is level planar. Hence, we must consider those components that are radial planar but not level planar. Therefore we introduce the concept of a ring:

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

A priori it is not clear 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 Fig. 6(a) consists of four biconnected components, one of which is a ring. We observe that a component can be nested in another and that, in the case of this graph, this is even necessary for a planar drawing. This only happens if the "outer" component is a ring. It also becomes clear that whether a component is a ring is not related to whether it contains a cycle. In fact every biconnected component with at least three vertices contains a cycle, but whether it is a ring depends on the levelling. If vertex 14 was on level 1, this graph would not contain a ring, because according to the ray in Fig. 6(b) there are no cut edges.



(a) A ring graph (b) Not a ring graph

Fig. 6. Rings depend on the levelling

Lemma 2. *If G is a level graph not containing any ring, the following statements are equivalent:*

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

Proof. The correctness follows directly from the definitions. □

Hence, if a graph does not contain a ring, we can use Leipert's level planarity test algorithm to test for radial planarity. For ring graphs, the algorithm has to be extended.

3.2 Rings

Before we describe in the next section how our algorithm stores the admissible permutations of the vertices on each circle, we discuss some more properties of rings first.

Lemma 3. *In any radial planar embedding of a ring graph the centre of the concentric levels lies in an inner face, the* centre face*.*

Proof. Suppose there exists a ring graph G that has a radial planar embedding $\mathcal{E}_l$ with the centre lying in the outer face. Then G contains a ring R. If we restrict $\mathcal{E}_l$ to R, the centre lies on the outer face of R. This means, there is a ray from the centre to infinity which crosses no edges. Therefore there are no cut edges and R is level planar, a contradiction to the definition of a ring made in Definition 1. □

Lemma 4. *A ring contains at least four vertices and four edges.*

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. Figure 5 shows two minimum rings. □

The nesting of rings is determined by some characterising parameters:

Definition 2. *Let G be a k-level graph containing a ring R. The minimum and maximum level with vertices of R are denoted by α_R and δ_R. These values are independent of the embedding. Let β_R be the maximum level with a vertex of the centre face in any radial planar embedding. Analogously, define γ_R as the minimum level with a vertex of the outer face of R in any radial planar embedding. See Fig. 7 for an example.*

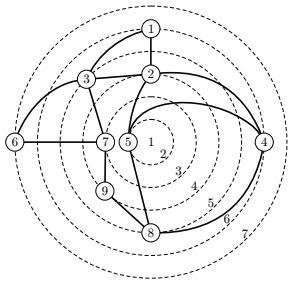


Fig. 7. Extreme levels of a ring. $\alpha_R = 2$, $\beta_R = 6$, $\gamma_R = 3$, $\delta_R = 7$

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

A priori it is not clear that there is a level optimal embedding for every ring. But our algorithm constructs such an embedding as we will see later in the proof of Lemma 10.

Lemma 5. *Any ring R spans at least two levels and its characterising parameters relate as $\delta_R > \gamma_R \geq \alpha_R$ and $\delta_R \geq \beta_R > \alpha_R$.*

Proof. Edges are not allowed to be horizontal, i. e., their source and target vertices always lie on different levels. Because of Lemma 4 a ring always contains edges and thus has vertices on at least two levels. Therefore the four relations follow directly from the definitions. □

Lemma 6. *Let G be a level graph consisting of two disjoint rings R and S. G is radial planar if and only if R and S are radial planar and*

$$\alpha_S > \gamma_R \wedge \beta_S > \delta_R \quad \textit{or} \quad \alpha_R > \gamma_S \wedge \beta_R > \delta_S.$$

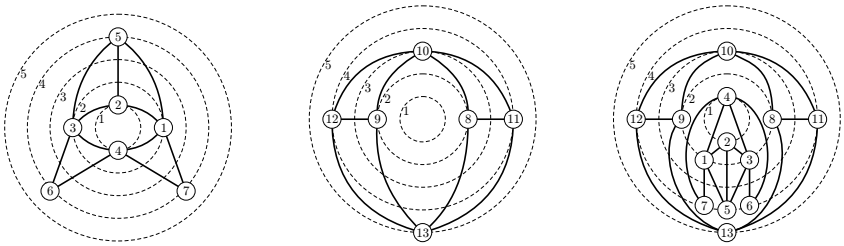
Proof. "⇒": Let G be a radial planar graph consisting of two disjoint rings R and S. Because subgraphs of radial planar graphs are always radial planar, R and S are radial planar. Each ring is biconnected and contains the centre according to Lemma 3. Thus one ring is fully contained within the centre face of the other in any planar embedding. In the case of R being contained within S, suppose $\alpha_S \leq \gamma_R$ or $\beta_S \leq \delta_R$. Then the border of the centre face of S intersects with the border of the outer face of R, a contradiction to the radial planarity of G. If S is contained in R, we analogously get the second part of the formula.

"⇐": Let G be a level graph consisting of two disjoint rings R and S with $\alpha_S > \gamma_R \wedge \beta_S > \delta_R$ or $\alpha_R > \gamma_S \wedge \beta_R > \delta_S$. Since these conditions are symmetric, we only consider the first case. To show the radial planarity of G, we construct an embedding of G by combining level optimal radial planar embeddings $\mathcal{E}_l^R$ and $\mathcal{E}_l^S$ of R and S. These embeddings only have in common the levels between α_S and δ_R, thus 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 outside of the outer border of R. Because of $\beta_S > \delta_R$, all vertices of R fit within the centre border of S, see Fig. 8. Note that it may be necessary to rotate and squeeze R, so that all vertices fit within the greatest cavity of S and vice versa. □

In other words, given appropriate embeddings of R and S, R must fit into the centre face of S or vice versa and cannot be placed side by side, as Leipert's algorithm would do.

3.3 R-Nodes

Our goal is to extend Leipert's algorithm to test for radial planarity. The basic ideas of that algorithm can be adopted. 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 $\mathcal{T}$. For dealing with rings, we introduce a new data structure *PQR-trees*.



(a) The "inner" ring R, $\gamma_R = 1$, $\delta_R = 4$

(b) The "outer" ring S, $\alpha_S = 2$, $\beta_S = 5$

(c) R nested in the centre face of S

Fig. 8. Nesting of rings

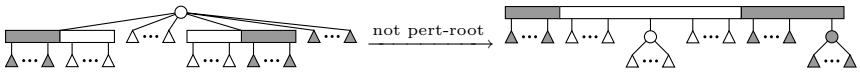
PQR-trees store the admissible edge permutations of radial planar graphs. PQR-trees are not related to SPQR-trees, used for incremental planarity testing [8]. Instead, PQR-trees are based on PQ-trees but contain a new "R" node type for the rings. *R-nodes* are similar to Q-nodes, but they have some new properties representing 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*. Since rings always contain the centre, it is possible to rotate a ring. This corresponds to rotating the graph around the centre, and is done by moving a subsequence of R-node children from the beginning of the children list to its end, or vice versa, while maintaining the relative order of the moved children. On a circular list this happens implicitly. Therefore R-nodes (as well as Q-nodes) can be implemented for example with the *improved symmetric list* data structure [1]. Then insertions, reversions and rotations can be done in constant time, which is crucial for the linear running time of this test.

Lemma 7. *For storing admissible edge permutations of radial planar graphs, it suffices that R-nodes occur only as the root of a PQR-tree.*

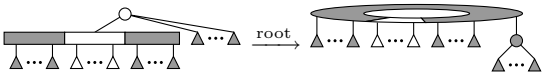
Proof. At any time in the wavefront sweep when a ring is encountered, it follows from radial planarity that there are no PQ-leaves left that originate from a component nested within the centre face of the ring. As a direct consequence, it is sufficient that R-nodes never have siblings and thus only occur at the root of PQR-trees. This follows from the definition of PQ-trees, since inner tree nodes must have at least two children, see [2, p. 339]. □

3.4 New Templates

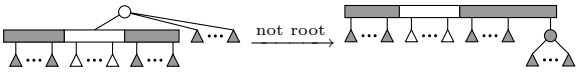
Due to the introduction of R-nodes we need twelve new templates to realise REDUCE on PQR-trees, some of them being analogous to Q-node templates,



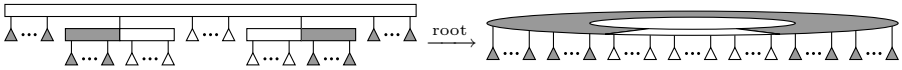
(a) Template P7



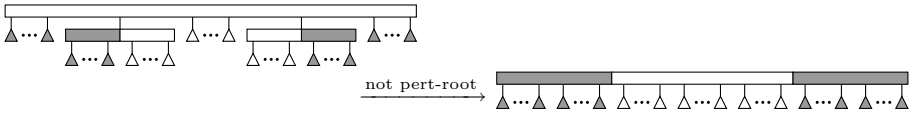
(b) Template P8



(c) Template P9

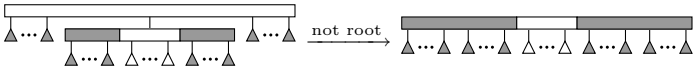


(d) Template Q4



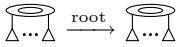
(e) Template Q5

(f) Template Q6

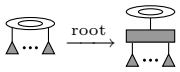


(g) Template Q7

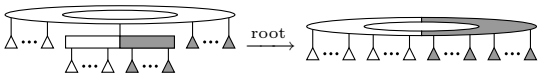
Fig. 9. Additional templates for testing radial level planarity, Part 1. Again the grey shading indicates that subtrees are pertinent



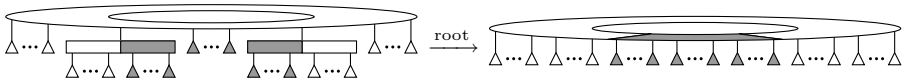
(a) Template R0



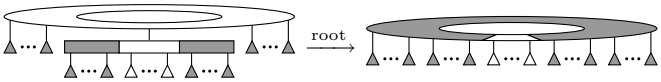
(b) Template R1



(c) Template R2



(d) Template R3



(e) Template R4

Fig. 10. Additional templates, Part 2

see Fig. 9 and Fig. 10. An R-node is only generated with one of the templates P8, Q4, or Q6, illustrated in Fig. 9(b), Fig. 9(d), and Fig. 9(f). Any of 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. It is clear, that in this cases it is not possible to apply any of the standard templates, i. e., the graph is no more level planar. This means that R-nodes are created only if it is necessary, i. e., if newly encountered edges change a represented biconnected component from level planar into a ring. Due to Lemma 7, 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 pert-root.

In analogy to the templates Q0–Q3 and Q6 it is necessary to provide new templates R0–R4, shown in Fig. 10, to treat source patterns having an R-node as its root. Before the application of an R-template 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. In R1 we introduce a new pseudo Q-node X' for technical reasons. This preserves the information that the PQR-tree represents a ring component and makes it possible to compute the value minML in order to know what fits "below" this ring component. The single meet level at the root is set to $\mathrm{ML}(X', X') = \mathrm{minML}$.

Definition 4. *For an R-node X with children X_1, X_2, $\ldots X_t$ define*

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

As we have seen, a Q-node may be *boundary partial*, i. e., it may have pertinent children at the front and at the end having at least one empty child in the middle. Preserving radial planarity, this can always be the case if the root of the PQR-tree is already an R-node or becomes an R-node during this reduction step and afterwards a rotation is possible, see Fig. 11. Then the front and the back is allowed to be connected via cut edges. If we can find a boundary partial Q-node in the PQ-tree after REDUCE, the graph is rejected as not radial level planar. This is even the case if this Q-node is the pertinent root. Note that the templates prohibit that a boundary partial Q-node is created at the root of a PQ-tree.

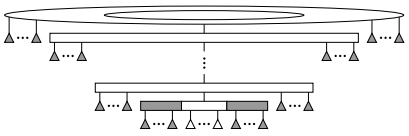


Fig. 11. Iterative merges of boundary partial Q-nodes

The presence of boundary partial Q-nodes forces us to provide the additional templates P7–P9 and Q5–Q7. P7 is the straightforward transformation of P6 if not applied to pert-root. The full children are grouped by a new P-node which is

inserted into the Q-node. It is both admissible to place it at the front or at the back. The difference here is whether the edges represented by the descendant leaves become cut edges later. The same holds for the new P-node created in P8 or P9. P8 can be applied only to the root, P9 otherwise. Template Q5 is basically the same as Q4 but treats non-roots. Q4 and Q5 are the inversion of Q3. Templates Q6 and Q7 are for Q-nodes having only full children except one boundary partial child, the first one is for the root and the second one for a non-root.

Now we are ready to show another important property of R-nodes with the next Lemma:

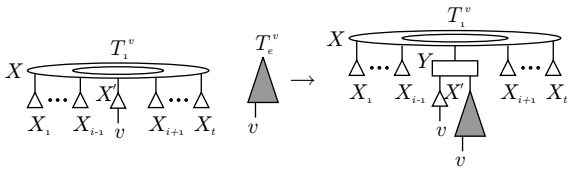
Lemma 8. *Once an R-node is created, it is preserved until its host PQR-tree is deleted.*

Proof. There is no template which destroys or replaces an R-node. Further, R1 ensures that an R-node never gets full, that means it is never replaced by an application of REPLACE_PERT. □

3.5 Merge Operations on PQR-Trees

Merge conditions stay essentially the same in PQR-trees as described in Sect. 2.1 if no R-node occurs. Because of Lemma 6, merge condition E, i. e., placing two PQ-trees next to each other with a new Q-root, cannot be applied if T_1^v or T_e^v has an R-root. As a consequence a merge operation may fail contrary to the non-radial case where condition E always is admissible if no other condition fits. For PQR-trees with an R-node as root we have to provide two additional merge conditions. The root of T_e^v must not be an R-node. If X is an R-node and thus the root of the host PQR-tree T_1^v, condition B and C collapse to the new condition C^{R}, because R-nodes can be rotated such that a merge can always be done in its interior. Similarly, if X is the root of the source 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 $\mathrm{ML}(X_i, X_{i+1}) < \mathrm{LL}(T_e^v)$. Replace X' with a Q-node Y having X' and T_e^v as children.



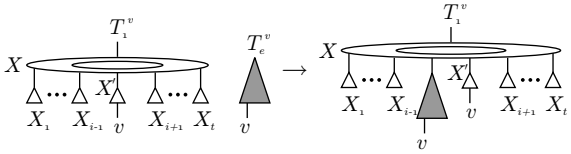
Merge Condition $\mathbf{D^R}$ The root of T_e^v is not an R-node. The node X is an R-node with ordered children $X_1, X_2, \ldots, X_t$, $X' = X_i$, $1 < i < t$, and

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

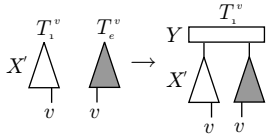
Attach T_e^v as a child of X between X_{i-1} and X_i. In case that

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

attach T_e^v as 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 is not an R-node. Reconstruct T_1^v by inserting a Q-node Y as new root with X' and T_e^v as children.



3.6 Merge of Processed Non-Rings

In level planarity testing, separate components can always be placed side by side without violating planarity. This is not necessarily true here. Consider a component of the input graph G containing at least one ring, cf. Sect. 3.2. The other components detected so far must fit into some inner face of the ring or its outer face. We first consider the case that these other components contain no rings. For the efficient execution of the necessary additional checks the algorithm maintains a variable minLL.

Definition 5. $\mathrm{minLL} = \min\{\,\mathrm{LL}(T) \mid T$ *is a completely processed PQR-tree with no R-root*$\,\}$. *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. If there is no such* 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 just one leaf (or just 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 identified as completely processed after REPLACE-SINGLE, minLL is updated with $\min\{\mathrm{minLL}, \mathrm{LL}(T)\}$. All processed

PQR-trees are discarded as in Leipert's test algorithm. It suffices to check if 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. Then, if there is a processed PQR-tree without an R-root, i. e., if minLL $< \infty$, we check if the largest of the newly created inner faces, i. e., the one of them starting at the lowest level, can include C. For this we use the same mechanism as Leipert uses for v-singular forms and compare minML with the new PML/QML value. If it fits, minLL $>$ PML or minLL $>$ QML, we set minLL $= \infty$. Otherwise we need not care whether another processed component smaller as C fits, for which its PQR-tree has been discarded. These will still fit later when a face for C is found. If no such face can be found, the graph is not radial level planar anyway. Recall that a processed PQR-tree with an R-root can never be included this way. The next section deals with merging of these.

3.7 Merge of Processed Rings

Let T^R be the only one, cf. Lemma 9, but maybe processed PQR-tree with an R-root. If none exists yet, T^R is undefined. As soon as the algorithm detects a ring and T^R becomes defined we obtain the invariant that T^R stays defined until termination, where T^R needs not always to stay the same PQR-tree. This behaviour is guaranteed because 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 3.

Algorithm 3: TREAT-NEW-RING

Input: PQR-tree T of a newly encountered ring, link vertex v, T^R, minLL
Output: boolean value indicating whether radial planarity is preserved

if $T^R \neq NULL$ **then**
  if (T^R *is not processed*) $\wedge$ (T^R *is not v-singular*) **then return false**
  minML $\leftarrow$ min(ML between the sons of the root of T^R)
  if (minML $\geq$ LL(T)) $\vee$ (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, the algorithm checks if T^R is completely processed. Otherwise G is not radial planar except the case that T^R is v-singular, which is covered by the same mechanism as in level planarity testing. Afterwards the algorithm checks whether minML is small enough that T fits "below" T^R and the tree with the smallest low indexed level minLL and thus all others fit "between" T and T^R, see Fig. 12. If one of the checks fails, G

is not radial planar because of Lemma 6. Finally T^R is updated. This algorithm leads to the next lemma.

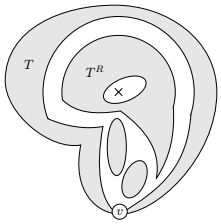


Fig. 12. Merge of rings

Lemma 9. *At any time in the radial level planarity test algorithm, there is at most one PQR-tree containing an R-node in $\mathcal{T}$ if the graph under test is radial level planar.*

3.8 Completion

If at termination of the test algorithm there is no PQR-tree T^R representing a ring graph, the graph is level planar. Otherwise, if no other trees occurred after T^R was detected, i. e., we have minLL $= \infty$, the graph is radial planar, too. It remains to check whether the other PQR-trees fit below T^R, i. e., if minML $<$ minLL. Otherwise, G is not radial planar.

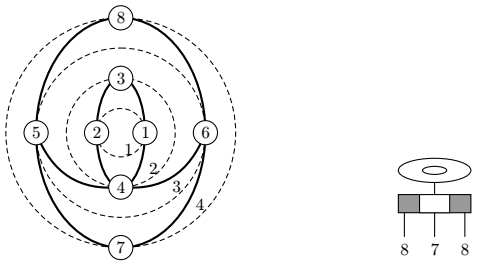
3.9 Correctness

As already stated in Sect. 3.2 it is necessary for the correctness of our algorithm that every computed embedding of a ring is level optimal.

Lemma 10. *The algorithm constructs a level optimal embedding for every ring contained in the graph.*

Proof. Consider a ring R contained in the 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 of two nested rings sharing a common vertex on a lower level than the link vertex of the outer ring. See Fig. 13 for an example. In this case the centre face of the outer ring is closed by the application of template R4.

As these four templates are only applied if none of the other templates 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.



(a) Vertex 4 links two rings

(b) Before closing the outer ring

Fig. 13. Linked and nested rings

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 are represented by the ML-values between two siblings. The meet levels stored between a node and its siblings are always lower or equal than the ones between its children. Thus the overall smallest values are stored between children of the root and minML represents the highest indentation of the outer border of R. The value of minML can only change when an inner face is closed by REPLACE-SINGLE. It may be possible that multiple options exist which faces are closed due to the freedom of rotation. This only depends on the templates applied in REDUCE. The only template which has multiple options is R1. Since R1 always preserves the minimum meet level, cf. Sect. 3.4, it is guaranteed that the highest possible indentation is preserved whenever possible. Thus $\gamma_R = \text{minML}$ is optimal in this embedding. □

Lemma 11. *The REDUCE operation, extended with the new templates shown in Fig. 9 and Fig. 10, calculates correctly the new set of admissible permutations for radial planarity.*

We omit the proof of this lemma because it is quite lengthy and rather simple. It must be shown first that no template violates radial planarity. This is obvious because the templates are constructed exactly that way. Further it must be shown that any radial 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 planar. This can be shown easily by regarding all possible constellations of node types and the order of empty, full, and (boundary) partial children. As a direct consequence we get our first main theorem:

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

4 Radial Planar Embedding

Leipert et. al. also have presented an algorithm for computing a level planar embedding for a level planar graph. This algorithm can be extended to compute a radial planar embedding for a radial planar graph by using PQR-trees. Again there are some differences that need attention.

4.1 Meet Levels between Ignored Siblings

As already mentioned in Sect. 2.2, PQ-trees contain ignored nodes, when calculating an embedding. Since we use the same strategy for calculating radial embeddings we also have to treat ignored nodes. This is especially important when calculating minML 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 calculated correctly. For example, when a Q-node with outermost ignored children is spliced into another one, the outer ML-values have to be initialised. Fortunately, this is straightforward and can always be done in constant time.

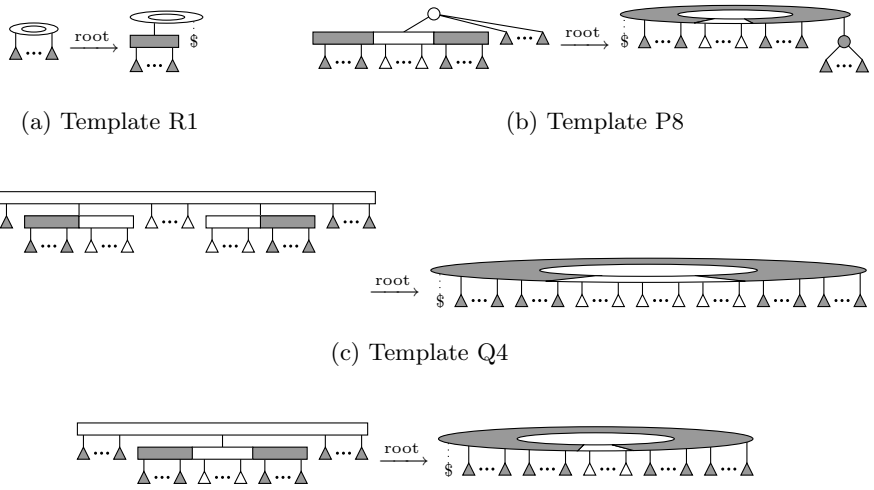
4.2 Embedding the Edges

As already mentioned in the introduction, we not only have to compute a vertex ordering $\leq_j$ on each level j but also the edge routing. It is not necessary to order the adjacent edges of each vertex like it is done in [4], but it suffices to designate cut edges. The following detection of cut edges is done by the *st*-embedding creation step described in Sect. 4.4.

At the beginning no edges are marked as cut edges. The recognition if an edge is a cut edge works as follows: A new node denoted by *ray indicator* and labelled with \$ marks for an R-node where the imaginary ray splits its children. The ray indicator is ignored throughout the application of the algorithm similar to sink indicators and it always stays a child of the R-node. It is created with every R-node by slightly modified templates P8, Q4, and Q6, see Fig. 14.

R1 also has to be modified. Recall that it creates a pseudo Q-node X'. Before this is done the R-node is rotated such that two siblings between which minML is located become the end vertices of X'. Otherwise level optimality can be lost. See Fig. 15 for an illustration.

It is possible that \$ divides the children of X' into two parts. Thus before creating X' it is necessary to drag one part over the \$ because it must stay a child of the R-node. All leaves of all pertinent subtrees that are dragged over the ray indicator represent cut edges. They can be found via DFS without violating the $\mathcal{O}(|V|)$ time bound, because after each drag operation the subtrees are removed from the PQR-tree by REPLACE_PERT anyway. The same operation is necessary if in REPLACE_PERT the ray indicator lies within the pertinent sequence. Again 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 Fig. 16(a). Figure 16(b) shows its corresponding PQR-tree before the reduction



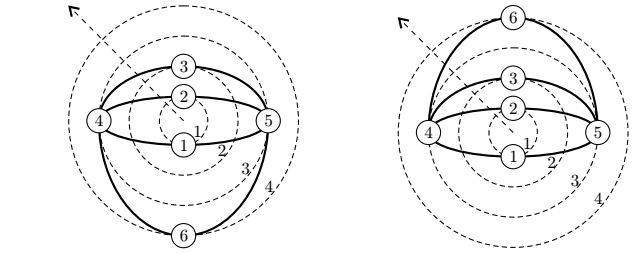
(a) Template R1

(b) Template P8

(c) Template Q4

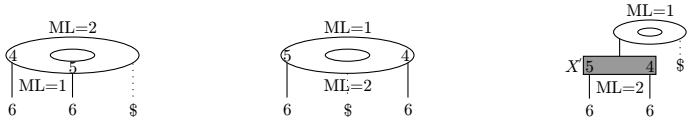
(d) Template Q6

Fig. 14. Radial level planarity with level embedding templates



(a) Not level optimal

(b) Level optimal



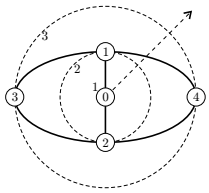
(c) Initial PQR-tree on level 3

(d) After rotation

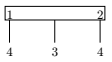
(e) Final PQR-tree

Fig. 15. Preserving level optimality

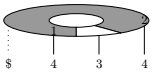
of all leaves labelled with 4, whereas Fig. 16(c) shows the resulting PQR-tree after this reduction. The applied template is Q4. As shown in Fig. 16(d) the leaf representing the edge $(1, 4)$ is dragged over \$ in REPLACE_PERT thus $(1, 4)$ becomes a cut edge.



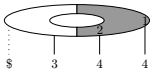
(a) The input graph



(b) The PQR-tree before Q4



(c) Figure 16(b) after REDUCE



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

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

4.3 Augmenting G to G_{st}

Instead of talking about processed PQR-trees as in Sect. 3, now we denote them as *ignored PQR-trees* because they consist of ignored nodes only, cf. Sect. 2.2. Here it is not sufficient to store only the LL-value of the highest ignored PQR-tree. We also have to store all ignored PQR-trees, because their sinks must be augmented with edges later. This can be the case if a ring is closed by template P8, Q4, or Q6. Then all sinks are augmented to the link vertex w on which REDUCE was called and that closes the ring. Further it can be the case that augmenting to a vertex v is necessary if a face is closed by REPLACE-SINGLE called on all leaves representing v. Hence, we maintain an *ignored PQR-tree collection* $\mathcal{T}^*$ which stores all ignored PQR-trees in addition to the *active collection* $\mathcal{T}$. The embedding of all ignored components within a ring always can be done without violating radial planarity. Let vertex u be a sink represented by a sink indicator in a PQR-tree in $\mathcal{T}^*$. Then $\phi(w) > \phi(u)$ holds.

When an R-node is created, all other PQR-trees, whether ignored or not, are nested into an inner face. This includes ignored PQR-trees with an R-root. Only a single PQR-tree T^R is left. In analogy to Lemma 9, this leads directly to the following property:

Lemma 12. *$\mathcal{T} \cup \mathcal{T}^*$ 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, but also all sinks in $\mathcal{T}^*$ are augmented to v and $\mathcal{T}^*$ is emptied. Similar to the test, if there exists an ignored R-rooted PQR-tree T^R, this step is omitted for a face different from the centre face, because 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.

As an optimisation, the test whether the LL-value of a newly created PQR-tree for the outer ring fits below the minML-value of an ignored inner ring can be omitted as well as the test if the other ignored PQR-trees fit between the two rings. This is done by the bottom up phase with the single hierarchy rooted at t. Still, the sinks have to be connected to the link vertex.

When a PQR-tree contains ignored nodes, the templates P8, Q4, and Q6 may not only be applied to the root of a PQR-tree. It may happen that the path from a Q-node X to the root of the tree 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 augmented to the link vertex. The $\mathcal{O}(|V|)$ time bound is not violated, because if the test on this situation fails either the input graph is not radial planar and the algorithm rejects or we have a situation as shown in Fig. 11. This case can be checked in $\mathcal{O}(1)$ time because in a PQR-tree no node chains exist 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 and thus are not traversed a second time. With this approach the calculation time stays linear in the sum again.

4.4 Computation of $\mathcal{E}_{st}$

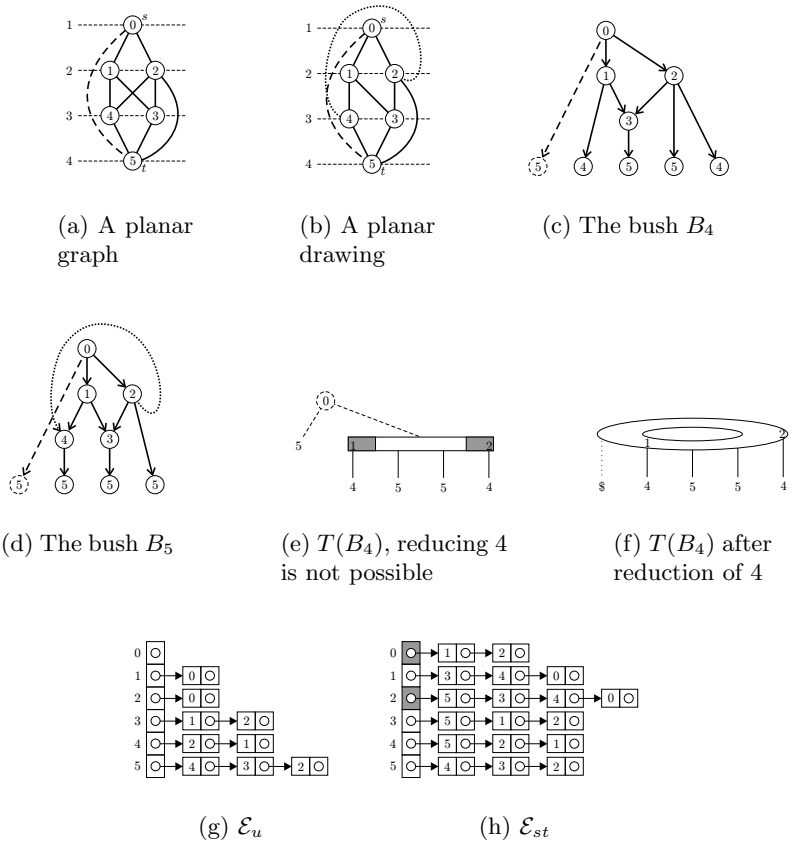
To compute an st-embedding $\mathcal{E}_{st}$ of the graph G_{st} (see Algorithm 2) the algorithm of Chiba et al. [4] is used. Unfortunately this algorithm, because it is based on the vertex addition method of [10, 21], needs an st-graph as input. But in our case G_{st} has no st-edge (s, t). If G is a ring graph, s and t are not in the same face in any planar level embedding $\mathcal{E}_l$ of G, i. e., s does not lie in the outer face as t does, cf. Lemma 3. Therefore the introduction of a new edge (s, t) as it is done in Leipert's algorithm may not be possible because it may destroy radial planarity and the st-embedding algorithm will fail. Thus we omit introducing (s, t), but only have an induced st-numbering of the vertices. After radial edge augmentation every vertex apart from 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 not be biconnected.

In the standard vertex addition method with computation of an embedding [4, 10, 21] the edge (s, t) behaves similar to the imaginary ray in the radial level planarity test. The difference is that the st-edge is physically existent and therefore no other edge is allowed to cross it without violating planarity. Thus

cyclic reduces, i. e., cut edges, are not allowed and need not be considered. Without (s, t), cyclic reduces are admissible. Thus our idea to solve the problem is the same as that of extending the level planarity test to the radial case. We upgrade the standard graph planarity test with embedding to use our new templates. We use R-nodes and thus the PQR-tree data structure to realise cyclic reduces. Again we omit the DFS of Chiba's algorithm to compute an st-embedding $\mathcal{E}_{st}$ out of the obtained upward st-embedding $\mathcal{E}_u$, cf. Sect. 2.2, here not only for the sake of efficiency but for the correctness. In our context, $\mathcal{E}_u$ is more an *inward embedding* than an upward embedding. Now with our approach it is possible to route edges around s. The routing around t is not admissible because we only consider monotone level planar graphs. Figure 17(b) is a radial planar drawing of the graph shown in Fig. 17(a) assuming the st-edge, shown as a dashed line, was not there. Due to the existence of cut edges Chiba's DFS may not deliver a valid edge ordering around each vertex, e. g., see the adjacencies of vertex 0 and 2 in Fig. 17(h), whereas in $\mathcal{E}_u$ the orderings of the incoming edges are correct, see Fig. 17(g). Therefore in Sect. 4.5 we use $\mathcal{E}_u$ to calculate a radial level planar embedding $\mathcal{E}_l$ instead of $\mathcal{E}_{st}$.

For example, Fig. 17(c) to Fig. 17(f) show the now admissible cyclic reduce of vertex 4 by an illustration of digraphs, the so called *bushes*, and their corresponding PQR-trees. If there is the edge (s, t), $T(B_4)$ cannot be reduced according to vertex 4 because no template fits. Otherwise template Q4 can be applied and we obtain Fig. 17(f). Afterwards replacing 4 with 5 by application of REPLACE_PERT can designate e. g. $(2, 4)$ as cut edge.

Let $l_1, l_2, \ldots, l_i$ be the leaves scanned in this order labelled with v and let v be the vertex on which REPLACE_PERT is called. The method UPWARD-EMBED of [4, p. 67f] relies on the fact that the order of the leaves/edges in frontier(pertinent sequence) which are removed from the PQR-tree by REPLACE_PERT and which afterwards are stored in $\mathcal{E}_u$ are in an admissible ordering except of reversion. The potential reversions of the parent Q-node in the further proceeding of the algorithm are handled by the direction indicators of [2]. This behaviour stays exactly the same with children of an R-node X. There exists one difference, however. If the ray indicator occurs within the pertinent sequence, we have to drag one part over it. But this is done previously to the removal of the pertinent sequence. In the further proceeding of the algorithm, after the removal of the pertinent sequence, there is the possibility of a rotation on X and thus of an implicit rotation of $l_1, l_2, \ldots, l_i$. But this only means rotation of the whole graph including the ray. Hence, the ordering of the stored sequence stays valid. Note that if an R-node has only pertinent children, then it is admissible to move the ray indicator in an arbitrary way leading to different outer faces, i. e., to different embeddings. This plays only a minor role because we are only interested in a single admissible embedding. Thus analogously to UPWARD-EMBED of Chiba et al. we obtain a valid inward embedding.



(a) A planar graph

(b) A planar drawing

(c) The bush B_4

(d) The bush B_5

(e) $T(B_4)$, reducing 4 is not possible

(f) $T(B_4)$ after reduction of 4

(g) $\mathcal{E}_u$

(h) $\mathcal{E}_{st}$

Fig. 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 indicated st-numbers

4.5 Computation of $\mathcal{E}_l$

In this section we assume w. l. o. g. that the incoming adjacent edges are sorted clockwise for every vertex in embedding $\mathcal{E}_u$. Before we present our algorithm for computing a radial level embedding we show a few lemmata.

Lemma 13. *Each vertex $v \in G_{st} - \{s\}$ hast at least one incoming non-cut edge.*

Proof. $G_{st} = (V_{st}, E_{st})$ is an 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 cut edge in REPLACE_PERT and in template R1 if the ray indicator lies within the pertinent sequence. Then there are PQ-leaves representing edges on each side of it and can only be put on one side of the sequence. Thus the edges of the part which are not dragged over \$ are non-cut edges. If \$ is already at the beginning or end of the pertinent sequence, no pertinent edges are cut edges. □

Lemma 14. *There exists at least one path from s to every vertex $v \in G_{st}$ not containing a cut edge.*

Proof. Follows directly from Lemma 13. □

Lemma 15. *A cut edge never lies between two non-cut edges in the ordered adjacency of a vertex v in any $\mathcal{E}_u$.*

Proof. Assume v has adjacent incoming edges in the ordering e_1, e_c, e_2, see Fig. 18. Let e_1 and e_2 be non-cut edges and e_c be a cut edge.

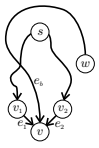


Fig. 18. No cut edge can be between two non-cut edges

Let v_1 be the source vertex of e_1 and v_2 the source vertex of e_2 respectively. Then $v_1 \neq v_2$. Thus there exist two paths, p_1 from s to v_1 and p_2 from s to v_2, according to Lemma 15 which are at least different at one edge. The cut edge e_c destroys planarity with crossing the bounds 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 its 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 end of the incoming adjacency list of its target node. It is called* counterclockwise *otherwise.*

Lemma 16. *No vertex v in $\mathcal{E}_u$ can be the target of both clockwise and counterclockwise cut edges.*

Proof. Assume v has two incoming cut edges, one being clockwise and the other being counterclockwise. Consider Fig. 19. Because edges have to be drawn monotonic, cut edges are not allowed to be placed below t. Thus they cross at least once above s which is a contradiction to planarity. □

Fig. 19. No vertex v has a incoming clockwise and a incoming counterclockwise cut edge at the same time assuming monotone planarity

According to the lemmata above we obtain Algorithm 4. $\mathcal{E}_l[j]$ denotes the vertex list of the radial level j, ordered by $\leq_j$.

Algorithm 4: CONSTRUCT-LEVEL-EMBED

Input: $\mathcal{E}_u, G_{st} = (V_{st}, E_{st})$
Output: $\mathcal{E}_l$

procedure *DFS(v, dir)*
  if $visited[v] =$ ***true*** **then return**
  $visited[v] \leftarrow$ **true**
  if $dir = left$ **then** insert v at the beginning of $\mathcal{E}_l[\phi(v)]$
  else insert v at the end of $\mathcal{E}_l[\phi(v)]$
  foreach *incoming edge e of v in direction dir* **do**
    if *e is a cut edge* **then**
      if *e is clockwise* **then** $insert(Q, source(e), left)$
      else $insert(Q, source(e), right)$
    else
      DFS($source(e), dir$)
    end
  end
end

foreach $v \in V_{st}$ **do** $visited[v] \leftarrow$ **false**
Queue Q *// stores pairs*
$insert(Q, t, right)$
while Q *not empty* **do** DFS($delete\_first(Q)$)

return $\mathcal{E}_l$

CONSTRUCT-LEVEL-EMBED starts with a sorted DFS from t in $\mathcal{E}_u$ not using cut edges. All vertices with at least one incoming cut edge are placed into a Queue Q. Every other newly detected vertex v is inserted at the end of $\mathcal{E}_l[\phi(v)]$. Afterwards a sorted DFS is started from all vertices $q \in Q$, in turn using only unvisited vertices. Clockwise cut edges are traversed from right to left and their respective source vertex w is inserted at the beginning of its level list $\mathcal{E}_l[\phi(w)]$. Counterclockwise cut edges are traversed from left to right and their respective source vertex w is inserted at the end of its level list $\mathcal{E}_l[\phi(w)]$. Now it is possible that some unused non-cut edges and therefore some unvisited vertices are reachable form w. They are inserted at the same end of the level lists as w. Note that source vertices of newly detected cut edges are treated in the same manner as above and are inserted into Q. The algorithm terminates when all vertices have been processed.

Theorem 2. *Algorithm 4 constructs a valid planar level embedding of the given inward embedding $\mathcal{E}_u$ and needs $\mathcal{O}(|V|)$ running time.*

Proof. The running time of $\mathcal{O}(|V|)$ is obvious, because the algorithms performs DFS only with different parts of the graph one after the other. Hence, we only need to show the correctness. The algorithm starts at t and traverses first the *trunk*, the face labelled with 1 in Fig. 20.

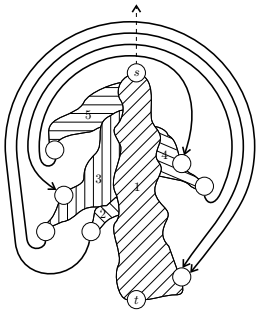


Fig. 20. Successive and sorted attachments of faces to the sides of the trunk

This is the same mechanism which Leipert et al. use in Algorithm 2. Afterwards it attaches the other faces successively to the left and to the right side. These attachments are sorted, on the left side of the trunk from right to the left, on the right side of the trunk from left to right, and at each case from bottom to the top. □

Example 1. Figure 21 shows $\mathcal{E}_l$ computed by Algorithm 4 from $\mathcal{E}_u$ shown in Fig. 17(g).

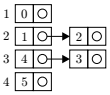


Fig. 21. $\mathcal{E}_l$ computed from $\mathcal{E}_u$ in Fig. 17(g)

5 Conclusion

We have presented a new algorithm for detecting radial planarity of a k-level graph in linear time. For this we have enhanced the PQ-tree data structure of [2] with a new node type, R-nodes, representing a ring component of the graph. The obtained data structure is called PQR-tree. Our algorithm is capable of computing a radial level planar embedding within the same linear time bounds as other planarity test algorithms [4, 16–18, 20]. To check the practicability of our algorithm we have realised a prototypical implementation of it in C++ using the Graph Template Library [11] with improved symmetric lists [1].

Further investigations are desired in order to expand the test algorithms for the various kinds of level planarity for detecting the so called *minimal non level planar subgraph patterns (MNLP-patterns)* if the tested graph is not level planar. These MNLP-patterns for level graphs are characterised in [12] and are the counterparts of the Kuratowski Graphs $K_{3,3}$ and K_5. See [19, 22] for detection of Kuratowski subgraphs. As already mentioned in the conclusion of [20, p. 211] the detection of MNLP-patterns can also be used to verify the results of a level planarity test. Because such a test is a non-trivial algorithm and thus it is not unlikely that an implementation is faulty, it is desirable not only to prove planarity with an embedding or drawing but also to show non-planarity on the basis of MNLP-patterns.

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

References

[1] C. Bachmaier and M. Raitner. Improved symmetric lists. Submitted for publication, August 2003.

[2] 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.

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

[4] 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.

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

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

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

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

[9] 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, ESA 2001*, volume 2161 of *LNCS*, pages 488–499. Springer, 2001.

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

[11] GTL. Graph Template Library. `http://www.infosun.fmi.uni-passau.de/GTL/`. University of Passau.

[12] P. Healy and A. Kuusik. Characterisation of level non-planar graphs by minimal patterns. Technical Report UL-CSIS-98-4, Department of Computer Science and Information Systems, University of Limerick, July 1998.

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

[14] 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.

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

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

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

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

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

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

[21] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In P. Rosenstiehl, editor, *Theory of Graphs, International Symposium, Rome, July 1966*, pages 215–232. Gordon and Breach, 1967.

[22] K. Mehlhorn and S. Näher. *LEDA, A Platform for Combinatorial and Geometric Computing*, chapter 8.7. Cambridge University Press, 1999.

[23] 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.

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