# Radial Level Planarity Testing and Embedding in Linear Time*
## (Extended Abstract)

Christian Bachmaier, Franz J. Brandenburg, and Michael Forster

University of Passau, 94030 Passau, Germany
{bachmaier,brandenb,forster}@fmi.uni-passau.de

**Abstract.** Every planar graph has a concentric representation based on a breadth first search, see [21]. 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 [12,13,15–18] 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, 20]. 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 $\phi\colon V \to \{1, 2, \ldots, k\}$, $1 \le k \le |V|$, that partitions the vertex set into $V = V^1 \dot\cup V^2 \dot\cup \cdots \dot\cup V^k$, $V^j = \phi^{-1}(j)$, $1 \le j \le k$, 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 $(u, v) \in E$. The level planarity problem [7,12,17] 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

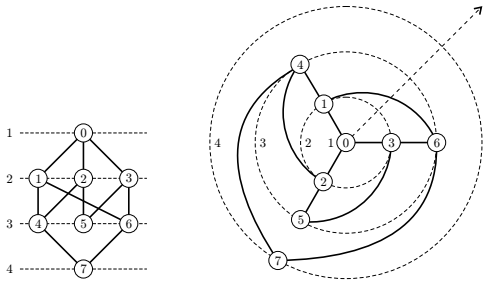
* This research has been supported in part by the Deutsche Forschungsgemeinschaft, grant BR 835/9-1.

$k$-level drawing with the above restrictions, an embedding of the graph has to be computed. Level planar embeddings are characterized 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.

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 [14] 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.

We generalise level planarity 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 1(b) shows a radial planar drawing of the graph in Fig. 1(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. 1.** A radial planar graph with a radial planar drawing

The more general approach of Dujmović et al. [9] leads to a linear time fixed paramter 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. [15–18] 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. Input graphs with $|E| > 3|V| - 6$ are rejected as not radial planar because radial planar graphs are planar by definition. For more details and proofs we refer to the long version [1] of this paper.

## 2 Related Work

The basis of our algorithm is the linear time algorithm of Leipert et al. [15–18] for level planarity testing and embedding, which is in turn based on previous work of Heath and Pemmaraju [12,13] and Di Battista and Nardelli [7]. As the vertex addition method for graph planarity tests [10,19], this algorithm makes heavy use of the *PQ-tree* data structure introduced by Booth and Lueker [3]. See [1, Sect. 2] for an introduction. Leipert's algorithm proceeds level by level and stores the admissible permutations of the vertices on each level efficiently in a set of PQ-trees. Also a level embedding can be computed in linear time. First, the input graph is augmented to an $st$-graph $G_{st} = (V_{st}, E_{st})$ and then an $st$-embedding is computed by the algorithm of Chiba et al. [4]. With this $st$-embedding, a level embedding is constructed by an ordered depth first search (DFS). See [1,12,13,15–18] for details.

**Theorem 1 (Leipert et al. [15–18]).** *There is an $\mathcal{O}(|V|)$ time algorithm for testing level planarity and computing a level planar embedding.*

## 3 Radial Level Planarity Test

In this section we describe our linear time algorithm for level planarity testing, using a new data structure, PQR-trees. This theorem is our main result. For its proof see [1].

**Theorem 2.** *There is an $\mathcal{O}(|V|)$ time algorithm for testing radial $k$-level planarity.*

### 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. 2 explains.

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:

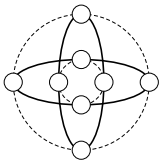
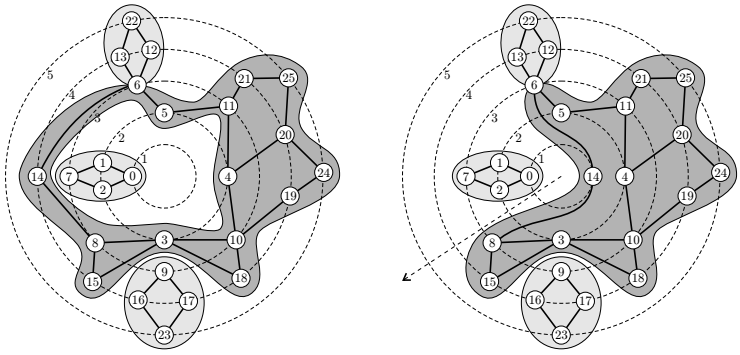


**Fig. 2.** A non-radial planar graph consisting only of radial planar connected components

**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. 3(a) consists of four biconnected components, where the single ring is indicated by the darker shading. A component can be nested in another and here this is even necessary for a planar drawing. This only happens if the "outer" component is a ring. Moreover rings are not related to cycles. 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. 3(b) there are no cut edges.



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

**Fig. 3.** Rings depend on the levelling

Rings make the difference between level planarity and radial planarity. If a level graph does not contain a ring, it is radial planar if and only if it is level planar. Moreover this is equivalent to each connected component being level planar (or radial planar). 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. Before we describe how our algorithm stores the admissible permutations of the vertices on each circle in the next section, we discuss some more properties of rings. Clearly, in any radial planar embedding of a ring graph the centre of the concentric levels lies in an inner face, the *centre face*. 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 $\alpha_R$ and $\delta_R$. These values are independent of the embedding. Let $\beta_R$ be the maximum level with a vertex of the centre face in any radial planar embedding. Analogously, define $\gamma_R$ as the minimum level with a vertex of the outer face of $R$ in any radial planar embedding. See Fig. 4 for an example.*

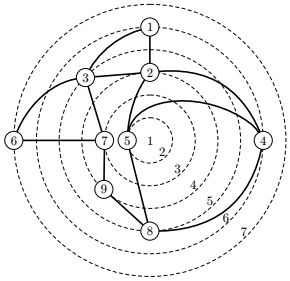



**Fig. 4.** Extreme levels of a ring. $\alpha_R = 2$, $\beta_R = 6$, $\gamma_R = 3$, $\delta_R = 7$

**Lemma 1.** *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*

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

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. Because $\beta_R$ and $\gamma_R$ depend on the embedding, it is clear that for leaving maximum space for placing other components the embedding has to be calculated such that $\beta_R$ is maximised and $\gamma_R$ is minimised. Such an embedding is called *level optimal*. A priori it is not clear that there is a level optimal embedding for every ring. But our algorithm constructs such an embedding, see [1].

### 3.2 R-Nodes

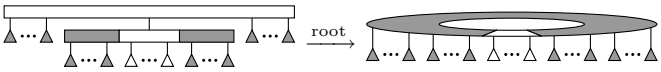
Our goal is to extend Leipert's algorithm to test for radial planarity. The basic ideas of that algorithm can be adopted. The input graph is traversed in a top

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*. 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].

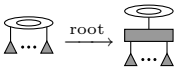
PQR-trees are based on PQ-trees, but contain a new "R" node type for the rings. Because of Lemma 1 it suffices that R-nodes occur only as the root of a PQR-tree. As usual [3], P-nodes represent all permutations of its children and Q-nodes the given children list or its reverse. *R-nodes* are similar 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 and as usual P-nodes and Q-nodes are drawn as circles and rectangles, respectively. 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 *rotation* which is new. 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. R-nodes can be implemented, for example, with the improved symmetric list data structure [2]. Insertions, reversions, and rotations can be done in constant time, which is crucial for the linear running time of the test.

### 3.3 New Templates

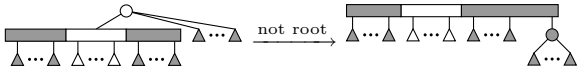
The main operation on PQ(R)-trees is REDUCE [3]. By the application of several templates REDUCE ensures that a given subset of *pertinent* PQ-leaves occurs consecutively in any of the stored permutations. In addition to the eleven templates of PQ-trees we need twelve new templates for PQR-trees.




(a) Template Q6




(b) Template R1




(c) Template P9

**Fig. 5.** Some of the new templates

Figure 5 shows some of them. A complete description is given in [1]. As an example, template Q6 demonstrates some key elements of the new templates. In addition to empty, partial, and full Q-nodes, there are *boundary partial* Q-nodes where all pertinent children are at the beginning or at the end separated by at least one empty child. Instead of this boundary partial child, there can be some empty children, optionally bounded by partial Q-nodes. This is handled by another template. Further it shows that there are templates which can 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 root of the pertinent subtree (pert-root). Finally it is illustrated how an R-node is created. Because the full children cannot be made consecutive it is not possible to apply any of the existing Q-templates, i. e., the biconnected component is no more level planar. Hence, R-nodes are created only by necessity, i. e., if newly encountered edges change the represented biconnected component from level planar into a ring.

Now that there are some templates which create R-nodes, it is necessary to treat R-nodes on the left side of a template. Therefore we introduce templates R0–R3. It may be necessary to rotate the R-node before applying an R-template. The R-templates are the straightforward transformations of the respective Q-node templates with the exception of R1 (see Fig. 5(b)), where a pseudo Q-node is introduced for technical reasons. This preserves the information that the PQR-tree represents a ring component and makes it possible to compute a value minML in order to know what fits "below" this round component. The single *meet level* at the root is set to minML. The meet level ML of two adjacent siblings with a Q-node or R-node as their parent describes how much space is left between them. Other components can be nested at this position only if they start on a higher level.

**Definition 3.** *For an R-node $X$ with children $X_1$, $X_2$, $\dots X_t$ define*

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

The presence of boundary partial Q-nodes forces us to provide the remaining templates P7–P9, Q5, Q7, and R4. See template P9 in Fig. 5(c) as an example. If a P-node has only full children with the exception of one boundary partial Q-node, 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 of the Q-node. The difference is only, whether the edges represented by the descendant leaves become cut edges later.

The templates are exactly designed to cover all cases occurring in PQR-trees while traversing a radial planar graph. In addition they maintain the invariant that once an R-node is created, it is preserved until its host PQR-tree is deleted. The following vertex addition step performed by a PQ(R)-tree operation REPLACE_PERT stays the same as in Leipert's algorithm.

### 3.4 Merge Operations on PQR-Trees

If there are PQ-leaves with the same label in more than one PQ-tree, these PQ-trees have to be merged according to merge conditions described in [1, 12,

13, 15–18]. Whether a PQ-tree $T$ can be merged into another one depends on its *low indexed level* $\mathrm{LL}(T)$, the lowest level with vertices of the represented component. A PQR-tree with a lower LL is said to be *higher* than a *smaller* one with a greater LL. The merge conditions remain essentially the same in PQR-trees if no R-node occurs. Because of Lemma 1, merge condition E, i. e., placing two PQ-trees next to each other with a new Q-root, cannot be applied if one of the merge candidates 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-root we have to provide two additional merge conditions. The root of the smaller PQR-tree must not be an R-node. If the root of the higher PQR-tree is an R-node, condition B and C collapse to the new condition $\mathrm{C^R}$, because R-nodes can be rotated such that a merge can always be done in its interior. Similarly, if the root of the source pattern of condition D is an R-node we obtain $\mathrm{D^R}$, see [1].

### 3.5 Merge of Processed Non-Rings

In level planarity testing, separate components can 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. 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 4.** $\mathrm{minLL} = \min\{\,\mathrm{LL}(T) \mid T$ *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* $\mathrm{minLL} = \infty$.

As soon as a PQR-tree $T$ is identified as completely processed, minLL is updated by $\mathrm{minLL} = \min\{\mathrm{minLL}, \mathrm{LL}(T)\}$. 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 $T$ 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 $C$. Whenever inner faces are closed and there is a processed PQR-tree with an R-root, i. e., $\mathrm{minLL} < \infty$, we have to check whether $C$ can be included. We use the same mechanism as Leipert uses for $v$-singular components, i. e., those whose PQR-tree has exactly one leaf. If it fits, we set $\mathrm{minLL} = \infty$. 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-node as root can never be included this way.

### 3.6 Merge of Processed Rings

The algorithm maintains the invariant that at any time there is at most one PQR-tree $T^R$ containing an R-node. If another PQR-tree $T$ gets an R-root during the

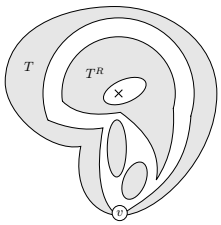



**Fig. 6.** Merge of rings

reduction of a *link vertex* $v$, we proceed as follows: 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 if $T^R$ is $v$-singular, which is solved as for level planarity. Afterwards the algorithm checks whether minML is small enough that $T$ fits "below" $T^R$ and the PQR-tree with the smallest low indexed level minLL and all others fit "between" $T$ and $T^R$. If one of the checks fails, $G$ is not radial planar as Lemma 1 states.

### 3.7 Completion

If at termination of the test algorithm there is no PQR-tree $T^R$ representing a ring graph, the graph is radial planar, because all other left PQR-trees can be placed side by side. Otherwise, if no other trees occurred after $T^R$ was detected, i. e., if 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.

## 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.

**Theorem 3.** *There is an $\mathcal{O}(|V|)$ time algorithm for computing a radial planar embedding of a radial $k$-level planar graph.*

### 4.1 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 suffices to designate cut edges while calculating the *st*-embedding. Since cut edges are defined as those crossing the imaginary ray, R-nodes save the current position of this ray with a new child, an ignored PQ-leaf similar to the direction indicators of [4]. It is called *ray indicator* and labelled with \$. All templates involving an R-node must be

aware of the ray indicator. If the pertinent sequence contains the ray indicator, this means that the edges represented by one part of the sequence must cross the ray. Thus they are moved over the ray indicator and are marked as cut edges, while the edges represented by the other part are non-cut edges.

### 4.2 Augmentation to an *st*-Graph

$G$ is augmented to an $st$-graph $G_{st}$ by a top down and a subsequent bottom up sweep of a slightly modified level planarity test. Each sink is connected to an appropriate vertex $v$ on a higher level. Therefore the corresponding leaves in the PQR-trees are not deleted but replaced with *sink indicators*, which are ignored by most parts of the algorithm. Nodes having only ignored children are also ignored. Since $v$ is not known before it is encountered, we do not discard processed PQR-trees, but save them in a list $\mathcal{T}^*$, similar to the list $\mathcal{T}$ of *active* PQR-trees. At most one PQR-tree $T^R \in \mathcal{T} \cup \mathcal{T}^*$ has an R-root. The necessary augmentations occur if an outer ring or a face is closed at a link vertex $v$.

Templates that create R-nodes must be applied to a node $X$ different from the root if all predecessors of $X$ have only one non-ignored child left. Then all PQ-leaves that are not descendants of $X$ are sink indicators and consequently are connected to the link vertex. Thus $X$ becomes the new R-root according to the applied template.

### 4.3 Computation of an Upward *st*-Embedding

If the graph has been augmented to an $st$-graph, we can generate a planar $st$-embedding. Leipert's algorithm achieves this by using the planar embedding algorithm of Chiba et al. [4] and a subsequent ordered DFS. Chiba's algorithm calculates a planar embedding for general planar graphs, where an edge $(s, t)$ ensures that $s$ and $t$ lie in the same face. Such an edge may violate radial planarity if the graph contains a ring. Fortunately, in our case the algorithm of Chiba also works without an $st$-edge because the levels and the faces of $s$ and $t$ are already fixed. It only has to be extended with the same new templates as the radial planarity test. The final DFS of Chiba's algorithm must be omitted, because cut edges may lead to an incorrect result. Instead we use the upward $st$-embedding $\mathcal{E}_u$, generated as an intermediate result, i. e., the order of the incoming edges for each vertex.

### 4.4 Computation of a Radial Embedding

Leipert's algorithm uses an ordered DFS to compute an ordering of the vertices for each level. Again we need an extension. In level planar embeddings there are no cut edges. Hence, the order of the incoming edges in $\mathcal{E}_u$ leads directly to the order of the source vertices in a level embedding $\mathcal{E}_l$. Due to the presence of cut edges this is not true in the radial case. Instead, the cut edges have to be treated in a special way. A cut edge cannot lie between two non-cut edges in the ordered

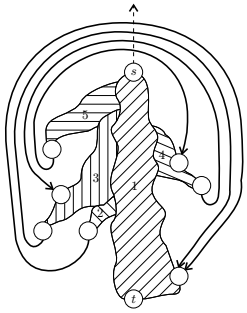



**Fig. 7.** Successive and ordered attachments of faces to the sides of the trunk

adjacency list of any vertex $v$ in $\mathcal{E}_u$. So cut edges occur only at the front or only at the back of the adjacency list. This leads to two different types of cut edges according to their position in the adjacency list. 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 vertex, and *counterclockwise* otherwise.

Our algorithm starts with an ordered DFS from $t$ in $\mathcal{E}_u$ using no 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 its level $\phi(v)$. Afterwards an ordered 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 $\phi(w)$. Counterclockwise cut edges are traversed from left to right and their respective source vertex $w$ is inserted at the end of its level $\phi(w)$. Now some unused non-cut edges and therefore some unvisited vertices can be reachable from $w$. 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. Figure 7 illustrates the algorithm. Since no vertex $v$ in $\mathcal{E}_u$ can be the target of both clockwise and counterclockwise cut edges, these edges do not cross and thus the algorithm calculates a correct embedding.

## 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 [3] with a new node type, R-nodes, representing a ring component of the graph. Further we have described the templates for our PQR-trees. We can compute a radial level planar embedding within the same linear time bounds as for planar embeddings [4] and level planar embeddings [15, 16, 18]. To check the practicability of our algorithm we have realised a prototypical implementation in C++ using the Graph Template Library [11].

## References


[1] C. Bachmaier, F. J. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time. Technical Report MIP-0303, University of Passau, June 2003.

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

[3] 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.

[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] L. S. Heath and S. V. Pemmaraju. Recognizing leveled-planar dags in linear time. In *Proc. Graph Drawing '95*, volume 1027 of *LNCS*, pages 300–311. Springer, 1996.

[13] 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.

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

[15] M. Jünger and S. Leipert. Level planar embedding in linear time. In *Proc. Graph Drawing '99*, volume 1731 of *LNCS*, pages 72–81. Springer, 1999.

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

[17] M. Jünger, S. Leipert, and P. Mutzel. Level planarity testing in linear time. In *Proc. Graph Drawing '98*, volume 1547 of *LNCS*, pages 224–237. Springer, 1998.

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

[19] 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.

[20] 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.

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