# Radiale Level-Planarität und -Einbettung in Linearzeit

Christian Bachmaier

Fakultät für Mathematik und Informatik
Universität Passau
D-94030 Passau
bachmaier@fmi.uni-passau.de

**Abstract:** Ein Graph mit einer geordneten $k$-Partitionierung seiner Knoten ist radial level-planar, wenn es eine strikte Auswärtszeichnung auf $k$ konzentrische Kreise ohne Kreuzungen gibt. Radiale Level-Planarität ist eine Erweiterung von Level-Planarität, bei der die Knoten auf $k$ horizontalen Linien und die Kanten strikt nach unten ohne Kreuzungen gezeichnet werden. Kennzeichnend für die Erweiterung sind Ringe, d. h. nicht level-planare zweifache Zusammenhangskomponenten.

Unsere Hauptergebnisse sind Linearzeitalgorithmen zum Test auf radiale Level-Planarität und zur Berechnung einer Einbettung. Wir führen die Datenstruktur PQR-Baum ein, bei der die neuen R-Knoten und die dazugehörigen Templates für die korrekte Behandlung von Ringen sorgen. Unsere Algorithmen sind daher eine Erweiterung von hoch entwickelten Level-Planaritäts- und -Einbettungs-Algorithmen auf Basis von PQ-Bäumen.

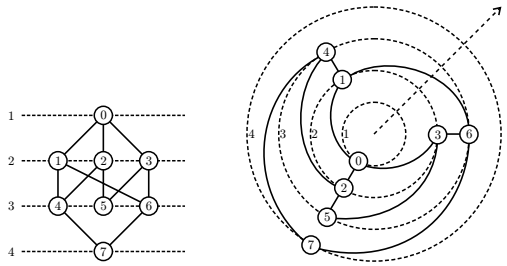
## 1 Einführung

Wir betrachten das Problem, einen Graphen mit einer gegebenen geordneten Knotenpartitionierung zu zeichnen. Diese Partitionierung kann auf anwendungsspezifischen Attributen (z. B. Hierarchien wie in [HMPT98]) oder struktureller Auswertung (z. B. Zentralitätsmaße wie in [BKW03]) beruhen oder durch einen Zeichenalgorithmus (z. B. das Sugiyama Framework [STT81]) eingeführt werden.

Formal ist ein *$k$-Level-Graph* $G = (V, E, \phi)$ mit einer Levelzuweisung $\phi\colon V \to \{1, \ldots, k\}$ mit $k \leq |V|$ gegeben, die seine Knoten in $k$ paarweise disjunkte Teilmengen $V = V^1 \dot\cup \cdots \dot\cup V^k$, $V^j = \phi^{-1}(j)$, $1 \leq j \leq k$ einteilt, so dass $\phi(u) \neq \phi(v)$ für jede Kante $(u, v) \in E$. Typischerweise werden die Knotenpositionen auf horizontale oder radiale Levels eingeschränkt um die Partitionierung sichtbar zu machen. Empirische Studien belegen, dass die Anzahl der Kreuzungen einer der Hauptfaktoren für die Lesbarkeit von Level-Zeichnungen ist [Pur97]. Das (horizontale) Level-Planaritätsproblem [DBN88, HP96, JLM98] ist die Frage, ob dieser Graph in der Ebene so gezeichnet werden kann, dass die Knoten des $j$-ten Levels auf der $j$-ten horizontalen Linie liegen und die Kanten strikt $y$-monotone Kurven sind, ohne sich zu schneiden. Die topologische Struktur einer derartigen Zeichnung wird durch eine Level-Einbettung beschrieben, die durch eine lineare Ordnung der Knoten auf jedem Level definiert wird. Level-Planarität wurde in der

Literatur eingehend untersucht. Als Hauptergebnisse sind Linearzeitalgorithmen für den Test auf Level-Planarität und für die Berechnung einer Einbettung [DBN88, HP96, HP99, Lei98, JL99, JL02, JLM98] zu nennen.

Unser Beitrag ist die Generalisierung von Level-Planarität zu radialer Level-Planarität. Dabei sind die Knoten auf $k$ konzentrische Kreise partitioniert. Ein $k$-Level-Graph ist *radial $k$-level-planar*, wenn es Permutationen der Knoten auf jedem radialen Level gibt, so dass die Kanten als strikt monotone Kurven von den inneren zu den äußeren Levels gezeichnet werden können, ohne sich zu schneiden. Diese Zeichnungen erweitern die radialen Baumzeichnungen von [Ead92], wo das Level eines Knotens durch seine Tiefe im Baum gegeben ist. Abbildung 1(b) zeigt eine radial level-planare Zeichnung des nicht level-planaren Graphen in Abb. 1(a). Ein anderes einfaches Beispiel ist ein in zwei Levels eingeteilter $K_{2,2}$, der radial level-planar ist, aber nicht 2-level-planar. Radial level-planare Zeichnungen sind nicht zu verwechseln mit den konzentrischen Darstellungen von planaren Graphen [Ull84]. Dort wird durch eine Breitensuche weder eine vorgegebene Leveleinteilung berücksichtigt, noch sind die Kanten monoton von innen nach außen.




(a) (b)

Abbildung 1: Ein radial level-planarer Graph mit einer radial level-planaren Zeichnung.

Jede level-planare Einbettung kann in eine radiale transformiert werden, indem man die Enden der Levels zu konzentrischen Kreisen „zusammenbiegt“. Das ermöglicht monotone Kanten vom Anfang eines Levels zum Ende eines anderen oder umgekehrt. Diese *Schnittkanten* schneiden genau einmal einen *Strahl* vom *Zentrum* der konzentrischen Levels nach Unendlich, der durch die Punkte läuft, an denen man die Levellinien „verklebt“ hat. Außerdem gibt es zwei Richtungen in denen Kanten um das Zentrum gelegt werden können. Daher muss eine *radiale Level-Einbettung* zusätzlich zu den Knotenordnungen Information über die Richtung der Schnittkanten abspeichern. Im Beispiel von Abb. 1(b) schneidet die Kante $(1, 6)$ den Strahl und ist damit eine Schnittkante im Uhrzeigersinn (wir folgen dazu den Kanten immer vom kleineren zum größeren Level). Es ist offensichtlich, dass ein radial level-planarer Graph ohne Schnittkanten auch level-planar ist. Also sind die Schnittkanten der essenzielle Unterschied zwischen radialer und horizontaler Level-Planarität.

# 2 Einordnung

Die Basis unseres Algorithmus ist der Linearzeitalgorithmus von Jünger, Leipert und Mutzel (JLM) [JL99, JL02, JLM98, Lei98] zum Test auf Level-Planarität. Dieser wiederum basiert auf vorhergehenden Arbeiten von Heath und Pemmaraju [HP96, HP99] und Di Battista und Nardelli [DBN88]. Wie die Knotenmethode zum Test auf normale Planarität [Eve79, LEC67] benutzen alle diese Methoden die *PQ-Baum*-Datenstruktur von Booth und Lueker [BL76]. Details dazu finden sich in [Bac04, BBF05]. Der JLM-Algorithmus geht Level für Level vor und speichert zulässige Permutationen der Knoten auf jedem Level effizient mit einer Menge von PQ-Bäumen. Durch ein komplexes Verfahren kann zusätzlich eine Einbettung in Linearzeit berechnet werden: Zuerst wird der Graph durch Hinzufügen von geeigneten Kanten in einen $st$-Graphen[1] umgewandelt. Dann wird eine $st$-Einbettung mit dem Algorithmus von Chiba et al. [CNAO85] berechnet. Daraus wird schließlich mit Hilfe einer geordneten Tiefensuche (DFS) eine Level-Einbettung erzeugt.

# 3 Radialer Level-Planaritäts-Test

## 3.1 Konzepte

Level-Planarität und radiale Level-Planarität scheinen ähnlich zu sein, dennoch gibt es aber essenzielle Unterschiede. Der JLM-Algorithmus verlässt sich auf die Tatsache, dass ein Level-Graph genau dann level-planar ist, wenn alle seine Zusammenhangskomponenten level-planar sind. Deshalb reicht es dort auch, jede Komponente auf Level-Planarität einzeln zu testen. Dies gilt im radialen Fall nicht mehr, wie man anhand von Abb. 2 sieht.

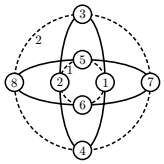


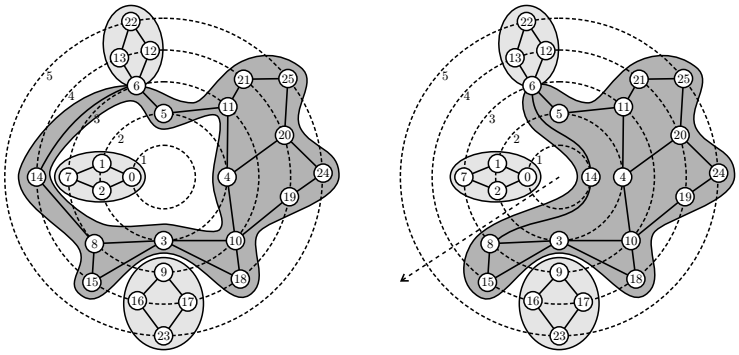

Abbildung 2: Nicht radialer level-planarer Graph bestehend aus radial level-planaren Komponenten.

Trivialerweise ist ein Graph radial level-planar, wenn er nur aus level-planaren Komponenten besteht. Daher müssen wir solche Komponenten (Ringe) betrachten, die radial level-planar sind, aber nicht level-planar. Ein *Ring* ist eine zweifache Zusammenhangskomponente eines Level-Graphen, die radial level-planar ist, aber nicht level-planar. Ein einen Ring enthaltender Level-Graph bezeichnen wir als *Ring-Graph*.

A priori ist nicht klar, ob eine Komponente ein Ring ist. Wir werden später sehen, wie Ringe erkannt werden. Nichtsdestotrotz untersuchen wir jetzt ein paar interessante Eigen-

[1] Ein $st$-Graph $G = (V, E, st)$ ist ein zweifach zusammenhängender gerichteter Graph mit einer Kante $(s, t) \in E$ und einer Nummerierung $st : V \to \{1, 2, \ldots, |V|\}$, so dass $st(s) = 1$, $st(t) = |V|$ und für alle $v \in V - \{s, t\}$ zwei Knoten $u, w \in V$ existieren mit $(u, v), (v, w) \in E$ und $st(u) < st(v) < st(w)$.

schaften von Ringen. Der Graph in Abb. 3(a) besteht aus vier zweifachen Zusammenhangskomponenten. Der einzige Ring wird durch die dunklere Schattierung angedeutet. Eine Komponente kann in eine andere *verschachtelt* werden, was hier auch unbedingt notwendig ist um die Planarität zu erhalten. Das passiert nur, wenn die „äußere“ Komponente ein Ring ist. Ringe sind nicht verwandt mit Zyklen. Jede zweifache Zusammenhangskompnente mit mindestens drei Knoten ist ein Zyklus, aber ob sie einen Ring bildet hängt von der Leveleinteilung ab. Wenn Knoten $14$ auf Level $1$ läge, dann würde der Graph keinen Ring enthalten, weil es bzgl. des Strahls in Abb. 3(b) keine Schnittkanten gibt.



(a) Ein Ring-Graph. (b) Kein Ring-Graph.

Abbildung 3: Ringe hängen von der Leveleinteilung ab.

Wir benutzen den JLM-Algorithmus, wenn der Eingabegraph keinen Ring enthält. Für Ring-Graphen muss der Algorithmus aber erweitert werden. Bevor wir im nächsten Abschnitt beschreiben, wie unser Algorithmus die zulässigen Permutationen der Knoten speichert, wenden wir uns noch einigen anderen Eigenschaften von Ringen zu. In jeder radialen Einbettung eines Ring-Graphs liegt das konzentrische Zentrum in einer inneren Fläche, der sog. *Zentrumsfläche*. Das Verschachteln von Ringen ist bestimmt durch einige charakteristische Parameter.

**Definition 1** *Für einen $k$-Level-Graphen $G$ mit einem Ring $R$ seien $\alpha_R$ und $\delta_R$ das minimale bzw. maximale Level von $G$ mit einem Knoten von $R$. Der* innere Radius $\beta_R$ *von $R$ sei das maximale Level mit einem Knoten der Zentrumsfläche von $R$ in einer bel. radial level-planaren Einbettung. Der* äußere Radius $\gamma_R$ *von $R$ sei das minimale Level mit einem Knoten der Außenfläche von $R$ in einer bel. radial level-planaren Einbettung.*

**Lemma 1** *Sei $G$ ein Level-Graph, bestehend aus zwei disjunkten Ringen $R$ und $S$. $G$ ist genau dann radial level-planar, falls $R$ und $S$ radial level-planar sind und $R$ in die Zentrumsfläche von $S$ passt oder umgekehrt, d. h. wenn*

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

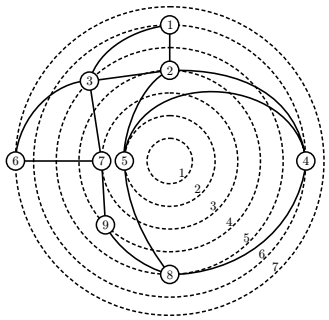



Abbildung 4: Ringextrema: $\alpha_R = 2$, $\beta_R = 6$, $\gamma_R = 3$, $\delta_R = 7$.

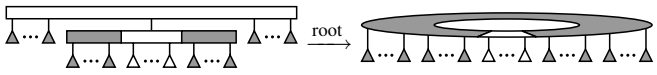
$R$ und $S$ können nicht einfach nebeneinandergelegt werden, wie es der JLM-Algorithmus macht. Weil $\beta_R$ und $\gamma_R$ von der Einbettung abhängen, muss bei der Berechnung der Einbettung darauf geachtet werden, möglichst viel Platz zur Platzierung andere Komponenten zu lassen, d. h., $\beta_R$ wird maximiert und $\gamma_R$ minimiert. Eine derartige Einbettung wird *level-optimal* genannt. A priori ist nicht klar, ob eine level-optimale Einbettung für jeden Ring existiert. Aber unser Algorithmus konstruiert eine, siehe [Bac04, BBF05].

### 3.2 R-Knoten

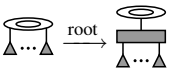
Das Ziel ist den JLM-Algorithmus zu erweitern. Die grundlegenden Ideen können dabei übernommen werden. Der Eingabegraph wird von oben nach unten abgearbeitet, was nun einer wellenförmigen Ausbreitung vom Zentrum entspricht. Der abgearbeitete Teil des Graphen wird durch eine Menge von PQR-Bäumen $\mathcal{T}$ repräsentiert. PQR-Bäume basieren auf PQ-Bäumen und enthalten einen neuen „R“ Knotentyp für die Ringe. Wie gewöhnlich [BL76] repräsentieren P-Knoten beliebige Permutationen ihrer Kinder und Q-Knoten die Reihenfolge ihrer Kinder und deren Umkehrung. *R-Knoten* ähneln Q-Knoten, aber sie haben einige Eigenschaften, die den Unterschied zwischen Ringen und anderen zweifachen Zusammenhangskomponenten repräsentieren. Ein R-Knoten wird als elliptischer Ring dargestellt, während (wie gewohnt) P-Knoten Kreise und Q-Knoten Rechtecke sind. Zulässige Operationen auf R-Knoten sind die Umkehrung der Reihenfolge der Kinder wie bei Q-Knoten und die neue *Rotation*. Da Ringe immer das Zentrum umschließen, darf man einen Ring drehen. Das entspricht einem Drehen des Graphen um das Zentrum und wird durch Versetzung einer Teilsequenz von Kindern des R-Knoten vom Anfang ans Ende der Kinderliste oder umgekehrt realisiert. Dabei wird die relative Reihenfolge der Kinder beibehalten. Auf einer zirkulär verketteten Liste geschieht dies implizit. R-Knoten können z. B. mit „Symmetrischen Listen“ [Bac04, BR04] implementiert werden. Dadurch kann in konstanter Zeit eingefügt, umgedreht und rotiert werden, was für die lineare Laufzeit des Tests entscheidend ist.

### 3.3 Neue Templates

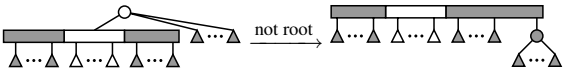
Die Hauptoperation auf PQ(R)-Bäumen ist REDUCE [BL76]. Durch die Anwendung verschiedener Templates, die den Baum lokal umbauen, stellt REDUCE sicher, dass eine gegebene Menge von Blättern des Baumes (welche den Knoten im Graphen entsprechen) in jeder kodierten Permutation aufeinanderfolgend ist. Zusätzlich zu den elf existierenden PQ-Baum-Templates brauchen wir zwölf neue zum Umgang mit PQR-Bäumen.




(a) Template Q6




(b) Template R1




(c) Template P9

Abbildung 5: Ausgewählte neue Templates.

Abbildung 5 zeigt einige Beispiele, der komplette Satz findet sich in [Bac04, BBF05]. Template Q6 enthält wichtige Schlüsselelemente: Zusätzlich zu den *leeren*, *partiellen* und *vollen* gibt es nun auch *rand-partielle* Q-Knoten, bei denen sich alle „dreckigen“ Kinder am Anfang und am Ende befinden. Dazwischen muss mindestens ein leeres Kind sein. Statt des rand-partiellen Kindes können auch leere Kinder auftreten, die optional durch partielle Q-Knoten begrenzt sind. Dafür gibt es aber ein anderes Template. Q6 ist eines der Templates, die nur auf die Wurzel eines PQR-Baumes angewendet werden dürfen – das ist ein Unterschied zur Einschränkung mancher PQ-Baum-Templates, die nur auf die Wurzel des dreckigen Unterbaums angewendet werden dürfen. Außerdem sieht man, wie R-Knoten entstehen. Mit den existierenden Q-Templates ist es nicht möglich, die vollen Kinder aufeinanderfolgend zu machen, d. h. die zweifache Zusammenhangskomponente ist nicht mehr level-planar. R-Knoten werden daher nur erzeugt, wenn es notwendig ist, d. h. wenn neu entdeckte Kanten eine level-planare Komponente zu einem Ring erweitern.

Da es nun einige Templates gibt, die R-Knoten erzeugen, ist es notwendig, R-Knoten auch auf der linken Seite von Templates zu behandeln. Wir führen deshalb R0–R3 ein. Vor der Anwendung eines R-Templates ist es i. A. notwendig, den R-Knoten zu rotieren. R-Templates sind die direkten Transformationen der jeweiligen Q-Templates mit Ausnahme von R1, wo ein Pseudo-Q-Knoten aus technischen Gründen eingeführt wird. Das erhält die Information, dass der PQR-Baum eine Ring-Komponente repräsentiert und ermöglicht die Berechnung einer Größenangabe, die angibt was „unter“ diese runde Komponente passt.

Die Existenz von partiellen Q-Knoten erfordert die Einführung der verbleibenden Templates P7–P9, Q5, Q7 und R4. Die Templates sind so ausgelegt, dass alle Fälle abgedeckt

werden, die beim Traversieren eines radial level-planaren Graphen auftreten. Zusätzlich erhalten sie die Invarianten, dass ein PQR-Baum mit einem R-Knoten einen R-Knoten enthält bis er gelöscht wird und dass in jedem Baum höchstens ein R-Knoten vorkommt und der nur als Wurzel. Das Hinzufügen neu entdeckter Kanten zur Datenstruktur REPLACE wird unverändert vom JLM-Algorithmus übernommen.

### 3.4 Verschmelzen von PQR-Bäumen

Falls PQ-Blätter, die ein und denselben Knoten im Graphen repräsentieren, in mehreren PQ-Bäumen auftauchen, müssen diese Bäume paarweise zu einem verschmolzen werden. Dazu gibt es die Verschmelzungs-Bedingungen A–E [HP96, HP99, Lei98, JL99, JL02, JLM98] die auf Größenangaben von und in den Bäumen basieren. Falls keine andere Bedingung passt, legt E die Komponenten einfach nebeneinander. Falls kein R-Knoten involviert ist, werden diese Operationen im Wesentlichen für PQR-Bäume übernommen. Falls jedoch einer der Bäume einen R-Knoten enthält, kann E nicht angewendet werden, d. h. im Gegensatz zum JLM-Algorithmus kann im radialen Fall eine Verschmelzung scheitern. Für PQR-Baum-Verschmelzungen, bei denen ein Baum einen R-Knoten enthält, stellen wir neue Bedingungen $C^R$ und $D^R$ zur Verfügung [Bac04, BBF05]. Jeweils ein R-Knoten in beiden Bäumen bedeutet, dass der Graph nicht radial level-planar ist.

Wenn ein Ring vorhanden ist, müssen andere (Nicht-Ring-)Komponenten in eine innere Fläche des Rings passen. Falls die andere Komponente ebenfalls ein Ring ist, muss der kleinere Ring in die Zentrumsfläche des größeren passen. Beide Tests führt unser Algorithmus immer dann aus, wenn eine innere Fläche durch REDUCE geschlossen wird. Auch hier werden bestimmte Größenangaben (ähnlich zu Lemma 1) verwendet, die größtenteils direkt in den Bäumen gespeichert werden. Nach einem positiven Test werden die abgearbeiteten Bäume aus $\mathcal{T}$ entfernt.

## 4 Radial level-planare Einbettung

JLM haben auch einen Algorithmus vorgestellt, der eine level-planare Einbettung für einen level-planaren Graphen erstellt. Auch er kann für die Berechnung einer radialen Level-Einbettung durch PQR-Bäume erweitert werden. Dabei gibt es aber wieder wesentliche Unterschiede zum horizontalen Fall, die es zu beachten gilt.

### 4.1 Kanteinbettung

Für eine Einbettung reicht die Knotenordnung alleine nicht aus. Zusätzlich müssen wir während der $st$-Einbettungsphase die Schnittkanten bestimmen. Da Schnittkanten definiert sind den Strahl zu schneiden, merken wir uns an jedem R-Knoten, wo der Strahl verläuft und zwar mit einem ansonsten ignorierten künstlichen Kind, dem *Strahl-Indikator*.

Falls eine dreckige Sequenz während eines REDUCE den Indikator enthält, müssen die Kinder, bzw. die entsprechenden Kanten im Graphen, einer (beliebigen) Seite den Strahl überqueren. Also werden sie über den Indikator „gezogen“ und als Schnittkanten markiert.

### 4.2 Erweiterung zu einem $st$-Graphen

Der Graph $G$ wird durch zwei Läufe eines modifizierten radialen Level-Planaritätstests, zuerst von oben nach unten und dann von unten nach oben, zu einem $st$-Graphen $G_{st}$ erweitert. Dabei wird jede Senke durch eine Kante zu einem Knoten $v$ verbunden, der später auf einem höheren Level die Fläche der Senke schließt. Da $v$ a priori nicht bekannt ist, werden die Blätter, die im PQR-Baum Senken des Graphen repräsentieren, durch anderweitig ignorierte *Senken-Indikatoren* ersetzt. Knoten, die nur ignorierte Kinder haben, werden ebenfalls ignoriert. Abgearbeitete/Ignorierte PQR-Bäume werden nicht gelöscht, sondern separat abgespeichert. Durch die Anwesenheit von ignorierten Knoten müssen u. U. R-Knoten generierende Templates nicht nur auf Wurzeln angewandt werden.

### 4.3 Berechnung einer Aufwärts-Einbettung

Nach der Erweiterung zu einem $st$-Graphen können wir eine planare $st$-Einbettung berechnen. JLM benutzen dazu den Algorithmus von [CNAO85], der eine planare Einbettung für allgemeine planare Graphen berechnet. Die Kante $(s, t)$ stellt dabei sicher, dass $s$ und $t$ in der gleichen Fläche liegen. Aber eine solche Kante verletzt die radiale Level-Planarität, wenn der Graph einen Ring enthält. Glücklicherweise kann aber auch [CNAO85] so erweitert werden, dass er mit PQR-Bäumen und nicht mehr mit PQ-Bäumen arbeitet und daher die $(s, t)$-Kante nicht braucht. Die DFS am Schluss muss allerdings entfallen, weil Schnittkanten deren Ergebnis verfälschen. Wir benutzen daher die als Zwischenergebnis berechnete Aufwärts-Einbettung[2], um darauf eine spezielle geordnete DFS anzuwenden, mit deren Hilfe direkt eine radial level-planare Einbettung bestimmt wird. Hier wird auch die Richtung der Schnittkanten, im oder gegen den Uhrzeigersinn, festgelegt.

## 5 Zusammenfassung

Wir haben einen neuen Algorithmus präsentiert, der radiale Level-Planarität eines $k$-Level Graphen in Linearzeit feststellt. Um die Praxistauglichkeit dieses Algorithmus zu testen, implementierten wir einen Prototyp in C++ auf Basis der Graph Template Library [GTL].

In [Bac04, BBF04] haben wir Erweiterungen der Algorithmen vorgestellt, die es erlauben, sowohl mit horizontalen als auch mit radialen Levels Kanten zwischen Knoten innerhalb des gleichen Levels zu betrachten. D. h. die Forderung nach strikt monotonen Kan-

[2] Eine Aufwärtseinbettung fixiert die Ordnung der eingehenden Kanten jedes Knotens.

ten nach unten/außen wurde abgewandelt auf schwache Monotonie. In [Bac04] wurde außerdem ein neues Zeichenverfahren für radiale Level-Graphen vorgestellt, welches die Knoten nach dem Median ihrer Nachbarn konzentrisch ausrichtet und Kanten als Spiralabschnitte zeichnet. Dadurch wird sichergestellt, dass keine Kreuzungen gezeichnet werden, die es in der Einbettung des Graphen gar nicht gibt. Im Gegensatz zu geradlinigen Kanten schneiden Spiralabschnitte keine Levellinien, die kleiner oder gleich des Levels ihres Quellknotens sind. In [BBF04] findet sich auch eine Abhandlung über Minoren[3] für radiale Level-Planarität im Stil von [HKL00].

## Literatur

[Bac04] C. Bachmaier. *Circle Planarity of Level Graphs*. Dissertation, Universität Passau, 2004.

[BBF04] C. Bachmaier, F. J. Brandenburg und M. Forster. Track Planarity Testing and Embedding. In P. Van Emde Boas et al., Hrsg., *Proc. Software Seminar: Theory and Practice of Informatics, SOFSEM 2004*, Jgg. 2, Seiten 9–17. MatFyzPres, 2004.

[BBF05] Christian Bachmaier, Franz Josef Brandenburg und Michael Forster. Radial Level Planarity Testing and Embedding in Linear Time. *Journal of Graph Algorithms and Applications*, 2005. Im Druck, Vorabversion unter `http://www.infosun.fmi.uni-passau.de/~chris/index.html#publications`.

[BKW03] U. Brandes, P. Kenis und D. Wagner. Communicating Centrality in Policy Network Drawings. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):241–253, 2003.

[BL76] K. S. Booth und 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.

[BR04] C. Bachmaier und M. Raitner. Improved Symmetric Lists. Bericht MIP-0409, Universität Passau, Oktober 2004.

[CNAO85] N. Chiba, T. Nishizeki, S. Abe und T. Ozawa. A Linear Algorithm for Embedding Planar Graphs Using PQ-Trees. *Journal of Computer and System Sciences*, 30:54–76, 1985.

[DBN88] G. Di Battista und E. Nardelli. Hierarchies and Planarity Theory. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(6):1035–1046, 1988.

[Ead92] P. Eades. Drawing Free Trees. *Bulletin of the Institute of Combinatorics and its Applications*, 5:10–36, 1992.

[Eve79] S. Even. *Algorithms*, Kapitel 7, Seiten 148–191. Computer Science Press, 1979.

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

[HKL00] P. Healy, A. Kuusik und S. Leipert. Characterization of Level Non-Planar Graphs by Minimal Patterns. In D.-Z. Du et al., Hrsg., *Computing and Combinatorics, COCOON 2000*, Jgg. 1858 von *LNCS*, Seiten 74–84. Springer, 2000.

[3] Minoren bei Graph-Planarität sind die Kuratowski-Subgraphen $K_{3,3}$ und $K_5$.

[HMPT98] C. Hundack, P. Mutzel, I. Pouchkarev und S. Thome. ArchE: A Graph Drawing System for Archeology. In G. Di Battista, Hrsg., *Proc. Graph Drawing 1997*, Jgg. 1353 von *LNCS*, Seiten 297–302. Springer, 1998.

[HP96] L. S. Heath und S. V. Pemmaraju. Recognizing Leveled-Planar Dags in Linear Time. In F. J. Brandenburg, Hrsg., *Proc. Graph Drawing 1995*, Jgg. 1027 von *LNCS*, Seiten 300–311. Springer, 1996.

[HP99] L. S. Heath und S. V. Pemmaraju. Stack and Queue Layouts of Directed Acyclic Graphs: Part II. *SIAM Journal on Computing*, 28(5):1588–1626, 1999.

[JL99] M. Jünger und S. Leipert. Level Planar Embedding in Linear Time. In J. Kratochvíl, Hrsg., *Proc. Graph Drawing 1999*, Jgg. 1731 von *LNCS*, Seiten 72–81. Springer, 1999.

[JL02] M. Jünger und S. Leipert. Level Planar Embedding in Linear Time. *Journal of Graph Algorithms and Applications*, 6(1):67–113, 2002.

[JLM98] M. Jünger, S. Leipert und P. Mutzel. Level Planarity Testing in Linear Time. In S. H. Whitesides, Hrsg., *Proc. Graph Drawing 1998*, Jgg. 1547 von *LNCS*, Seiten 224–237. Springer, 1998.

[LEC67] A. Lempel, Shimon Even und I. Cederbaum. An Algorithm for Planarity Testing of Graphs. In P. Rosenstiehl, Hrsg., *Theory of Graphs, International Symposium, Rome*, Seiten 215–232. Gordon and Breach, 1967.

[Lei98] S. Leipert. *Level Planarity Testing and Embedding in Linear Time*. Dissertation, Mathematisch-Naturwissenschaftliche Fakultät der Universität zu Köln, 1998.

[Pur97] H. C. Purchase. Which Aesthetic has the Greatest Effect on Human Understanding? In G. Di Battista, Hrsg., *Proc. Graph Drawing 1997*, Jgg. 1353 von *LNCS*, Seiten 248–261. Springer, 1997.

[STT81] K. Sugiyama, S. Tagawa und M. Toda. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.

[Ull84] J. D. Ullman. *Computational Aspects of VLSI*, Kapitel 3.5, Seiten 111–114. Computer Science Press, 1984.

## Werdegang des Autors

Christian Bachmaier, geb. 1975, absolvierte ab 1996 ein achtsemestriges Studium der Informatik mit Nebenfach Medizin-Informatik an der Universität Passau, das er Ende 2000 mit einem Diplom mit Auszeichnung und dem Fakultätspreis der Fakultät für Mathematik und Informatik abschloss. Nach einer kurzen Tätigkeit in der Wirtschaft ist er seitdem als wissenschaftlicher Mitarbeiter am Lehrstuhl von Prof. Dr. Franz J. Brandenburg tätig. Im Juli 2004 erfolgte nach dreieinhalb Jahren seine Promotion in Informatik mit Auszeichnung und der Bewertung „summa cum laude“. Während der Entstehung dieses Dokuments befindet sich der Autor in einem Forschungsaufenthalt unter der Betreuung von Prof. Dr. Ulrik Brandes an der Universität Konstanz. Die wissenschaftlichen Interessen von Christian Bachmaier liegen in den Bereichen Algorithmus-Engineering, effiziente Datenstrukturen, Graphalgorithmen und Zeichnen von Graphen.