

Layout-Algorithmen – eine Übersicht

Oliver Kopp
Universität Stuttgart
Institut für Softwaretechnologie
Universitätsstrasse 38
70569 Stuttgart
koppo@stud1.informatik.uni-stuttgart.de

Abstract

This article gives an overview on the existing automatic graph drawing algorithms. It starts presenting aesthetic criterias and continues in outlining algorithms for drawing trees, planar, layered and general graphs. Finally further literature notes are given.

1. Einführung

In dieser Seminararbeit geht es um das automatisierte Layout von Graphen. Ein Layout ist die visuelle Repräsentation eines Graphen $G = (V, E)$.

Graphen mit weniger als zehn Knoten lassen sich relativ einfach von Hand zeichnen. Falls jedoch grössere Graphen mit hundert oder mehr Knoten visualisiert werden sollen, so ist dies nur automatisiert möglich. Nach einer Vorstellung von Ästhetik-Aspekten und derer Messbarkeit vor, Layout-Algorithmen nach Möglichkeit erfüllen sollen, werden bekannte Vertreter von Algorithmen vorgestellt, die z.B. auch im Compilerbau zum Einsatz kommen. Abschliessend werden die Ergebnisse zusammengefasst und ein Ausblick auf mögliche Vertiefungen gegeben.

2. Ästhetik

Ein gutes Layout definiert sich durch die einfache Lesbarkeit des beschriebenen Graphen: Die Eigenschaften des Graphen, z.B. der Zusammenhang oder der Weg zwischen zwei Knoten, soll einfach erkennbar sein und der Betrachter soll komplexe Zusammenhänge schnell erfassen und leicht lernen können [2, S.20].

Die Ästhetik eines Graphen kann durch folgende Kriterien quantifiziert werden (nach [12, S.2], [15, S.3]):

- Anzahl der Kantenschnitte
Die Anzahl der Schnitte soll möglichst gering sein.

- Anzahl der Kantenknickpunkte
Die Anzahl der Knickpunkte soll möglichst gering sein.
- Anzahl der Schnitte zwischen Kanten und Knoten
Die Anzahl der Schnitte soll möglichst gering sein.
- Länge der Kanten
Die Länge der Kanten soll möglichst minimal sein und möglichst gleich groß.
- Richtung der Kanten
Die Mehrheit der Kanten soll in die selbe Richtung zeigen.
- Verteilung der Knoten
Die Knoten sollen möglichst gleich verteilt sein.
- Benötigte Fläche
Die Fläche, die das Layout benötigt soll möglichst gering sein.
- Verhältnis von Länge zu Breite des Layouts
Das Verhältnis von Länge zu Breite soll möglichst eins betragen.
- Winkelauflösung
Die benutzten Winkel der an- und abgehenden Kanten sollen minimal sein und die Winkel selbst maximal groß. Die Winkel selbst werden zwischen benachbarten Kanten eines Knotens gemessen.
- Anzahl der übernommenen Symmetrien
Enthält der zu visualisierende Graph Symmetrien, so sollen diese auch im Layout wiedergespiegelt werden.
- Anzahl der erhaltenen Ordnungen
Sind die Kinder eines Knotens geordnet, soll sich diese Ordnung im Layout widerspiegeln.

Im Allgemeinen ist es nicht möglich, zwei dieser Kriterien gleichzeitig zu optimieren. Abbildung 1 zeigt zwei sich signifikant unterscheidende Layouts von K_4 . Links wurden die Kantenschnitte minimiert und rechts die Symmetrien wiedergespiegelt. Bei Spezialfällen lassen sich auch mehrere Kriterien gleichzeitig optimieren, wie unten am Fall der Bäume zu sehen ist.

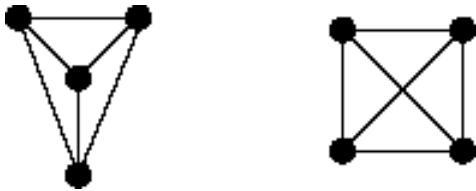


Abbildung 1. Zwei verschiedene Layouts für K_4

Die Komplexität der Berechnung ist ein weiteres Kriterium, das nicht direkt mit Ästhetik, sondern vielmehr den Aufwand der Berechnung betrifft. Ein hoher Aufwand ist ein Problem, da die Umsetzung der meisten o.g. Kriterien in NP liegen. Es wurden verschiedene Heuristiken entwickelt, die versuchen, den Aufwand zu minimieren. Ein Auszug aus den bekanntesten Methoden wird im Folgenden näher erläutert.

3. Bäume

Bäume werden normalerweise planar, geradlinig und aufwärts gezeichnet. Dabei bedeutet „aufwärts“, dass die Väter über den Kindern gezeichnet werden. Weiterhin ist es zielführend, die Fläche zu minimieren, Symmetrien und isomorphe Teilbäume ersichtlich zu machen (vgl. [5, S.16]).

3.1. Reingold-Tilford-Algorithmus

Der Reingold-Tilford-Algorithmus ist ein rekursiver Algorithmus zum Zeichnen von binären Bäume in Linearzeit (nach [5, S.18]):

- 1: Zeichne linken Teilbaum L
- 2: Zeichne rechten Teilbaum R
- 3: **wenn** es zwei Kinder gibt **dann**
- 4: Platziere die Zeichnungen der Teilbäume mit horizontalem Abstand 2
- 5: Platziere die Wurzel eine Ebene höher und in der Mitte der Kinder
- 6: **sonst**
- 7: Platziere die Wurzel mit horizontalem Abstand 1 von dem Kind

Dieser Algorithmus erfüllt fünf der sechs Ästhetik-Kriterien. Wie in Abbildung 2 zu sehen ist, optimiert er

nicht die Fläche, die bei ihm auf $O(n^2)$ beschränkt ist.

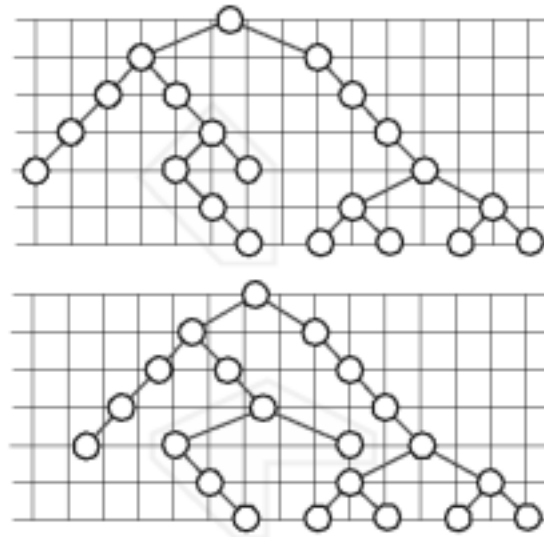


Abbildung 2. oben: Ergebnis des Reingold-Tilford-Algorithmus, unten: flächenoptimiert

Die Breite eines Binärbaums lässt sich in polynomieller Zeit mittels linearer Programmierung minimieren. Das Problem der minimalen Breite wird jedoch NP-hart, falls man auf ganzzahlige Koordinaten beschränkt ist.

3.2. Chans Algorithmus

Verzichtet man auf die Forderung, dass die Kinder auf der gleichen Ebene angeordnet werden sollen, so kann mit der Methode von Chan ([4],[11, S.49ff.]) eine fast-lineare maximale Flächennutzung erreicht werden.

Prinzipiell geht Chans Algorithmus wie der Reingold-Tilford-Algorithmus vor. Der Unterschied liegt in der Platzierung der Kinder und der Wurzel:

- 1: Zeichne linken Teilbaum L
- 2: Zeichne rechten Teilbaum R
- 3: **wenn** $|L| < |R|$ **dann**
- 4: Kombiniere Layout mit der „left-rule“
- 5: **sonst**
- 6: Kombiniere Layout mit der „right-rule“

Die „left-“ und „right-rules“ sind in Abbildung 3 verdeutlicht. Durch diese Regeln wird jedoch noch keine lineare Flächennutzung erreicht. Um dies zu erreichen, wird der Baum an einer bestimmten Stelle unterteilt und die Kinder an dieser Stelle mit erweiterten left- bzw. right-rules kombiniert.

Die erweiterte left-rule besagt, dass der rechte Teilbaum um einen Betrag verschoben werden darf, so lange die Wurzel dieses Teilbaums rechts von dessen Vater v bleibt. Abbildung 4 veranschaulicht die Unterteilung

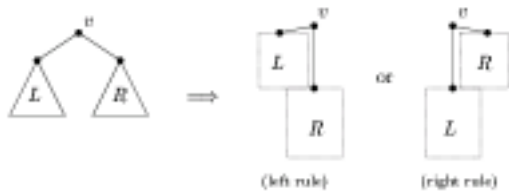


Abbildung 3. Basisregeln der Chan-Methode

des Baums. Diese Unterteilung hat eine nahezu lineare

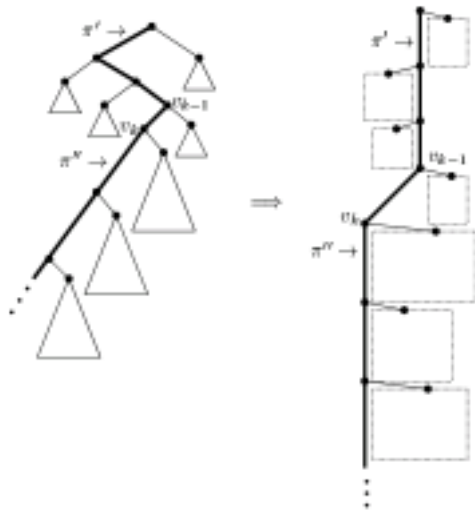


Abbildung 4. Die Unterteilung des Baums in der Chan-Methode

Flächenbegrenzung zur Folge (vgl. [4, S.8]).

3.3. Walker-Algorithmus

Der Reingold-Tilford-Algorithmus wurde von Walker 1990 erweitert, um Bäume mit Grad > 2 in Linearzeit zu erzeugen [3]. Bei der Erweiterung muss darauf geachtet werden, dass benachbarte Kinder immer den gleichen Abstand haben und nicht links- oder rechtsbündig ausgerichtet werden (vgl. Abbildung 5 a) und b)). Eine Mittelwertbildung von zwei Läufen (von links nach rechts und von rechts nach links) bringt keine Besserung, da einzelne Knoten geclustert werden (vgl. Abbildung 5 c)). Nach Walkers Methode werden die Kinder gleichmäßig ausgerichtet (vgl. Abbildung 5 d)).

4. Planare Graphen

Allgemeine planare Graphen können in linearer Zeit gezeichnet werden (vgl.[10, S.2]). Das Ergebnis ist jedoch

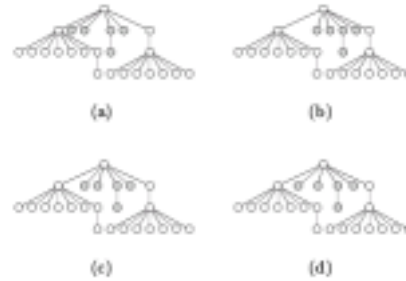


Abbildung 5. Illustration der Schwierigkeiten der Kind-Ausrichtung

meist eine Zeichnung mit mit kleinen Winkeln, was den Graphen nicht ästhetisch aussehen lässt. Manche planare Graphen lassen eine konvexe Zeichnung zu, d.h. alle Innenflächen und der Gesamtgraph sind konvex, wodurch die Winkel nicht zu klein werden (vgl. [13, S.66ff.] oder [11, S.33ff.]).

Auch wenn das Problem der kleinen Winkel gelöst ist, können die Koordinaten in R liegen und Häufungen von Knoten auftreten. Für den Spezialfall eines dreifach-zusammenhängenden Graphen löst der Algorithmus von Kant dieses Problem (vgl. [10]).

Sind die Graphen auf den Grad vier beschränkt, so ist eine weitere Möglichkeit, sie darzustellen die orthogonale Einbettung. Die Kanten sind dabei aufeinanderfolgende horizontale und vertikale Segmente, d.h. es existieren Kantenknickpunkte.

5. Hierarchische Graphen

Ein hierarchischer Graph ist ein zusammenhängender Graph, in dem Ebenen L_0, \dots, L_k die Knoten partitionieren. Jede Kante verbindet nur Knoten in aufeinanderfolgende Ebenen (vgl. [17]). Der Sugiyama-Algorithmus (vgl. [11, S.87ff]) ist der bekannteste Vertreter von Algorithmen, die hierarchische Graphen layouts.

Der Algorithmus erhält als Eingabe einen gerichteten Graphen, der im Laufe des des Algorithmus hierarchisiert wird. So ist der Algorithmus auch zur Visualisierung von Graphen geeignet, die sich von hierarchischen Graphen nur geringfügig unterscheiden.

Der Algorithmus besteht aus fünf Phasen:

1. Zyklen entfernen
2. Knoten Ebenen zuweisen
3. Minimierung der Kantenkreuzungen
4. Absolute Knotenposition festlegen
5. Kanten zeichnen

Die erste beiden Schritte dienen dazu, den Graphen zu hierarchisieren. Falls die Knoten eine Größe > 0 haben, ist

der letzte Schritt erforderlich, in dem eventuell zusätzliche Kantenkickpunkte eingeführt werden.

Lässt man die Knotenpositionen in einer Schicht fest und ändert nur die Positionen der anderen Schicht, so ist das Problem der Kreuzungen NP-hart. Deshalb existieren verschiedene Heuristiken, dieses Problem zu lösen. Nahezu jede basiert auf der Kreuzungszahlmatrix. Diese gibt für jedes Knotenpaar einer Ebene an, wieviele Kreuzungen entstehen, wenn ein Knoten v vor bzw. hinter einem anderen Knoten w liegt. Für Abbildung 6 ist die Kreuzungszahlmatrix:

C	e	f	g
e	0	2	1
f	1	0	2
g	0	3	0

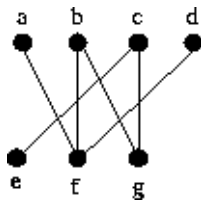


Abbildung 6. Beispielgraph zur Demonstration der Kreuzungszahlmatrix

Als Beispiel sei der greedy-swich-Algorithmus aufgeführt, welcher sich an der Funktionsweise von Bubblesort orientiert (vgl. [11, S.105]): Für jedes benachbarte Knotenpaar v, w (v liegt vor w) wird geprüft, ob die Kreuzungszahl geringer ist, wenn w vor v liegt. Falls dieses zutrifft, werden die beiden Knoten vertauscht. Dies wird so lange durchgeführt, bis keine Vertauschungen mehr durchgeführt werden.

Nach Abschluss des dritten Schritts ist bekannt, wie die Knoten relativ zueinander liegen. Anschließend müssen die absoluten Positionen festgelegt werden. Die Y-Koordinate kann auf ein Vielfaches der Ebene festgesetzt werden. Schwieriger wird es bei der X-Koordinate. Eine Heuristik zur Bestimmung der X-Koordinate ist die „Pendelmethode“ (vgl. [14, S.97ff.]): Es wird vorwärts von Ebene zu Ebene und danach wieder rückwärts das Verfahren angewandt. Dabei sind die Positionen der Vorgängerknoten fest und die dazugehörigen Kanten ziehen an den Knoten. Der Graph ist quasi ein Multi-Pendel: Die Knoten entsprechen den Kugeln und die Kanten den Fäden. Fixiert man die Kugeln der ersten Schicht an der Decke, so schwingen die unteren Kugeln unter Einfluß der Gravitationskraft in balancierte Positionen. Es wird nun jede Kugel „ausgependelt“, bis sich keine Veränderungen mehr ergeben.

6. Kräftebasierte Algorithmen

Die kräftebasierte Methode ist nach [8, S.2] eine verbreitete Methode, um allgemeine Graphen zu zeichnen. Algorithmen, die auf dieser Methode basieren, bestehen aus zwei Komponenten: Dem Energiemodell, mit dem die Qualität des Layouts quantifiziert wird und zweitens dem Algorithmus, der daraus ein Optimum für das Layout errechnet. Am Ende der Berechnung ist das System in einem Gleichgewichtszustand: Die Summe der Kräfte, die auf einen Knoten wirken ist Null. Alternativ formuliert, ist die potentielle Energie minimal. Es wird davon ausgegangen, dass minimale Energie in einem solchen System einem ästhetischen Layout entspricht. In der konkreten Visualisierung werden die Kanten immer geradlinig gezeichnet, so dass sich das Problem auf die Positionierung der Knoten reduziert.

6.1. Spring-Embedder

Der Spring-Embedder war erste praktikable Methode, um allgemeine Graphen zu zeichnen. In diesem Modell wird jeder Knoten durch einen elektrisch geladenen Ring und jede Kante durch verbindende Federn ersetzt. Jedes Knotenpaar stößt sich gegenseitig ab und benachbarte Knoten, die durch eine Feder verbunden sind, werden zusammengezogen. Die Methode versucht ein Gleichgewicht zwischen diesen beiden sich entgegengerichteten Kräften zu finden.

Der Optimierungsalgorithmus funktioniert wie folgt: Es wird eine vordefinierte Anzahl von Schritten ausgeführt. Der Initialschritt platziert die Knoten an zufällige Positionen. In jedem Schritt werden für jeden Knoten die Kräfte errechnet, die auf ihn wirken. Danach wird er in diese Richtung bewegt. Die Länge der Bewegung wird von Schritt zu Schritt geringer.

Dadurch werden folgende Ästhetikkriterien erfüllt ([7, S.262]):

- Keine Überlappung der Knoten
- Kurze Kanten
- Minimierung der Kantenkreuzungen
- Geringer Überlappung der Kanten und der Knoten

6.2. Kamada und Kawais Methode

In Kamada und Kawais Modell wird ein Graph als ein System von Federn modelliert. Jedes Knotenpaar wird durch eine Feder, deren Länge im Grundzustand der graphentheoretischen Distanz beträgt, verbunden. Die tatsächliche Länge der Federn ist die aktuelle Distanz der

Knoten. Der Optimierungsalgorithmus minimiert die globale Energie E

$$E = \sum_{u,v \in V} \frac{1}{d_{u,v}^2} (l(u,v) - Ld_{u,v})^2$$

Dabei ist $l(u,v)$ die Länge der Feder zwischen u und v , L die Länge einer einzelnen Kante und $d_{u,v}$ die graphentheoretische Distanz zwischen u und v ist.

Die Methode von Kamada und Kawai erzeugt Layouts mit einer ähnlichen Qualität wie der Spring-Embedder, jedoch mit dem Vorteil, dass sie ohne Aufwand auf Graphen mit gewichteten Kanten angepasst werden kann.

6.3. Erweiterungen und andere Einsatzgebiete

Durch die quadratische Zeitkomplexität beider Verfahren, eignen sie sich für Graphen mit bis zu 50 Knoten. Sollen grössere Graphen dargestellt werden, so muss der Algorithmus optimiert werden. Ein Ansatz hierfür ist, entfernt wirkende Kräfte zusammenzufassen. Werden nur Kräfte zwischen benachbarten Knoten betrachtet, so lässt sich der Aufwand auf $O(n(\log n)^2)$ beschränken (vgl. [6, S.9]).

Kräftebasierte Verfahren sind auch in anderen Gebieten einsetzbar. In [9] wird gezeigt, wie Kräfte dazu benutzt werden können, um von einem optimalen Layout in ein nächstes überzuleiten, das um Knoten erweitert oder reduziert wurde. Diese Überleitung wichtig, damit der Benutzer sein Bild des Graphens einfacher modifizieren kann und gleichzeitig der aktuelle Graph ästhetisch visualisiert ist.

7. Vergleich

In diesem Abschnitt sollen drei repräsentative Vertreter von Layoutalgorithmen verglichen werden. Folgende Tabelle stellt ausgewählte Ästhetik-Kriterien dar und wie sie in den entsprechenden Algorithmen realisiert werden. Dabei bedeutet „J“, dass das Kriterium voll umgesetzt wird, „(J)“, dass es auf die verwendete Heuristik ankommt, ob das Kriterium umgesetzt wird und schließlich „N“, dass der Algorithmus keine Optimierung danach vornimmt (vgl. [16, S.46f.]):

Algorithmus	Rheingold-Tilford	Sugiyama	Spring-Embedder
Graphtyp	Baum	Hierarchisch	Allgemein
Aufwand	$O(n)$	$O(n^2)$	$O(n^2)$
Erhaltung von Isomorphie	J	(J)	N
Erhaltung von Hierarchien	J	J	N
Minimierung der Kantenschnitte	J	J	N
Einheitliche Kantenlängen	N	N	(J)

Es ist zu sehen, dass der Grad der Allgemeinheit des Graphs bestimmt, wie viele Ästhetik-Kriterien realisiert werden. Dies ist auch nachvollziehbar, denn je allgemeiner ein Graph ist, desto weniger Informationen über die Struktur und ein dazu gehöriges ästhetisches Layout existieren.

8. Darstellung von großen Graphen

In diesem Abschnitt wird ein anderes Gebiet der Graphenvisualisierung vorgestellt. Die Darstellung von Layouts, deren Ausdehnung die Bildschirmgröße nicht übersteigt, stellt keine Schwierigkeit dar. Falls sich der Graph jedoch über viele Bildschirmseiten erstreckt, müssen Methoden gefunden, diese so dargestellt werden, dass der Betrachter ohne grössere Schwierigkeiten durch ihn browsen kann. Dabei wird davon ausgegangen, dass der Betrachter zum einen den TBD!!

Zur Darstellung des relevanten Ausschnitts und des restlichen Graphen gibt es prinzipiell drei verschiedene Ansätze: Die lineare Sicht, die Fischaugensicht (vgl. [14, S.209ff]) und die Multi-Level-Technik (vgl. [6, S.2]). In Abbildung 7 werden die beiden erstgenannten Ansätze gegenübergestellt.



Abbildung 7. Lineare Sicht und zwei verschiedene Fischaugensichten

Bei der linearen Sicht wird der Graph stufenlos vergrößert bzw. verkleinert. Dabei wird das Bild nicht verzerrt. Weiterhin kann der Detaillierungsgrad der derzeitigen Anzeige variiert werden, so dass die bei der aktuellen Sicht nicht benötigten Details ausgeblendet werden können. Zur Orientierung des Benutzers kann eine „Minimap“ eingeblendet werden, in der der Benutzer den kompletten Graphen sieht und mittels eines Rechtecks angezeigt bekommt, in welchem Bereich er sich gerade befindet.

Die Fischaugensicht stellt das Bild des Graphen verzerrt da. Dadurch erlaubt sie dem Benutzer in einem Fenster eine Orientierung und alle Details des aktuellen Bereichs zu sehen. Der Punkt des Interesses wird vergrößert im Detail dargestellt. Je weiter Knoten vom Fokuspunkt entfernt liegen, desto kleiner erscheinen sie im Anzeigefenster. Die Abbildungsfunktion muss bijektiv sein, damit der Knoten anklickbar ist. Meistens werden für diese Abbildung \sin , \arctan oder $\frac{Kx}{Ax+1}$ gewählt. Die Abbildungsfunktion wird

teilweise auch in zwei Bereiche unterteilt, wobei Knoten nahe bei dem Fokuspunkt linear skaliert werden und weiter entfernte Knoten stärker verkleinert, damit der Bereich um den Knoten herum weiterhin gut sichtbar bleibt.

Die Multi-Level-Technik geht einen anderen Weg: Sie versucht, den Graph in verschiedene Abstraktionsebenen einzuteilen. Jede Ebene repräsentiert dabei eine Abstraktionsschicht. Während des Browsens wechselt man die Schichten je nach benötigter Abstraktionsebene. Besteht keine Information über die existierenden Schichten, so muss diese bestimmt werden. Eine Heuristik für die Aufteilung in Partitionen ist die so genannte „Min-Cut k -Way Partition“ (vgl. [11, S.197]). Es werden k Cluster so gebildet, dass die Anzahl der Kanten, die die gebildeten Cluster verbinden, minimal wird. Um die Zeit der Berechnung gering zu halten, werden minimale und maximale Größen der zu bildeten Cluster vorgegeben.

9. Zusammenfassung und Ausblick

In dieser Seminararbeit wurden einige Layoutalgorithmen für spezielle und allgemeine Graphen skizziert.

Zuerst wurde das Layout von Bäumen betrachtet. Durch den Walker-Algorithmus ist eine Methode gefunden worden, Bäume planar, geradlinig, hierarchisch und ordnungserhaltend in Linearzeit darzustellen. Damit wurde das Problem des effizienten Layouts von Bäumen gelöst. Die Schwierigkeit, hierarchische Graphen darzustellen, ist durch Sugiyamas Methode auch weitgehend gelöst.

Die Visualisierung von planaren Graphen stellte eine Herausforderung dar, da Algorithmen, die ästhetische Layouts erzeugen, nur für bestimmte Typen von planaren Graphen einsetzbar sind.

Die anschliessend vorgestellten kräftebasierten Verfahren sind ein allgemeines Verfahren, die auch auf planare Graphen angewendet werden können, wodurch zwar im Allgemeinen keine Kreuzungsfreiheit garantiert werden kann, jedoch ästhetische Layouts.

Da ein Mensch dreidimensionale Strukturen besser als zweidimensionale durchdringen kann, sollte Information dreidimensional dargestellt werden. Es gibt jedoch es für dreidimensionale Graphen noch wenige spezielle Layoutalgorithmen und deshalb sollte in diesem Gebiet intensiv geforscht werden. Insbesondere der Bereich der Übertragung der zweidimensionalen in dreidimensionale Algorithmen und der Verringerung der Komplexität des visualisierten Graphen.

Das jährlich stattfindende „Graph Drawing Symposium“ und die dabei ausgerufenen Wettbewerbe, einen gegebenen Graphen ästhetisch zu zeichnen, zeigen, dass die Aufgabe, ein ästhetisches Layout für Graphen zu finden, inzwischen eine eigene Disziplin der Informatik, mit immer noch ungelösten Problemen ist.

Für weitergehende Studien sei auf www.graphdrawing.org und das kommentierte Literaturverzeichnis von Battista et al (siehe [1]) verwiesen.

Literatur

- [1] G. D. Battista, P. Eades, R. Tamassia, und I. G. Tollis. Algorithms for Drawing Graphs: an Annotated Bibliography. Technical report, Department of Computer Science, Brown University, 1994. <http://www.cs.brown.edu/rt/gd-biblio.html>. (Zitiert in Abschnitt 9.)
- [2] F. J. Brandenburg. Graph Drawing – past - present - future. Technical report, University of Passau, 2002. <http://www.csse.monash.edu.au/gfarr/research/GraphDrawing02-Mel.ppt>. (Zitiert in Abschnitt 2.)
- [3] C. Buchhein, M. Jünger, und S. Leipert. Improving Walker’s Algorithm to Run in Linear Time. Technical report, Universität zu Köln, 1999. <http://citeseer.nj.nec.com/buchheim02improving.html>. (Zitiert in Abschnitt 3.3.)
- [4] T. M. Chan. A Near-Linear Area Bound for Drawing Binary Trees. Technical report, University of Waterloo, 2001. <http://citeseer.nj.nec.com/chan01nearlinear.html>. (Zitiert in Abschnitten 3.2 and 3.2.)
- [5] I. F. Cruz und R. Tamassia. *Graph Drawing Tutorial*. Brown Univeristy, 1998. <http://www.cs.brown.edu/people/rt/papers/gd-tutorial/gd-constraints.pdf>. (Zitiert in Abschnitten 3 and 3.1.)
- [6] P. Gajer, M. Goodrich, und S. Kobourov. A fast multi-dimensional algorithm for drawing large graphs, 2000. <http://citeseer.nj.nec.com/gajer00fast.html>. (Zitiert in Abschnitten 6.3 and 8.)
- [7] R. Hadany und D. Harel. A Multi-scale Algorithm for Drawing Graphs Nicely. Aus P. Widmayer, G. Neyer, und S. Eidenbenz, editors, *Graph-Theoretic Concepts in Computer Science, 25th International Workshop, WG '99, Ascona, Switzerland, June 17-19, 1999, Proceedings*, Ausgabe 1665 von *Lecture Notes in Computer Science*, Seiten 262–277. Springer, 1999. (Zitiert in Abschnitt 6.1.)
- [8] D. Harel und Y. Koren. A Fast Multi-Scale Method for Drawing Large Graphs. *Journal of Graph Algorithms and Applications*, 6(3):179–202, 2002. <http://citeseer.nj.nec.com/harel00fast.html>. (Zitiert in Abschnitt 6.)
- [9] M. L. Huang, P. Eades, und J. Wang. Online Animated Graph Drawing using a Modified Spring Algorithm. Technical report, The University of Newcastle, 1998. <http://citeseer.nj.nec.com/huang98online.html>. (Zitiert in Abschnitt 6.3.)
- [10] G. Kant. Drawing Planar Graphs Using the Canonical Ordering. Technical report, Utrecht University, 1992. <http://citeseer.nj.nec.com/kant96drawing.html>. (Zitiert in Abschnitt 4.)
- [11] M. Kaufmann und D. Wagner, editors. *Drawing Graphs — Methods and Models*. Springer, 2001. (Zitiert in Abschnitten 3.2, 4, 5, 5 and 8.)

- [12] O. Niggeman und B. Stein. A Meta Heuristic for Graph Drawing. Technical report, Universität Paderborn, 2000. <http://oliver-niggemann.de/work/publications/avi00.pdf>. (Zitiert in Abschnitt 2.)
- [13] T. Nishizeki und N. Chiba. *Planar Graphs: Theory and Algorithms*. North-Holland, 1988. (Zitiert in Abschnitt 4.)
- [14] G. Sander. *Visualisierungstechniken für den Compilerbau*. Verlag Pirrot, 1997. (Zitiert in Abschnitten 5 and 8.)
- [15] S. Sim. Automatic Graph Drawing Algorithms. Technical report, University of Toronto, 1996. <http://citeseer.nj.nec.com/sim96automatic.html>. (Zitiert in Abschnitt 2.)
- [16] K. Sugiyama. *Graph Drawing And Applications*. World Scientific, March 2002. (Zitiert in Abschnitt 7.)
- [17] Verschiedene. Dictionary of Algorithms and Data Structures. <http://www.nist.gov/dads/>. (Zitiert in Abschnitt 5.)