

Zeichnen gerichteter Graphen*

Professor Dr. Petra Mutzel
Lehrstuhl für Algorithm Engineering
Fachbereich Informatik LS11, Universität Dortmund
Otto-Hahn-Str. 14, 44227 Dortmund

Die Schichtenmethode, nach ihrem Erfinder auch die „Sugiyama“-Methode genannt (vgl. [10]), eignet sich gut zum Zeichnen gerichteter azyklischer Graphen. Diese können so gezeichnet werden, daß ihre hierarchische Struktur gut zur Geltung kommt. Abbildung 1 zeigt eine mit der Schichtenmethode erstellte Zeichnung. Die Knoten sowie die Kantenknicke des gerichteten Graphen liegen auf sogenannten „Schichten“, und die Kanten werden möglichst so gezeichnet, daß sie in eine Richtung, z.B. von oben nach unten, zeigen.

Die Schichtenmethode gliedert sich in drei Schritte:

- (1) Verteilung der Knoten auf die Schichten: Hier wird versucht eine Einteilung zu erhalten, so daß für jede (gerichtete) Kante (u, v) gilt: $\text{Schicht}(u) < \text{Schicht}(v)$. Die y -Koordinaten der Knoten sind mit der Schicht festgelegt. Sei k die Anzahl der Schichten. Dann erhalten die Knoten, die Schicht i zugeordnet sind ($i = 1, \dots, k$), die y -Koordinate $k - i$.
- (2) Umordnung der Knoten innerhalb der Schichten: Hierbei wird versucht, die Anzahl der Kantenkreuzungen zu minimieren.
- (3) Bestimmung der x -Koordinate der Knoten in den Ebenen: Hier möchte man u.a. die Anzahl der Kantenknicke minimieren.

Im einzelnen werden wir zu jedem der drei Schritte einen oder mehrere verschiedene Algorithmen angeben.

1 Schichteneinteilung

Wir transformieren den azyklischen Graphen (d.h. der Graph besitzt keine gerichteten Kreise) in eine einfache Hierarchie.

Eine k -stufige Hierarchie ist ein gerichteter Graph $G = (V, A)$, für den die folgenden Bedingungen gelten:

- (i) V ist in k disjunkte Untermengen partitioniert, d.h. $V = V_1 \cup V_2 \cup \dots \cup V_k$ mit $V_i \cap V_j = \emptyset$ für $i \neq j$. V_i bezeichnet die i -te Ebene und k die Tiefe der Hierarchie.
- (ii) Für alle $e = (v_i, v_j) \in A$, $v_i \in V_i$, $v_j \in V_j$ gilt: $i < j$.

Eine k -stufige Hierarchie heißt *einfach*, falls zusätzlich gilt:

- (iii) A ist in $k-1$ disjunkte Teilmengen partitioniert, d.h. $A = A_1 \cup A_2 \cup \dots \cup A_{k-1}$ mit $A_i \cap A_j = \emptyset$ für $i \neq j$, wobei $A_i \subseteq V_i \times V_{i+1}$, $i = 1, \dots, k-1$.

*Unterrichtsmaterial anlässlich der Schnupperuni 2005 des FB Informatik der Universität Dortmund

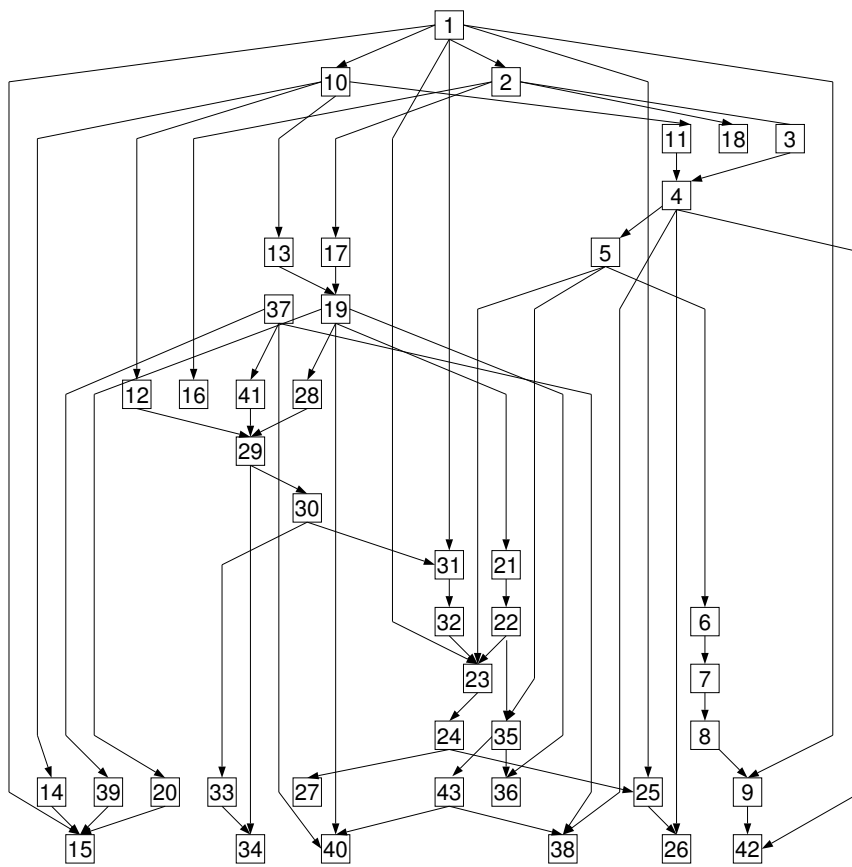


Abbildung 1: Eine hierarchische Zeichnung eines gerichteten Graphen.

1.1 Topologisches Sortieren

Eine Numerierungsfunktion $\text{num}: V \rightarrow \mathbb{N}$ heißt *topologische Ordnung* der Knoten eines Graphen $G = (V, A)$, falls gilt: $\text{num}(v) < \text{num}(w)$ für alle Kanten $(v, w) \in A$.

Eine topologische Sortierung existiert nur für azyklische Graphen. Falls der Graph nicht azyklisch ist, kann man eine kleinste Anzahl von Kanten „umdrehen“ (d.h. aus (v, w) wird (w, v)), so daß G azyklisch wird. Das Problem, eine kleinste solche Menge von Kanten zu finden, ist NP-schwierig und wird auch „feedback arc set Problem“ genannt. Dieses Problem wollen wir hier nicht behandeln, man kann sich jedoch leicht heuristische Verfahren ausdenken.

Der folgende Algorithmus liefert eine topologische Sortierung. Dabei bezeichnet $\text{indeg}(v)$ die Anzahl der Kanten, die von anderen Knoten zu v führen. Ist $\text{indeg}(v) = 0$ für einen Knoten, so wird dieser auch als *Wurzel* bezeichnet. Der Algorithmus wählt sich iterativ einen Wurzelknoten v ; dieser erhält eine Nummer $\text{num}(v)$, die sich in jedem Schritt erhöht. Danach wird v aus dem Graphen entfernt. Dadurch entstehen eventuell neue Wurzeln im Graphen $G' = (V', A')$.

Algorithmus Top_Sort();

- (1.0) $V' := V; A' := A; \text{count} := 0;$
- (2.0) While (es existiert ein $v \in V'$ mit $\text{indeg}(v) = 0$) {
- (2.1) $\text{num}(v) := \text{count};$
- (2.2) $\text{count}++;$
- (2.3) $V' := V' \setminus \{v\};$
- (2.4) Für alle $(v, w) \in A'$: $A' := A' \setminus \{(v, w)\}$ und update $\text{indeg}(v);$
- (3.0) If ($\text{count} < |V|$) then: G ist zyklisch;

Der Algorithmus bricht ab, wenn kein Knoten mehr existiert, der keine Vorgänger besitzt. Falls noch unbesuchte Knoten existieren, müssen diese Teil eines Kreises sein, denn kreisfreie Subgraphen besitzen immer Knoten mit $\text{indeg} = 0$. Falls G azyklisch ist, so liefert der Algorithmus eine k -stufige Hierarchie mit $k = |V|$, nämlich: $\text{num}(v)$ gibt die Schicht an, in der Knoten v zu liegen kommt. Der Algorithmus kann leicht abgeändert werden, so daß eine k -stufige Hierarchie mit $k \leq |V|$ erzeugt werden kann. Man kann, z.B., in einem Schritt alle Wurzeln auswählen und diesen diesselbe Nummer num zuteilen.

1.2 Tiefensuche zum Aufbau einer Hierarchie

Eine weitere Methode besteht darin, den Graphen rekursiv zu durchsuchen. Dies leistet der folgende rekursive „Depth-First-Search“ Algorithmus. Die Methode untersucht den Graphen in der Art, daß sie jeweils so tief wie möglich gehen möchte.

Algorithmus Bestimme_Tiefe();

- (1) Für alle $v \in V$: $\text{besucht}[v] := \text{false};$
- (2) Für alle $v \in V$: if not $\text{besucht}[v]$ then DFS($v, 1$);

Prozedur DFS(v ,tiefe);

- (1) besucht[v] :=true;
- (2) tiefe[v] :=tiefe;
- (3) Für alle $(v, w) \in A$: If (not besucht[w]) then DFS(w ,tiefe+1);

Die Schicht des Knotens v wird durch tiefe[v] bestimmt. Diese hängt von der Reihenfolge der Knotenwahl ab. Die Schichteneinteilung führt nicht immer zu einer Hierarchie, d.h. manche Kanten zeigen von unten nach oben, selbst wenn der Graph azyklisch ist.

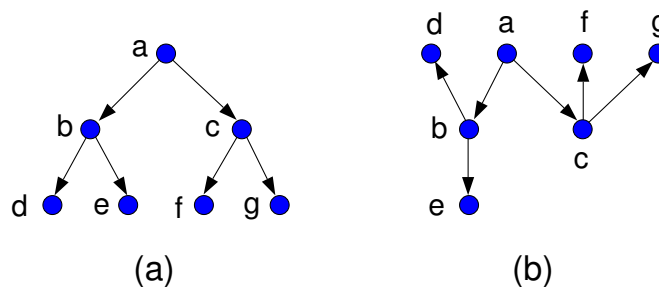


Abbildung 2: Das Ergebnis der Schichteneinteilung hängt beim Bestimme_Tiefe()-Algorithmus von der Bearbeitungsreihenfolge der Knoten ab. Hier sind zwei mögliche Ergebnisse des DFS-Algorithmus desselben Graphen gezeigt.

Beispiel: Betrachte den Graphen in Abbildung 2. Durchläuft man den Graphen in der Reihenfolge a, b, d, e, c, f, g , so erhält man die Schichteneinteilung, die in Abbildung 2(a) dargestellt ist. Durchläuft man ihn jedoch in der Reihenfolge d, f, g, a, b, e, c , so erhält man die Schichteneinteilung, die in Abbildung 2(b) gezeigt ist. Hierbei treten einige rückwärtsgerichtete Kanten auf, obwohl der Graph azyklisch ist, also hierarchisch gezeichnet werden könnte (d.h. alle Kanten von oben nach unten).

Da wir für die folgenden Schritte eine Hierarchie benötigen, müssen wir uns um die folgenden Fälle kümmern.

Die Kante $e = (v, w)$, $v \in V_i$, $w \in V_j$:

- (i) $i < j$: o.k.
- (ii) $i > j$: die Kante $e = (v, w)$ wird temporär durch (w, v) ersetzt
- (iii) $i=j$: Umleitung der Kante e über die nächste Ebene $i+1$ (bzw. $i-1$), d.h. Erzeugung eines neuen Knotens d auf Tiefe $i+1$ und neue Kanten (v, d) und (w, d) .

1.3 Umwandlung in eine einfache Hierarchie

Die nachfolgenden Schritte (Kreuzungsreduktion und Knotenpositionierung) vereinfachen sich wesentlich, wenn nur Kanten zwischen benachbarten Ebenen auftreten. Um eine einfache Hierarchie zu erhalten, ersetzen wir jede Kante der Länge l (d.h., Kante $(v, w) \in A$, $v \in V_i$, $w \in V_j$ mit $l := j - i$) durch l Kanten der Länge 1. Die neu eingeführten Knoten heißen „Dummy-Knoten“.

Dadurch wird eine Vereinfachung der weiteren Berechnungen erreicht, da nur noch benachbarte Ebenen betrachtet werden müssen. Die Kantenknicke entsprechen nun den Dummy-Knoten, wodurch verhindert wird, daß Kanten durch Knoten laufen.

2 Umordnung der Knoten: Kreuzungsreduktion

Gegeben ist nun eine einfache Hierarchie. Eine Kante in einer einfachen Hierarchie wird als direkte gerade Linie zwischen den beiden Endknoten gezeichnet. Wichtig für die Lesbarkeit einer Zeichnung ist die geringe Anzahl an Kantenüberkreuzungen. Die Anzahl der Kantenüberkreuzungen ergibt sich als die Summe der sich paarweise überschneidenden Kanten. Diese hängt von der Reihenfolge der Knoten in den Schichten ab.

Selbst das Kreuzungsminimierungsproblem für zwei Schichten ist bereits NP-schwierig. In der Praxis geht man folgendermaßen vor. Man fixiert die erste Schicht, bestimmt eine Ordnung der Knoten auf der 2.-ten Schicht, so daß die Anzahl der Kreuzungen zwischen beiden Schichten minimiert wird. (Auch dieses Problem ist NP-schwierig.) Dann fixiert man die 2.-te Schicht und sortiert die 3.-te Schicht, etc. So traversiert man den Graphen von oben nach unten und von unten nach oben, bis sich die Anzahl der Kreuzungen nicht mehr vermindert.

Im folgenden beschreiben wir zwei Verfahren, die in der Praxis zur Kreuzungsminimierung bei einer fixierten Schicht verwendet werden. Doch zunächst geben wir zwei Methoden an, die die Anzahl der Kantenkreuzungen bei gegebenen fixierten Schichten berechnen.

2.1 Berechnung der Anzahl der Kreuzungen zwischen zwei Schichten

Um festzustellen, ob eine Anordnung der Knoten innerhalb der Schichten zu einer Verbesserung oder Verschlechterung gegenüber des momentanen Zustands führt, muß die Anzahl der Kantenüberkreuzungen berechnet werden. Eine einfache Methode hierfür ist es, für jedes Kantenpaar zu testen, ob sie eine Kreuzung besitzen oder nicht. Dies benötigt jedoch viel zu lange Rechenzeit, denn es wären mindestens E^2 Operationen notwendig. Wir stellen im folgenden drei verschiedene Methoden vor, die diesen Aufwand um ein Vielfaches verringern.

2.1.1 Berechnung durch Matrizen

Wir betrachten die Adjazenzmatrix des 2-Schichten-Subgraphens $G_i = ((V_i, V_{i+1}), A_i)$. Hierbei entsprechen die Zeilen den Knoten der Schicht V_i und die Spalten der Schicht V_{i+1} . Der Eintrag der Matrix in Zeile v und Spalte w , m_{vw} , ist 1 genau dann wenn die Kante (v, w) in dem Subgraphen existiert und 0 sonst.

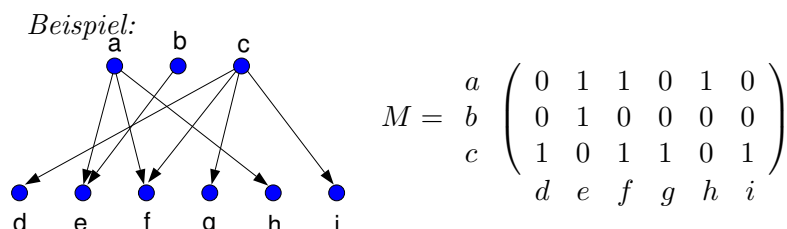


Abbildung 3: Ein Graph und dessen Adjazenzmatrix M .

Die Permutation der Schicht V_i sei gegeben durch $\sigma_i = v_1, \dots, v_{|V_i|}$, die Permutation der Schicht V_{i+1} durch $\sigma_{i+1} = w_1, \dots, w_{|V_{i+1}|}$. Seien $e_1 = (v_j, w_\beta)$ und $e_2 = (v_k, w_\alpha)$, $v_j, v_k \in V_i$,

$w_\beta, w_\alpha \in V_{i+1}$, und $j < k$. Die Anzahl der Überkreuzungen zwischen den Nachbarn der beiden Knoten v_j und v_k ist gegeben durch

$$C(v_j, v_k) = \sum_{\alpha=1}^{|V_{i+1}|-1} \sum_{\beta=\alpha+1}^{|V_{i+1}|} m_{j\beta} m_{k\alpha}.$$

Dies entspricht der Summe der 1-Einträge in Zeile v_j , die rechts über den Einträgen von v_k stehen.

Beispiel: Die Anzahl der Überkreuzungen zwischen Nachbarn der Knoten a und b ist gegeben durch $C(a, c) = 3 + 1 + 1 = 5$. Oder anders formuliert, die Kante (c, d) ist in 3 Kreuzungen mit Nachbarn von c involviert, die Kanten (c, f) und (c, g) in jeweils eine Überkreuzung, sowie (c, i) in keine Überkreuzung.

Die Laufzeit der Berechnung ist $O((|V_{i+1}||V_i|)^2)$. Die Anzahl der Kreuzungen ist oft viel geringer als die maximal mögliche Anzahl von $|V_i||V_{i+1}|$. Schneller ist die folgende Methode.

2.1.2 Plane Sweep Methode

Wir wollen die Anzahl der Kantenüberkreuzungen im 2-Schichten-Graphen $G_i = ((V_i, V_{i+1}), A_i)$ berechnen. Eine vertikale Gerade („Sweepline“) wird von links nach rechts über den Raum bewegt. Zunächst jedoch werden die Knotenkoordinaten entsprechend der Position der Knoten in der Permutation angeordnet, so daß eine x -Koordinate von höchstens einem Knoten belegt wird. Die Sweepline bewegt sich von Knoten zu Knoten entlang wachsender x -Koordinaten. In jedem Schritt ist die Kreuzungszahl derjenigen Kanten, die sich vollständig links der Sweepline befinden, schon bekannt, während die Kreuzungszahl der restlichen Kanten erst noch berechnet werden muß. In jedem Sweepline-Schritt wird die Anzahl der neu traversierten Kantenüberkreuzungen zur Summe hinzuaddiert.

Laufzeit der Berechnung: $O(|V_i| + |V_{i+1}| + |A_i| + c_i)$, wobei $|A_i|$ die Anzahl der Kanten zwischen den beiden Schichten bezeichnet und c_i die Anzahl der Überkreuzungen.

Da jedoch die Anzahl der Überkreuzungen quadratisch von der Anzahl der Kanten abhängen kann, kann auch diese Rechenzeit noch sehr lange dauern. Das folgende Verfahren enthält keinen quadratischen Faktor mehr und ist somit um ein Vielfaches schneller als die beiden oben vorgestellten Verfahren.

2.1.3 BJM-Verfahren

Das BJM-Verfahren wurde 2002 von Barth, Jünger und Mutzel (s. [1]) vorgestellt und revolutionierte das Kreuzungszählen sowohl in Theorie als auch in der Praxis. Das Verfahren ist einfach, kann in wenigen Zeilen beschrieben und programmiert werden und hat eine sehr schnelle Rechenzeit.

Die Idee beruht auf der Beobachtung, dass die Anzahl der Kreuzungen äquivalent zu der Anzahl von Inversionen einer Folge ist, die zunächst aus den Permutationen der Knotenfolgen berechnet wird. Man erhält die Folge, indem man alle Kanten lexikographisch nach ihren Anfangs- und dann nach ihren Endknoten sortiert. Z.B. erhält man aus dem Graphen aus Abb. 3 die Kantenfolge $(a, e), (a, f), (a, h), (b, e), (c, d), (c, f), (c, g), (c, i)$. Danach entfernt man die Anfangsknoten und erhält die Zahlenfolge $\langle e, f, h, e, d, f, g, i \rangle$. In dieser zählt man nun einfach die Inversionen, d.h., wenn ein größeres Element in der Folge vor einem kleineren steht, dann

wird dies jeweils als eine Inversion gezählt. Z.B. steht das vierte Element 'e' hinter f und h und erzeugt somit zwei Inversionen. Das Element 'd' erzeugt vier Inversionen, u.s.w. Insgesamt enthält diese Folge 8 Inversionen.

Wie kann man nun die Anzahl der Inversionen einer Folge zählen? Das geht sehr leicht mit dem einfachen Algorithmus „Insertion Sort“. Dieser arbeitet die Folge von links nach rechts ab, nimmt sich ein Element vor und schiebt dieses schrittweise soweit nach vorne bis links von ihm nur noch kleinere oder gleiche Elemente stehen. Die Laufzeit dieses einfachen Algorithmus ist im schlimmsten Fall jedoch auch quadratisch, denn sie stimmt ziemlich genau mit der Anzahl der Inversionen der Folge überein.

Ein anderer einfacher Sortier-Algorithmus, der jedoch rekursiv arbeitet ist der „Merge-Sort“. Dieser hat Laufzeit $O(|A_i| \log |A_i|)$, hängt also nicht von der Anzahl der Inversionen bzw. Kreuzungen ab. Die Idee hierbei ist es, die Folge in zwei Hälften zu teilen, und dann zunächst die linke Hälfte, dann die rechte Hälfte zu sortieren. Dies geschieht auch jeweils mittels Teilung. Irgendwann bestehen die beiden Hälften nur noch aus einem Element, dann werden die sortierten Teilfolgen wieder gemischt. Dies ist einfacher als zu sortieren: Es genügt ein Durchlauf; man nimmt jeweils das kleinere Element aus beiden Teilfolgen und man erhält die gemischte Folge.

Ein einfacherer nicht-rekursiver Algorithmus ist der mittels einer Datenstruktur, des sogenannten „Accumulator-Trees“ (s. [1]),. Dies ist ein Baum, der sich an jeder Stelle merkt, wieviele Elemente bereits in dem Teilbaum rechts von dieser Stelle enthalten sind. Mit Hilfe dieser Datenstruktur kann der „Insertion Sort“ Algorithmus zu einer Laufzeit von $O(|A_i| \log |V_i|)$ gebracht werden. Experimentelle Tests haben gezeigt, dass dieser Algorithmus nicht nur theoretisch, sondern auch praktisch der schnellste ist.

2.2 Barycenter Heuristik zur Kreuzungsreduktion

Gegeben ist die fixierte Ordnung der Schicht V_i . Diese sei $1, \dots, v_i \rightarrow \sigma(1) \dots \sigma(v_i)$. Wir berechnen für alle Knoten w in Schicht V_{i+1} deren *Barycenter-Wert*:

$$B(w) := \frac{1}{|N(w)|} \sum_{v \in N(w)} \sigma(v),$$

wobei $N(w) = \{(v, w) \in A_i \mid v \in V_i\}$. Danach werden die Knoten der Schicht V_{i+1} gemäß ihrem Barycenter-Wert sortiert.

Beispiel: Betrachte den Graphen in Abbildung 3. Der Barycenter-Wert des Knotens a ist $\frac{(2+3+5)}{3} = 3\frac{1}{3}$. Der Wert des Knotens b ist 2 und des Knotens c ist $\frac{(1+3+4+6)}{4} = 3.5$. Eine Sortierung der Kanten entlang ihrer Barycenter-Werte ergibt die Reihenfolge b, a, c .

2.3 Median Heuristik zur Kreuzungsreduktion

Gegeben ist die fixierte Ordnung der Schicht V_i . Diese sei $1, \dots, v_i \rightarrow \sigma(1) \dots \sigma(v_i)$. Diesmal berechnen wir für alle Knoten w in Schicht V_{i+1} deren *Median-Wert*, der der Position des mittleren Nachbarn von w entspricht. Danach werden die Knoten der Schicht V_{i+1} gemäß ihrem Median-Wert sortiert.

Beispiel: Die Median-Werte der Knoten in Abbildung 3 betragen 3 für Knoten a , 2 für Knoten b und 3 (bzw. 4) für Knoten c . Demnach kann die Reihenfolge der Knoten in Schicht $i + 1$ entweder b, a, c oder b, c, a sein.

Beide Verfahren, die Barycenter sowie die Median Heuristik werden häufig in der Praxis verwendet, da sie im Vergleich zu den anderen heuristischen Verfahren sehr gute Ergebnisse liefern und auch sehr schnell sind.

2.4 Optimale 2-Schichten Kreuzungsminimierung

Unter anderem sind wir aktiv in der Forschung auf dem Gebiet der Kreuzungsminimierung tätig. Mittels Methoden der mathematischen Programmierung bzw. der ganzzahligen Optimierung können wir das Problem der 2-Schichten-Kreuzungsminimierung bei einer fixierten Schicht in nur wenigen Sekunden für die praxisrelevanten Graphen optimal lösen (vgl. Folien, sowie [6]). Stichworte hierbei sind: „Linear Ordering Problem“, Ganzzahlige Optimierung, Lineare Programmierung, Schnittebenenverfahren, sowie „Branch-and-Cut“.

3 Positionierung der Knoten

Zur Bestimmung der x -Koordinaten gibt es verschiedene Verfahren. Ziel ist es hier, die x -Koordinaten der Knoten so festzulegen, daß die Anzahl der Kantenknicke minimal ist sowie die Breite der Zeichnung nicht zu breit wird. Insbesondere sollen die langen Kanten, die über mehrere Schichten gehen (durch Dummy-Knoten verbunden) höchstens zwei Knicke haben, die sich am Anfang und am Ende der Kante befinden. Dazwischen sollte diese Kante möglichst vertikal verlaufen. Es gibt mehrere in der Praxis verwendete Verfahren. Ein vereinfachtes Verfahren wird im folgenden Abschnitt dargestellt.

3.1 Knotenpositionierung mittels Prioritätswerten

Hier werden zunächst den Knoten die folgenden Anfangskoordinaten zugeordnet. Jeder Knoten an Position k in der Permutation der Schicht i erhält die Anfangs- x -Koordinate $k + 1$. Nun werden die Positionen der Knoten auf allen Schichten schrittweise verbessert. Ähnlich wie bei der Kreuzungsminimierung, wird auch hier zunächst die Position der Knoten der ersten Schicht festgelegt, und daraus gute Positionen der Knoten der 2.-ten Schicht berechnet. Danach werden diese fixiert und die Positionen der 3.-ten Schicht berechnet, etc. Dies wird solange durchgeführt, bis es keine Verbesserungen mehr gibt.

Wir nehmen nun an, daß die Positionen der Schicht i fixiert sind, und wir die x -Koordinaten der Knoten der Schicht $i+1$ berechnen wollen. Dazu ordnen wir jedem Knoten w aus Schicht $i+1$ seinen Prioritätswert zu. Dieser ergibt sich aus der Anzahl der benachbarten Knoten zu Schicht i . Ist w ein Dummy-Knoten und ist sein einziger Nachbar in Schicht i auch ein Dummy-Knoten, dann wollen wir an dieser Stelle möglichst keine Kantenknicke erzeugen. Deswegen geben wir dem Knoten w in diesem Fall einen sehr hohen Prioritätswert.

Nun wählen wir den Knoten mit der höchsten Priorität und setzen ihn an die Wunschstelle. Sukzessive wählen wir nun die Knoten mit der nächsthöheren Priorität, etc. Beim Auswählen der Knotenposition des Knotens w müssen wir beachten, daß die Position ganzzahlig sein sollte, und nie zwei verschiedene Knoten dieselbe Position erhalten dürfen. Außerdem muß die Sortierung der Knoten auf der Schicht erhalten bleiben. Positionen der Knoten mit niedrigerer Priorität dürfen verschoben werden, während die Positionen der Knoten mit höherer Priorität nicht mehr verändert werden dürfen.

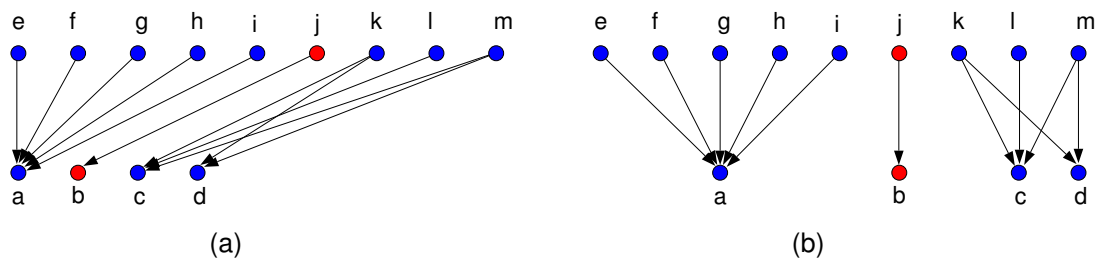


Abbildung 4: (a) Die Anfangspositionen der Knoten der $(i + 1)$ -ten Schicht sowie (b) die Endpositionen der Knoten.

Beispiel: Betrachte den Graphen in Abbildung 4. Abbildung 4(a) zeigt die Anfangsposition der Knoten. Die Prioritäten der Knoten in Schicht $i + 1$ sind 5, 100, 3 und 2, wobei b ein Dummy-Knoten ist, und daher besonders hohe Priorität bekommt. Nun wählen wir Knoten b aus und setzen ihn an seine Wunschstelle, also genau unter j . Dadurch müssen wir die Positionen der Knoten c und d aktualisieren. Die neuen Positionen der Knoten wechseln also von $\{1, 2, 3, 4\}$ auf $\{1, 6, 7, 8\}$. Knoten a besitzt die zweithöchste Priorität und wird auf seine Wunschposition 3 gesetzt. Nun wählen wir Knoten c und setzen ihn auf Position 8. Eine Aktualisierung aller Positionen ergibt $\{3, 6, 8, 9\}$. Auch die Wunschposition des Knotens d ist 8, diese Position ist jedoch schon besetzt, deswegen müssen wir Position 9 auswählen. Die Endpositionen sind in Abbildung 4(b) gezeigt.

4 Weitere Informationen

Weitere Informationen zum Zeichnen gerichteter Graphen entnehmen Sie bitte [5, 10, 7, 9]. Allgemeinverständliche Übersichtsartikel zum Zeichnen von Graphen finden sich in [8, 2, 4]. Eine Bibliographie über die Forschungsprobleme im Gebiet des Graphen Zeichnens ist [3]. Weitere Informationen sowie Software befindet sich auf dem Server unter der Adresse <http://www.mpi-sb.mpg.de/~mutzel/>.

Literatur

- [1] W. Barth, M. Jünger, and P. Mutzel. Simple and efficient bilayer cross counting. *Journal of Graph Algorithms and Applications (JGAA)*, 8(2):179–194, 2004. <http://www.cs.brown.edu/publications/jgaa/>.
- [2] F.J. Brandenburg, M. Jünger, and P. Mutzel. Algorithmen zum automatischen Zeichnen von Graphen. *Informatik Spektrum*, 20(4):199–207, 1997.
- [3] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.
- [4] P. Eades and P. Mutzel. Graph drawing algorithms. In M. Atallah, editor, *CRC Handbook of Algorithms and Theory of Computation*, chapter 9, pages 9–1–9–26. CRC Press, 1999.

- [5] P. Eades and K. Sugiyama. How to draw a directed graph. *Journal of Information Processing*, pages 424–437, 1991.
- [6] M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica, Special Issue on Graph Drawing*, 16(1):33–59, 1996.
- [7] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications (JGAA)* (<http://www.cs.brown.edu/publications/jgaa/>), 1(1):1–25, 1997.
- [8] P. Mutzel. *Automatisiertes Zeichnen von Diagrammen*, pages 425–431. Jahrbuch 1995 der Max-Planck-Gesellschaft, Vandenhoeck & Ruprecht Verlag, Göttingen, 1995.
- [9] G. Sander. Graph layout through the VCG tool. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 194–205. Springer-Verlag, 1995.
- [10] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Trans. Syst. Man Cybern.*, SMC-11(2):109–125, 1981.