Christian-Albrechts-Universität zu Kiel

Bachelorarbeit

Algorithmen zur Layerzuweisung

Philipp Döhring

29. September 2010

Institut für Informatik
Arbeitsgruppe Echtzeitsysteme und Eingebettete Systeme

Prof. Dr. Reinhard von Hanxleden

betreut durch:
Dipl.-Inf. Miro Spönemann

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Kiel,

_____

Zusammenfassung

Das hierarchische Layout ist ein heute gängiges und in vielen Bereichen zum Standard gewordenes Verfahren zur übersichtlichen Darstellung von Graphen mit der Layerzuweisung als zentraler Phase. In dieser Arbeit wird anhand von konkreten Algorithmen ein Überblick über diesen durchaus anspruchsvollen Abschnitt des hierarchischen Ansatzes mit besonderem Blick auf die ästhetischen Anforderungen, die hierbei berücksichtigt werden müssen, gegeben. Die bisher bekannten Vorgehensweisen zur Layerzuordnung erweitern wir anschließend um eine Vor- und Nachbearbeitungsphase, mittels welcher auch besonders große Knoten sinnvoll berücksichtigt werden können.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Das hierarchische Layout ist ein heute relativ bekanntes und häufig verwendetes Verfahren zur Darstellung von auf gerichteten Graphen basierenden Diagrammen. Hierbei werden die einzelnen Knoten unterschiedlichen horizontalen oder vertialen Hierarchieebenen (*Layer*) zugeordnet und alle Knoten eines Layers entsprechend mit der selben X-, respektive Y-Koordinate versehen. In dieser Arbeit werden Layer dabei grundsätzlich als vertikale Einteilungen der Graphen verstanden, wobei ein Beispiel eines derartig visualisierten Graphen in Abbildung 1.1 dargestellt wurde. Als typisches Anwendungsbeispiel dieser Repräsentationsform könnte man das Layout von Zustands- oder Arbeitsfluss-Diagrammen ansehen.

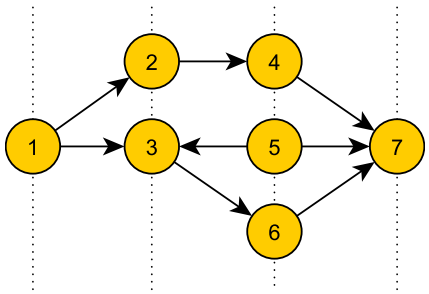


Abbildung 1.1: Ein Graph mit vertikaler Layereinteilung

Damit eine Visualisierung leicht von Betrachtern überblickt und verstanden werden kann, muss das Layout dieser Schaubilder dabei eine Reihe von ästhetischen Kriterien wie Kompaktheit oder die Verwendung möglichst kurzer Kantenzüge erfüllen, da solche bei Nichteinhaltung sehr häufig die Übersichtlichkeit und Lesbarkeit der Graphen stören. Hier spielt besonders die Layerzuweisung als Teilabschnitt des hierarchischen Layouts eine entscheidende Rolle, da in diesem die äußere Form der finalen Visualisierung und damit auch seine qualitativen Eigenschaften bereits weitestgehend festgelegt werden. Wurde eine Layerzuweisung der Knoten nicht im Vorfeld bestimmt, ist es notwendig, eine solche mittels Algorithmen zu ermitteln, wobei hier die weitestmögliche Erfüllung aller optischen Ansprüche als die zentrale Aufgabe dieser Layerzuweiser anzusehen ist. Bis dato wurden hier ganz unterschiedliche Ansätze entwickelt, welche diesem Anliegen nachkommen, jedoch gestaltet sich die Bestimmung eines Layouts, das allen ästhetischen Anforderungen genügt, aufgrund

der Komplexität dieses Problems als schwierig. Viele der geforderten Eigenschaften stehen zum Beispiel in einem gegenseitigen Abhängigkeitsverhältnis, oder manche schließen sich sogar weitestgehend aus. So kann es beispielsweise vorkommen, dass aufgrund einer kompakten Darstellung häufige und unschöne Kantenkreuzungen entstehen oder eine geringe Höhe der Visualisierung zu besonders langen Kantenzügen führt. Gerade wegen dieser Problematik ist die Forschung nach neuen Ansätzen zur Lösungsfindung noch lange nicht abgeschlossen.
Diese Arbeit soll hierbei einen Einblick in diese Thematik geben, in welcher wir eine Auswahl konkreter, bisher entwickelter Verfahren zur Layerzuweisung vorstellen und diese auf die ästhetische Qualität ihres Ausgabelayouts hin untersuchen. Darunter befinden sich mit dem Netzwerk-Simplex-Algorithmus von Gansner *et al.* [6] und dem Verfahren mittels LP-Löser auch zwei Ansätze, welche im Zuge dieser Arbeit implementiert wurden. Da ein Diagramm, welches mit den Algorithmen des hierarchischen Ansatzes ästhetisch ansprechend dargestellt werden soll, oftmals auch Knoten von unterschiedlicher Größe besitzt, werden wir hier zudem ein von uns entwickeltes Verfahren vorstellen, welches diese differierenden Maße seiner Komponenten bei der Layerzuweisung effektiv mit berücksichtigt.

1.1 Struktur dieses Dokuments

In Abschnitt 1.2 soll zunächst ein kurzer Überblick über themenverwandte Arbeiten zur Layerzuweisung im hierarchischen Layout gegeben werden. Da für das Verständnis dieser Arbeit auch einige Definitionen erforderlich sind, werden wir diese in Abschnitt 1.3 vorstellen, gefolgt von einem Überblick über den gesamten hierarchischen Ansatz in Kapitel 2, von welchem die Layerzuweisung ein Teilabschnitt darstellt. In Kapitel 3 leiten wir dann das eigentliche Thema dieser Arbeit ein. Wir werden dabei in Abschnitt 3.1 zunächst der Frage nachgehen, welche optischen Anforderungen typischerweise an ein Layout gestellt werden und wie man diese in der Layerzuordnung optimieren kann. Mit der Betrachtung des Longest-Path- und des Coffman-Graham-Layerzuweisers in Abschnitt 3.2, bzw. Abschnitt 3.3 werden dabei Möglichkeiten vorgestellt, wie man das Layout in seiner Höhe und Breite begrenzen kann. Anhand von gleich zwei weiteren Algorithmen, dem Netzwerk-Simplex-Ansatz und dem Verfahren mittels LP-Löser, sollen anschließend in Abschnitt 3.4 Ansätze zur Minimierung der Kantenlänge gezeigt werden. Eine effiziente Implementierung dieser beiden Verfahren wird dabei vorgestellt. Anhand einer Ergebnis- und Laufzeitanalyse in Abschnitt 3.5 werden diese beiden Verfahren sowohl untereinander als auch mit anderen Layerzuweisern verglichen. Kapitel 4 geht anschließend der Frage nach, inwiefern Knoten mit unterschiedlicher Breite bei der Layerzuweisung sinnvoll berücksichtigt werden können und welche Probleme sich daraus ergeben. Die Schwächen des bisherigen Ansatzes werden dabei zunächst in Abschnitt 4.1 aufgezeigt und anschließend ein besseres, von uns entwickeltes Verfahren in Abschnitt 4.2 vorgestellt. Eine Zusammenfassung in Kapitel 5 greift abschließend nochmals die wesentlichen Inhalte und Erkenntnisse dieser Arbeit auf und weist auf mögliche zukünftige For-

schungsbereiche im Themenfeld der Layerzuweisung hin.

1.2 Themenverwandte Arbeiten

Das Themenfeld der Layerzuweisung ist ein relativ überschaubares Gebiet im hierarchischen Ansatz und die Menge an Literatur, die über die Behandlung konkreter Layerzuweisungsverfahren hinausgeht, ist leider nur sehr gering. Aus diesem Grund können wir dem Leser leider nur wenige themenverwandte Materialien mit an die Hand geben.

- In E.G. Coffman's und R.L. Graham's Werk *Optimal Scheduling for Two-Processor Systems* [2] zeigen sich Parallelen von der Layerzuweisung zur Aufgabenverteilungen in Mehrprozessorsystemen. Die dort präsentierte Lösung impliziert dabei auch direkt eine Layerzuteilungsstrategie und mündete in dem Coffman-Graham-Algorithmus, welchen wir in Abschnitt 3.3 vorstellen.
- Georg Sander stellt in seinem Werk *Layout of Compound Directed Graphs* [11] ein Verfahren vor, mittels welchem auch mehrschichtige Graphen mit dem hierarchische Layout visualisiert werden können. Ein wesentlicher Teil dieser Arbeit beschreibt dabei auch die erforderlichen Erweiterungen, die zu diesem Zweck in Layerzuweisungsphase vorgenommen werden müssen.
- In *Layered Drawings of Directed Graphs in Three Dimensions* [8] erweitern Seok-Hee Hong und Nikola S. Nikolov den bisher für zwei-dimensionale Grafiken ausgerichteten hierarchischen Ansatz um die dritte Dimension und zeigen, wie man mit diesem Graphen auch in 3D darstellen kann.

Über diese genannten Werke hinausgehend existieren noch eine Reihe von Publikationen, die sich mit spezifischen Layerzuweisungsstrategien auseinandersetzen, bzw. neue Verfahren auf diesem Gebiet vorstellen. Auf diese möchten wir jedoch hier nicht weiter eingehen, da deren Vielzahl den Rahmen einer Auflistung sprengen würde und wir diese an passender Stelle in dieser Arbeit noch weitergehend erwähnen werden [14, 4, 9, 10, 2, 6, 7].

1.3 Grundlegende Definitionen

In diesem Kapitel werden die grundlegenden Definitionen und Notationen gegeben, welche für das weitere Verständnis dieser Arbeit entscheidend sind. Auf die Nennung der elementaren Definitionen für Graphen wird hierbei verzichtet, da diese dem Leser in der Regel schon bekannt sein sollten. Viele wurden dabei aus dem Werk von Healy und Nikolov [7] übernommen, mit einigen Anpassungen der Notation und Ergänzungen aus anderen Quellen [4, 9].

Sei im Folgenden $G = (V, E)$ ein gerichteter Graph und $v \in V$. $\delta^+(v) := \{(v, u)|(v, u) \in E\}$ definiert die Menge aller eingehenden Kanten von v und $\delta^-(v) := \{(u, v)|(u, v) \in$

$E\}$ die Menge aller ausgehenden Kanten von v. $\delta(v) := \delta^+(v) \cup \delta^-(v)$ wird als die Menge aller Kanten inzident zu $v \in V$ notiert. $|\delta^+(v)|$ heißt *eingehender Grad* (engl.: *indegree*) von v und $|\delta^-(v)|$ heißt *ausgehender Grad* (engl.: *outdegree*) von v. Ein Knoten ohne ausgehende Kanten wird als *Senke* des Graphen und ein Knoten ohne eingehende Kanten wird als *Quelle* des Graphen bezeichnet.
Eine *Layerzuweisung* eines azyklischen, gerichteten Graphen $G = (V, E)$ ist eine Partition von V in Teilmengen $L_1, L_2, \ldots, L_h$, sodass für alle $(u, v) \in E$ mit $u \in L_i$ und $v \in L_j$ $i > j$ gilt. Ein azyklischer Graph mit einer Layerzuweisung wird *geschichteter Graph* (engl.: *layered graph*) genannt. Die *Breite* eines geschichteten Graphen bezeichnet die Anzahl seiner Layer. Die *Höhe* des Graphen bezeichnet die Anzahl der Knoten in dem größten Layer, d.h. $\max_{1 \leq i \leq h} |L_i|$. Der *Span* einer Kante $(u, v) \in E$ mit $u \in L_i$ und $v \in L_j$ ist definiert als $i - j$. Die Layerzuweisung wird als *streng* (engl.: *proper*) bezeichnet, falls keine Kante einen Span größer eins hat. Der *Mindestlayer* (engl.: *floor*) eines Knotens $v \in V$ ist der niedrigste Index all derjenigen Layer, in denen v platziert werden kann. Der *Höchstlayer* (engl.: *roof*) eines Knotens $v \in V$ ist hingegen der höchste Index aller Layer, in denen v platziert werden kann.

1.4 Implementierungsgrundlage

Sämtliche Implementierungen im Zusammenhang mit dieser Bachelorarbeit wurden auf Basis des KIELER[1]-Projekts durchgeführt. Dies ist ein von der Arbeitsgruppe Echtzeitsysteme und Eingebettete Systeme der Christian-Albrechts-Universität zu Kiel entwickeltes Softwarewerkzeug zum modellbasierten Design komplexer Systeme, dessen besondere Eigenschaft in dessen Layout-Möglichkeiten liegt, welche es dem Benutzer ermöglichen, Modelle mittels verschiedener Layout-Ansätze, darunter auch dem hierarchischen, übersichtlich darzustellen. Algorithmen für alle übrigen Phasen des hierarchischen Ansatzes (siehe Kapitel 2) sind in diesem bereits aus vorangegangenen studentischen Arbeiten vorhanden und mussten nicht zusätzlich implementiert werden. Der Quellcode dieses Projekts, unter welchem sich auch die für die Implementierung zugrunde liegende Datenstruktur befindet, ist dabei frei zugänglich und kann unter der unten angegebenen Adresse eingesehen werden.

[1] Kiel Integrated Environment for Layout Eclipse Rich Client,
`http://www.informatik.uni-kiel.de/rtsys/kieler`

2 Das hierarchische Layout

Das hierarchische Layout wurde erstmals in den Arbeiten von Sugiyama, Tagawa und Toda [13] aus dem Jahre 1981 beschrieben und stellt einen relativ simplen Algorithmus für das Layout von gerichteten Graphen dar. Auch heute noch wird dieser Ansatz aufgrund seiner optisch ansprechenden Ergebnisse sehr häufig verwendet und ist noch immer Gegenstand wissenschaftlicher Forschung und Erweiterung (siehe Abschnitt 1.2). Für einen gegebenen Graphen führt der Algorithmus in angegebener Reihenfolge die folgenden Schritte aus:

1. **Zyklenentfernung:** Zyklen werden durch Richtungsvertauschung möglichst weniger Kanten entfernt. In der finalen Zeichnung erhalten diese wieder ihre ursprüngliche Orientierung (beschrieben in Abschnitt 2.1).

2. **Layerzuordnung:** Für den Graphen wird eine strenge Layerzuweisung bestimmt. Dies ist möglich, da der Graph durch die erste Phase azyklisch ist (beschrieben in Abschnitt 2.2, Kapitel 3 und Kapitel 4).

3. **Kreuzungsminimierung:** Die Reihenfolge der Knoten in einem Layer wird so verändert, dass die Anzahl der Kantenüberschneidungen möglichst minimiert wird (beschrieben in Abschnitt 2.3).

4. **Knotenpositionierung** Aus den bisher nur relativen Knotenpositionen (Zuordnung der Knoten zu den Layern, Reihenfolge in den Layern) werden konkrete Y-Koordinaten erzeugt, wobei Knotenüberlappungen vermieden werden müssen (beschrieben in Abschnitt 2.4).

5. **Kantenführung:** Die Kanten werden mittels Knickpunkte (engl. *bendpoints*) im Graph positioniert, wobei es in Abhängigkeit von Phase 3 zu einer möglichst geringen Anzahl Kantenüberkreuzungen kommen soll und keine Kante einen Knoten schneiden darf (beschrieben in Abschnitt 2.5).

In den nun folgenden Sektionen wollen wir jede Phase etwas genauer betrachten. Hierzu wurden bereits eine Vielzahl unterschiedlicher Algorithmen entwickelt [4, 9, 6, 11, 12], jedoch würde eine Behandlung all dieser Verfahren den Rahmen dieser Arbeit deutlich sprengen, sodass wir hier nur eine enge Auswahl besprechen möchten und für eine tiefergehende Studie auf die entsprechende Literatur verwiesen werden muss.

2.1 Zykelentfernung

Die Aufgabe dieser Phase ist die Eliminierung sämtlicher Zyklen aus dem Graphen durch temporäres Vertauschen der Orientierung möglichst weniger Kanten. Das Finden einer minimalen Menge umzukehrender Kanten ist hierbei nicht nur vom algorithmischen Standpunkt aus erstrebenswert, sondern hat auch einen visuellen Aspekt, da jede in dieser Phase umorientierte Kante in der finalen Darstellung des Graphen gegen die vorherrschende Flussrichtung zeigt. Die Bestimmung einer optimalen Lösung, d.h. das Finden eines maximal azyklischen Subgraphen, ist jedoch NP-schwer [9]. Folglich muss man sich einer Reihe von Heuristiken bedienen, um dennoch ein akzeptables Ergebnis zu erhalten. Ein relativ intuitiver Ansatz wäre hier eine Tiefensuche auf dem Graphen auszuführen und alle identifizierten Rückkehrkanten in ihrer Orientierung umzukehren. Dieser Algorithmus hätte zwar eine lineare Laufzeit, jedoch würden im schlechtesten Fall, abhängig von der Wahl des Startknotens, $|E| - |V| - 1$ Kanten umgekehrt [4]. Ein Beispiel, bei dem dieser Aspekt nicht unbedeutend ist, ist in Abbildung 2.1 gegeben. Würde man dort bei Knoten 1 beginnen, würden vier Kanten umgedreht, bei Knoten 2 wären es jedoch nur zwei.

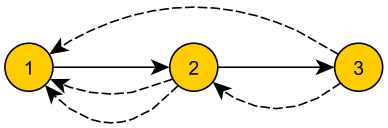


Abbildung 2.1: Beispiel einer vom Startknoten abhängigen Zyklenentfernung

Ein wesendlich besserer Ansatz stellt hier der *Greedy-Cycle-Removal*-Algorithmus (zu Deutsch: "gierige Zykelentferung") dar, welcher ebenso eine lineare Ausführungszeit hat, jedoch höchstens $\frac{1}{2}|E|$ Kanten umkehrt und nicht von der Wahl eines Startknotens oder oder einer initialen Orientierung abhängig ist [9].

Algorithmus 2.1: Greedy-Cycle-Removal

```
1  Prozedur Greedy−Cycle−Removal(G: gerichteter Graph)
2      S_l = ∅; S_r = ∅
3      G ≠ {∅, ∅} tue
4          Während G eine Senke v enthält tue
5              füge δ⁻(v) zu S_r hinzu und entferne v und δ⁻(v) aus G
6          entferne alle isolierten Knoten aus G;
7          Während G eine Quelle v enthält tue
8              füge δ⁺(v) zu S_l hinzu und entferne v und δ⁺(v) aus G;
9          Falls G ≠ {∅, ∅} dann
10             sei v ein Knoten mit maximalem Wert |δ⁺(v)| − |δ⁻(v)|;
11             füge δ⁺(v) zu S_l hinzu und entferne v und δ(v) aus G;
12         Verbinde S_l und S_r zur Knotensequenz S;
13 Ende
```

Nachdem hier im Graph nach Abschluss der zweiten Während-Schleife nur noch Knoten und Kanten enthalten sind, die einen Zykel bilden, werden in der letzten Schleife durch die Wahl desjenigen Knotens, der die geringste Differenz aus rechtsgerichteten zu linksgerichteten Kanten besitzt, eine möglichst geringe Anzahl Kanten in ihrer Orientierung vertauscht.

2.2 Layerzuweisung

Diese Phase wollen wir hier nur kurz betrachten, da wir in Kapitel 3 und Kapitel 4 noch genauer auf die besonderen Aspekte und Ansätze eingehen werden. Zweck dieser Phase ist die Zuteilung der Knoten des durch Phase 1 azyklischen Graphen zu einzelnen Layern. Da viele Algorithmen zur Kantenkreuzungsminimierung (Phase 3) eine strenge Layerzuweisung voraussetzen, wird für jede Kante, dessen Start- und Endknoten sich nicht in aufeinander folgenden Layern befinden, ein virtueller Knoten für jeden übersprungengen Layer eingefügt (*Normalisierung*). Abbildung 2.2 zeigt hier ein Beispiel.
Eiglsperger *et al.* nennen hierzu jedoch auch Ansätze, wie man mittels einer angepassten Kantenkreuzungsminimierung die Gesamtzahl der einzufühgenden Knoten reduzieren kann [5]. Deren Grundidee ist hierbei relativ simpel. Da alle virtuellen Knoten, die eine lange Kante untergliedern, die selbe Y-Koordinate besitzen, lassen sich diese zu gemeinsamen Segmenten zusammenfassen. Der Vorteil dieses Verfahrens liegt nun darin, dass die nachfolgenden Phasen alle virtuellen Knoten eines Segments als ein einzelnes Objekt betrachten können, d.h. alle Knoten gemeinsam zu verschieben oder platzieren vermögen, wodurch sich die Gesamtlaufzeit des hierarchischen Ansatzes verringert.

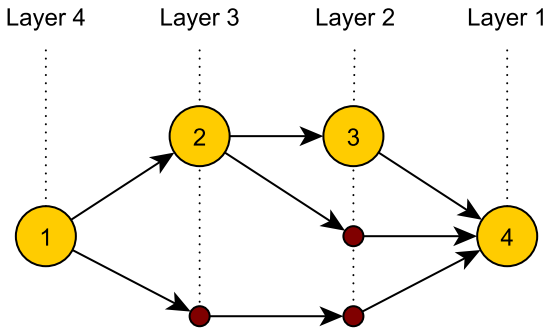


Abbildung 2.2: Strenge Layerzuweisung: Die Kanten $(1, 4)$ und $(2, 4)$ werden mittels Einfügen von virtuellen Knoten unterteilt.

2.3 Kreuzungsminimierung

Nachdem die Knoten in Phase 2 zunächst ungeordnet zu den Layern hinzugefügt wurden, soll hier nun für diese eine Ordnung definiert werden, bei der eine möglichst kreuzungsfreie Darstellung des Graphen ermöglicht wird. Dies ist aus ästhetischen Aspekten erstrebenswert, da Graphen mit Kantenkreuzungen häufig als schwerer zu lesen empfunden werden. Jedoch ist das Finden einer optimalen Lösung selbst für Graphen mit nur zwei Layern NP-vollständig [4], sodass auch hier eine Reihe von Heuristiken zur Anwendung kommen müssen, welche das Problem wenn auch suboptimal zu lösen vermögen. Viele dieser Ansätze folgen dabei einem ähnlichen Schema: Man wähle für einen beliebigen Layer, in unserem Beispiel L_1, eine beliebige Reihenfolge seiner Knoten. Für alle $i = 2, 3, 4, \ldots, n$ wird nun die Knotenreihenfolge in L_{i-1} fix gehalten, während die Knoten in L_i zwischen L_{i-1} und L_i neu geordnet werden. Wenn Layer L_n erreicht wurde, kehren wir auf umgekehrtem Wege wieder zu L_1 zurück, d.h. wir nehmen alle Knoten in L_i als fest an und variieren die Reihenfolge in L_{i-1}. Diesen Vorgang wiederholen wir so lange, bis sich die Anzahl kreuzender Kanten nicht weiter verringert.
Für die Berechnung der konkreten Knotenreihenfolgen werden hier am häufigsten Verfahren verwendet, die Schwerpunkt- oder Median-Berechnungen in den einzelnen Layer durchführen. Ausgehend von zuvor berechneten Werten der Knoten des Layer mit fester Ordnung kann hierbei direkt eine Reihenfolge für alle Knoten des Layers mit variabler Ordnung bestimmt werden. Der große Vorteil dieser Verfahren ist ihre Effizienz. Während viele andere, einfachere Heuristiken eine quadratische Laufzeit haben, lassen sich diese bereits in linearer Zeit implementieren. Einer dieser, von der Performanz her schlechteren Verfahren ist z.B. der *Adjacent-Exchange*-Algorithmus (zu Deutsch: "adjazenter Austausch"), der auf das schrittweise Vertauschen einzelner Knoten basiert, sofern hierdurch eine Verringerung der Kantenkreuzungen erreicht wird.

Algorithmus 2.2: Adjacent-Exchange

```
1 Prozedur Adjacent−Exchange(G: gerichteter Graph mit Layern L1 und L2 und fester
2                                  Knotenreihenfolge in L1)
3     Wähle eine initiale Knotenreihenfolge in L2
4     Wiederhole
5         Für alle benachbarten Knoten (u, v)
6             Falls c_{u,v} > c_{v,u} tue
7                 vertausche u und v
8     bis die Anzahl an Kantenkreuzungen nicht weiter abnimmt
9 Ende
```

$c_{u,v}$ bezeichnet hierbei die Anzahl an Kreuzungen, die Kanten inzident zu u mit Kanten inzident zu v verursachen, wenn u eine geringere Y-Koordinate als v besitzt.

2.4 Knotenpositionierung

In dieser Phase werden für jeden Knoten vertikale Koordinaten bestimmt, wobei besonders ästhetische Aspekte wie eine gradlinige Kantenführung befolgt werden müssen. Hierzu existiert eine optimale Lösung, die sich wie folgt beschreiben lässt [9].
Sei hierzu $p = v_1, v_2, \ldots, v_k$ ein Pfad in G, wobei $v_2, v_3, \ldots, v_{k-1}$ virtuelle Knoten seien. Einen solchen Pfad bezeichnen wir im Folgenden als *Kantenpfad*. Würde dieser Pfad gradlinig gezeichnet werden, so gilt

$$y(v_k) - y(v_1) = \frac{i-1}{k-1}(y(v_k) - y(v_1)) \tag{2.1}$$

für alle $1 < i < k$, wobei $y(v_i)$ die Y-Koordinate von v_i bezeichnet. Weiter lässt sich nun die Abweichung des Pfades von einer geraden Linie bestimmen. Diese ist gerade

$$diff(p) = \sum_{i=2}^{k-1}(y(v_i) - \overline{y}(v_i))^2 \tag{2.2}$$

mit $\overline{y}(v_i)$ als die gewählte Y-Koordinate von v_i. Als lineares Optimierungsproblem erhalten wir somit

$$\sum_{p \;\; ist \;\; Kantenpfad} diff(p) \tag{2.3}$$

unter der Nebenbedingung $y(w) - y(v) > \rho(w, v)$, welche die Minimaldistanz zwischen zwei aufeinander folgenden Knoten definiert.
Jedoch birgt dieser Ansatz den Nachteil, dass seine Zielfunktion quadratisch ist und daher eine optimale Knotenpositionierung nur für kleine Graphen bestimmt werden kann. Eine Alternative bieten hier Heuristiken, welche dabei häufig dem gleichen, unten skizzierten Schema folgen.

Algorithmus 2.3: Knotenpositionierung

```
1 Prozedur Knotenpositionierung(G: gerichteter Graph)
2     Wähle initiale Koordinaten für alle Knoten
3     Während bestimmte Abbruchbedingung nicht erfüllt ist tue
4         positioniere Knoten;
5         begradige Layout;
6         kompaktiere Layout;
7 Ende
```

Ausgehend von einer initialen Knotenpositionierung wird diese solange verbessert, bis ein gewisser Grad an Zielerfüllung (angedeutet durch das Abbruchkriterium) erreicht ist.

2.5 Kantenführung

In dieser letzten Phase werden die Kanten mittels Knickpunkte im Graph platziert. In der Literatur werden hier verschiedene Ansätze genannt, von denen jeder einen großen Einfluss auf das Erscheinungsbild des Graphen hat und dessen Charakter individuell prägt [4, 9, 12]. Eine relativ klassische Methode stellt hierbei die *Polyline*-Kantenführung dar, bei der alle Kanten durch gerade Linien gezeichnet werden und an den Stellen, an denen virtuelle Knoten lange Kanten spalten, Knickpunkte eingefügt werden (siehe Abbildung 2.2). Aber auch andere, wesentlich komplexere Ansätze wie die Kantenführung mittels Bézier-Kurven werden aufgrund ihrer optisch vorteilhaften Eigenschaften häufig verwendet. Bei dieser Form der Visualisierung werden Kanten an den Stellen, an denen sich eigentlich Knickpunkte befinden, gebogen, wobei sämliche Ecken aus dem Graphen entfernt werden und seinem Erscheinungsbild eine besonders weiche und geschwungene Struktur verliehen wird. Abbildung 2.3 zeigt hier ein Beispiel.

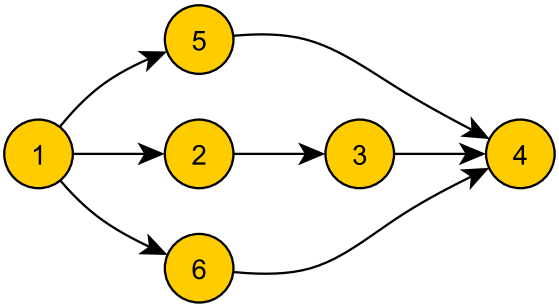


Abbildung 2.3: Beispiel eines mittels Bézier-Kurven visualisierten Graphen

3 Optimale Layerzuweisung

Mit der Layerzuweisung, in der zweiten von fünf Phasen, wird bereits relativ früh im Sugiyama-Ansatz das optische Erscheinungsbild des Graphen weitestgehend festgelegt. Entsprechend früh müssen hier auch ästhetische Aspekte, die das spätere Layout des Graphen befolgen sollte, mit berücksichtigt werden. Im Folgenden wollen wir hier verschiedene Ansätze präsentieren, welche sich dieses Problems angenommen haben. Zwei dieser Verfahren wurden dabei auch im Zuge dieser Arbeit implementiert. Dies ist zum einen der Netzwerk-Simplex-Algorithmus, welchen wir in Unterabschnitt 3.4.2 beschreiben, und zum anderen ein Ansatz, welcher sich eines LP-Lösers bedient, der in Unterabschnitt 3.4.1 näher erläutert wird.

3.1 Ästhetische Anforderungen an die Layerzuweisung

Das Ziel eines Layouts ist es, einen Graphen so darzustellen, dass seine Topologie in möglichst kurzer Zeit verstanden werden kann und es dem Leser ermöglicht wird, die Bedeutung des Dargestellten schnell zu erfassen. Als häufige Anforderungen an das spätere Design des Graphen werden insbesondere die folgenden Aspekte genannt [4, 6, 9]:

1. Der geschichtete Graph sollte möglichst kompakt sein, d.h. seine Höhe und Breite sollte gering bleiben. Ist das Layout in seiner Höhe oder Breite zu groß, passt es nicht mehr auf eine Bildschirmseite. Die Navigation in diesem dabei erschwert, da man entweder ständig den momentan auf den Monitor sichtbaren Ausschnitt verschieben oder das Diagramm auf die Bildschirmgröße verkleinern muss. Ersteres sorgt hier dafür, dass die Übersicht über den Gesamtzusammenhang verloren geht und im letzteren Fall kann es passieren, dass die Darstellung so klein wird, dass einzelne Beschriftungen nicht mehr problemlos gelesen werden können.

2. Die Anzahl an virtuellen Knoten sollte möglichst gering bleiben. Hierfür werden gleich mehrere Gründe genannt:

   - Bei den klassischen Verfahren wie der Polyline-Kantenführung, treten Knicke ausschließlich an den Stellen auf, an denen sich virtuelle Knoten befinden. Eine hohe Anzahl ist jedoch nicht besonders hilfreich für das Verständnis, denn die Lesbarkeit steigt, wenn die Anzahl der Knicke möglichst gering wird. Obwohl die Anzahl der Knicke auch noch während

der Knotenpositionierung mittels Verschiebung der virtuellen Knoten auf eine Linie mit seinen adjazenten Knoten verringert werden kann, ist es dennoch effektiver, die Anzahl der virtuellen Knoten von vornherein zu reduzieren, da andernfalls stets noch zwei Knickpunkte in jeder Kante, die mindestens durch einen virtuellen Knoten unterteilt wurde, verbleiben.

- Die Anzahl der virtuellen Knoten ist ein Maß für die Länge einer Kante. Kurze Kanten machen es dabei dem Betrachter einfacher, adjazente Knoten im Graphen zu finden. Dies ist damit begründet, dass es für das Auge einfacher ist, eher kurzen als langen Kanten zu folgen [4].

- Da die Laufzeit der nachfolgenden Phasen von der Gesamtanzahl der Knoten im Graphen, d.h. sowohl realen als auch virtuellen Knoten, abhängt, ist es erstrebenswert die Knotenmenge zu begrenzen, um den Gesamtalgorithmus möglichst performant zu halten. Dies stellt dabei keinen optischen, sondern eher einen praktischen Effekt dar.

Die gleichzeitige Optimierung des Graphlayouts bezüglich all dieser Eigenschaften ist jedoch nicht möglich. Beispielsweise ist das Finden einer Darstellung, die gleichzeitig sowohl eine minimale Höhe als auch Breite besitzt, NP-vollständig [4]. Aber auch anschaulich lässt sich dieses Problem verdeutlichen. So könnte z.B. eine geringe, bzw. minimale, Höhe der Darstellung dazu führen, dass viele Knoten in dem selben Layer platziert werden müssen. Dies jedoch bewirkt wiederum eine höhere Breite einiger Layer, wodurch das Gesamtlayout verbreitert wird. Anstatt also das Layout des Graphen in allen optischen Kriterien gleichzeitig zu optimieren, finden sich in der Literatur eher Algorithmen, die die Layerzuweisung bezüglich eines einzelnen ästhetischen Aspekts isoliert betrachtet verbessern [14, 4, 9, 10, 2, 6, 7].

3.2 Minimierung der Breite

Ein sehr einfaches und bekanntes Verfahren zur Minimierung der Breite ist der *Longest-Path-Layerer*, zu Deutsch "Längster-Pfad-Layerzuweiser". Dieser platziert alle Senken des Graphen in den letzten, d.h. äußersten rechten Layer L_1. Alle übrigen Knoten werden Layern zugewiesen, dessen Index äquivalent zu der Anzahl der Knoten in einem längsten Pfad zu einer Senke plus eins ist.

Algorithmus 3.1: Longest-Path-Layerzuweiser

```
1  Prozedur Longest−Path−Layerzuweiser(G: gerichteter Graph)
2      Für alle v in G tue
3          besuche(v)
4      k := max{Höhe (v) : v ∈ V}
5      platziere alle v in Layer L_{k−h(v)+1}
6  Ende

8  Prozedur besuche(v: Knoten)
9      Falls Höhe(v) ≠⊥ dann
10         h_m := 1
11         Für alle Kanten e = (v, u), u ≠ v tue
12             besuche(u)
13             h_m := max{h_m, h(u)}
14         Höhe(v) := h_m
15 Ende
```

Dieser Algorithmus bestimmt zwar eine Layerzuweisung von minimaler Breite, jedoch birgt dieser auch Nachteile. So kommt es zum Beispiel oft vor, dass das Layout aufgrund der hohen Knotenmenge, die besonders L_1, also der letzte Layer, aufnehmen muss, relativ hoch wird. Ebenso produziert dieses Verfahren häufig eine große Menge an virtuellen Knoten. Dies kann andeutungsweise in Abbildung 3.1 erkannt werden.

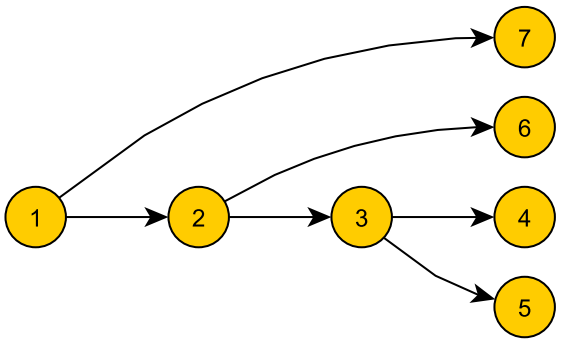


Abbildung 3.1: Ein mittels Longest-Path-Algorithmus geschichteter Graph

3.3 Minimierung der Höhe

Bei der Minimierung der Höhe der Layerzuweisung untergliedert man die vorhandenen Ansätze in zwei Kategorien, nämlich in solche, die virtuelle Knoten bei der Bestimmung der Knotenmenge eines Layers mit berücksichtigen, und solche, die diese Knoten vollständig außenvorlassen. Im zweiten Fall ist die Begrifflichkeit dabei eigentlich etwas irreführend, da in der Definition der Layerzuweisung keinerlei Restriktionen bezüglich der Anzahl der Knoten in einem Layer gegeben sind und es sich

auch um kein eigenständiges Optimierungsproblem handelt, da es für jeden Graphen eine Layerzuweisung gibt, die eine minimale Höhe, nämlich eine solche von 1, besitzt. Hierzu müsste man lediglich jedem Knoten einen separaten Layer zugeweisen. Vielmehr ist der Ausdruck hier im Sinne einer Beschränkung der Höhe auf einen festen Wert zu verstehen. Wir möchten uns hier auf eine Betrachtung eines Ansatzes, welcher die virtuellen Knoten vernachlässigt, beschränken, da die Vorstellung von Algorithmen zur Höhenminimierung nicht im Fokus dieser Arbeit liegt. Für Weiteres sei auf die Arbeit von Tarassov *et al.* verwiesen [14].

Für die Bestimmung einer Layerzuweisung mit begrenzter Höhe wird besonders häufig der *Coffman-Graham*-Algorithmus verwendet. Ausgehend von einem azyklischen Graphen und einem ganzzahligen Parameter $w > 0$ bestimmt dieser eine Layerzuordnung des Graphen mit einer Höhe von höchstens w, wobei virtuelle Knoten hier, wie erwähnt, nicht eingerechnet werden. Ein Beispiel eines auf diese Weise geschichteten Graphen ist in Abbildung 3.2 gegeben, welche dabei auch direkt den Unterschied zum Longest-Path-Layerzuweiser verdeutlicht.

Algorithmus 3.2: Coffman-Graham-Layerzuweiser

```
1  Prozedur Coffman−Graham−Layerzuweiser(G = (V, E): gerichteter Graph, w ∈ ℕ)
2      initial ist jeder Knoten ohne Label
3      Für i = 1 bis |V| tue
4          wähle ein v ∈ V ohne Label, sodass {π(v) : (u, v) ∈ E} minimal ist;
5          π(v) = i;
6      k = 1; L_1 = ∅ ;U = ∅;
7      Während U ≠ V tue
8          wähle ein u ∈ V \ U, sodass jeder Knoten in {π(v) : (u, v) ∈ E} ist in U
9                  und π(u) ist maximal;
10         Falls |L_k| < w und für jede Kante (u, v) gilt w ∈ L_1 ∪ L_2 ∪ ... L_{k−1}
11             dann füge u zu L_k hinzu;
12         sonst k = k + 1; l_k = {u};
13         füge u zu U hinzu;
14 Ende
```

Der Algorithmus lässt sich dabei in zwei Phasen untergliedern. In der ersten wird jedem Knoten $u \in V$ eine natürliche Zahl als eindeutiges Label $\pi(u)$ zugeordnet. Dabei erhält eine Quelle v des Graphen die 1, d.h. $\pi(v) = 1$. Nachdem alle Label $1, 2, \ldots k-1$ zu Knoten zugeteilt wurden, wird Label k einem Knoten v so vergeben, dass die folgenden Eigenschaften erfüllt sind:

1. v hat zuvor noch kein Label erhalten.

2. Zu allen Knoten u mit $(u, v) \in E$ wurden bereits Labels zugeteilt.

3. Unter allen Knoten, die die obigen beiden Kriterien erfüllen, wird jener ausgewählt, dessen Menge der Label seiner unmittelbaren Vorgänger, d.h. $\{\pi(u) : (u, v) \in E\}$, in der lexikographischen Ordnung "$<$" maximal ist. Bei dieser gilt für beliebige Mengen S und T von Labels $S < T$ genau dann, wenn

- $S = \emptyset$ und $T \neq \emptyset$ oder
- $T \neq \emptyset$, $S \neq \emptyset$ und $\max(S) < \max(T)$ oder
- $T \neq \emptyset$, $S \neq \emptyset$, $\max(S) = \max(T)$ und $S \setminus \max(S) < T \setminus \max(T)$

In der zweiten Phase werden die Layer, beginnend mit dem äußersten rechten (L_1), nach und nach mit Knoten bestückt. Zur Befüllung von L_k wird dabei ein Knoten v ausgewählt, der noch in keinem Layer platziert wurde und die Quellknoten all seiner eingehender Kanten bereits in $L_1, L_2, \ldots L_{k-1}$ positioniert wurden. Existieren mehrere solcher Knoten, wird derjenige gewählt, dessen Label den größten Wert besitzt. Gibt es keinen derartigen Knoten oder hat L_k bereits eine Höhe von w, gehen wir zu Layer L_{k+1} über und wiederholen die Prozedur.

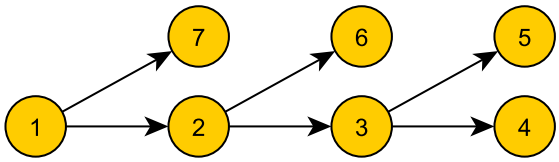


Abbildung 3.2: Der Graph aus Abbildung 3.1, diesmal mittels Coffman-Graham-Algorithmus und $w = 2$ geschichtet.

3.4 Minimierung der Kantenlänge

Da die Länge einer Kante im finalen Layout des Graphen wesentlich von dem Abstand derjenigen Layer abhängt, zu denen deren Quell- und Zielknoten zugeordnet sind, ist dieses Problem äquivalent zur Minimierung der eingesetzten virtuellen Knoten während der Layerzuweisung. Leider ist dies nicht der einzige Faktor, der die spätere Kantenlänge beeinflusst. Auch die konkrete Knotenpositionierung in Phase 4 und Kantenführung in Phase 5 haben hier einen wesentlichen Einfluss. Die Optimierung der Layerzuweisung stellt jedoch die einzige Möglichkeit dar, die Kantenlänge unabhängig von der gewählten Strategie bei der Kantenführung zu beeinflussen.
Nachfolgend wollen wir zwei unterschiedliche Verfahren zur Minimierung der Kantenlänge vorstellen, zum einen die Lösung mittels LP-Löser und zum anderen der Netzwerk-Simplex-Algorithmus, welche ebenso im Rahmen dieser Bachelorarbeit implementiert wurden. Wir beginnen dabei jeweils mit einer theoretischen Betrachtung des jeweiligen Algorithmus und gehen dann im Anschluss auf die konkreten Implementierungsdetails über. Auf eine Betrachtung eines dritten denkbaren Ansatzes, welcher von Nikolov *et al.* [10] entwickelt wurde und auf dem schrittweisen Verschieben einzelner Knoten basiert, wollen wir hier verzichten, da Versuche der genannten Autoren gezeigt haben, dass deren Verfahren das Problem der Kantenlängenminimierung lediglich suboptimal zu lösen vermag und von der Ergebnisqualität nicht an die Güte der von uns betrachteten Algorithmen heranreicht.

3.4.1 Lösung mittels LP-Löser

Ein relativ simpler Ansatz zur Kantenlängenminimierung besteht hier in der Verwendung eines LP-Lösers, d.h. eines Programms, welches algorithmisch die optimale Lösung einer linearen Optimierungsaufgabe bestimmt, sofern diese existiert. Die Layerzuweisung ergibt sich dabei unmittelbar aus dem Ergebnis des LP-Lösers und lässt sich somit ohne weitere Nachbearbeitung direkt umsetzen.

Kantenlängenminimierung als lineares Problem

Die Minimierung der Kantenlänge lässt sich als diskretes Optimierungsproblem

$$\min \sum_{(u,v)\in E} (\lambda(u) - \lambda(v)) \tag{3.1}$$

unter den Nebenbedingungen

$$\lambda(u) - \lambda(v) \geq 1 \quad \forall (u,v) \in E \tag{3.2}$$

$$\lambda(v) \in \mathbb{N} \qquad \forall v \in V \tag{3.3}$$

darstellen, wobei $\lambda(u)$ den Index des Layers bezeichnet, in dem Knoten u platziert wurde. Die Minimierung der Zielfunktion bewirkt somit, dass Knoten, die durch eine adjazente Kante miteinander verbunden sind, in der Layerzuordnung enger aufeinander folgen, wodurch die Anzahl der virtuellen Knoten und somit auch die Gesamtlänge der Kanten minimiert wird.
In der Literatur finden sich hier oft noch Erweiterungen dieses Ansatzes [9, 6]. Möchte man zum Beispiel bewirken, dass einige Kanten besonders kurz werden, kann man einzelnen Kanten eine Gewichtung oder Priorität $\omega > 1$ zuzuweisen. Deren Länge fließt nun um ein Vielfaches in den Zielfunktionswert ein, wodurch diese Kanten bevorzugt kurz gehalten werden. Als klassisches Gegenstück hierzu lässt sich auch ein Parameter einführen, der zur Folge hat, dass bestimmte Kanten mindestens eine vorgegebene Länge $\rho \in \mathbb{N}$ besitzen. Als erweitertes Optimierungsproblem erhält man somit

$$\min \sum_{(u,v)\in E} \omega(u,v)(\lambda(u) - \lambda(v)) \tag{3.4}$$

unter den Nebenbedingungen

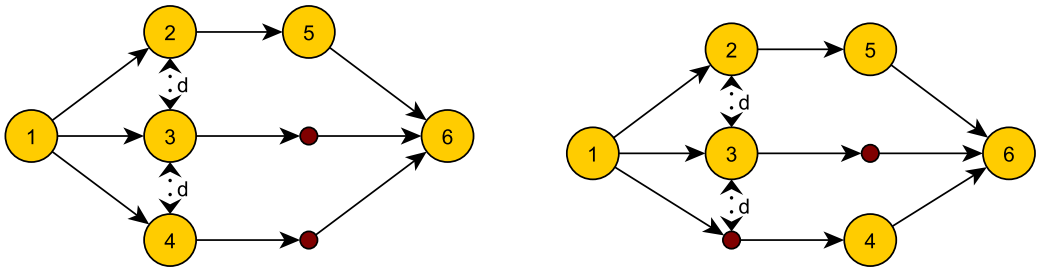
$$\lambda(u) - \lambda(v) \geq \rho(u,v) \quad \forall (u,v) \in E \tag{3.5}$$

$$\lambda(v) \in \mathbb{N} \qquad \forall v \in V \tag{3.6}$$

Da jedoch die Priorisierung einzelner Kanten oftmals deren Länge auf Kosten der Gesamtlänge aller Kanten minimiert, wollen wir uns hier auf die Behandlung des Problems ohne besondere Kantengewichtung oder Mindestlänge beschränken. Eine Lösung dieser Optimierungsaufgabe ist dabei auf jeden Fall ganzzahlig, da die Matrix der Nebenbedingungen vollständig unimodular ist [9].

Implementierung

Für die Implementierung wurde das Programm *lp_solve* in der Version 5.5 verwendet, welches ein kostenloses Software-Werkzeug zum Lösen von linearen Optimierungsaufgaben darstellt und als Programmierschnittstelle für viele verschiedene Sprachen, darunter C und Java, inklusive vollständigem Quellcode, frei verfügbar ist. Für die Lösungsbestimmung wird hierbei standardmäßig der Simplex-Algorithmus verwendet [1]. Leider werden über die Performanz des verwendeten Programms auf dessen Internetpräsenz[1] keine Angaben gemacht, jedoch ist bekannt, dass der Simplex-Algorithmus bereits an sich im schlechtesten Fall eine exponentielle Laufzeit besitzt [1].
Als Erweiterung dieses Ansatzes, und um auch das ästhetische Kriterium der Kompaktheit der Layerzuweisung zu befriedigen, wenden wir abschließend auf die vom LP-Löser erhaltene Lösung ein Verfahren an, welches von Gansner *et al.* in ihrer Arbeit beschrieben wird [6]. Da zu dem gegebenen Optimierungsproblem häufig mehrere Layerzuweisungen existieren, die einen identischen Zielfunktionswert besitzen, kann man einzelne Knoten, deren eingehender Grad ihrem ausgehenden Grad gleicht und zu mehreren Layern im Graph gültig zugeordnet werden können, zu demjenigen Layer zuordnen, der eine minimale Anzahl an bereits zugewiesenen Knoten besitzt, ohne dass die Optimalität der Lösung hierbei beeinträchtigt wird.



(a) Eine Visualisierung, dessen Höhe noch verringert werden kann

(b) Der selbe Graph nach Anwendung der Heuristik

Abbildung 3.3: Nachträgliche heuristische Minimierung der Höhe

Durch diese relativ einfache Heuristik soll versucht werden, die Breite des Layouts zu minimieren und nach Möglichkeit die Knotenmenge in allen Layern ungefähr ausgeglichen zu halten. Das Prinzip dieser Heuristik soll dabei anhand von Abbildung 3.3 näher erläutert werden, in welcher das linke Schaubild den Graphen vor der Anwendung und das rechte den selben nach der Terminierung des Verfahrens, d.h. nach der Verschiebung von Knoten 4, dargestellt. Der im Zuge der Knotenpositionierung bei der Visualisierung des Graphen streng einzuhaltende Mindestabstand wurde hierbei als d gekennzeichnet. Ebenso wurden sämtliche virtuelle Knoten, mittels denen später jede layerüberspannende Kanten in kürzere unterteilt wird, hier dargestellt.

[1] `http://lpsolve.sourceforge.net/`

Wie man erkennt, hat sich durch die nachträgliche Platzierung von Knoten 4 in den anderen gültigen Layer zwar die Gesamthöhe der Layerzuweisung unter Berücksichtigung aller Knoten nicht verringert und verbleibt weiterhin bei drei, jedoch konnte im rechten Schaubild der Graph dennoch unter Einhaltung des Mindestabstandes mit einer geringeren Höhe gezeichnet werden. Die Maße der virtuellen Knoten sind hierfür verantwortlich, denn diese besitzen im Gegensatz zu den regulären Knoten eine Höhe und Breite von null. Vergleicht man nun die tätsächliche Höhe der beiden dargestellten Diagramme, so erhält man für das linke eine Höhe von $3k + 2d$ und für das rechte eine Höhe von $2,5k + 2d$, wobei k die Höhe eines regulären Knotens bezeichnet. Zwar mag dieser Unterschied auf den ersten Blick nicht sehr ins Gewicht fallen und erbrachte in unserem Beispiel lediglich eine Höhenersparnis von rund 10%. Versuche der genannten Autoren mit größeren Graphen, in denen der Größenunterschied zwischen den einzelnen Layern in der Regel ausgeprägter ist, haben jedoch gezeigt, dass diese Heuristik relativ gute Ergebnisse erzielt.

3.4.2 Lösung mittels Netzwerk-Simplex-Algorithmus

Das Netzwerk-Simplex-Verfahren ist eng mit dem eben bereits angesprochenen Simplex-Verfahren zur algorithmischen Lösungsbestimmung von linearen Optimierungsproblemen verwandt und stellt eine Spezialisierung dieses Ansatzes dar. In seiner ursprünglichen Form lassen sich mittels diesem Probleme wie die des minimalen Kostenflusses lösen [1]. Gansner *et al.* haben diesen Algorithmus dabei so modifiziert, dass man mit diesem auch das Layerzuweisungsproblem bezüglich minimaler Kantenlänge relativ handlich lösen kann. Für diesen Algorithmus wurde eine polynomielle Laufzeit noch nicht nachgewiesen, jedoch benötigt dieser für die Bestimmung einer optimalen Lösung in der Regel nur wenige Iterationen [6].
Eine nähere Beschreibung des allgemeinen Netzwerk-Simplex-Algorithmus wollen wir hier nicht leisten, da dies den Fokus der Arbeit verlassen würde. Der interessierte Leser sollte hierfür die entsprechende Literatur zu Rate ziehen [1, 3].

Der Netzwerk-Simplex-Algorithmus

Bevor wir mit der näheren Beschreibung des Netzwerk-Simplex-Ansatzes beginnen können, sollen zunächst einige algorithmenrelevanten Begriffe und Beobachtungen näher erläutert werden.

Wie bereits bekannt, muss eine Layerzuweisung die Eigenschaft besitzen, dass für jede Kante $(u, v) \in E$ mit $u \in L_i$ und $v \in L_j$ stets $i > j$ gilt. Bei einer gegebenen Layerzuweisung bezeichnet die *minimale* Länge einer Kante die kleinstmögliche Differenz der Indizes derjenigen Layer, in denen ihr Quell- und Zielknoten platziert werden kann. Die Differenz aus der minimalen und gegenwärtigen Länge einer Kante wird ihr *Schlupf* genannt. Eine Kante heißt *eng*, wenn ihr Schlupf null ist.

Ein spannender Baum über einen Graphen impliziert dabei bereits eine gültige Layer-

zuweisung, respektive eine Menge von äquivalenten Zuweisungen, denn eine solche lässt sich relativ einfach aus einem gegebenen Spannbaum erzeugen. Bezeichne hierzu mit $min(u)$ die minimale Länge einer Kante $u \in E$. So können von einem beliebigen Knoten v und dessen vorgegebenem Layer L_i alle bislang noch nicht zu einem Layer zugeteilten Knoten u, die über eine eingehenden Kante mit v verbunden sind, dem Layer $L_{i-min(u)}$ zugeordnet werden und sämtliche übrigen zu v adjazenten und noch nicht zugeteilten Knoten w in dem Layer $L_{i+min(w)}$ platziert werden. Dieses Verfahren wird rekursiv auf alle $v \in V$ ausgeführt, bis auf diese Weise alle Knoten auf die Layer verteilt wurden.
Würde man nur die Kanten des spannenden Baumes betrachten, wären diese eng und die Layerzuweisung somit optimal bezüglich einer minimalen Kantenlänge. Um nun auch dieses Verfahren auf die nicht-Baumkanten auszuweiten, lässt sich jeder Baumkante ein ganzzahliges Attribut *Schnittwert* zuweisen, welches wie folgt definiert ist: Würde man diese Kante aus dem Spannbaum entfernen, würde der Baum in zwei Zusammenhangskomponenten zerfallen, dem Kopf-Teil, welcher den Zielknoten der entnommenen Kante enthält, und dem Schwanz-Teil, welcher den Quellknoten enthält. Der Schnittwert einer Kante bezeichnet nun die Anzahl aller Kanten, die vom Schwanz- zum Kopf-Teil verlaufen, einschließlich der Baumkante selbst, abzüglich der Anzahl derjenigen Kanten, die vom Kopf- zum Schwanz-Teil führen. All diese Kanten (mit Ausnahme der als entfernt angenommenen), stellen hierbei nicht-Baumkanten dar.
Ein negativer Schnittwert impliziert nun typischerweise (jedoch aufgrund von Entartung nicht immer), dass die Gesamtkantenlänge durch Verlängerung der zugehörigen Baumkante verringert werden kann. Wir verlängern dabei diese Kante solange, bis eine nicht-Baumkante eng geworden ist, und tauschen die Baumkante durch diese aus. Ein alternativer spannender Baum ist entstanden, der ebenso eine gültige Layerzuweisung repräsentiert. Diesen Vorgang wiederholen wir solange, bis keine Baumkanten mehr einen negativen Schnittwert besitzen, was bedeutet, dass die implizierte Layerzuweisung optimal ist. Der Algorithmus selber terminiert dabei in einer endlichen Anzahl von Schritten [1, 3].

Die Funktionsweise dieses bisher nur abstrakt definierten Algorithmus wollen wir nun anhand von konkretem Pseudocode nochmals nachvollziehen, welcher auch die Grundlage für die angefertigte Implementierung dieses Algorithmus war.

Algorithmus 3.3: Netzwerk-Simplex (Spannender Baum)

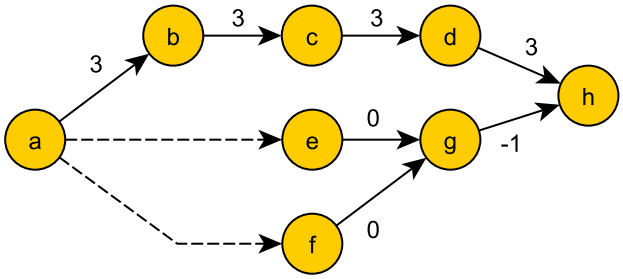
```
1  Prozedur spannenderBaum()
2      initLayerzuweisung();
3      Während engerBaum() < |V| tue
4          e = eine nicht−Baumkante inzident zum Baum mit minimalem Schlupf;
5          delta = schlupf(e);
6          Falls inzidenter Knoten ist Zielknoten von e dann
7              delta = −delta;
8          Für alle Knoten v aus dem Baum tue
9              erhöhe die Layer von v um delta;
10     schnittwerte();
11 Ende
```

Der Algorithmus beginnt mit der initialen Bestimmung einer gültigen Layerzuweisung für alle Knoten des Graphen. Eine solche lässt sich hierbei auf verschiedene Weise bestimmen und ist in ihrer konkreten Erzeugungsstrategie nicht festgelegt. Eine Möglichkeit ist hier, alle Quellknoten in den äußersten rechten Layer L_1 zu platzieren und alle übrigen Knoten dem Layer zuzuordnen, dessen Index der Länge des längsten Pfades zu einer Senke plus eins entspricht, welches mit dem Longest-Path-Verfahren übereinstimmt. Aber auch die Verwendung von übrigen Layerzuweisungsstrategien wie zum Beispiel die Verwendung des Coffman-Graham-Layerzuweisers stellen hier denkbare Optionen dar. Ausgehend von einem beliebigen, aber festen, Knoten u des Graphen werden in der Funktion *engerBaum()* nachfolgend alle Knoten traversiert, die von u über die ausschließliche Durchquerung enger Kanten erreicht werden können. Die Anzahl jener so erhaltenen Knoten wird zurückgegeben und ein spannender Baum des hierbei induzierten Subgraphen wird gebildet, bei dem alle Kanten, die zu jenem Baum gehören, entsprechend markiert werden. Konnten bei der initialen Ausführung dieser Prozedur nicht alle Knoten erreicht werden, wird eine nicht-enge Kante bestimmt, welche einen minimalen Schlupf besitzt, und alle Knoten des spannenden Baumes entsprechend so in ihrer Position verschoben, dass diese Kante fortan eine minimale Länge besitzt. Dieser Vorgang wird solange wiederholt, bis ein vollständiger spannender Baum über alle Knoten des Graphen bestimmt werden konnte, wobei in jeder Iteration mindestens ein weiterer Knoten zum Spannbaum hinzugefügt werden kann. Dieser besteht dabei nur aus engen Kanten, d.h. würde man nur diese betrachten, stellt die so definierte Layerzuweisung bereits eine optimale Lösung bezüglich minimaler Kantenlänge dar. Die durch diesen spannenden Baum implizierte Layerzuweisung wird im folgenden Verlauf des Algorithmus durch Verschieben einzelner Knoten zwischen den Layern solange verändert, bis die resultierende Zuweisung eine minimale Gesamtkantenlänge besitzt, und stellt somit eine Basislösung dar, die es zu optimieren gilt. Die daran anschließende Funktion *schnittwerte()* bestimmt hierzu die Schnittwerte für jede Kante gemäß der obigen Definition.

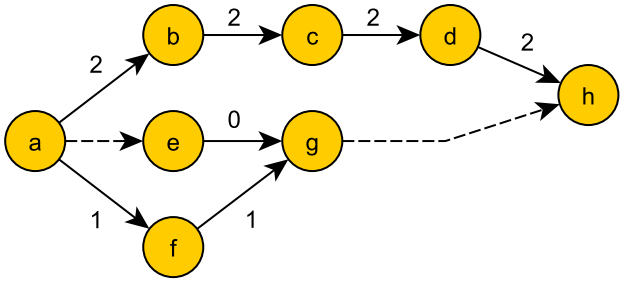
Algorithmus 3.4: Netzwerk-Simplex

```
1 Prozedur Netzwerk−Simplex(G = (V, E): gerichteter Graph)
2     spannenderBaum();
3     Während (e = auszutauschendeKante())≠⊥ tue
4         f = ersetzendeKante();
5         tauscheAus(e,f);
6         schnittwerte();
7     normalisiere();
8     balanciere();
9 Ende
```

Die Funktion *auszutauschendeKante()* gibt nun eine Baumkante zurück, die einen negativen Schnittwert besitzt, oder nichts (im Pseudocode als $\perp$ notiert), falls keine solche Kante existiert. Im letztgenannten Fall ist die durch den momentanen Spannbaum implizierte Layerzuweisung bereits optimal. Eine weitere Verbesserung der Lösung ist somit nicht mehr möglich und die Schleife wird verlassen. Die Prozedur kann hierbei eine beliebige Kante ausgeben. Eine Bevorzugung einer solchen mit geringstem Schnittwert ist nicht erforderlich, da ein negativer Wert lediglich andeutet, dass durch die Verkürzung dieser Kante der Zielfunktionswert in irgendeinem Maße verringert wird. Nachdem nun eine Kante bestimmt wurde, die den spannenden Graphen verlassen soll, wird mittels der Funktion *ersetzendeKante()* eine nicht-Baumkante ermittelt, die an dessen Stelle als neue Baumkante eingesetzt wird. Alle Kanten, die von der Kopf- zur Schwanz-Komponente verlaufen, werden hierbei berücksichtigt, wobei unter diesen eine solche mit minimalem Schlupf gewählt wird. Die beiden Kanten werden anschließend ausgetauscht, d.h. die ersetzende Kante wird durch Verschieben aller Knoten des Kopf- oder Schwanz-Teils auf ihre minimale Länge verkürzt und in den spannenden Baum eingefügt. Eine erneute Berechnung der Schnittwerte vervollständigt diese Iteration. Da nach Abschluss der Während-Schleife nicht notwendigerweise der Layer mit dem geringsten Index den Index 1 besitzt, wird dies in der Funktion *normalisieren()* nachgeholt. Der Name dieser Prozedur ist dabei etwas irreführend, da ebenso das Einfügen von virtuellen Knoten in die Layer (siehe Abschnitt 2.2) als solches bezeichnet wird. Diese Benennung wurde jedoch aus dem Original so übernommen. Abschließend wenden wir in der *balancieren()*-Funktion noch eine Heuristik an, welche wir auch schon bei der Lösung mittels LP-Löser verwendet haben. Um die gegebene Layerzuweisung auch noch in seiner Breite in beschränktem Maße zu optimieren, werden hier einzelne Knoten, die einen identischen eingehenden wie ausgehenden Grad besitzen und in mehreren Layern äquivalent platziert werden können, in einen erreichbaren Layer mit einer minimalen Anzahl an Knoten verschoben. Da für jede hierbei länger werdende Kante gleichsam eine gegenüberliegende Kante in ihrer Länge sinkt, wird die Optimalität der Layerzuweisung bezüglich der Gesamtkantenlänge dabei nicht beeinträchtigt. Ein kleiner Ausschnitt der arbeitenden Algorithmus ist dabei in dabei in Abbildung 3.4 zu beobachten.



(a) Ein Beispielgraph mit initialer Layerzuweisung und angegebenen Schnittwerten



(b) Der selbe Graph nach Terminierung des Algorithmus

Abbildung 3.4: Ein Beispiel des laufenden Netzwerk-Simplex-Algorithmus

Implementierung

An dieser Stelle soll eine effiziente Implementierung des Netzwerk-Simplex-Algorithmus vorgestellt werden, welche auch im Rahmen dieser Bachelorarbeit auf diese Weise angefertigt wurde. Da sich viele Teilfunktionen des Algorithmus relativ intuitiv mittels des vorliegenden Pseudocodes in linearer Laufzeit umsetzen ließen und daher kaum weiterer Erwähnung bedürfen, möchten wir uns hier auf die Betrachtung derjenigen Funktionen beschränken, die sich nicht ohne weitere Vorüberlegungen in linearer Zeit implementieren lassen. Hierbei handelt es sich zum einen um die *schnittwerte()*-Prozedur und zum anderen um die Bestimmung derjenigen Knotenmenge, welche den Kopf-Teil, respektive den Schwanz-Teil einer Kante ausmachen. Diese wurde dabei bisher so noch nicht als eigenständige Methode vorgestellt, jedoch werden die Informationen, die diese liefert, in vielen Teilfunktionen des Algorithmus wie beispielsweise in der *tauscheAus()*- und auch der *schnittwerte()*-Prozedur verwendet. Die hier erwähnten Implementierungsansätze wurden dabei aus der Arbeit von Gansner *et al.* [6] übernommen und lassen sich anhand dieser auch nochmals nachvollziehen.

Die Berechnung der Schnittwerte aller Kanten macht häufig einen signifikanten Anteil an der Gesamtlaufzeit des Algorithmus aus. Daher ist es besonders wichtig, dass diese Funktion so effizient wie möglich umgesetzt wird, um die Ausführungszeit des

gesamten Algorithmus so gering wie möglich zu halten. Eine naive Implementierung wäre hier für jede Baumkante alle Knoten zu traversieren, jeweils zu markieren, welche von diesen sich im Kopf- bzw. Schwanz-Teil der Kante befinden, und anschließend aus diesen Informationen den Schnittwert zu bestimmen. Dieser Ansatz hätte jedoch eine Laufzeit von $O(|V| \cdot |E|)$.

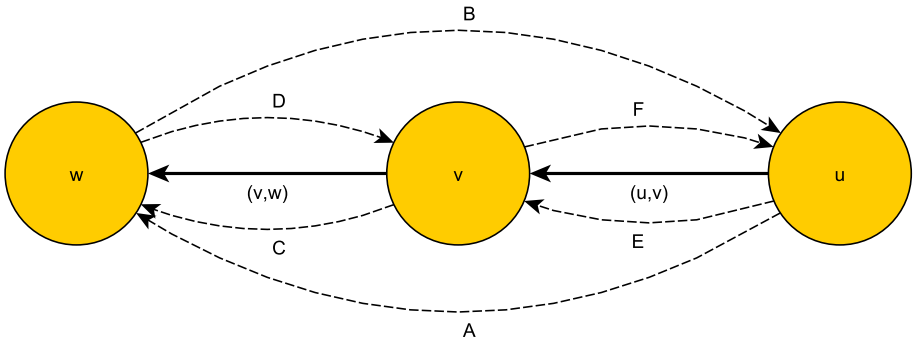


Abbildung 3.5: Inkrementelle Bestimmung der Schnittwerte

Der Schnittwert einer Kante lässt sich jedoch wesentlich effizienter berechnen, wenn man bedenkt, dass für dessen Bestimmung lediglich die Schnittwerte aller übrigen Baumkanten bekannt sein müssen, die gemeinsam zu einem ihrer Endpunkte inzident sind. Um sicherzustellen, dass stets ein solcher Fall vorliegt, muss die Berechnungsreihenfolge der Kanten von den Blättern des spannenden Baumes nur nach Innen führen. Es ist dabei trivial, den Schnittwert einer Kante zu ermitteln, die inzident zu einem Blatt des Spannbaums ist, da entweder deren Kopf- oder Schwanz-Komponente aus nur einem Knoten besteht. Wie man nun jedoch den Schnittwert einer Baumkante im Inneren des Spannenden Baumes bestimmen kann, soll zum besseren Verständnis zunächst für eine konkrete Konstellation beispielhaft mit Hilfe von Abbildung 3.5 erklärt werden. Die übrigen Fälle ergeben sich daraus analog. Alle Baumkanten sind hier als durchgezogene, nicht-Baumkanten jedoch als gestrichelte Pfeile mit jeweils angedeuteter Orientierung dargestellt. Der Schnittwert der Kante (u, v) sei hierbei bekannt und gegeben durch

$$\begin{aligned} s_{(u,v)} &= |\{(u,v)\}| + |\{A,E\}| - |\{B,F\}| && (3.7)\\ &= |\{(u,v)\}| + |\{A\}| + |\{E\}| - |\{B\}| - |\{F\}| && (3.8) \end{aligned}$$

Dann gilt für den Schnittwert von (v, w) gerade

$$\begin{aligned} s_{(v,w)} &= |\{(v,w)\}| + |\{A,C\}| - |\{B,D\}| && (3.9)\\ &= |\{(v,w)\}| + |\{A\}| + |\{C\}| - |\{B\}| - |\{D\}| && (3.10)\\ &= s_{(u,v)} + |\{(v,w)\}| + |\{C\}| + |\{F\}| - |\{(u,v)\}| - |\{D\}| - |\{E\}| && (3.11)\\ &= s_{(u,v)} + |\{(v,w)\}| + |\{C,F\}| - |\{(u,v)\}| - |\{D,E\}| && (3.12) \end{aligned}$$

Wir können also den Schnittwert einer jeden Kante unter der alleinigen Kenntnis der Werte der anderen inzidenten Kanten ausdrücken. Betrachtet man nun auch die übrigen Fälle, so lässt sich auf deren Grundlage ein allgemeingültiger Algorithmus zur Berechnung des Schnittwerts aufstellen. Sei dazu im Folgenden e die Baumkante, für welche ihr Schnittwert zu berechnen ist. Seien weiter die Baumkanten $b_1, \ldots, b_n$ und nicht-Baumkanten $c_1, \ldots, c_m$ inzident zu einem der Endpunkte von e, welchen wir als v bezeichnen, und die Schnittwerte von $b_1, \ldots, b_n$ bekannt. Dann ergibt sich die Berechnungsvorschrift:

Algorithmus 3.5: Effiziente Schnittwertberechnung

```
1  Prozedur schnittwert(e: Baumkante)
2      s_e = 1;
3      Für b ∈ {b_1, ..., b_n} tue
4          Falls e.Ziel = b.Ziel oder e.Quelle = b.Quelle tue
5              s_e = s_e − s_b + 1;
6          sonst s_e = s_e + s_b − 1;
7      Für alle c ∈ {c_1, ..., c_m} tue
8          Falls v.Quelle = e.Quelle tue
9              Falls c.Quelle = e.Quelle tue
10                 s_e = s_e + 1;
11             sonst s_e = s_e − 1;
12         sonst
13             Falls c.Quelle = e.Quelle tue
14                 s_e = s_e − 1;
15             sonst s_e = s_e + 1
16 Ende
```

Diese Prozedur besitzt hierbei offensichtlich eine lineare Laufzeit zur Menge der inzidenten Kanten, mittels welcher die Berechnung des Schnittwertes von e durchgeführt wird. Da im Zuge der Bestimmung der Schnittwerte aller Baumkanten jede Kante höchstens zweimal traversiert wird, besitzt dieser Ansatz somit eine lineare Gesamtlaufzeit für die Wertermittlung sämtlicher Baumkanten des Graphen.

Bei der Bestimmung der Knotenmenge, die die Kopf- bzw. Schwanz-Komponente einer Baumkante bilden, bedient man sich einer anderen, aber ebenso effektiven Technik. Die Grundidee besteht hier darin, auf dem Graphen, von einem beliebigen Knoten ausgehend, eine *Postorder*-Traversierung durchzuführen und dabei zu jedem Knoten sowohl seine Postorder-Nummer (pn) als auch die niedrigste Nummer desjenigen Knotens zu speichern, der während der Durchwanderung des Graphen von diesem Knoten nachfolgend erreicht wurde (npn). Bei dieser Form der Tiefensuchdurchquerung des Baumes wird dem momentan treversierten Knoten w dabei zunächst die aktuelle und mit jedem besuchten Knoten zu inkrementierende Durchgangsnummer als *npn*-Wert zugeteilt und anschließend alle zu diesem adjazenten und noch nicht besuchten Unterbäume traversiert. Kehrt die Tiefensuche nach deren Abarbeitung zu w zurück, wird diesem Knoten dessen *pn*-Attribut zugeordnet, welches auch hier der aktuellen Durchgangsnummer entspricht. Abbildung 3.6 zeigt

hier ein Beispielgraphen, welcher mittels einer solchen Traversierung durchwandert wurde. Die Knoten sind dabei mit (pn,npn) beschriftet.

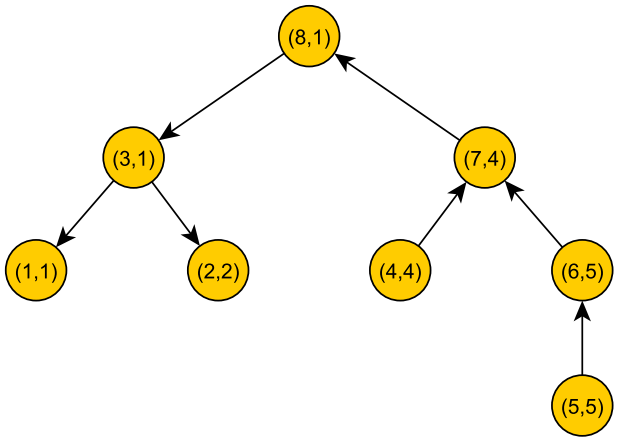


Abbildung 3.6: Postorder-Traversierung eines Beispielgraphen

Mit Hilfe dieser Werte lässt sich nun für jede Kante relativ leicht in konstanter Zeit ermitteln, ob ein gegebener Knoten zum Kopf- oder Schwanz-Teil dieser Kante zugehörig ist. Offensichtlich kann dabei ein jeder Knoten nicht zu beiden Komponenten gehören. Deswegen reicht es hier aus, nur einen Algorithmus anzugeben, der überprüft, ob ein Knoten zur Kopf-Komponente gehört. Der andere Fall ergibt sich dann durch Negation des Rückgabewertes.

Algorithmus 3.6: Effiziente Komponentenzugehörigkeit

```
1  Prozedur istInKopfteil(e: Baumkante, v: Knoten)
2      Falls e.Quelle.npn ≤ v.pn und v.pn ≤ e.Quelle.pn
3              und e.Ziel.npn ≤ v.pn und v.pn ≤ e.Ziel.pn tue
4          Falls e.Quelle.pn < e.Ziel.pn tue
5              gib falsch zurück;
6          sonst gib wahr zurück;
7      sonst
8          Falls e.Quelle.pn < e.Ziel.pn tue
9              gib wahr zurück;
10         sonst gib falsch zurück;
11 Ende
```

Mit der jetzigen Betrachtung der *schnittwerte()*- und der *istInKopfteil()*-Funktion besitzen nun alle wesentlichen Subprozeduren des Netzwerk-Simplex-Ansatzes eine lineare Laufzeit und stellen damit die Grundlage für eine effiziente Ausführung des Algorithmus dar. Inwieweit nun jedoch dessen Laufzeit und Qualität des Ausgabelayouts mit der Performanz der übrigen Layerzuweisern mithalten kann, soll nun im Folgenden geklärt werden.

3.5 Laufzeit- und Ergebnisanalyse

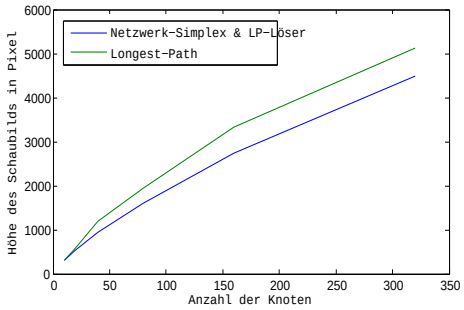
In diesem Abschnitt möchten wir das Laufzeitverhalten und die Ergebnisgüte einer Auswahl verschiedener Layerzuordnungsverfahren testen und gegenüberstellen, wobei wir jeweils die Kategorien Kantenlänge und Breite des finalen Layouts sowie Ausführungszeit der Layerzuweiser unterscheiden. Als Algorithmen standen uns hierfür der Longest-Path-Zuweiser aus Abschnitt 3.2, welche im KIELER-Projekt bereits vorhanden war, sowie der Netzwerk-Simplex-Algorithmus und der Ansatz mittels LP-Löser, welche erst im Rahmen dieser Arbeit implementiert wurden, zur Verfügung. Auf eine Untersuchung der Layoutbreite haben wir hierbei sinnvollerweise verzichtet. Zwar besitzt der Longest-Path-Ansatz dessen Minimierung als charakteristische Aufgabe, jedoch impliziert eine Layerzuweisung von minimaler Kantenlänge, welche vom Netzwerk-Simplex-Algotithmus oder dem Verfahren mittels LP-Löser erzeugt wird, auch eine solche von minimaler Breite [4]. Eine Gegenüberstellung der Resultate in dieser Eigenschaft ist daher nicht notwendig, da dessen Wert bei allen der oben genannten Ansätzen identisch wäre.
Tarassov *et al.* haben dabei in ihrer Arbeit ähnliche Laufzeit- und Layoutanalysen von Layerzuweisungsansätzen durchgeführt, wobei diese zusätzlich auch den Coffman-Graham-Ansatz in ihre Untersuchungen mit einbezogen haben [14]. Deren Kategorien sind dabei leider nicht identisch mit den von uns verwendeten, wodurch ein direkter Vergleich mit deren Resultate nicht möglich ist.

Im Folgenden wollen wir nun auf die Ergebnisse unserer Analysen näher eingehen. Für jede der genannten Kategorien wurden dabei zwei Messreihen erstellt. In der einen wurde jeder Algorithmus auf eine Folge von Graphen mit steigender Knotenanzahl bei einem gleich bleibendem Kantenverhältnis von 1:2 angewandt (jeweils linkes Schaubild) und in der anderen wurde eine Menge von Graphen mit einer festen Anzahl von 40 Knoten erzeugt, bei welcher wir die Kantenmenge stetig anwachsen ließen (rechtes Diagramm). Zur Ermittlung der einzelnen Werte wurden dabei für jeden Messpunkt jeweils fünf Zufallsgraphen der angegebenen Größe erstellt und anschließend der arithmetische Mittelwert der dabei jeweils gemessenen Eigenschaft der resultierenden Layerzuweisung verwendet. Bei der Ermittlung der Laufzeiten der getesteten Algorithmen wurde dabei besonders auf eine Vermeidung von Speichereffekten geachtet, d.h. für die Ausführungszeit wurde der niedrigste Messwert nach zehnmaliger Ausführung des Ansatzes auf dem selben Graphen verwendet.

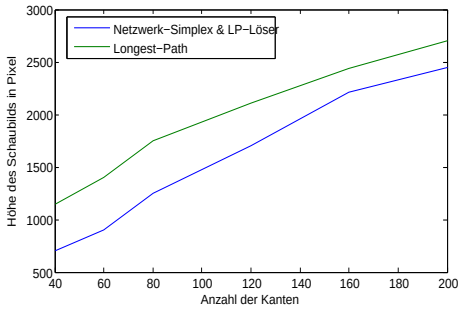
3.5.1 Höhe der Layerzuweisung

Abbildung 3.7 zeigt die ermittelten Werte bezüglich der Höhe der Darstellung, wobei das linke Schaubild die Messreihe mit kontinuierlich anwachsender Graphgröße darstellt und die rechte Abbildung jene mit fester Knotenmenge bei unterschiedlicher Kantenzahl visualisiert. Man erkennt dabei, dass die Höhe bei Verwendung des Netzwerk-Simplex oder des LP-Lösers, wessen Werte in unseren Tests stets im Rah-

men einer Abweichung von höchstens 3% identisch ausfielen und daher hier nur als eine Kurve dargestellt wurden, kontinuierlich einen niedrigeren Wert besitzt als bei Anwendung des Longest-Path-Ansatzes. Dieses Phänomen deckt sich dabei mit den in Abschnitt 3.2 gewonnenen Erkenntnissen, wonach bei diesem die Höhe des Layouts besonders im äußersten rechten Layer besonders groß ausfallen kann, da in diesem sämtliche Senken des Graphen platziert werden. Ein Effekt, der beim Netzwerk-Simplex- oder LP-Löser-Verfahren ebenso eine geringere Höhe der Layerzuordnung begünstigt, stellt dabei zudem auch die *balance()*-Methode dar, welche in beiden Ansätzen integriert wurde und welche wir in Abschnitt 3.4.2 näher beschrieben haben. Existieren zu einer gegebenen Graphen mehrere optimale Layerzuweisungen bezüglich minimaler Kantenlänge, so verschiebt diese, wenn möglich, einzelne Knoten in einen gültigen Layer mit einer möglichst geringen Anzahl an zugewiesenen Knoten, ohne die Gesamtkantenlänge dabei zu verändern.



(a) Steigende Knotenanzahl bei gleichbleibendem Kantenverhältnis



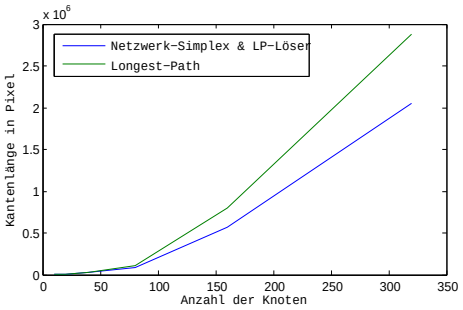
(b) Steigende Kantenanzahl bei gleichbleibender Knotenzahl

Abbildung 3.7: Layouthöhe der finalen Darstellung

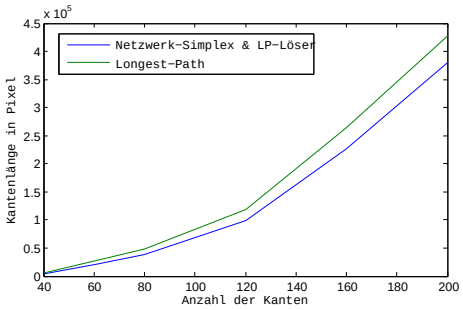
3.5.2 Länge der Kanten

Abbildung 3.8 stellt die Ergebnisse der Verwendung der genannten Layerzuweiser in Bezug auf die resultierende Gesamtkantenlänge des finalen Layouts dar, wobei wir für die Darstellung die Polyline-Kantenführung gewählt haben. Auch in diesem Fall fällt das Ergebnis ziemlich eindeutig für den Netzwerk-Simplex-, respektive den LP-Löser-Ansatz aus, wobei auch hier deren Ergebnisse bis auf wenige Prozentpunkte gleich ausfielen, sodass wir auch hier deren Werte der Übersichtlichkeit halber nur mittels einer Kurve dargestellt haben. So nimmt die Gesamtkantenlänge der Schaubilder bei dessen Verwendung stets einen niedrigeren Wert als bei dem Einsatz des Longest-Path-Algorithmus an, jedoch fällt der Unterschied zwischen diesen bei deren Anwendung auf Graphen mit lediglich steigender Kantenzahl nicht in dem Maße aus wie bei der ersten Messreihe und ist auch hier auf den oben beschriebenen Effekt zurückzuführen. Ein solches Ergebnis ist dabei keineswegs verwunderlich, denn die erstgenannten Verfahren haben als konkretes Optimierungsziel die Minimierung der

Kantenlänge, wohingegen der Longest-Path-Ansatzes allein die Breite der Layerzuordnung optimiert. Dies stellt somit lediglich einen Beweis für die Effektivität dieser Verfahren dar.



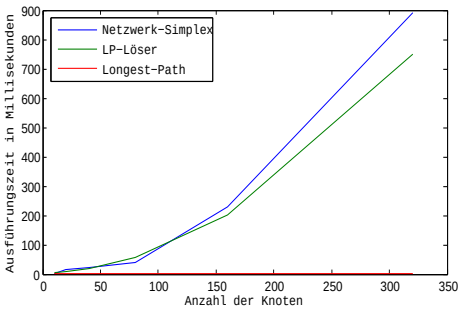
(a) Steigende Knotenanzahl bei gleichbleibendem Kantenverhältnis



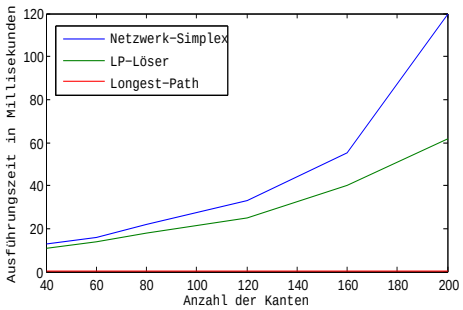
(b) Steigende Kantenanzahl bei gleichbleibender Knotenzahl

Abbildung 3.8: Kantenlänge der finalen Darstellung

3.5.3 Algorithmische Laufzeit



(a) Steigende Knotenanzahl bei gleichbleibendem Kantenverhältnis



(b) Steigende Kantenanzahl bei gleichbleibender Knotenzahl

Abbildung 3.9: Algorithmische Laufzeit der betrachteten Layerzuweiser

Bezüglich der Laufzeit der Algorithmen ergibt sich in Abbildung 3.9 ein relativ überraschendes Ergebnis. So ist die Ausführungszeit des LP-Löser-Layerzuweisers bei kleineren Graphen mit der des Netzwerk-Simplex-Verfahrens vergleichbar. Steigt jedoch die Größe des Graphen, so vergrößert sich zunehmend der Abstand zwischen diesen zugunsten des erstgenannten Ansatzes. Zwar ist Performanz des LP-Lösers von dessen konkreter Implementierung in hohem Maße abhängig, wobei wir in diesem Fall

die Programmierschnittstelle des Programms *lp_ solve* gewählt haben, jedoch scheint es dennoch verwunderlich, dass dieser relativ schnell zu umzusetzende Ansatz eine in der Regel niedrigere Ausführungszeit besitzt als der doch im Gegenzug vergleichsweise aufwendig umzusetzende Netzwerk-Simplex-Layerzuweiser, zumahl dieser bereits über sämtlicher Performanzverbesserungen verfügt, welche wir in Abschnitt 3.4.2 vorgestellt haben. Als eindeutig performantester Layerzuweiser hat sich hier jedoch der Longest-Path-Ansatz herausgestellt, welcher für alle von uns erzeugten Testgraphen eine Ausführungszeit von unter einer Millisekunde besitzt.

3.5.4 Optische Gegenüberstellung

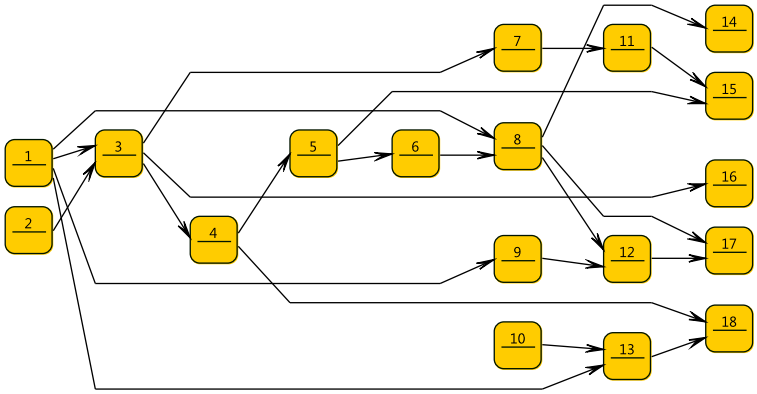


Abbildung 3.10: Ein mittels Longest-Path-Ansatz geschichteter Graph

Nachdem wie in den vorangegangenen Abschnitten die genanten Layerzuweiser eher anhand von eher harten Kriterien isoliert voneinander verglichen haben, möchten wir hier dem Leser auch eine Gegenüberstellung der Layoutergebnisse bieten, bei welcher die zuvor ermittelten Messergebnisse auch in dessen Zusammenspiel beobachtet werden können. Abbildung 3.10 und Abbildung 3.11 zeigen hierzu einen etwas größeren Graphen, der jeweils mit dem Netzwerk-Simplex- und dem Longest-Path-Ansatz geschichtet wurde. Die Grafiken stammen dabei direkt aus dem KIELER-Projekt[2], in welches der von uns implementierte Netzwerk-Simplex-Ansatz integriert wurde und in welchem eine Implementierung des Longest-Path-Layerzuweisers schon vorhanden ist. Auf eine zusätzliche Einbeziehung des vom LP-Löser erzeugten Layouts haben wir hierbei verzichtet, da in der Regel dessen ermittelte Layerzuweisungen und damit auch die späteren Designs der Schaubilder bis auf wenige vernachlässigbare Unterschiede indifferent zu den Ergebnissen des Netzwerk-Simplex-Algorithmus sind.

[2] siehe Abschnitt 1.4

In Abbildung 3.10 erkennt man das typische Design eines mittels Longest-Path-Layerzuweisers geschichteten Graphen. Sämtliche Senken des Graphen wurden hierbei in dem ersten, d.h. äußersten rechten Layer, platziert, wohingegen alle übrigen Knoten dem Layer zugeteilt wurden, dessen Index der Anzahl der Knoten in einem längsten Pfad zu einer Senke plus eins entspricht. Hierdurch entstehen diese oftmals besonders langen Kantenzüge, welche das Layout als besonders breit gezogen wirken lassen. Da dieser Graph dabei mit fünf Knoten im ersten Layer besonders viele Senken im Vergleich zur Gesamtknotenzahl besitzt, ist zudem auch die Höhe des Schaubilds relativ üppig. Auch dies stellt ein klassisches Merkmal dieses Layerzuweisers dar.

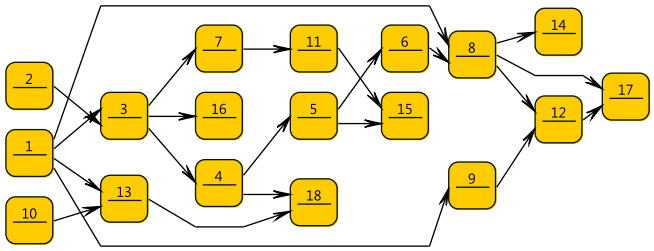


Abbildung 3.11: Der selbe Graph aus Abbildung 3.10 mittels Netzwerk-Simplex-Verfahren geschichtet.

Abbildung 3.11 zeigt im Gegensatz hierzu das Ergebnis des Netzwerk-Simplex-Ansatzes, welcher dabei eine Layerzuweisung mit minimaler Kantenlänge ermittelt hat. Das Layout wirkt wesentlich gedrungener, welche das Finden von jeweils adjazenten Knoten erleichtern soll, da man nicht mehr langen, sondern hier nur kurzen Kantenzügen folgen muss. Als ein ansehnlicher Nebeneffekt dieser Optimierung besitzt die Darstellung eine im Vergleich zu oberen Ansatz deutlich kompaktere Form, da die Höhe der Layerzuweisung auch unter Einbeziehung der virtuellen Knoten fünf nicht überschreitet.

4 Layerzuweisung breiter Knoten

In den bisherigen Betrachtungen der Layerzuweisungsphase, respektive des hierarchischen Layouts, haben wir Knoten stets als gleichgroße und gleichförmige Elemente des Graphen kennen gelernt und verstanden. Diese Sichtweise soll sich nun in diesem Kapitel ändern und wir möchten uns einer Facette der Layerzuweisung widmen, welche bislang in der gängigen Literatur zur Layerzuordnung so noch nicht thematisiert wurde. Gemeint ist hier die Behandlung von Knoten unterschiedlicher Breite. Im Folgenden wollen wir hierbei die Layerzuweisungseigenschaften des hierarchischen Ansatzes beim Vorhandensein solcher Knoten untersuchen, sowie konkrete Vorschläge zu dessen Verbesserung nennen.
Besitzt ein Diagramm ungleich breite Knoten, so kommen auf einen Layout-Ansatz häufig sehr viel schwerer zu erfüllende Anforderungen zu. Zwar soll beispielsweise die finale Darstellung des Graphen stets vielen ästhetischen Ansprüchen wie Kompaktheit und der Verwendung möglichst kurzer Kantenzüge genügen, jedoch trägt die zusätzliche Komplexität, welche mit der Einführung unterschiedlicher Breitenmaße im Graphen verbunden ist, gerade dazu bei, dass sich das Finden einer optisch idealen Lösung ungemein erschwert. Während bislang bei Graphen mit Knoten äquivalenter Größe stets alle Elemente, was deren äußere Form anbelangt, als identische Einheiten des zu visualisierenden Schaubilds angesehen werden können, müssen nun ebenso Eigenschaften wie die individuelle Höhe und Breite eines jeden Knotens und dessen Zusammenspiel im Graphen bei dessen Positionierung mit berücksichtigt werden. Im Folgenden wollen wir nun die spezifische Strategie, welche das hierarchische Layout bei dem Umgang mit breiten Knoten verfolgt, näher erläutern und dabei von dessen Schwächen ausgehend konkrete Ansätze zu dessen Verbesserung nennen.

4.1 Das bisherige Layout und seine Schwächen

Das hierarchische Layout handhabt Schaubilder mit Knoten unterschiedlicher Breite bislang relativ simpel. In der Layerzuweisungsphase werden alle Knoten als solche äquivalenter Größe und Gestalt betrachtet und ungeachtet ihrer wahren Maße gemäß der gewählten Zuweisungsstrategie, z.B. Longest-Path, in den Layern platziert. In der Knotenpositionierungsphase findet anschließend dessen konkrete Bestimmung der Y-Koordinate auf der Zeichenfläche statt. Die Breite eines jeden Layers ergibt sich dabei aus der Länge seines breitesten zugewiesenen Knotens. Alle Knoten des selben Layers werden im Anschluss gemäß dem Index seines zugewiesenem Layers und unter Berücksichtigung der Maße aller übrigen Knoten mittig auf einer parallelen Linie zur Y-Achse platziert. Abbildung 4.1 zeigt hierbei ein Beispiel eines mittels

mit Netzwerk-Simplex-Algorithmus visualisierten Graphen mit Knoten differierender Breite. Die Knoten 1 bis 5 besitzen in diesem Schaubild eine identische Größe, lediglich Knoten 6 ist hier inklusive des vordefinierten Objektabstandes zwischen den Diagrammelementen mehr als dreimal so breit wie die übrigen.

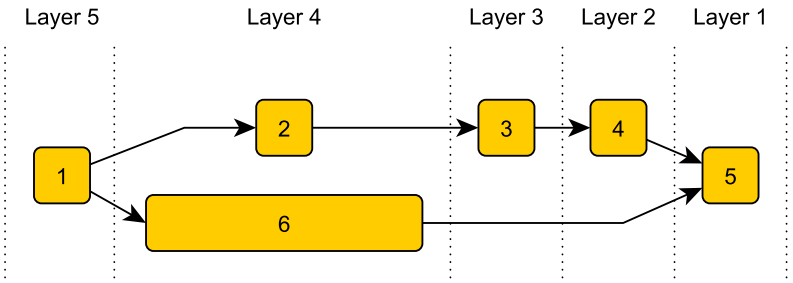


Abbildung 4.1: Ein geschichteter Graph mit unterschiedlich breiten Knoten

An diesem Schaubild erkennt man relativ schnell einen deutlichen Nachteil des hierarchischen Layouts. Zwar besitzt diese Darstellung gemäß den Maßstäben der Layerzuweisung eine minimale Kantenlänge, jedoch stellt dies keineswegs eine ästhetisch ansprechende Lösung dar. So wird beispielsweise die Übersichtlichkeit durch dessen starke vertikale Ausdehnung beeinträchtigt und die drei besonders langen Kanten $(1, 2)$, $(2, 3)$ und $(6, 5)$ sind nicht gerade förderlich für die Verständlichkeit der Topologie des Graphen. Auch die Verwendung eines anderen Layerzuweisungs-Algorithmus, welcher andere Optimierungsziele als der Minimierung der Kantenlänge verfolgt, vermag nicht, diese Visualisierung im ästhetischen Sinne weiter zu verbessern. Die besondere Definition der Layerzuteilung ist hierfür verantwortlich, denn diese verbietet die gleichzeitige Zuteilung von Knoten zu mehreren benachbarten Layern.

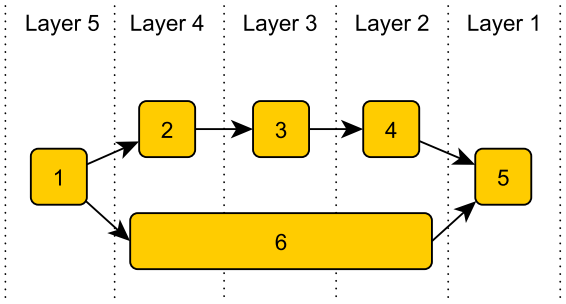


Abbildung 4.2: Eine optimale Darstellung des Graphen aus Abbildung 4.1

4.2 Ein neuer Ansatz der Layerzuweisung

Eine optisch perfekte Darstellung des selben Graphen ist in Abbildung 4.2 zu finden. Die Darstellung entspricht dabei jener aus Abbildung 4.1 bis auf einen entscheidenden Unterschied. Der Knoten mit Index 6 wurde hier zu gleichen Teilen auf die Layer 2, 3 und 4 verteilt. Der weiße Zwischenraum, der von den Kanten $(1, 2)$ und $(2, 3)$ überspannt wird, wurde somit verringert und die dortige frei gewordene Zeichenfläche konnte für die Platzierung der Knoten 3 und 4 verwendet werden, wodurch eine gemäß den gegebenen ästhetischen Kriterien optimale Visualisierung des Graphen entstanden ist. Diese Lösung ist jedoch gemäß der momentanen Definition der Layerzuweisung illegal, denn ein Knoten wurde hier mehreren Layern zugewiesen. Aus diesem Grund soll uns diese Darstellung als Motivation gelten, das bisherige Verfahren dahingehend zu verbessern, dass auch die gleichzeitige Zuordnung von Knoten zu verschiedenen, paarweise benachbarten Layern zwecks der Optimierung der ästhetischen Qualität des Schaubildes zulässig ist.

4.2.1 Neudefinition der Layerzuweisung

Eine Definition, die diese Forderung dennoch erfüllt, können wir wie folgt aufstellen und bezeichnen diese im Folgenden als *mehrschichtige* Layerzuweisung.

Sei $G = (V, E)$ ein gerichteter, azyklischer Graph. Eine *mehrschichtige Layerzuweisung* eines azyklischen, gerichteten Graphen $G = (V, E)$ ist eine Menge von Teilmengen $L = \{L_1, \dots L_n\}$ von V, welche dessen Layer genannt werden, und für die die folgenden Eigenschaften erfüllt sind:

1. Jeder Knoten $v \in V$ befindet sich mindestens in einem Layer.

2. Wurde v mehreren Layern zugewiesen, so stellt deren aufsteigend sortierter Index eine streng monoton wachsende Folge dar, bei welcher die Differenz zweier aufeinanderfolgender Werte exakt eins beträgt.

Von der ursprünglichen Definition der Layerzuweisung unterscheidet sich unsere dabei nur in einen einzigen Aspekt. So erlaubt die zweite Eigenschaft die Zuteilung eines, z.B. besonders breiten, Knotens zu mehr als einem Layer, wobei alle Layer, die einem Knoten zugewiesen wurden, lediglich einen fortlaufenden Index besitzen müssen. Mit dessen Hilfe lässt sich nun auch ein Schaubild wie in Abbildung 4.2 im hierarchischen Ansatz legal erzeugen.
Nachdem wir nun die bisherige Definition gemäß unseren Vorstellungen einer ästhetisch hochwertigeren Layerzuweisung erweitert haben, können wir nun der Frage nachgehen, wie wir diesen Neuansatz in das hierarchische Layout, respektive gezielt in die Phase der Layerzuteilung algorithmisch integrieren können. Damit wir hier nicht sämtliche existierende Layerzuweiser neu entwickeln müssen, wollen wir hier eine Lösung anstreben, bei welcher wir die bisher bekannten Ansätze weitestgehend wiederverwenden können. Die oben genannte starke Ähnlichkeit zur ursprünglichen

Definition kommt uns hierbei zu Gute. So stellt die einzige Veränderung die obige zweite Eigenschaft dar. Deren Erfüllung können wir aber auch ohne explizite Anpassung der Layerzuweiser sicherstellen, in dem wir die bisherigen Algorithmen um eine vorbereitende Phase ergänzen, die diese Aufgabe übernimmt. Um die Zuteilung eines Knotens zu mehreren Layern auch gemäß der ursprünglichen Definition zu ermöglichen, besteht hier das Verfahren, für welches wir uns entschieden haben, darin sämtliche Knoten, welche breiter sind als der schmalste Knoten des Schaubilds, temporär für die Phase der Layerzuweisung in mehrere kleinere aufzuteilen und diese wiederum mittels Kanten miteinander zu verbinden. Abbildung 4.3 zeigt ein solches Beispiel, bei welchem der Beispielgraph aus Abbildung 4.2 auf diese Weite bearbeitet wurde. Knoten 6 wurde hier in die drei Subknoten 6.1, 6.2 und 6.3 untergliedert.

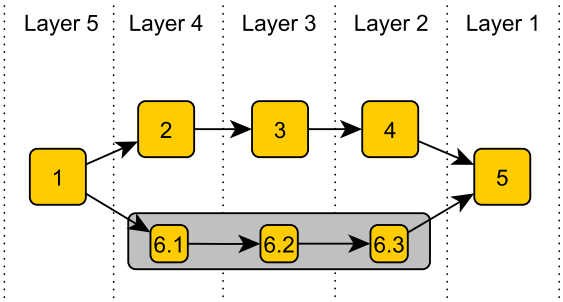


Abbildung 4.3: Der breite Knoten 6 wird in mehrere kleinere Subknoten unterteilt.

Folglich besitzen alle Knoten des Diagramms eine ähnliche Größe und können wie gewohnt von den Layerzuweisern auch unter Einhaltung der geforderten ästhetischen Kriterien ansprechend den Layern zugewiesen werden, wobei jeder zu einem Knoten zugehöriger Subknoten in einem seperaten Layer platziert wurde. Bei der Zuordnung muss lediglich beachtet werden, dass die verknüpfenden Kanten zwischen diesen stets minimale Länge besitzen, d.h. sich alle Subknoten in aufeinanderfolgenden Layern befinden. Dies lässt sich jedoch relativ leicht durch kleine Änderungen an den Layerzuweisern selbst oder eine nachträgliche Bearbeitung der Layerzuweisung nach dessen Ausführung realisieren. Damit diese vorübergehende Veränderung der Knotenmenge auf das spätere Diagramm keine Auswirkungen besitzt, müssen diese Unterteilungen direkt nach der Layerzuordnung oder in einer darauffolgenden Phase wieder rückgängig gemacht werden, indem der ursprüngliche breite Knoten aus seinen Unterknoten wieder zusammengesetzt wird. Diesen bisher doch eher abstrakt beschriebenen Vorgang wollen wir nun nochmals anhand von konkretem Pseudocode verdeutlichen.

Algorithmus 4.1: Knotenteilung

```
1  Prozedur teileBreiteKnoten(G = (V, E))
2      e = Breite des kürzesten Knotens;
3      Für alle v ∈ V tue
4          d = 0;
5          Solange v.Breite > e tue
6              d = d + 1;
7              v.Breite = v.Breite − e;
8          Erzeuge neue Liste l = ∅
9          Solange d > 0 tue
10             Erzeuge neuen Knoten u;
11             Falls l ≠ ∅ tue
12                 k = Letztes Element von l;
13                 Erzeuge neue Kante (k, u);
14             Füge u ans Ende von l an;
15             d = d − 1
16         Falls l ≠ ∅ tue
17             i = Erstes Element von l;
18             j = Letztes Element von l;
19             Für alle z ∈ δ+(e) tue
20                 z.Quelle = j;
21             Erzeuge neue Kante (e, i);
22 Ende
```

Die Funktion beginnt dabei mit der Bestimmung der geringsten Breite e aller Knoten des gegebenen Graphen, welche hierbei als Referenzwert dient, mittels welchem die Anzahl der Knoten bestimmt wird, in die ein jeder Knoten im Graphen unterteilt wird. Hierzu traversiert die Prozedur alle Knoten und bestimmt dabei, um ein wie Vielfaches der momentan bearbeiteten Knoten v breiter ist als e, wessen Wert in der Variablen d gespeichert wird. In der zweiten Solange-Schleife werden nun die konkreten zu v zugehörigen Subknoten erzeugt, wobei all diese zunächst mittels einzelnen, gleich orientierten Kanten (*Verbindungskanten*) zu einer langen Kette, welche im Code durch die Liste von Knoten l repräsentiert wird, verbunden werden. Diese Knoten besitzen dabei höchstens eine ein- und ausgehende Kante. Nach deren Generierung aller d neuen Knoten werden alle ausgehenden Kanten von v entfernt und an das letzte Element in l so angefügt, dass deren neuer Quellknoten i beträgt, wobei die Zielknoten dieser Kanten unverändert bleiben. Anschließend wird v mittels einer weiteren Kante mit dem erste Knoten von l verbunden. Der ehemals lange Knoten v wurde somit in $d + 1$ kleinere Knoten aufgeteilt, deren Breite e nicht übersteigt. Im direkten Anschluss an die Termination dieser Prozedur kann nun der gewählte Layerzuweiser wie gewohnt ausgeführt werden, welcher eine konkrete Schichtung des Graphen bestimmt, die dabei der Definition einer mehrschichtigen Layerzuordnung erfüllt, sofern alle Verbindungskanten die Länge von eins behalten. Ist diese Eigenschaft dabei nicht auf Anhieb erfüllt, lässt sich dabei jedoch relativ intuitiv eine nachbereitende Phase entwickeln, die die Layerzuweisung entsprechend korrigert. Ein relativ simpler Ansatz wäre hier, alle Layer von rechts nach links zu traversie-

ren, die Erfüllung für alle Verbindungskanten abzuprüfen und bei denen, die diese Eigenschaft verletzen, deren adjazente Knoten entsprechend soweit heranzuschieben, bis die Kante wieder eine Länge von eins besitzt. Für die Laufzeit der *teileBreite-Knoten()*-Prozedur lässt sich dabei eine solche von $O(g \cdot |V|)$ feststellen, wobei g die Anzahl an Knoten definiert, in welche der breiteste Knoten des Graphen untergliedert wird. Dies lässt sich damit begründen, dass jeder Knoten zweimal traversiert und in höchstens g kleinere Unterknoten aufgeteilt wird.

4.2.2 Hervorhebung der Schichtung

Nachdem wir nun auch die Zuweisung eines Knotens zu mehreren Layern sowohl algorithmisch als auch per Definition ermöglicht haben, möchten wir nun jedoch den Fokus auf eine ästhetische Unzulänglichkeit lenken, die mit dieser verbunden ist. Die Grundidee des hierarchischen Layouts ist es, die Menge der Knoten in vertikale Hierarchieebenen zu unterteilen und durch deren klare optische Unterscheidbarkeit eine Strukturierung des Schaubilds zu erzeugen, die dazu geeignet ist, die Topologie des Graphen übersichtlich und leichter verständlich zu veranschaulichen. Betrachtet man nun jedoch Abbildung 4.4, so stellt man fest, dass diese zwar die Definition der mehrschichtigen Layerzuweisung erfüllt ist, jedoch ist eine eindeutige Einteilung der Knoten in vertikale Schichten nicht mehr erkennbar.

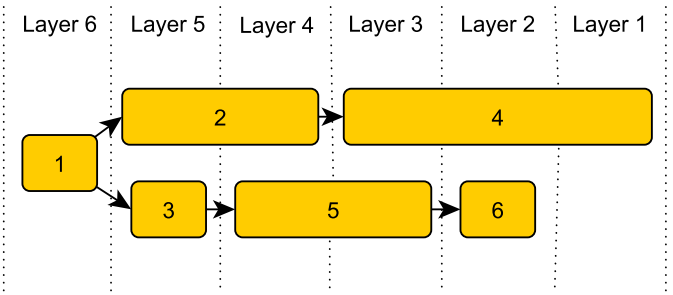


Abbildung 4.4: Ein mittels mehrschichtiger Layerzuweisung dargestellter Graph

Als eindeutige vertikale Hierarchieebenen des Diagramms sind in diesem Schaubild lediglich zwei zu erkennen. Die eine, die den Knoten 1 enthält, und die andere, der die restlichen Knoten des Graphen zugeteilt wurden, wobei gerade dessen Elemente relativ unsortiert und unstrukturiert angeordnet wirken. Dieser ästhetische Makel motiviert uns, dieses Verfahren und die damit verbundene Definition der mehrschichtigen Layerzuordnung weiter zu verbessern.

Eine bessere Darstellung des selben Graphen, die dabei die auch die eindeutige Erkennbarkeit der Hierarchieebenen wahrt, ist in Abbildung 4.5 gegeben. In dieser wurde die Layerzuteilung aus Abbildung 4.4 dahingehend verbessert, dass die Knoten 5 und 6 unter Knoten 4 verschoben wurden, wodurch im Schaubild nun drei, anstelle

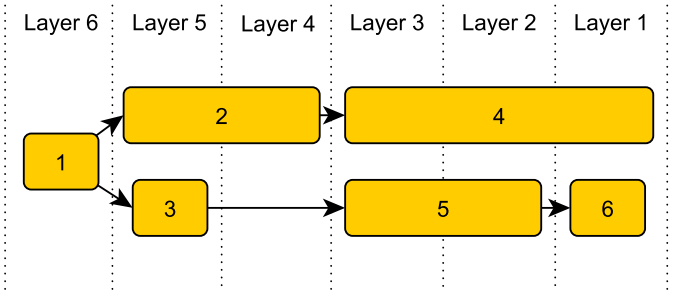


Abbildung 4.5: Eine hierarchieebenentreue Darstellung des Graphen aus Abbildung 4.4

von zwei, Schichtungsebenen identifiziert werden können. Eine weitere Unterteilung ist dabei nicht mehr sinnvoll möglich, da jede Ebene bereits eine minimale Breite besitzt, die durch die horizontale Ausdehnung des breitesten Knotens definiert wird. Zwar hat sich sich die Kantenlänge durch die Verlängerung von $(3, 5)$ um ein gewisses Maß erhöht, jedoch ist dies unserer Ansicht nach als akzeptabel zu betrachten, da im Gegenzug das Design des Diagramms hin zu einer strukturierteren Darstellung verbessert wurde.
Diese bisher eher beispielhafte Beschreibung der ästhetischen Optimierung der mehrschichtigen Layerzuweisung wollen wir nun in eine allgemeingültige Definition gießen, mit welcher auch gleichzeitig einige neue Begriffe vorgestellt werden.

Bezeichne mit $L(w)$ die Menge aller Layer, zu denen ein Knoten $w \in V$ zugewiesen wurde. Dann heißt eine mehrschichtige Layerzuweisung eine *segmentierte* Layerzuweisung, wenn für alle Paare von Knoten $u, v \in V$ mit $L(u) \cap L(v) \neq \emptyset$, ein nicht notwendigerweise verschiedener Knoten $w \in V$ existiert mit $(L(u) \cup L(v)) \subseteq L(w)$.
Ein *Segment* $S \subset L$ bezeichnet eine Menge von Layer, zu denen ein Knoten $v \in V$ existiert mit $L(v) = S$ und für welche kein anderer Knoten $w \in V$ existiert mit $S \subset L(w)$. Zwei Knoten $u, v \in V$ heißen *schneidend*, wenn $L(u) \cap L(v) \neq \emptyset$, $L(u) \not\subseteq L(v)$ und $L(v) \not\subseteq L(u)$ gelten. Ein Knoten v heißt *Segmentknoten*, falls ein Segment S existiert mit $L(v) = S$.

Diese Begriffe wollen wir nun anhand der Abbildung 4.5 nochmals verdeutlichen. So stellen hier gerade die Mengen der Layer $\{L_1, L_2, L_3\}$, $\{L_4, L_5\}$ und $\{L_6\}$ die Segmente dieses Diagramms dar und Knoten 1, 2 und 4 bezeichnen deren Segmentknoten. Würde man zusätzlich noch einen Knoten zugleich in die Layer L_3 und L_4 platzieren, so würde sich dieser mit Knoten 2 und 4 schneiden. Diese Zuweisung wäre dabei gemäß der Definition der segmentierten Layerzuweisung ungültig.

Nachdem wir nun die Begriffsbestimmung der Layerzuordnung hin zu einer optisch besseren Schichtung des Graphen weitergehend verbessert haben, gilt es auch in diesem Fall, eine algorithmische Lösung bestimmen, mit welcher man diese in die

Phase der Layerzuteilung sinnvoll integrieren kann. Wir können hier auf die Definition der mehrschichtigen Layerzuteilung aufbauen, da wir für diese bereits ein Verfahren zur Einbettung in das hierarchische Layout entwickelt haben. Da sich eine Layerzuweisung des Graphen, welche diese Definition erfüllt, von der segmentierten lediglich in dem Aspekt unterscheidet, dass in dieser einige Knoten zwecks der Bildung von Segmenten leicht nach links oder rechts verschoben werden, genügt es hier, eine nachgelagerte Phase zu entwickeln, die die Umwandlung einer mehrschichtigen Layerzuteilung in eine segmentierte übernimmt. Einen solchen Ansatz wollen wir im Folgenden vorstellen. Dieser eignet sich dabei zur Anwendung nach der Termination des Longest-Path-, LP-Löser- oder Netzwerk-Simplex-Zuweisers, da dieser die von denen angewandte Layerzuweisungsstrategie respektiert, d.h. alle Senken des Graphen im ersten Layer belassen werden und die Kantenlänge in der Regel nur minimal vergrößert wird. Lediglich beim Einsatz des Coffman-Graham-Ansatzes muss dieser dahingehend modifiziert werden, dass durch das nachträgliche Verschieben einiger Knoten keinem Layer mehr als die vordefinierten ω Knoten zugeteilt werden. Eine solche Anpassung lässt sich jedoch relativ intuitiv durchführen.

Algorithmus 4.2: Korrektur der Layerzuweisung

```
1  Prozedur segmentiereZuweisung(geschichteter Graph G = (V, E)
2                  mit Layer L = {L_1 ... L_n})
3      i = 1;
4      Während i < n tue
5          Bestimme ehemals breitesten Knoten v in L_i;
6          Für alle Knoten w ∈ L_i ... L_{i+v.Breite-1} tue
7              Falls v schneidet w tue
8                  Platziere w in L_{v.Breite};
9                  aktualisiereNachbarn(w,v.Breite);
10         i = i + v.Breite
11 Ende

13 Prozedur aktualisiereNachbarn(v: Knoten, s: Index eines Layers aus {1 ... n}
14     Für alle Knoten w ∈ δ⁻(v) tue
15         i = Index des zu w zugewiesenen Layers;
16         Platziere w in L_{max{i,s}};
17         aktualisiereNachbarn(w,s + 1);
18 Ende
```

Der Algorithmus traversiert alle Layer von rechts nach links, d.h. beginnt bei demjenigen mit dem niedrigsten Index. Die Prozedur bestimmt dabei für einen Layer L_k zunächst sämtliche Knoten, die mit einem ihrer Subknoten diesem zugeordnet wurde, und wählt unter diesen denjenigen aus, der die größte Breite besitzt. Dieser stellt dabei ein Segmentknoten des späteren Schaubilds dar und dessen zugeteilte Layer bilden ein Segment des Graphen. Jeder Knoten muss nun bezüglich seiner zugeordneten Layer entweder vollständig in diesem enthalten oder disjunkt sein. Jeder Knoten, der diese Eigenschaft verletzt, wird dabei soweit nach links verschoben, bis dieser keinen gemeinsamen Layer mehr mit dem Segmentknoten besitzt, wobei

ebenfalls sämtlichen Knoten, welche zu diesem über eine eingehende Kante adjazent sind, weitergehend in Layer mit höheren Indices platziert werden müssen, um nicht bereits die ursprüngliche Definition der Layerzuweisung zu verletzen, welche besagt, dass der Zielknoten einer jeden Kante zu einem Layer mit niedrigerem Index zugewiesen werden muss als sein Quellknoten. Diese Aufgabe übernimmt dabei die *aktualisiereNachbarn()*-Prozedur, welche mittels einer Tiefensuche all diese Knoten durchwandert und gegebenenfalls in einen gültigen Layer verschiebt. Dieser Vorgang wiederholt sich dabei zyklisch solange, bis alle Layer des Graphen traversiert wurden und somit eine auch nach den Regeln der segmentierten Layerzuteilung gültige Zuordnung ermittelt wurde. Als Laufzeit dieses Algorithmus lässt sich eine quadratische Ausführungszeit festhalten, da für jeden Knoten, der in der *segmentiereZuweisung()*-Funktion verschoben wird, in der *aktualisiereNachbarn()*-Prozedur eine Tiefensuche über alle Knoten ausgeführt wird, die zu diesem über eine eingehende Kante adjazent sind.

4.3 Optische Gegenüberstellung

An dieser Stelle wollen wir nochmals die unterschiedlichen Effekte, welche die Anwendung der drei bisher betrachteten Definitionen der Layerzuweisung auf einen gegebenen Graphen ausüben, anhand eines Beispielgraphen untersuchen, welcher mit der Software des KIELER-Projekts[1] erzeugt und mittels unseren Implementierungen geschichtet wurde. Als Basis-Layerzuweiser wurde hier jeweils der Netzwerk-Simplex-Algorithmus verwendet.

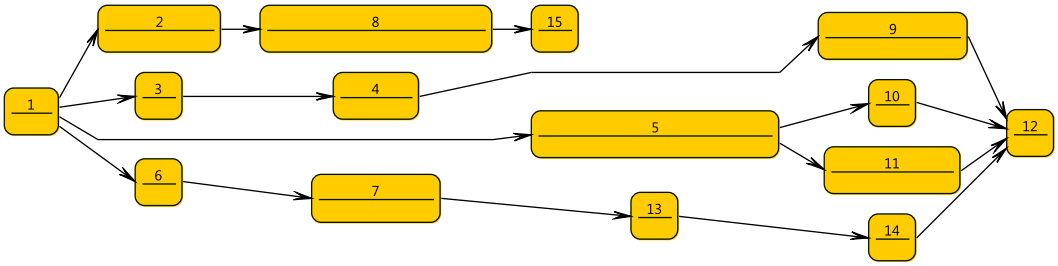


Abbildung 4.6: Ein Layout gemäß der ursprunglichen Layerzuweisungsdefinition

Abbildung 4.6 zeigt das typische Bild eines Graphen, welcher unter Verwendung der ursprünglichen Definition der Layerzuordnung visualisiert wurde. Die Breite eines Layers ist hier durch die Breite des längsten ihm zugewiesenen Knotens gegeben, wobei sämtliche kürzeren Knoten des Layers mittig in diesem platziert wurden. Aufgrund der in diesem Beispiel sehr auffälligen Größenunterschiede der Knoten sind dabei sehr lange Kanten entstanden und die Zeichenfläche wurde, erkennbar an dem vielen frei gebliebenen weißen Zwischenraum, nur sehr spärlich ausgenutzt. Als ein

[1] siehe Abschnitt 1.4

weiterer ästhetischer Makel dieser Darstellung ist das schlecht ausgewogene Verhältnis zwischen Höhe und Breite anzusehen, welche das Diagramm relativ unkompakt erscheinen lässt. So besitzt die Darstellung eine etwa viermal so große horizontale wie vertikale Ausdehnung.

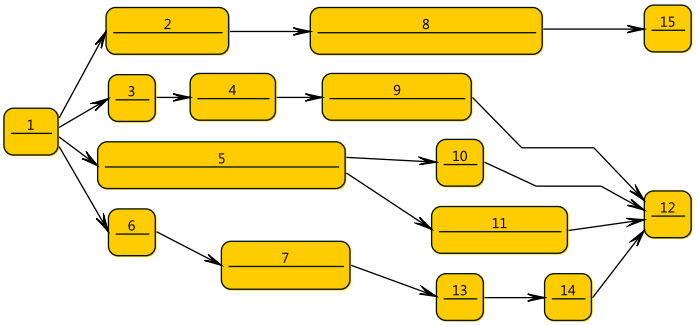


Abbildung 4.7: Ein Layout gemäß der mehrschichtigen Layerzuweisung

In Abbildung 4.7 ist der selbe Graph aus Abbildung 4.6 zu betrachten, welcher hier unter Anwendung der mehrschichtigen Layerzuweisung visualisiert wurde. Diese erlaubt die Zuordnung eines Knotens zu mehreren Layern, wodurch die Breite des Schaubilds erkennbar verkleinert und die Gesamtlänge der Kanten verringert wurde. Bei dieser Darstellung ist jedoch eine eindeutige Schichtung des Graphen nicht zu erkennen. So wirkt gerade der mittlere Abschnitt des Diagramms als ungeordnet und dessen Knoten als beliebig platziert.

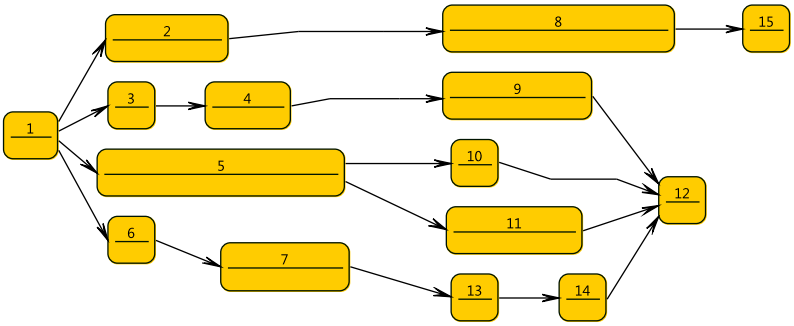


Abbildung 4.8: Ein Layout gemäß der segmentierten Layerzuweisung

Abbildung 4.8 zeigt das Layout des selben Graphen, welcher mittels der segmentierten Layerzuordnung geschichtet wurde. In diesem erkennt man vier deutlich voneinander unterscheidbaren Segmente, welche sich durch die zugehörigen Segmentknoten 1, 5, 8 und 15 identifizieren lassen. Die Gesamtkantenlänge wurde dabei zu Gunsten der Schichtungsdarstellung nur minimal verlängert und die Breite des Layouts ist annähernd identisch mit der aus Abbildung 4.7.

5 Zusammenfassung und Ausblick

Die Phase der Layerzuweisung stellt einen, im Bereich des Layouts von auf Graphen basierenden Diagrammen, hoch interessanten Forschungszweig dar. Zwar wurden für diese Phase des hierarchischen Ansatzes bereits unterschiedliche Ansätze und Algorithmen entwickelt, jedoch besitzen diese häufig noch markante Schwächen, die oftmals in der mangelnden gleichzeitigen Optimierbarkeit aller geforderten ästhetischen Kriterien begründet liegen. Der Longest-Path-Layerzuweiser beispielsweise minimiert zwar die Breite eines Layouts, jedoch ist die Höhe der resultierenden Layerzuordnung häufig keineswegs optimal. Auf der anderen Seite begrenzt zwar der Coffman-Graham-Algorithmus diese vertikale Ausdehnung des Schaubilds, aber dessen Breite artet dabei oft sehr stark aus. Eine optimale Lösung wäre hier sicherlich das Finden eines Ansatzes, welches das Produkt aus Höhe und Breite minimiert, jedoch ist die Bestimmung eines solchen Layouts NP-vollständig [4]. Die Ergebnisse des Netzwerk-Simplex-Ansatz sehen hier zwar vielversprechend aus, da eine minimale Kantenlänge eine Grundvoraussetzung für die Kompaktheit des Layouts darstellt und diese bereits eine Layerzuweisung von geringster Breite impliziert [9], jedoch garantiert dieser noch lange keine Layerzuweisung von minimalen Gesamtmaßen. Weitere Entwicklungsarbeit ist in diesem Bereich also dringend nötig, um die optische Qualität der Visualisierung noch weiter zu verbessern.
Mit der Layerzuweisung besonders großer Knoten haben wir ein Thema angeschnitten, welches so bislang in der Literatur noch nicht behandelt wurde. Dabei erscheint die bisherige Vernachlässigung dieses Aspekts doch etwas unverständlich, denn die Schwächen des aktuellen Ansatzes bei der Layerzuweisung von Knoten differierender Breite sollten es eigentlich für jeden relativ offensichtlich machen, dass hier noch Nachholbedarf besteht. Mit unseren Ausführungen haben wir hier jedoch gerademal an der Oberfläche dessen gekratzt, was hier noch an Entwicklungspotential möglich ist. Löst man sich beispielsweise von der Vorstellung, die bisher bekannten Layerzuweiser auch für die Zuteilung unterschiedlich großer Knoten zu verwenden und diese lediglich mittels einer Vor- oder Nachbearbeitungsphase für verschieden breite Knoten gemäß unserer Neudefinition der Layerzuweisung kompatibel zu machen, so sind in Zukunft sicherlich noch andere, deutlich bessere Layerzuordnungsansätze zu erwarten. Darüber hinaus haben wir uns auch nur mit einer einzelnen Facette des Layouts verschieden großer Knoten beschäftigt. Die Frage nach der sinnvollen Integration von Knoten differierender Höhe ist hier eine ganz andere. Hier müsste man auch zunächst untersuchen, ob, respektive inwiefern man diese Thematik in die bestehenden Ansätze integrieren kann und welche Änderungen an den bisherigen Algorithmen dafür erforderlich sind. Als erste Idee könnte man zum Beispiel den Coffman-Graham-Algorithmus so modifizieren, dass dieser nicht mehr lediglich die

simple Anzahl der Knoten in einem Layer beschränkt, sondern auch dessen Gesamthöhe, zum Beispiel in Zentimetern oder Pixel, berücksichtigt. Aber auch hier gibt es sicherlich noch bessere Verfahren.
Wie man also erkennt, ist das Entwicklungspotential der Layerzuweisung und insbesondere des hierarchischen Layouts an sich noch lange nicht ausgeschöpft. Auch trotz des enormen Fortschritts, den diese Phase seit der ursprünglichen Erwähnung in Sugiyama *et al.* [13] genommen hat, unterliegt dieser Ansatz immer noch einem Prozess fortwährender Forschung und Erweiterung. Inwiefern sich dieses Potential noch in konkrete Ergebnisse manifestieren wird, bleibt dabei noch abzuwarten.

Literaturverzeichnis

[1] Vasek Chvatal. *Linear programming.* Freeman, New York, 1983.

[2] E.G. Coffman and R.L. Graham. Optimal scheduling for two-processor systems. In *Acta Informatica 1*, pages 200–213. Springer-Verlag, 1972.

[3] W. H. Cunningham. A network simplex method. In *Mathematical Programming 11*, pages 105–116. North-Holland Publishing Company, 1976.

[4] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, 1998.

[5] Markus Eiglsperger, Martin Siebenhaller, and Michael Kaufmann. An efficient implementation of Sugiyama's algorithm for layered graph drawing. In János Pach, editor, *GD 2004: 12th International Symposium on Graph Drawing*, volume 3383 of *LNCS*, pages 155–166. Springer-Verlag, 2004.

[6] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A technique for drawing directed graphs. *Software Engineering*, 19(3):214–230, 1993.

[7] Patrick Healy and Nikola Nikolov. How to layer a directed acyclic graph. In *Graph Drawing (GD 2001), LNCS 2265*, pages 16–30. Springer-Verlag, 2002.

[8] Seok-Hee Hong and Nikola S. Nikolov. Layered drawings of directed graphs in three dimensions. In *Information Visualisation 2005: Asia-Pacific Symposium on Information Visualisation (APVIS2005)*, pages 69–74, 2005.

[9] Michael Kaufmann and Dorothea Wagner, editors. *Drawing Graphs: Methods and Models.* Number 2025 in LNCS. Springer-Verlag, Berlin, Germany, 2001.

[10] Nikola S. Nikolov and Alexandre Tarassov. Graph layering by promotion of nodes. In *Discrete Applied Mathematics 154*, pages 848–860, 2006.

[11] Georg Sander. Layout of compound directed graphs. Technical Report A/03/96, Universität des Saarlandes, FB 14 Informatik, 66041 Saarbrücken, June 1996.

[12] Miro Spönemann. On the automatic layout of data flow diagrams. Diploma thesis, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, March 2009. `http://rtsys.informatik.uni-kiel.de/~biblio/downloads/theses/msp-dt.pdf`.

[13] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, February 1981.

[14] Alexandre Tarassov, Nikola S. Nikolov, and Jürgen Branke. A heuristic for minimum-width graph layering with consideration of dummy nodes. In *Proceedings of the Third International Workshop on Efficient and Experimental Algorithms (WEA 2004), LNCS 3059*, pages 570–583. Springer-Verlag, may 2004.