Platzieren von Zusammenhangskomponenten mit hierarchischen Ports

Daniel Jahn

Masterarbeit
Dezember 2017

Echtzeitsysteme und Eingebettete Systeme
Prof. Dr. Reinhard von Hanxleden
Institut für Informatik
Christian-Albrechts-Universität zu Kiel

Betreut durch
Dipl.-Inf. Christoph Daniel Schulze

Selbstständigkeitserklärung

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

Kiel,

Zusammenfassung

Datenflussdiagramme sind ein weit verbreitetes Werkzeug, um Abläufe von Systemen zu modellieren. Das Platzieren von Objekten muss dabei sorgfältig überlegt sein, damit der Betrachter das Modell leichter verstehen kann.

Für das Projekt Eclipse Layout Kernel (ELK) wird im Rahmen dieser Arbeit ein Problem gelöst, welches die Lesbarkeit von Diagrammen mit Zusammenhangskomponenten und hierarchischen Ports verbessern soll. Die üblichen Algorithmen für die Ebenenzuweisung sind nicht dafür optimiert, Zusammenhangskomponenten eines hierarchischen Knotens bei den Porteinschränkungen Fixed Order und Fixed Position zu berücksichtigen.

Um das Problem zu lösen, werden zwei Phasen des ebenenbasierten Layoutansatzes verändert. Die erste Phase ist die Ebenenzuweisung, welche dafür sorgen soll, dass nicht zu viele Knoten in eine Ebene gepackt und nicht zu viele Ebenen für einen Graphen erstellt werden. Als Zweites wird die Kreuzungsminimierung angepasst, um die Anzahl von Kreuzungen zwischen Zusammenhangskomponenten zu reduzieren, weil dadurch die Lesbarkeit eines Graphen verbessert werden kann.

Die Ergebnisse des entwickelten Ansatzes zeigen, dass die Graphen noch nicht kompakt genug erstellt werden, was aber durch vorgeschlagene Lösungen weiter verbessert werden kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungen

KIELER Kiel Integrated Environment for Layout Eclipse RichClient

ELK Eclipse Layout Kernel

KIML KIELER Infrastructure for Meta Layout

KLighD KIELER Lightweight Diagrams

SCCharts Sequentially Constructive Charts

GrAna Graph Analysis

KiCo KIELER Compiler

IDE Entwicklungsumgebung

FAS Feedback Arc Set

Kapitel 1

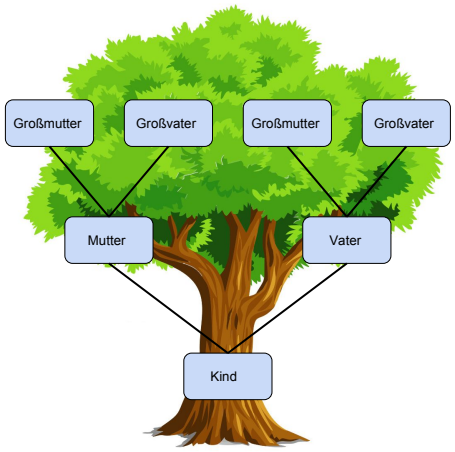
Einleitung

Graphen werden oft genutzt, um bestimmte Beziehungen und Informationen zwischen Objekten zu modellieren. Für ein besseres Verständnis wird meist vorausgesetzt, dass der Graph visuell repräsentiert wird. Beispiele von Graphen sind Stammbäume, Freundschaftsbeziehungen in sozialen Netzwerken, eingebettete Systeme, die mit Hilfe von Sequentially Constructive Charts (SCCharts) [HDM+13] erstellt werden können, oder Datenflussdiagramme, welche den Ablauf komplexer Softwaresysteme beschreiben. In Abbildung 1.1 wird jeweils eine visuelle Repräsentation für die vier eben genannten Graphen gezeigt. Um einen Graphen darstellen zu können, kann zum Beispiel das automatische Graphenzeichnen genutzt werden, welches das Problem behandelt, geometrische Repräsentationen von Graphen automatisch zu erstellen [DET+99]. Der Grund dafür, dass Diagramme benutzt werden, ist, dass sie vermeintlich besser lesbar sind als textuelle Programme [Kla12]. Allerdings ist dies nur der Fall, wenn Diagramme bestimmte Ästhetikkriterien erfüllen, welche die Lesbarkeit erhöhen [Pur97].

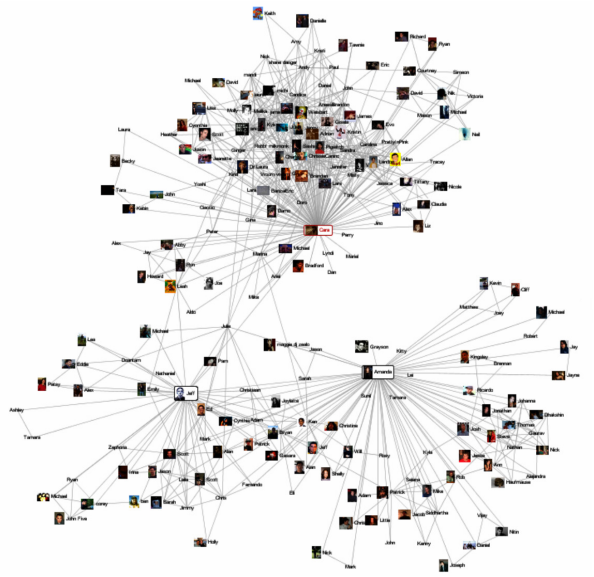
Für das Erstellen von Diagrammen gibt es bislang zwei Alternativen: Entweder werden die Diagramme mit Hilfe eines *Drag & Drop*-Editors erstellt oder es kann eine textuelle Sprache verwendet werden. Bei der ersten Variante werden Elemente für den Graphen aus einer Werkzeugkiste per Maus ausgewählt und in die vorgegebene Zeichenfläche gezogen. Für das Hinzufügen neuer Elemente in ein bereits vorhandenes Diagramm ist es meist notwendig, manuell Platz für das neue Element zu schaffen, was bereits bei kleinen Diagrammen zeitaufwändig sein kann. Jeder, der bereits einen Drag & Drop-Editor verwendet oder anderweitig Grafiken am Computer erstellt hat, kennt das Problem, dass man viel Zeit investiert, um die Elemente im Diagramm ordentlich zueinander auszurichten, damit das Diagramm ansprechend und nachvollziehbar für andere Betrachter ist. Ein weiterer zeitlicher Faktor bei dieser Variante ist die Handhabung des Editors. Die zu verwendeten Elemente müssen zunächst in der Werkzeugkiste gefunden werden. Durch einen benutzerunfreundlichen Aufbau kann viel Zeit verloren gehen, wenn zum Beispiel durch viele Auswahlmenüs geklickt werden muss, um ein bestimmtes Element zu finden.

Beim Ansatz der textuellen Sprache wird ein Graph mit Hilfe von der Sprache zur Verfügung gestellter Schlüsselwörter textuell beschrieben. Durch einen Layoutalgorithmus wird der Graph aus der textuellen Beschreibung danach automatisch visualisiert. Hierfür muss die textuelle Sprache erlernt werden. Die meisten Editoren bieten eine Autovervollständigung, die dem Anwender die als nächstes gültigen Schlüsselwörter vorschlägt und ein schnelleres Schreiben ermöglicht.

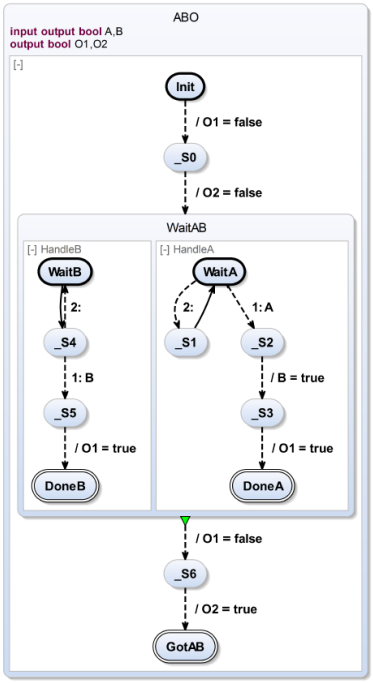
Fuhrmann et al. [FH10] haben eine Studie über automatisches Layout von *SyncCharts*, welches der Vorgänger von SCCharts ist, durchgeführt. Dabei wurden 30 Probanden in drei



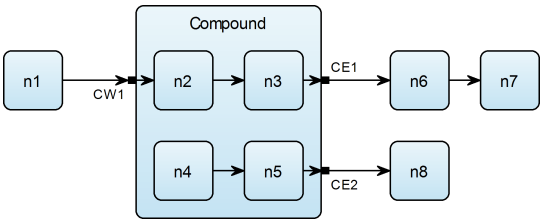
(a) Stammbaum, basierend auf [All].



(b) Freundschaftsbeziehungen im sozialen Netzwerk. Zu sehen in [HB05].



(c) SCChart, zu sehen in [HDM+13].



(d) Datenflussdiagramm

Abbildung 1.1. Vier Beispiele für Graphen.

Gruppen eingeteilt. Eine Gruppe war mit der Syntax von SyncCharts vertraut, hatte aber nur wenig Erfahrungen mit graphischen Editoren; die zweite Gruppe hatte an einem praktischen Kurs teilgenommen; die letzte Gruppe bestand aus Mitarbeitern, die Kenntnisse für beide Ansätze hatten. Die Aufgabe bestand darin, jeweils einen SyncChart mit Hilfe von erstens einem Drag & Drop-Editor mit manuellem Layout, zweitens einem Drag & Drop-Editor mit automatischem Layout und drittens mit Hilfe einer textuellen Sprache und automatischem Layout zu erstellen. Die Editierungszeit von SyncCharts hatte sich bei der zweiten Aufgabe um durchschnittlich 33% im Vergleich zur ersten verbessert. Durch das Nutzen des textuellen Editors in Aufgabe drei wurde diese Zeit sogar um weitere 15% verbessert.

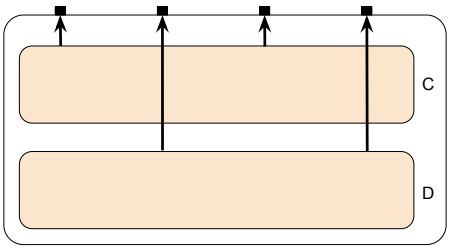
Weiterhin hat Klauske [Kla12] in seiner Arbeit das Bearbeiten von *Simulink* Modellen untersucht. Hier lag der Anteil des Aufwandes für das manuelle Layout basierend auf Erfahrungen zwar nur bei 25%, was aber trotzdem eine signifikante Zeiteinsparung mit der Hilfe von automatischem Layout erbringen würde.

Zu beachten ist, dass ich keine Studie gefunden habe, die analysiert, ob das Verwenden von automatischem Layout bei Datenflussdiagrammen einen zeitlichen Vorteil gegenüber einem Drag & Drop-Editor bietet. Die zwei eben genannten Beobachtungen für SyncCharts und Simulink Modelle liefern aber ein gutes Indiz dafür, dass dies auch bei Datenflussdiagrammen der Fall sein kann. Allgemein gesagt ist das automatische Graphenzeichnen ein wichtiges Forschungsgebiet, welches durchaus Vorteile gegenüber einem manuellen Layout erzielt. Der Anwender muss sich hier keine Gedanken mehr über die visuelle Repräsentation des Graphen machen und ihm steht somit mehr Zeit für die eigentliche Bedeutung des Diagramms zur Verfügung.

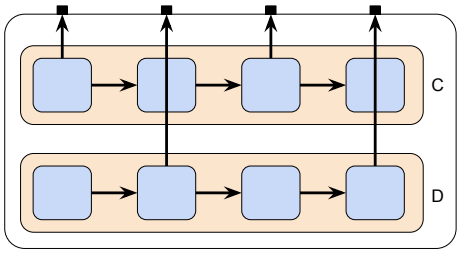
1.1. Problemstellung

In der Arbeitsgruppe *Echtzeitsysteme und Eingebettete Systeme* wird das Projekt Kiel Integrated Environment for Layout Eclipse RichClient (KIELER) unter anderem dafür genutzt, ein solches automatisches Layout zu erzeugen. Mit Hilfe von Plugins in der Entwicklungsumgebung *Eclipse* werden verschiedene Layoutalgorithmen als Open Source-Software zur Verfügung gestellt. Je nach Benutzeranforderung kann der jeweils passende Algorithmus mit diversen Optionen gewählt werden. Die Beispielgraphen aus Abbildung 1.1 unterscheiden sich in ihrer Darstellungsform und weisen auch strukturelle Unterschiede auf, weshalb zum Teil verschiedene Layoutalgorithmen genutzt werden müssen. Was die Graphen aber alle gemeinsam haben ist, dass sie aus Knoten bestehen, die üblicherweise als Rechteck oder Kreis repräsentiert werden. Diese Knoten können durch Kanten miteinander verbunden sein.

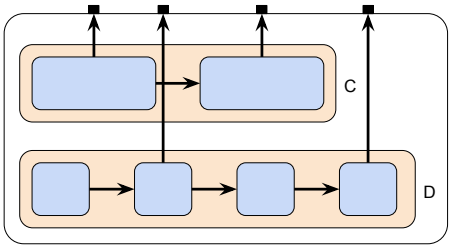
Ein Stammbaum wird mit einem Baumlayouter von zum Beispiel Reingold et al. [RT81] erstellt. Ein Baum beginnt mit der sogenannten Wurzel und wird im Verlauf immer breiter und größer, was die Äste und Blätter repräsentiert. Freundschaftsbeziehungen in sozialen Netzwerken können mit Hilfe eines kräftebasierten Layouts dargestellt werden. Hierbei werden abstoßende und anziehende Kräfte zwischen Knoten berechnet und diese dementsprechend zusammen- oder auseinander gezogen [FR91].



(a) Abstrakte Darstellung der Zusammenhangskomponenten. Basierend auf einem unpublizierten Diagramm von Schulze.



(b) Innerhalb befinden sich Knoten, die gekreuzt werden könnten.



(c) Es können auch zu kleine Abstände zwischen Knoten und Kanten entstehen.

Abbildung 1.2. Probleme bei der Platzierung von Zusammenhangskomponenten.

SCCharts und Datenflussdiagramme nutzen den ebenenbasierten Ansatz von Sugiyama et al. [STT81], welcher in dieser Arbeit im Fokus steht. Hierbei werden Knoten in sogenannte Ebenen unterteilt und der Lesefluss verläuft von links nach rechts. Die anderen beiden Algorithmen eignen sich hierfür nicht, da sie entweder eine Wurzel voraussetzen oder keinen guten Lesefluss bieten, der in diesem Bereich sehr wichtig ist, weil das Verfolgen von Signalen oder Kontrollflüssen im Vordergrund steht.

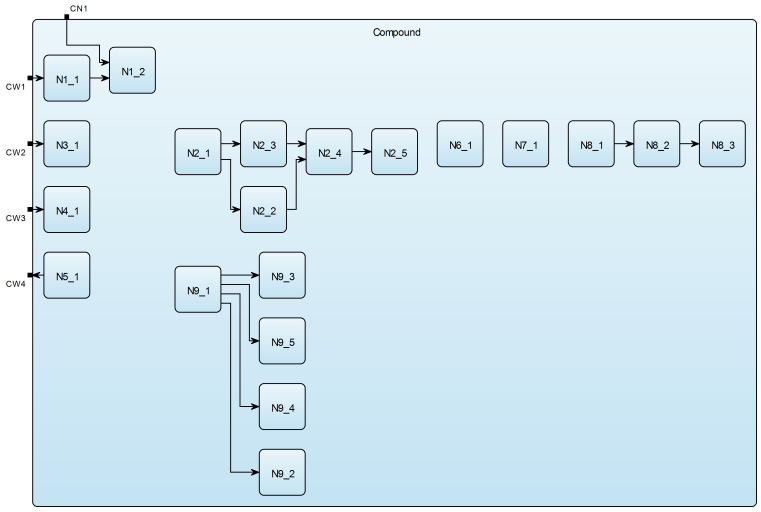
Für das weitere Verständnis sind ein paar Begrifflichkeiten notwendig. Schauen wir uns dafür noch einmal Abbildung 1.1d an. Der Knoten *Compound* beinhaltet weitere Knoten $n2$, $n3$, $n4$ und $n5$ und wird deshalb als *hierarchischer Knoten* bezeichnet. Innerhalb des hierarchischen Knotens ist $n2$ mit $n3$ verbunden, aber nicht mit $n4$ und $n5$. Alle miteinander verbundenen Knoten werden als *Zusammenhangskomponente* bezeichnet. Die Zusammenhangskomponenten sind in dem Beispiel über Ports $CW1$, $CE1$ und $CE2$ mit dem restlichen Graphen verbunden. Diese Ports können verschiedene Einschränkungen haben. Zunächst genügen uns die Einschränkungen Fixed Side und Fixed Order. Bei Fixed Side müssen die Ports auf einer vorgegebenen Seite des hierarchischen Knotens platziert werden, aber ihre Reihenfolge darf vertauscht werden. Bei Fixed Order darf zusätzlich die Reihenfolge nicht vertauscht werden.

Als nächstes betrachten wir Abbildung 1.2a. Hier sehen wir zwei Zusammenhangskomponenten *C* und *D*, wobei der Inhalt zunächst ausgeblendet ist, denn in dem Algorithmus, der die Einschränkung Fixed Side behandelt, werden die Zusammenhangskomponenten als *"black box"* betrachtet. Der Algorithmus hat also keine Informationen darüber, wo Knoten und Kanten innerhalb dieser Box platziert sind. Wenn wir nun die Einschränkung Fixed Order verwenden, dann haben wir die Möglichkeit, entweder die Kanten zu den Ports um die Zusammenhangskomponente C herum zu platzieren, was den Lesefluss und die Verständlichkeit des Diagramms stark beeinflussen würde, oder durch die Zusammenhangskomponente selbst hindurch zu platzieren. Bei Letzterem können, wie in Abbildung 1.2b zu sehen ist, Kreuzungen zwischen Knoten und Kanten entstehen, da dem Layoutalgorithmus die Knoten- und Kantenplatzierungen innerhalb anderer Zusammenhangskomponenten unbekannt sind. Ebenfalls können dadurch, wie in Abbildung 1.2c zu sehen ist, die Abstände zwischen Kanten und Knoten zu klein sein, was die Lesbarkeit des Diagramms ebenfalls einschränkt. Das Verfahren, welches für die Einschränkungen Fixed Side genutzt wird, heißt *Connected Components Processing* und wurde von Schulze et al. [RSG+16] entwickelt. Dieses Verfahren hat eine gute Laufzeit, aber kann auf Grund der eben genannten Probleme nicht für die Einschränkung Fixed Order genutzt werden.

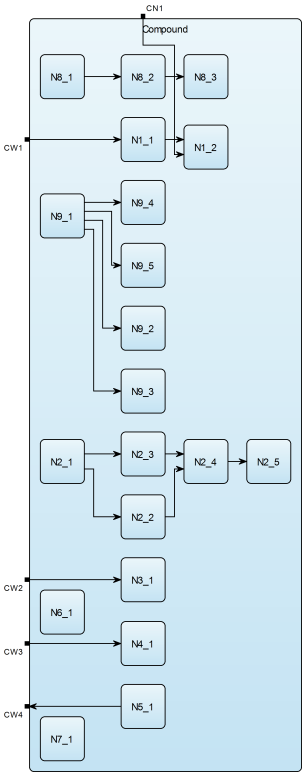
Wenn man einen Graphen mit den Einschränkungen Fixed Order darstellen möchte, wird deshalb ein anderes Verfahren verwendet, welches nicht dafür optimiert ist Zusammenhangskomponenten zu berücksichtigen. In Abbildung 1.3a sehen wir einen Graphen, der die Einschränkung Fixed Side vorgegeben hat, und das Connected Components Processing nutzt. In Abbildung 1.3b ist das aktuelle Ergebnis desselben Graphen mit der Einschränkung Fixed Order dargestellt. Hier sehen wir, dass der Layoutalgorithmus bei einer Vielzahl von Zusammenhangskomponenten dazu tendiert, eine sehr hohe Zeichnung zu erstellen. Ziel meiner Arbeit ist es also, den Layoutalgorithmus so anzupassen, dass er Zusammenhangskomponenten mit Porteinschränkungen, die eine feste Reihenfolge vorgeben, besser behandeln kann. Das Ergebnis des Beispielgraphen sollte dann eine Darstellung wie in Abbildung 1.3c sein. Hier sehen wir, dass die Darstellung bei einer wachsenden Zahl an Zusammenhangskomponenten nicht einfach in die Höhe wächst.

1.2. Aufbau der Arbeit

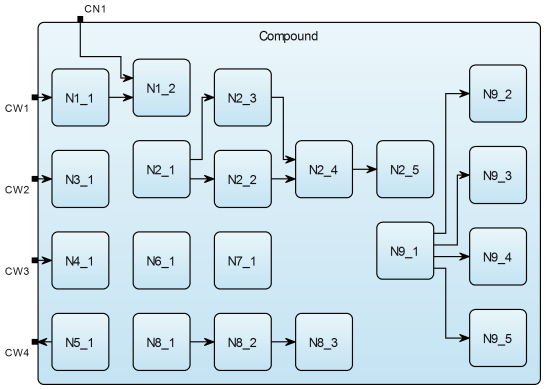
Zunächst werden in Kapitel 2 die Grundlagen dieser Arbeit erläutert. Dazu gehören unter anderem graphentheoretische Begriffe, die wichtig für das Verständnis der präsentierten Lösungen sind. Ebenfalls werden in dem Kapitel die zugrundeliegenden genutzten Technologien erläutert, welche diese Arbeit überhaupt erst ermöglicht haben. Kapitel 3 beschreibt verwandte Arbeiten für die beiden Hauptkapitel. Das Kapitel 4 ist das erste Hauptkapitel. Der ebenenbasierte Ansatz ist in insgesamt fünf Phasen aufgeteilt. In meiner Arbeit müssen zwei Phasen angepasst werden, um das eben vorgestellte Problem zu lösen. Dieses Kapitel behandelt also die erste Phase, die angepasst werden muss. Zunächst wird erklärt, warum die im vorherigen Kapitel vorgestellten verwandten Arbeiten für mein Problem nicht nutzbar



(a) Beispiel mit FIXED SIDE Einschränkung.



(b) Beispiel mit FIXED ORDER Einschränkung.



(c) Beispiel mit FIXED ORDER Einschränkung und Berücksichtigung von Zusammenhangskomponenten.

Abbildung 1.3. Probleme der aktuellen Implementierung von der Einschränkung FIXED ORDER.

sind. Danach wird ein anderer Ansatz vorgestellt, um Knoten bei den problematischen Porteinschränkungen besser in Ebenen einzuteilen. In Kapitel 5 wird die zweite Phase vorgestellt, die verändert werden muss. Diese Phase kümmert sich darum, Kreuzungen von Kanten, die zu einem Port führen, und Kanten anderer Zusammenhangskomponenten zu reduzieren. Das sorgt dafür, dass das Diagramm für den Betrachter leichter zu verstehen ist. Kapitel 6 vergleicht die in dieser Arbeit vorgestellte Lösung mit den bisherigen Ansätzen im Hinblick auf mehrere Kriterien. Als letztes wird in Kapitel 7 eine Zusammenfassung meiner Arbeit präsentiert und es werden weiterführende Arbeiten vorgestellt, die zu weiteren Verbesserungen meines Ansatzes führen könnten.

Kapitel 2

Grundlagen

In diesem Kapitel werden zunächst alle wichtigen Begriffe erläutert, die für das Verständnis meines Themas nötig sind. Diese Definitionen werden in Abbildung 2.1 veranschaulicht. Danach werden Ästhetikkriterien vorgestellt, deren Einhalten die Lesbarkeit von Graphen für den Betrachter erhöhen sollen. Als Letztes werden verwendete Technologien vorgestellt, durch die meine Arbeit überhaupt erst möglich geworden ist. Manche Definitionen nehmen Bezug auf das Projekt Eclipse Layout Kernel (ELK), um zu erklären, wie sie in dem Projekt behandelt werden. ELK ist ein Projekt, welches Layoutalgorithmen enthält und für dieses Projekt wurde der ebenenbasierte Ansatz erweitert. Eine genauere Definition von ELK folgt in Abschnitt 2.4.2.

2.1. Definitionen

Für die meisten Definitionen wurde das Buch von Battista et al. [DET+99] als Vorlage verwendet. Ansonsten orientieren sich die Definitionen an denen aus der Vorlesung Graphenzeichnen der Christian-Albrechts Universität zu Kiel.

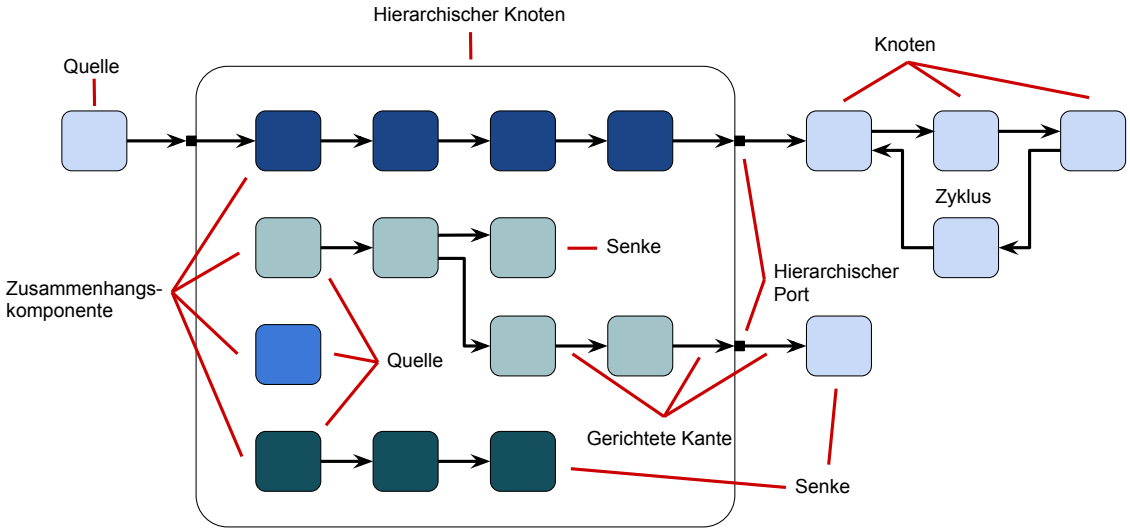


Abbildung 2.1. Alle für die Arbeit relevanten Elemente eines Graphen. Die Zusammenhangskomponenten sind jeweils unterschiedlich gefärbt.

2.1. Definition. Ein *gerichteter Graph* $G = (V, E)$ besteht aus einer endlichen Menge an Knoten $V = \{v_1, ..., v_n\}$ und einer Menge an gerichteten Kanten $E = \{e_1, ..., e_m\}$. Alle Kanten haben dabei eine zugewiesene Richtung. Das bedeutet, dass $e = (u, v) \in VxV$ ein geordnetes Paar bildet. Dabei ist e die ausgehende Kante von u und die eingehende Kante von v.

Bei einem gerichteten Graphen wird ein Knoten, der nur eingehende Kanten hat, als *Quelle* (engl. *source*) und ein Knoten, der nur ausgehende Kanten hat, als *Senke* (engl. *sink*) bezeichnet.

2.2. Definition. Ein gerichteter *Pfad* der Länge n in einem gerichteten Graphen $G = (V, E)$ ist eine Sequenz $(v_1, ..., v_n)$ von Knoten in G, wobei $(v_i, v_{i+1}) \in E$ für $1 \leqslant i < n$. Das bedeutet also, dass ein Knoten v_n über eine oder mehrere gerichtete Kanten aus dem Graphen von einem anderen Knoten v_1 erreichbar ist.

Bei einem *ungerichteten* Pfad, können die Kanten in beide Richtungen durchlaufen werden.

2.3. Definition. Ein Graph enthält einen *Zyklus*, wenn es einen gerichteten Pfad mit $v_n = v_1$ gibt, und heißt *azyklisch*, wenn er keinen Zyklus enthält.

2.4. Definition. Als *hierarchischer Knoten* wird ein Knoten bezeichnet, der in sich selbst einen weiteren Graphen enthält.

2.5. Definition. Ein Graph $G = (V, E)$ wird *zusammenhängender Graph* genannt, falls alle Knoten v über einen ungerichteten Pfad erreichbar sind. Falls G kein zusammenhängender Graph ist, dann besteht G aus mehreren zusammenhängenden Graphen. Jeder dieser zusammenhängenden Graphen wird auch als *Zusammenhangskomponente* bezeichnet.

2.6. Definition. Die ursprüngliche Definition eines Graphen beinhaltet keine Ports. Diese wurden als Erweiterung hinzugefügt, weil sie wichtig für Datenflussdiagramme sind. Jedem Knoten v werden Ports auf seinem äußeren Rand zugewiesen und diese Ports dienen als Start- und Endpunkte für Kanten. Dabei gibt es vier Portseiten. Die obere Portseite wird *Norden* genannt und dem Uhrzeigersinn folgend wird die rechte Seite *Osten*, die untere Seite *Süden* und die linke Seite *Westen* genannt.

Sobald wir Ports an einen hierarchischen Knoten platzieren, nennen wir diese Ports auch *hierarchische Ports*. Diese Ports dienen als Verbindung von Zusammenhangskomponenten zu anderen Knoten außerhalb des hierarchischen Knotens. In ELK ist es nicht verpflichtend, normale Ports zu nutzen. Intern hat jede Kante als Start- und Endpunkt einen Port, aber nur für hierarchische Knoten muss explizit ein Port für die Kantenverbindung genutzt werden.

2.7. Definition. Für jeden Knoten kann für seine Ports eine von fünf Porteinschränkungen als Option gesetzt werden. Im Folgenden werden diese Porteinschränkungen erläutert:

FREE Bei der Wahl dieser Einschränkung können Ports auf einer beliebigen Seite des hierarchischen Knotens platziert werden. Auch die Reihenfolge der Ports auf den jeweiligen Seiten ist frei wählbar.

FIXED SIDE Hier wird für jeden Port eine feste Seite vorgegeben, aber die Reihenfolge der Ports auf den jeweiligen Seiten kann beliebig gewählt werden.

FIXED ORDER Bei dieser Einschränkung ist sowohl die Seite der Ports, als auch deren Reihenfolge festgelegt.

FIXED POSITION Hier ist die Position der Ports auf dem Rand des Knotens fest vorgegeben.

FIXED RATIO Wenn diese Option gewählt ist, werden die Ports in einem festen Seitenverhältnis zum zugehörigen Knoten platziert. Das bedeutet, dass die Position der Ports abhängig von der Größe des Knotens ist. Diese Option wird in der Praxis kaum verwendet und dementsprechend in dieser Arbeit auch nicht weiter betrachtet.

2.2. Ebenenbasiertes Layout

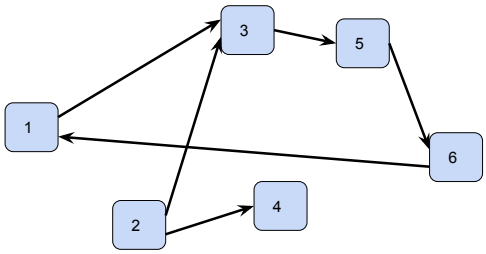
Der ebenenbasierte Layoutansatz hebt Datenflussrichtungen hervor und eignet sich somit für Datenflussdiagramme. In dem Paper von Sugiyama et al. [STT81] werden für das Erstellen einer Darstellung aus dem Graphen drei Phasen genutzt. Der Layoutansatz nimmt an, dass ein Graph azyklisch ist und Kanten gerade gezeichnet werden. In weiteren Arbeiten für dieses Themengebiet wurde dieser Ansatz um zwei zusätzliche Phasen erweitert, welche auch in ELK genutzt werden. Eine Phase sorgt dafür, dass der Graph temporär azyklisch gemacht wird und die andere ermöglicht es, alternative Kantenführungen zu verwenden. Somit besteht der Algorithmus in ELK aus fünf Phasen. Ein weiterer Unterschied bei ELK ist, dass der Datenfluss im ebenenbasierten Ansatz von links nach rechts verläuft, wohingegen er bei Sugiyama von oben nach unten verläuft.

Da die Phasen der Ebenenzuweisung und Kreuzungsminimierung für diese Arbeit am wichtigsten sind, werden die Algorithmen dieser Phasen erst in Kapitel 3 genauer erläutert. Im Folgenden werden die fünf Phasen beschrieben und in Abbildung 2.2 werden die jeweiligen Resultate der einzelnen Phasen veranschaulicht.

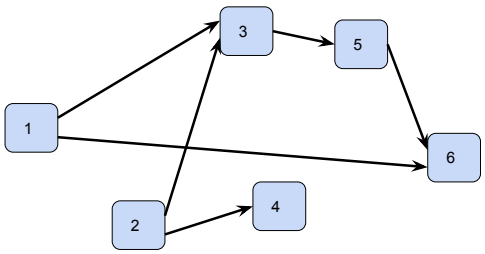
Phase 1: Zyklen brechen

In der ersten Phase erhalten wir als Eingabe einen gerichteten Graphen $G = (V, E)$. In dieser Phase werden mögliche vorhandene Zyklen entfernt und als Ergebnis wird ein gerichteter azyklischer Graph $G' = (V, E')$ zurückgegeben. Es werden keine Kanten entfernt, aber falls der Graph Zyklen enthielt, dann beinhaltet E' nun die entsprechenden umgedrehten Kanten. Diese werden markiert, damit sie am Ende des Algorithmus wieder in die richtige Richtung umgedreht werden können.

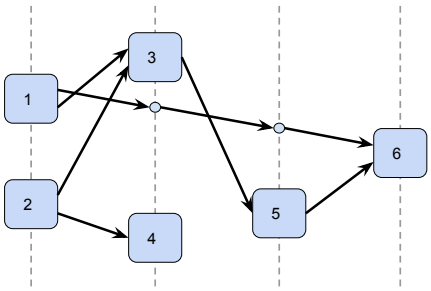
Um dieses Problem zu lösen gibt es das sogenannte Feedback Arc Set (FAS), welches eine Menge von Kanten FAS $\subseteq$ E bezeichnet, deren Umkehrung den Graphen azyklisch macht. Dafür muss als erstes eine lineare Ordnung aller Knoten des Graphen erstellt werden und danach können entweder alle Kanten, die nach links zeigen, oder alle Kanten, die nach rechts zeigen, umgedreht werden. Das Problem hierbei ist, dass je nach gewählter Ordnung zu viele Kanten umgedreht werden könnten. Es muss also die Ordnung gefunden werden, die die wenigsten Kanten umdreht. Dies ist ein *NP-schweres* Problem, weshalb hier auf



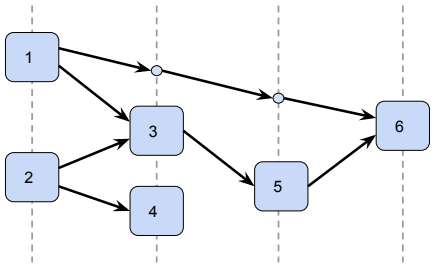
(a) Der Ursprungsgraph.



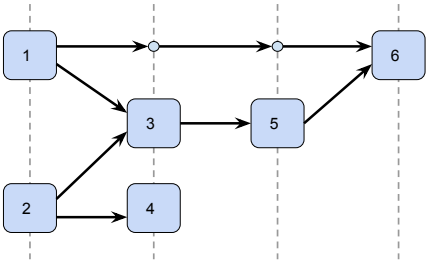
(b) Phase 1: Zyklen brechen. Die Kante von Knoten 6 zu Knoten 1 wurde umgedreht.



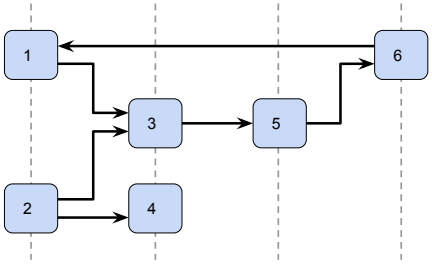
(c) Phase 2: Ebenenzuweisung.



(d) Phase 3: Kreuzungsminimierung.



(e) Phase 4: Knotenplatzierung.



(f) Phase 5: Kantenplatzierung.

Abbildung 2.2. Ergebnisse der fünf Phasen, die durchlaufen werden.

Heuristiken zurückgegriffen wird, damit eine akzeptable Laufzeit erreicht werden kann. Eades et al. [ELS93] haben eine Heuristik entwickelt, mit der dieses Problem in $\mathcal{O}(|E|)$ gelöst wird, und dabei werden höchstens zweimal so viele Kanten umgedreht wie bei der optimalen Lösung.

Phase 2: Ebenenzuweisung

In dieser Phase wird für die Knoten aus V eine Einteilung $L_1, ..., L_n$ in *Ebenen* (engl. *layer*) erstellt, so dass für (u, v) mit $u \in L_i$ und $v \in L_j$ gilt, dass $i < j$ ist. Hierbei beschreibt jedes L_i und jedes L_j eine Ebene.

Eine Zuweisung wird als *ordentliche Zuweisung* bezeichnet, falls alle ausgehenden Kanten eines Knotens ihren Zielknoten in der direkt nachfolgenden Ebene haben.

Als Eingabe für diese Phase wird ein azyklischer gerichteter Graph $G = (V, E)$ erwartet und als Ausgabe erhalten wir einen ordentlich ebenenzugewiesenen Graphen $G' = (V', E', L')$. Da die Ebenenzuweisung einen ordentlich ebenenzugewiesenen Graphen zurückliefern soll, müssen für manche Graphen *Dummyknoten* und *Dummykanten* hinzugefügt werden. In Abbildung 2.2c sehen wir ein Beispiel hierfür. Im späteren Verlauf des Algorithmus werden die Dummyknoten und Dummykanten wieder entfernt und die ursprüngliche lange Kante wiederhergestellt.

Es gibt verschiedene Algorithmen für die Ebenenzuweisung. Der bekannteste und einfachste ist der *Longest Path Layerer* [RDM+87]. Dieser und weitere Algorithmen für diese Phase werden in Kapitel 3 genauer erläutert, da diese Phase ein Kernthema der Arbeit ist.

Die Herausforderung aller Algorithmen dieser Phase besteht darin, nicht zu viele Ebenen zu erzeugen und nicht zu viele Knoten in eine Ebene zu packen, weil dadurch ungünstige Seitenverhältnisse erzeugt werden können. Ein weiteres Problem ist, dass durch die Algorithmen oft zu lange Kanten erzeugt werden. Lange Kanten verringern die Lesbarkeit eines Diagramms und die Ausführungszeit wird durch eine erhöhte Zahl an Dummyknoten und Dummykanten beeinträchtigt. Allgemein gibt es für das Kürzen zu langer Kanten die Möglichkeit, im Nachhinein Algorithmen zum Zusammenschieben des Diagramms zu nutzen.

Phase 3: Kreuzungsminimierung

Die Kreuzungsminimierung erhält als Eingabe einen ordentlich ebenenzugewiesenen Graphen $G = (V, E, L)$ und berechnet für diesen eine Ordnung der Knoten in allen Ebenen, so dass die Zahl der Kantenkreuzungen so klein wie möglich ist.

Die Minimierung der Kantenkreuzungen ist ein NP-schweres Problem, weshalb zum Beispiel die *Layer Sweep*-Heuristik von Sugiyama et al. [STT81] zum Einsatz kommt. Hierbei werden Knoten in benachbarten Ebenen vertauscht, um so die Zahl der Kantenkreuzungen zu verringern. Dies wird in mehreren Durchläufen wiederholt, bis sich die Anzahl der Kreuzungen nicht mehr verändert. Dieser und weitere Algorithmen werden in Kapitel 3 näher erläutert, da diese Phase ein weiteres Kernthema der Arbeit ist.

Weiterhin werden durch die Kreuzungsminimierung auch die Ports so sortiert, dass möglichst wenig Kreuzungen entstehen, sofern die Porteinschränkungen der einzelnen Knoten auf Free oder Fixed Side gesetzt sind.

Phase 4: Knotenplatzierung

Für die Knotenplatzierung erhalten wir als Eingabe einen ebenenzugewiesenen Graphen $G = (V, E, L)$, dessen Knoten in allen Ebenen geordnet sind. Als Ausgabe erhalten wir y-Koordinaten für alle $v \in V$. Weiterhin gilt nach dieser Phase für $u, v \in L$, falls u über v bezüglich der Reihenfolge ist, dass $y_u + h_u < y_v$ gelten muss. Dabei bezeichnet y_u die y-Koordinate und h_u die Höhe des Knotens u. Diese Bedingung stellt also sicher, dass sich zwei Knoten aus derselben Ebene nicht überlappen und ihre Reihenfolge beibehalten.

Für die Knotenplatzierung wird in ELK standardmäßig der *BK node placement* Algorithmus von Brandes und Köpf [BK02] verwendet. Dieser sorgt dafür, dass

- die Reihenfolge der Knoten beibehalten wird,
- lange Kanten möglichst gerade gezeichnet werden,
- maximal zwei Kantenknicke pro Kante entstehen,
- Kanten kurz gehalten werden sollen und
- Knoten balanciert platziert werden.

Die Hauptidee des Algorithmus ist es, vier extreme Layouts zu berechnen und dann einen Mittelwert der extremen Layouts für das finale Layout zu verwenden.

Phase 5: Kantenführung

Bei der Kantenführung erhalten wir als Eingabe einen ebenenzugewiesenen Graphen $G = (V, E, L)$, dessen Knoten eine y-Koordinate zugewiesen bekommen haben. In dieser Phase bekommen alle Knoten $v \in V$ zusätzlich eine x-Koordinate zugewiesen und es wird eine Kantenführung für alle $e \in E$ berechnet. Die x-Koordinaten können erst in dieser Phase berechnet werden, da vorher noch nicht feststeht, wie viel Platz für die Kanten zwischen den Ebenen benötigt wird. In ELK gibt es aktuell drei verschiedene Kantenführungen. Entweder *orthogonale Kanten*, *Splines* oder *Polylines*. In Abbildung 2.3 werden die drei verschiedenen Kantenführungen gezeigt.

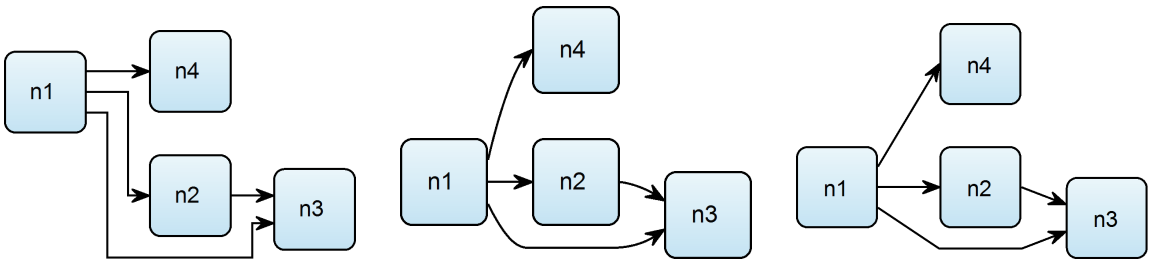
Standardmäßig werden in ELK orthogonale Kanten verwendet. Die Implementierung hierfür ist durch Sander [San04] und Battista et al. [DET+99] inspiriert worden.

2.2.1. Dummyknoten

In ELK gibt es sechs verschiedene Arten von Dummyknoten. Im Folgenden werden diejenigen Arten erklärt, die für diese Arbeit relevant sind.

Externer Port Dieser Dummyknoten repräsentiert einen Port des hierarchischen Knotens.

Die Größe des hierarchischen Knotens ergibt sich aus der Gesamtgröße des beinhalteten



(a) Orthogonal **(b)** Splines **(c)** Polylines

Abbildung 2.3. Drei verschiedene Kantendarstellungen.

Graphen. In Abbildung 2.4a sehen wir ein Beispiel für die erstellten externen Portdummyknoten. Für die östliche und westliche Portseite sind die externen Portdummies immer in der ersten beziehungsweise letzten Ebene des Graphen platziert. Die nördlichen und östlichen Portdummies müssen in der jeweiligen Ebene ganz oben beziehungsweise ganz unten eingeordnet sein. In Abbildung 2.4b werden die Dummyknoten aufgelöst, wodurch an ihrer Position der hierarchische Port platziert wird.

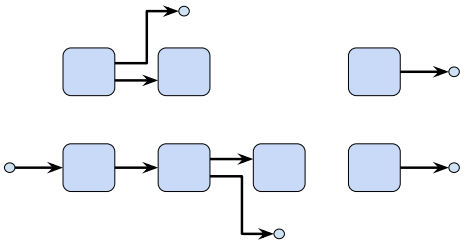
Lange Kante Für lange Kanten müssen Dummyknoten hinzugefügt werden, damit eine ordentliche Zuweisung des Graphen eingehalten werden kann. Ein Beispiel hierfür ist in Abbildung 2.2c zu sehen.

Nord- und Südport Kantenverbindungen zu nördlichen und südlichen Ports können nicht wie normale Kantenverbindungen geführt werden, ansonsten gibt es überlappende Kantensegmente. Eine Folge davon wäre, dass der Graph nicht mehr eindeutig lesbar ist. In ELK wurden deshalb Dummyknoten für Kantenverbindungen mit diesen Ports eingefügt und in Abbildung 2.4c sehen wir ein Beispiel dafür. Eine Kante ist mit dem jeweiligen Dummyknoten verbunden und hat zunächst keine Verbindung mit dem Port, mit dem sie eigentlich verbunden sein soll. Der Dummyknoten hat deshalb als Eigenschaft gespeichert, mit welchem Port er verbunden sein soll. Diese Verbindung ist in der Abbildung mit einer grauen gestrichelten Linie symbolisiert. In der Kreuzungsminimierung wird eine Reihenfolge der Dummyknoten berechnet und im späteren Verlauf des ebenenbasierten Ansatzes werden die originalen Verbindungen wiederhergestellt. Das Resultat davon ist in Abbildung 2.4d zu sehen.

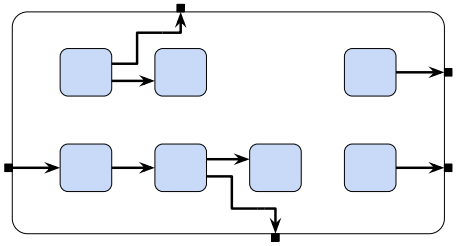
In der Abbildung ist auch eine lange Kante e_1 zu sehen. Für lange Kanten ist es erlaubt, zwischen Nord- und Südports platziert zu werden. Das Erlauben hiervon kann Kreuzungen an anderen Stellen vermeiden oder die Kantenlänge kürzen.

2.2.2. Zwischenprozessoren

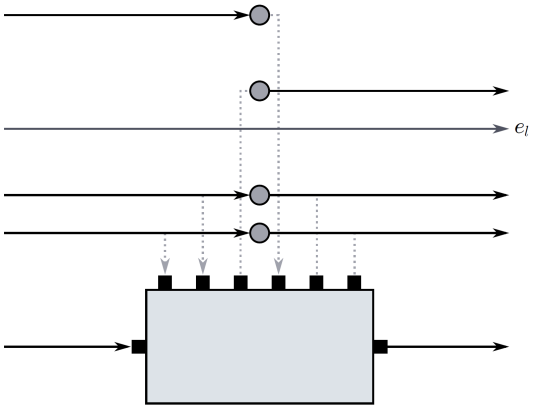
Ein großer Vorteil des ebenenbasierten Ansatzes ist die hohe Modularität. Die einzelnen Phasen können leicht durch alternative Implementierungen ausgetauscht werden, die zum



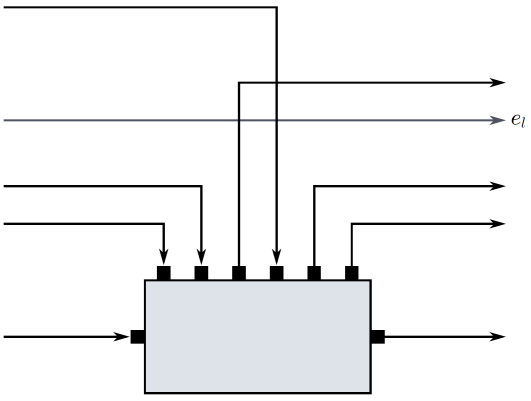
(a) Graph mit externen Portdummies.



(b) Auflösen der externen Portdummies.



(c) Die zugehörigen Nordportdummies eines Knotens. Zu sehen in [Sch11].



(d) Auflösen der Nordportdummies. Zu sehen in [Sch11].

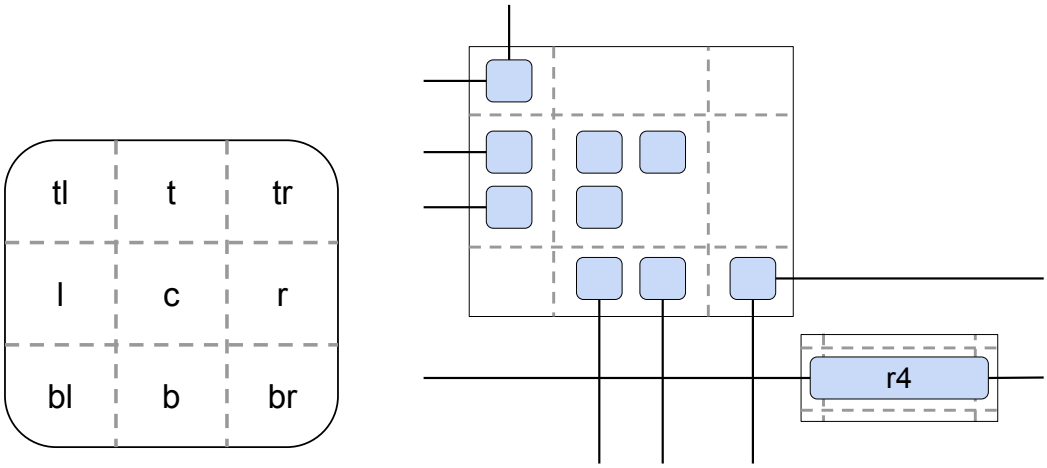
Abbildung 2.4. Veranschaulichung der Dummyknoten.

Beispiel andere Ziele für das resultierende Diagramm verfolgen. Vor und nach den einzelnen Phasen gibt es Slots für sogenannte *Zwischenprozessoren*. Die Idee hierfür ist es, die einzelnen Phasen so simpel wie möglich zu halten. Alle Berechnungen, die nicht die Hauptziele der jeweiligen Phase verfolgen, werden dadurch ausgelagert.

Ein Beispiel hierfür ist das Erstellen einer ordentlichen Zuweisung. Wie Schulze in seiner Diplomarbeit [Sch11] beschrieben hat, würde das Erstellen einer ordentlichen Zuweisung während der Phase der Ebenenzuweisung zu einem Problemen führen: Bei unterschiedlichen Algorithmen für die Ebenenzuweisung müsste der Code für das Erstellen einer ordentlichen Zuweisung dupliziert werden. Durch ein Auslagern dieses Schrittes in einen Zwischenprozessor wird dieses Problem gelöst.

2.2.3. Connected Components Processing

Wenn für einen hierarchischen Knoten die Porteinschränkung FREE oder FIXED SIDE gewählt wurde, dann wird in ELK das sogenannte *Connected Components Processing*, welches von



(a) Eine Gruppe mit den neun Sektoren. Basierend auf einem unpublizierten Diagramm von Schulze.

(b) Platzierung der Komponenten innerhalb der Sektoren. Basierend auf einem Diagramm von Ruegg et al. [RSG+16].

Abbildung 2.5. Platzieren der Komponenten mit Hilfe von Cell Packing.

Schulze et al. [RSG+16] entworfen wurde, verwendet. Wie in Abschnitt 1.1 erklärt wurde, kann dieses Verfahren nicht für die anderen Porteinschränkungen verwendet werden.

Beim Connected Components Processing wird der Graph innerhalb des hierarchischen Knotens unterteilt, bevor er an den Layoutalgorithmus übergeben wird. Innerhalb des hierarchischen Knotens werden alle Zusammenhangskomponenten gesucht und diese werden einzeln an den Layoutalgorithmus übergeben. Alle Phasen des ebenenbasierten Ansatzes arbeiten dann jeweils nur mit einer Zusammenhangskomponente. Nachdem die Phasen des ebenenbasierten Ansatzes für die Zusammenhangskomponenten durchlaufen sind, wird das sogenannte *Cell Packing* für das eigentliche Platzieren der Zusammenhangskomponenten verwendet. Dafür wird der hierarchische Knoten, wie in Abbildung 2.5a zu sehen ist, in neun Sektoren aufgeteilt, um ein überschneidungsfreies Platzieren zu ermöglichen. In die jeweiligen Sektoren werden Zusammenhangskomponenten mit Verbindungen zu bestimmten Portseiten des hierarchischen Knotens eingeordnet. Die Sektoren in den Ecken können nur jeweils eine Komponente enthalten, wohingegen in alle anderen Sektoren beliebig viele Komponenten eingeordnet werden können

Komponenten, die sowohl eine Verbindung mit einem Port auf der Nordseite und einem Port auf der Westseite haben, werden in den Sektor *tl* platziert. Komponenten, dessen Portverbindungen sich ausschließlich auf der Nordseite befinden, werden in den Sektor *t* platziert. Komponenten ohne Portverbindungen werden in den Sektor *c* eingeteilt. Falls eine Komponente zum Beispiel Verbindungen mit den Seiten Westen und Osten hat, dann wird die Komponente in mehrere Sektoren eingeteilt. Bei diesem Beispiel werden die Sektoren *l*, *c* und *r* genutzt. Wenn die Komponente nun zusätzlich eine Verbindung mit der Seite Süden

hat, dann werden die Sektoren *bl*, *b* und *br* genutzt und bei Verbindungen mit allen Portseiten müssen alle Sektoren verwendet werden.

Beim Einordnen von Komponenten, die in mehrere Sektoren platziert werden müssen, können Konflikte entstehen. Wenn einer der Sektoren, in die die Komponente mit verschiedenen Portseitenverbindungen platziert werden soll, bereits mindestens eine andere Komponente enthält, dann entsteht ein solcher Konflikt. Bei einem Konflikt wird eine neue Gruppe bestehend aus weiteren neun Sektoren diagonal unterhalb der vorherigen Gruppe platziert. Falls in dieser Gruppe erneut ein Konflikt entsteht, werden weitere Gruppen diagonal erzeugt bis alle Komponenten platziert sind. In Abbildung 2.5b können wir sehen, dass für r_4 eine solche neue Gruppe erzeugt werden musste, da r_4 in den Sektoren *l*, *c* und *r* platziert werden muss, aber diese sind in der ersten Gruppe bereits belegt. Durch das diagonale Platzieren ist sichergestellt, dass in den neuen Gruppen keine Kreuzungen mit bereits platzierten Knoten entstehen können.

Ein Nachteil dieses Ansatzes kommt zum Tragen, wenn jede Komponente einen Konflikt aufweist. Dann werden alle Komponenten diagonal platziert und es wird sehr viel ungenutzte Fläche erzeugt.

2.3. Ästhetikkriterien

Graphen können immer auf verschiedene Weisen dargestellt werden. Die Darstellungsform hat dabei Einfluss auf die Verständlichkeit für den Betrachter [Pet95]. Damit ein Graph auf eine möglichst lesbare Art dargestellt wird, hat Purchase [Pur97] Ästhetikkriterien analysiert und mit Hilfe einer Studie fünf Hypothesen über diese Ästhetikkriterien untersucht. Die Hypothesen sind im Folgenden aufgelistet:

Kantenknicke Eine erhöhte Anzahl an Kantenknicken in einem Graph führt zu einem schlechteren Verständnis des Graphen.

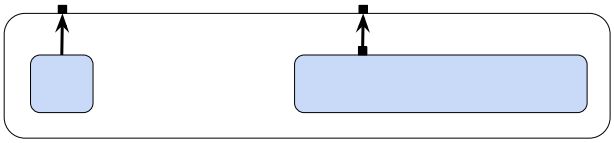
Kantenkreuzungen Eine erhöhte Anzahl an Kantenkreuzungen in einem Graph führt zu einem schlechteren Verständnis des Graphen.

Winkel Das Maximieren des kleinsten Winkels zwischen ausgehenden Kanten eines Knotens in einem Graphen erhöht das Verständnis des Graphen.

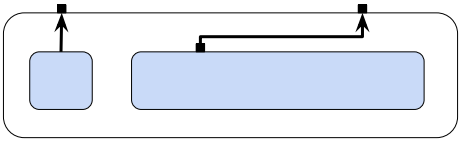
Orthogonalität Das Fixieren von Knoten und Kanten in einem orthogonalen Grid erhöht das Verständnis des Graphen.

Symmetrie Durch das Erhöhen der Symmetrie eines Graphen wird das Verständnis des Graphen verbessert.

Am aussagekräftigsten waren in der Studie von Purchase die Ergebnisse für die Ästhetik der Kantenkreuzungen. Eine negative Ausprägung dieser Ästhetik führt bei den getesteten Graphen dazu, dass der Anwender wesentlich mehr Zeit für die gestellten Aufgaben benötigt



(a) Weniger Kantenknicke, aber mehr ungenutzte Fläche.



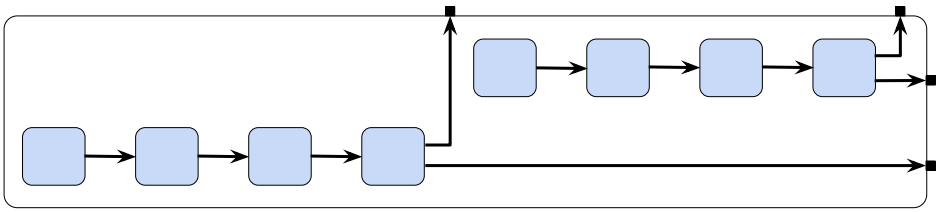
(b) Mehr Kantenknicke, aber weniger ungenutzte Fläche.

Abbildung 2.6. Graph mit der Einschränkung Fixed Position.

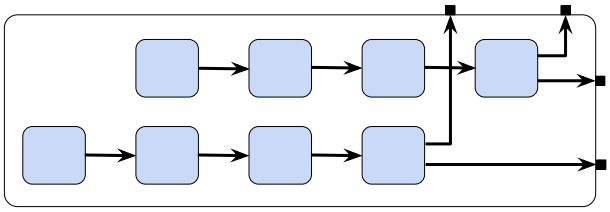
und mehr Fehler produziert. Andererseits gibt es bei einer positiven Ausprägung dieses Kriteriums keine relevanten Auffälligkeiten im Vergleich mit den anderen Ästhetiken. Kreuzungen sind also für die getesteten Graphen nur genau dann problematischer als andere Ästhetiken, wenn viele Kreuzungen im Graphen vorhanden sind.

In einem weiteren Paper von Purchase [Pur02] verdeutlicht sie aber noch einmal, dass das extreme Einhalten eines Kriteriums auch nicht unbedingt förderlich für das Verständnis ist. Viele Algorithmen versuchen zum Beispiel alle Kreuzungen zu entfernen oder alle Knoten auf einem unsichtbaren Grid zu platzieren. Beim Betrachten der Graphen aus Purchase eben genannter Studie [Pur97] fällt auf, dass sich Kriterien gegenseitig ausschließen. Um alle Kreuzungen aus einem Graphen zu entfernen, müssen in dem Beispiel zusätzliche Kantenknicke hinzugefügt werden. Purchase sagt daher, dass es besser wäre für alle Kriterien einen gewissen Grad einzuhalten, da es sehr wahrscheinlich ist, dass Graphen mit einer geringen Zahl an Kreuzungen genauso gut lesbar sind wie Graphen mit gar keinen Kreuzungen. Diese Aussage wird allerdings nicht weiter untersucht und somit gibt es keine genauen Werte dafür, bis zu welchem Grad eine Ästhetik eingehalten werden sollte.

Die von Purchase vorgestellten und untersuchten Ästhetikkriterien nutzen als Grundlage immer einen zusammenhängenden Graphen. Da sich meine Arbeit aber mit Graphen bestehend aus mehreren Zusammenhangskomponenten auseinandersetzt, ist ein weiterer Faktor die Nutzung der Fläche des Diagramms. Wie man in Abbildung 2.6 sieht, kann die ungenutzte Fläche und hierbei auch die Gesamtfläche des Diagramms durch zusätzlich eingeführte Kantenknicke reduziert werden. Ebenso kann die Gesamtfläche und die ungenutzte Fläche in Abbildung 2.7 durch das Einführen von zusätzlichen Kreuzungen verringert werden. Wie bereits erwähnt, sollte dieses Vorhaben die Lesbarkeit des Graphen nicht weiter beeinträchtigen, sofern anderweitige Kantenkreuzungen so gut wie möglich entfernt werden.



(a) Keine Kantenkreuzungen, aber mehr ungenutzte Fläche.



(b) Eine Kantenkreuzung, aber weniger ungenutzte Fläche.

Abbildung 2.7. Graph mit der Einschränkung FIXED ORDER.

2.4. Verwendete Technologien

2.4.1. Eclipse

Eclipse[1] ist eine quelloffene Entwicklungsumgebung (IDE), welche dazu genutzt wird, Software verschiedenster Art zu entwickeln. Die erste Eclipse-Version wurde von der *Eclipse Foundation* am 7. November 2001 veröffentlicht. Es wird ein *Plugin*-System verwendet, um die Benutzerumgebung veränderbar zu machen. Ursprünglich ist Eclipse darauf ausgelegt worden, Software in der Programmiersprache Java zu entwickeln. Durch Plugins kann Eclipse aber zum Beispiel dafür erweitert werden, mit Hilfe anderer Programmiersprachen entwickeln zu können. Dazu gehören zum Beispiel Sprachen wie C, C++, C#, Haskell, JavaScript, PHP und viele weitere. Im *Marktplatz* von Eclipse können solche Plugins heruntergeladen oder selbst zur Verfügung gestellt werden. Der Kern von Eclipse ist in Java programmiert, was eine plattformunabhängige Benutzung ermöglicht.

Mittlerweile kann Eclipse auch in der *Cloud* eingesetzt werden, wodurch im Browser mit Hilfe eines gehosteten *Arbeitsbereichs* entwickelt werden kann. In dem Arbeitsbereich werden die Projekte lokal abgespeichert. In der Cloud befindet sich der Arbeitsbereich dann bei dem jeweiligen Cloud-Anbieter.

[1] `http://www.eclipse.org`

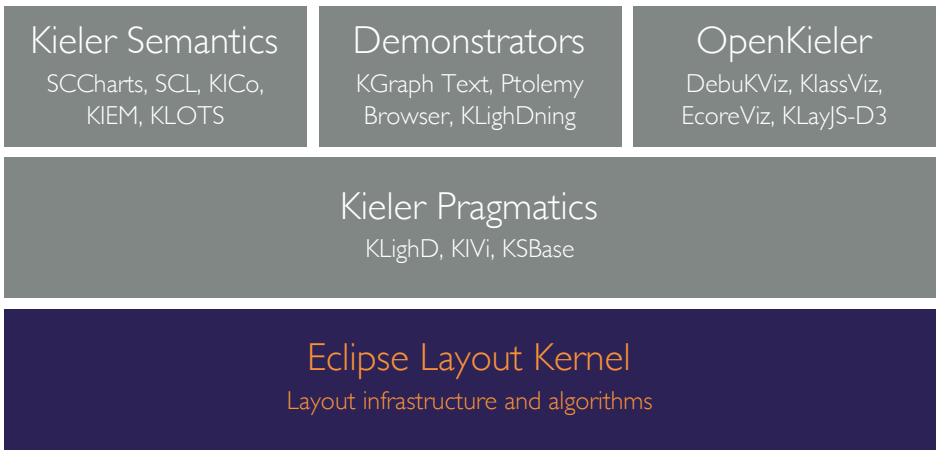


Abbildung 2.8. Aufbau von KIELER.

2.4.2. Kiel Integrated Environment for Layout Eclipse RichClient (KIELER)

KIELER[2] ist ein Projekt, welches von der Arbeitsgruppe *Echtzeitsysteme und Eingebettete Systeme* der Christian-Albrechts-Universität zu Kiel als eine akademische, quelloffene Experimentierplattform genutzt wird. Das komplette Projekt basiert auf der Entwicklungsumgebung Eclipse und wird dort über mehrere Plugins integriert. Das Ziel des Projekts ist es, das graphische modellbasierte Design komplexer Systeme zu verbessern. In Abbildung 2.8 sehen wir die aktuellen Forschungsbereiche an denen im Rahmen von KIELER gearbeitet wird.

Ein großer Bereich von KIELER ist die *Pragmatik* [HF10], wodurch die Verständlichkeit von Diagrammen, sowie die Entwicklung und die Analyse komplexer Modelle verbessert werden soll.

Für die Visualisierung von Graphen wurde KIELER Lightweight Diagrams (KLighD) [SSH13] entwickelt. KLighD sorgt dafür, dass die Elemente des Graphen eine visuelle Repräsentation bekommen und berechnet Größen von Knoten, die zum Beispiel Text beinhalten, dynamisch.

Ursprünglich wurde im Bereich der Pragmatik KIELER Infrastructure for Meta Layout (KIML) als Schnittstelle zwischen Layoutalgorithmen und Editoren für das automatische Layout graphischer Modelle verwendet. KIML wurde nun durch Eclipse Layout Kernel (ELK)[3] abgelöst. Dadurch wurde das Projekt zu Eclipse ausgelagert und ist somit kein Bestandteil mehr von KIELER. Die Funktionalität hat sich dabei nicht verändert, wodurch ELK weiterhin als Schnittstelle zwischen den Layoutalgorithmen und dem Editor dient. Die Funktionsweise von ELK sehen wir in Abbildung 2.9. Damit automatisches Layout verwendet werden kann, wird als Erstes aus dem Editor ein Graph an ELK weitergeleitet. ELK beinhaltet viele selbstentwickelte Layoutalgorithmen, kann aber auch mit quelloffenen Layoutbibliotheken wie zum Beispiel GraphViz[4] umgehen. Mit Hilfe der Layoutalgorithmen werden Koordinaten für

[2] https://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/Overview

[3] http://www.eclipse.org/elk

[4] http://www.graphviz.org/

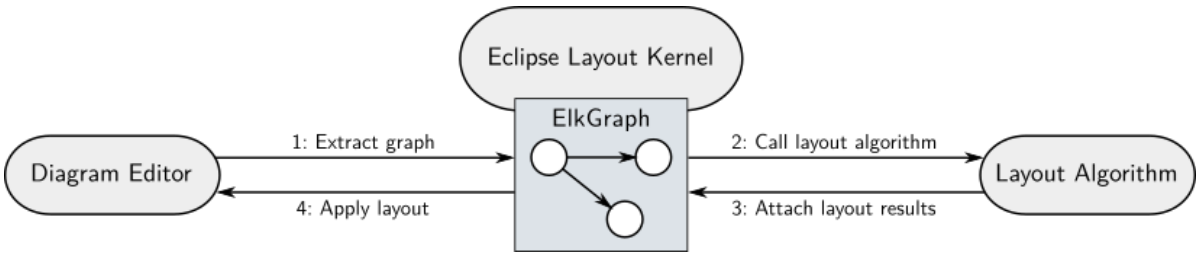


Abbildung 2.9. Funktionsweise von ELK [Ker].

die einzelnen Elemente des erhaltenen Graphen berechnet. Die Ergebnisse werden dann an den Editor zurückgeliefert. Im Rahmen dieser Arbeit wird ein Layoutalgorithmus von ELK weiterentwickelt.

Der zweite große Bereich von KIELER, die *Semantik* [HMA+13], nutzt die in ELK entwickelten Layoutalgorithmen und liefert somit unter Anderem Anwendungsbeispiele für die Pragmatik. Mit Hilfe des KIELER Compiler (KiCo) können Modelle für sicherheitskritische Systeme mit dem Tool SCCharts entwickelt werden. KiCo kann dabei zum Beispiel C-Code aus dem modellierten System generieren, der dann in anderen Anwendungen weiterverwendet werden kann.

Zu dem Bereich der *Demonstrators* gehören Editoren, die genutzt werden, um die entwickelten Technologien aus den Bereichen ELK, Pragmatik und Semantik zu explorieren und zu testen.

Im Bereich *OpenKieler* landen Projekte aus dem Bereich Demonstrators, die besonders sinnvoll für Endbenutzer sind.

2.4.3. Graph Analysis (GrAna)

Das Projekt GrAna erlaubt es für alle Graphenformate, die in ELK bekannt sind, Analysen über den Graphen auszuführen. Hierbei können verschiedene strukturelle Eigenschaften der Graphen, wie zum Beispiel die Anzahl der Knoten und Kanten, gezählt werden. Weiterhin können Eigenschaften über das finale Aussehen der Graphen analysiert werden. Dazu gehört zum Beispiel das Zählen von Kantenkreuzungen, das Berechnen der Gesamtfläche oder das Berechnen der ungenutzten Fläche des Diagramms. Die Ergebnisse von GrAna werden anschließend in einer Datei abgespeichert.

Kapitel 3

Verwandte Arbeiten

Da in dieser Arbeit zwei Phasen des ebenenbasierten Ansatzes angepasst wurden, stelle ich in diesem Kapitel Algorithmen vor, die sich in den beiden Phasen bisher etabliert haben. In den jeweiligen Hauptkapiteln wird dann erklärt, weshalb sich diese Algorithmen für eine Lösung des Problems aus Abschnitt 1.1 nicht eignen.

3.1. Ebenenzuweisung

Longest Path Layerer Der bekannteste und einfachste Algorithmus für die Ebenenzuweisung ist der *Longest Path Layerer* [RDM+87]. In Algorithmus 1 ist dieser in Pseudocode beschrieben. Die Idee ist es, den längsten Pfad im Graphen als Gesamtzahl der Ebenen zu verwenden. Senken werden in die letzte Ebene gepackt und anschließend aus der Liste der zu betrachtenden Knoten entfernt. Dadurch entstehen neue Senken, die in die vorherige Ebene gepackt werden. Dieses Verfahren wird so lange wiederholt, bis alle Knoten in Ebenen eingeteilt wurden. Die Laufzeit des Algorithmus liegt dabei in $\mathcal{O}(n)$, wobei n der Anzahl der Knoten entspricht.

Da beim ebenenbasierten Ansatz in ELK der Datenfluss von links nach rechts und nicht von oben nach unten verläuft, findet der Longest Path Layerer eine Ebenenzuweisung mit kleinster Breite. Ein typisches Problem dieses Ansatzes ist, dass er eine schlechte Anzahl an Dummyknoten und damit unnötige lange Kanten produziert. Ein Beispiel dafür sehen wir für den Knoten $n9$ in Abbildung 3.1a. Der Knoten könnte zwei Ebenen weiter links platziert werden, wodurch die Zahl der Dummyknoten und somit auch die Kantenlänge verringert werden würde.

Network Simplex Dieser Algorithmus wurde von Gansner et al. [GKN+93] entworfen. Hierbei werden die Dummyknoten des Graphen minimiert, was in einer kürzeren Kantenlänge resultiert. Wie wir in Abbildung 3.1b sehen, existiert die lange Kante zu dem Knoten $n9$ nicht mehr, welche ein Resultat von dem Longest Path Layerer war. Die Idee des Network Simplex-Algorithmus ist es, einen spannenden Baum zu nutzen, um den Knoten einen Rang zuzuweisen. Durch *Schnittwerte* in dem spannenden Baum können die Ränge der Knoten so berechnet werden, dass die Kantenlängen möglichst kurz gehalten werden. Für die Laufzeit des Algorithmus wurde noch keine Polynomialzeit bewiesen, aber üblicherweise führt der Algorithmus schnell genug zu einem Ergebnis.

Algorithm 1: Longest Path Layerer

```
1 lp := längster Pfad im Graph;
2 unprocessed := Liste aller noch zu betrachtenden Knoten;
3 while unprocessed nicht leer do
4 |   Weise alle Senken der Ebene L_lp zu;
5 |   Entferne alle Senken aus unprocessed und entferne deren eingehende und
  |     ausgehende Kanten für den nächsten Durchlauf;
6 |   lp = lp - 1;
```

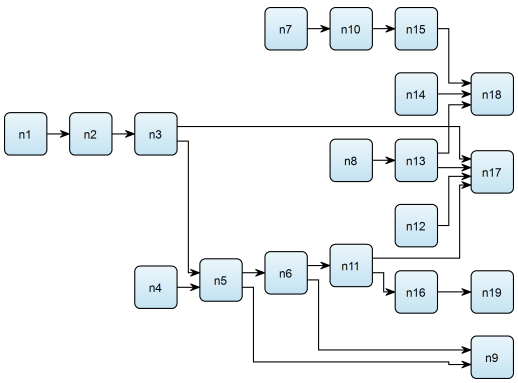
Coffman-Graham Dieser Algorithmus wurde von Coffman und Graham [CG72] entworfen. Die Idee für den Algorithmus ist es, eine obere Schranke für die maximale Anzahl an Knoten in einer Ebene zu haben. Diese Schranke wird dann vom Algorithmus eingehalten. Die Laufzeit des Algorithmus liegt in $\mathcal{O}(n^2)$, wobei n der Anzahl der Knoten entspricht. Wenn wir Dummyknoten dazu zählen, dann wird die Höhe durch diesen Algorithmus im schlechtesten Fall fast doppelt so groß wie das Optimum. Weiterhin tendiert dieser Algorithmus dazu, eine hohe Anzahl an Dummyknoten zu produzieren.

Minimum-Width Wie eben bereits beschrieben wurde, wird beim Coffman-Graham-Algorithmus eine obere Schranke für die Anzahl an Knoten in einer Ebene genutzt. Der Nachteil des Algorithmus ist es, dass er eine Vielzahl an Dummyknoten produziert und die Höhe bei Berücksichtigung von Dummyknoten wesentlich größer werden kann. Nikolov et al. [NTB05] haben den Minimum-Width-Algorithmus entwickelt, bei dem unter Berücksichtigung der Dummyknoten die Anzahl der Knoten in einer Ebene minimiert wird. Hierbei ist zu beachten, dass in ELK die Höhe und nicht die Breite des Graphen durch Minimum-Width verringert wird, da der Datenfluss beim ebenenbasierten Ansatz in ELK von links nach rechts und nicht von oben nach unten verläuft.

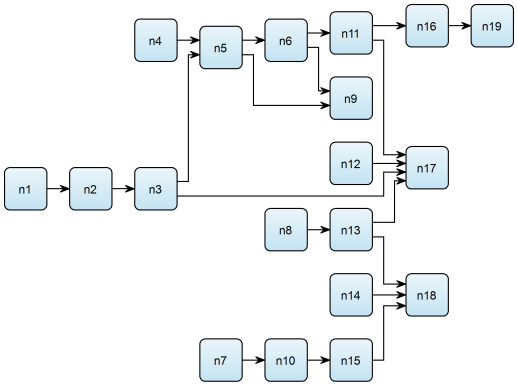
Stretch-Width Dieser Algorithmus wurde ebenfalls von Nikolov et al. [NTB05] entwickelt und ist eine wesentlich einfachere Variante des Minimum-Width-Algorithmus. Wie beim Coffman-Graham-Algorithmus wird eine obere Schranke für die maximale Anzahl an Knoten in einer Ebene angegeben. Falls der Algorithmus eine Stelle erreicht, an der ein Knoten nicht mehr in Ebenen einsortiert werden kann, ohne die obere Schranke zu überschreiten, dann wird die obere Schranke um eins erhöht und somit kann der Algorithmus die Knoten weiter in die Ebenen einteilen.

3.2. Kreuzungsminimierung

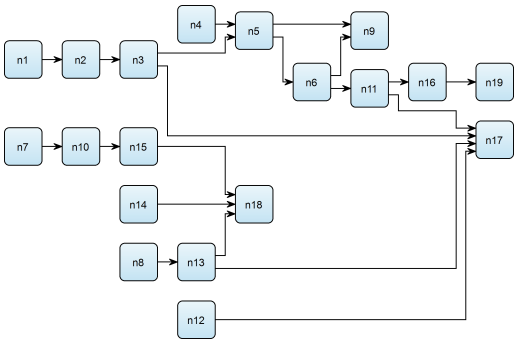
In der Praxis ist der erfolgreichste Algorithmus für die Kreuzungsminimierung der *Layer Sweep*-Algorithmus [STT81; San99]. Die Idee des Algorithmus wird in Algorithmus 2 in Pseudocode angegeben. Der Algorithmus durchläuft die Ebenen des Graphen iterativ in beide Richtungen. Dabei werden jeweils zwei benachbarte Ebenen betrachtet, wobei eine Ebene



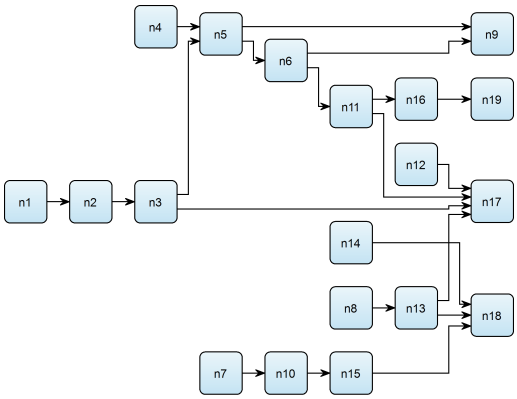
(a) Longest Path erzeugt 8 Ebenen.



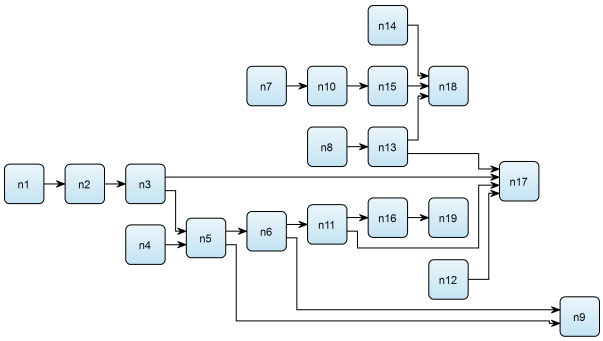
(b) Network Simplex erzeugt 8 Ebenen.



(c) Coffman-Graham erzeugt 9 Ebenen.



(d) Minimum-Width erzeugt 8 Ebenen.



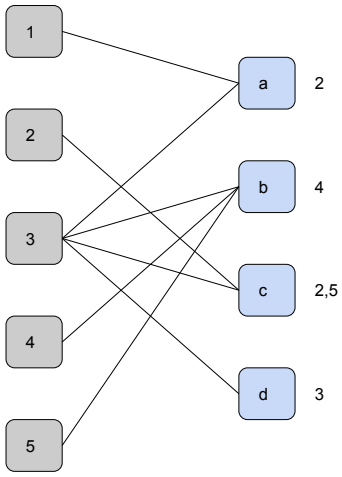
(e) Stretch-Width erzeugt 10 Ebenen.

Abbildung 3.1. Beispiel der fünf vorgestellten Algorithmen für die Ebenenzuweisung.

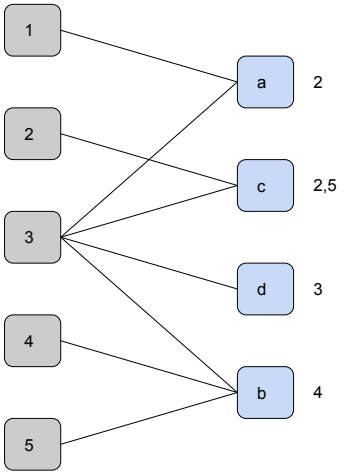
Algorithm 2: Layer Sweep

```
1 for 1 to R do
2     foreach Ebene i von links nach rechts do
3         foreach Knoten aus i do
4             Berechne Barycenter-Wert;
5         Sortiere Knoten aus i anhand der Barycenter-Werte;
6     foreach Ebene j von rechts nach links do
7         Gleiche Vorgehensweise wie in der ersten ForEach-Schleife;
8     Speichere die aktuelle Reihenfolge der Knoten, falls sich die Zahl der Kreuzungen
      verringert hat;
```

fixiert ist und in der anderen Ebene Knoten vertauscht werden, um die Kantenkreuzungen zwischen den beiden Ebenen zu minimieren. Das Vertauschen der Knoten kann zum Beispiel durch *Barycenter*-Werte realisiert werden, wie wir es in Abbildung 3.2 sehen. Dabei wird den Knoten in der fixierten Ebene ein Rang entsprechend ihrer Reihenfolge zugewiesen und die Knoten aus der freien Ebene erhalten einen Durchschnittswert, der sich aus den Rängen der verbundenen Knoten aus der fixierten Ebene berechnet. Das Sortieren anhand der Barycenter-Werte führt in der Abbildung zu einer Verringerung der Kreuzungen um sieben. Der Layer Sweep-Algorithmus muss diese Kreuzungen zählen können, damit die Reihenfolge der Knoten mit den geringsten Kreuzungen vom Algorithmus ausgegeben werden kann. Das Zählen der Kreuzungen kann hierbei zum Beispiel durch die Methode von Barth et al. [BJM02] durchgeführt werden. Falls nach dem Durchlaufen der Ebenen die aktuelle Reihenfolge der Knoten in weniger Kreuzungen als die bisherige beste Reihenfolge der Knoten resultiert, dann wird die Reihenfolge abgespeichert. Das Iterieren durch die Ebenen wird mehrfach wiederholt, bis eine vorher festgelegte Anzahl an Schritten erreicht wurde. Am Ende wird das beste Ergebnis der Durchläufe zurückgegeben.



(a) Bei dieser Reihenfolge der Knoten entstehen acht Kreuzungen.



(b) Nach dem Sortieren der Knoten anhand der Barycenter-Werte entsteht nur noch eine Kreuzung.

Abbildung 3.2. Vertauschen von Knoten durch Barycenter-Werte. Die linke Ebene ist in den Abbildungen jeweils fixiert und die rechte frei. Die Barycenter-Werte sind rechts neben den Knoten aus der freien Ebene angegeben.

Kapitel 4

Ebenenzuweisung

Dieses Kapitel beschreibt, wie die erste Phase verändert werden muss, um das Problem aus Abschnitt 1.1 zu lösen. Zunächst werden die Algorithmen aus Kapitel 3 auf einen Graphen mit mehreren Zusammenhangskomponenten angewendet und erklärt, wie diese mit Zusammenhangskomponenten umgehen. Danach werden Lösungsansätze vorgestellt und geklärt, ob es sinnvoll ist, diese Ansätze weiter zu verfolgen. Als Letztes werden die einzelnen Schritte erklärt, die in der gewählten Lösung durchgeführt werden müssen, um die Zusammenhangskomponenten in Ebenen zu verteilen.

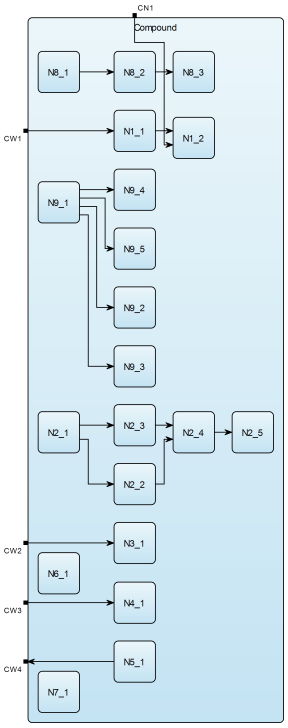
4.1. Ebenenzuweisung mit Zusammenhangskomponenten

In Abbildung 4.1 sehen wir die Ergebnisse der Ebenenzuweisungsalgorithmen aus Kapitel 3 angewandt auf einen Graphen mit mehreren Zusammenhangskomponenten.

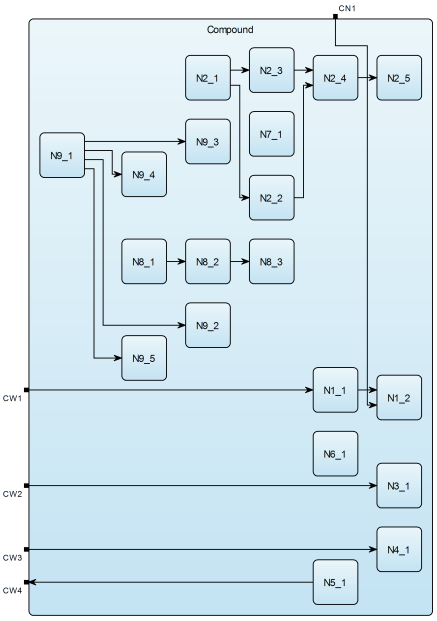
Network Simplex und Longest Path Diese beiden Algorithmen liefern für den gezeigten Graphen dieselben Ergebnisse und wurden somit in einer Abbildung zusammengefasst. Hier können wir erkennen, dass die Breite des Diagramms von der breitesten Zusammenhangskomponente abhängt. Die Höhe des Graphen wächst durch die Anzahl der Zusammenhangskomponenten. Somit können sich bei einer Vielzahl von Zusammenhangskomponenten schlechte Seitenverhältnisse ergeben. Ein weiterer Nachteil dieser Algorithmen ist es, dass bei vielen Nord- und Südportverbindungen mit dem hierarchischen Knoten die Lesbarkeit des Diagramms stark eingeschränkt wird, da diese Verbindungen potentiell eine Vielzahl von anderen Komponenten kreuzen müssen.

Minimum-Width Dieser Algorithmus ist am vielversprechendsten, da er auch bei großen Diagrammen nur selten lange Kanten innerhalb von Zusammenhangskomponenten erzeugt. Die einzigen Nachteile hierbei sind, dass ebenfalls lange Kanten zu Komponenten mit Ost- oder Westportverbindungen entstehen und viel ungenutzte Fläche im Diagramm entstehen kann, da die Komponenten nicht kompakt über die Ebenen verteilt werden.

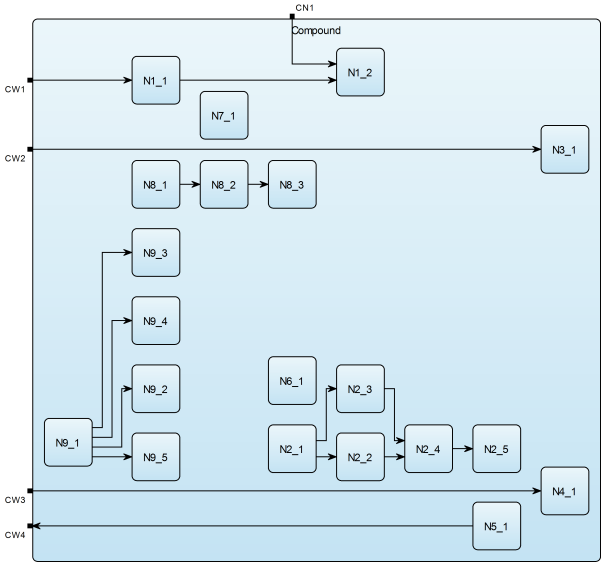
Coffman-Graham Dieser Algorithmus erzeugt viele Dummyknoten. Das führt vor allem bei größeren Diagrammen dazu, dass viele lange Kanten innerhalb von Zusammenhangskomponenten erzeugt werden. Die Knoten der Zusammenhangskomponenten sind dadurch weit voneinander entfernt, was die Lesbarkeit der Komponente selbst einschränkt. Ebenfalls könnte dies zu einer größeren Gesamtfläche und viel ungenutzter Fläche des Diagramms führen. Ein weiterer Nachteil ist, dass die Portverbindungen der Ost- und



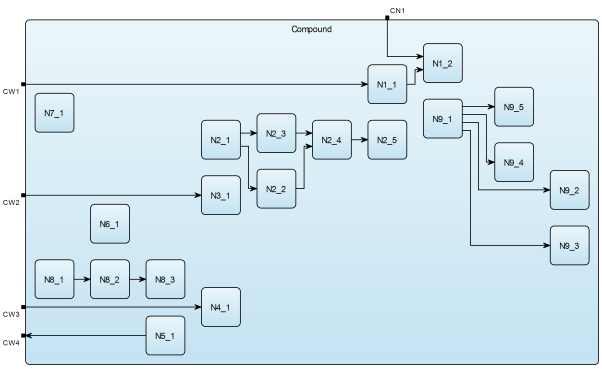
(a) Network Simplex und Longest Path erzeugen 4 Ebenen.



(b) Minimum-Width erzeugt 6 Ebenen.



(c) Coffman-Graham erzeugt 8 Ebenen.



(d) Stretch-Width erzeugt 10 Ebenen.

Abbildung 4.1. Verschiedenen Algorithmen für die Ebenenzuweisung angewandt auf einen Graphen mit vielen Zusammenhangskomponenten.

Westports nicht berücksichtigt werden. Die Komponenten werden nicht nah an die jeweilige Portverbindung platziert, woraus sich weitere lange Kanten ergeben.

Stretch-Width Dieser Algorithmus führt zu breiteren Ergebnissen des Minimum-Width-Algorithmus. Die Ergebnisse verhalten sich sehr ähnlich zu denen von Coffman-Graham.

Aus den eben genannten Beobachtungen können Ziele für eine bessere Ebenenzuweisung abgeleitet werden. Ein Ziel ist es, die Zusammenhangskomponenten kompakt darzustellen, so dass keine unnötig langen Kanten in die Komponenten eingefügt werden, damit die Komponenten lesbar bleiben. Weiterhin sollten Komponenten mit Ost- oder Westportverbindungen möglichst nah an ihre zugehörigen Ports platziert werden, ansonsten werden ebenfalls unnötig lange Kanten produziert. Diese verschlechtern die Lesbarkeit, limitieren den Platz für andere Objekte oder erzeugen zusätzliche Kantenkreuzungen. Ebenfalls sollen Komponenten über die Ebenen verteilt werden, sodass die einzelnen Ebenen möglichst gleich viele Knoten beinhalten.

4.2. Lösungsansätze

Anders als beim Connected Components Processing erhalten wir in der Phase der Ebenenzuweisung als Eingabe den kompletten Graphen innerhalb des zugehörigen hierarchischen Knotens. Es gibt verschiedene Möglichkeiten, wie man mit dem Graphen umgehen kann. Im Folgenden werden zwei Möglichkeiten vorgestellt.

Gesamter Graph Bei diesem Ansatz wird eine Ebenenzuweisung direkt für alle Knoten durchgeführt. Hierbei hat man nur wenig Informationen vorliegen, da vorher nicht bekannt ist, wie viele Ebenen und Knoten pro Ebene die Komponenten benötigen. Dadurch ist es schwieriger abzuschätzen, wann eine neue Ebene eröffnet werden muss, oder wie viele Knoten maximal in einer Ebene sein sollten. Die in Abbildung 4.1 gezeigten Ergebnisse verfolgen diesen Ansatz.

In Zusammenhangskomponenten unterteilen Bei diesem Ansatz wird der Graph zuerst temporär in die einzelnen Zusammenhangskomponenten unterteilt und es wird eine Ebenenzuweisung für jede Komponente durchgeführt. Danach werden die Ergebnisse in gemeinsam genutzte Ebenen zusammengeführt. Das Resultat beinhaltet den kompletten Inhalt des hierarchischen Knotens, der an die nächste Phase weitergeleitet wird. Bei diesem Ansatz haben wir mehr zugrundeliegende Informationen, da wir durch die einzelnen Ebenenzuweisungen wissen, wie viele Ebenen und Knoten pro Ebene die Komponenten jeweils benötigen. Dadurch können für den finalen Graphen Abschätzungen darüber vorgenommen werden, wie viele Ebenen und Knoten pro Ebene benötigt werden, um alle Zusammenhangskomponenten unterzubringen. Ich habe mich für diesen Ansatz entschieden und erhoffe mir ein kompaktes Diagramm und eine bessere Verteilung der Komponenten über die Ebenen.

Als nächstes werden verschiedene Ideen vorgestellt, mit denen das Problem des Platzierens der Komponenten gelöst werden könnte.

Breite oder Höhe minimieren Wie in Abschnitt 2.3 beschrieben geht es in dieser Arbeit auch darum, kompakte Diagramme zu erstellen, um möglichst wenig ungenutzte Fläche zu erhalten. Ein naiver Ansatz dafür ist das Minimieren der Breite oder der Höhe des Graphen. Für diese beiden Lösungen haben wir allerdings schon den Longest Path Layerer und den Minimum-Width-Algorithmus kennengelernt. Für den Longest Path Layerer haben wir bereits festgestellt, dass in den meisten Fällen für Zusammenhangskomponenten keine guten Ergebnisse erzielt werden, da die Diagramme hauptsächlich in die Höhe wachsen. Bei Minimum-Width besteht das Problem, dass Komponenten nicht dicht an ihre jeweiligen verbundenen Ports platziert werden.

Tetrisprinzip Ein anderer Ansatz wäre es, den Komponenten eine Umrandung zu geben und diese Umrandungen wie bei *Tetris* möglichst kompakt zusammenzustecken. Das Problem hierbei ist, dass es keine effizienten Algorithmen zum Lösen dieses Ansatzes gibt. Weiterhin hat man in der Phase der Ebenenzuweisung noch kein finales Aussehen der Komponenten berechnet. Erst in späteren Phasen wäre eine Umrandung repräsentativ für eine Komponente.

Kostenfunktion Als letzte Idee kann man über eine Kostenfunktion nachdenken. Diese berechnet Kosten für die Komponenten an verschiedenen Positionen und wählt dann die Position mit den geringsten Kosten. Hierbei können Komponenten dicht an ihren jeweiligen Portverbindungen platziert werden, wodurch sich die Lesbarkeit des Diagramms erhöhen sollte. Daher habe ich mich für diesen Ansatz entschieden und werde das Vorgehen des Algorithmus in den folgenden Sektionen erklären.

4.3. Schritte der Ebenenzuweisung

Für den gewählten Ansatz wird der Graph zunächst in die Zusammenhangskomponenten unterteilt. Als Nächstes wird, wie in Abbildung 4.2 zu sehen ist, eine Ebenenzuweisung für jede Komponente durchgeführt und die Ergebnisse werden in gemeinsam genutzte Ebenen zusammengeführt. Beim Zusammenführen werden die Zusammenhangskomponenten in einer bestimmten Reihenfolge behandelt. Das eigentliche Platzieren in die Ebenen wird durch eine Kostenfunktion gelöst. Die Kostenfunktion orientiert sich an vorher berechneten angestrebten Werten, damit abgeschätzt werden kann, wann eine Ebene zu viele Knoten beinhaltet und die Komponente somit in einer anderen Ebene beginnen muss.

Um das zu lösende Problem zunächst etwas zu vereinfachen, wird davon ausgegangen, dass alle Knoten dieselbe Größe haben. Dadurch muss zunächst nicht bedacht werden, in welche Ebene große Knoten eingeordnet werden sollten. Das Problem bei Knoten mit unterschiedlicher Größe ist, dass die Breite einer Ebene von dem breitesten Knoten der Ebene abhängt. Das führt dazu, dass kleine Knoten, die in dieselbe Ebene eingeordnet werden, lange

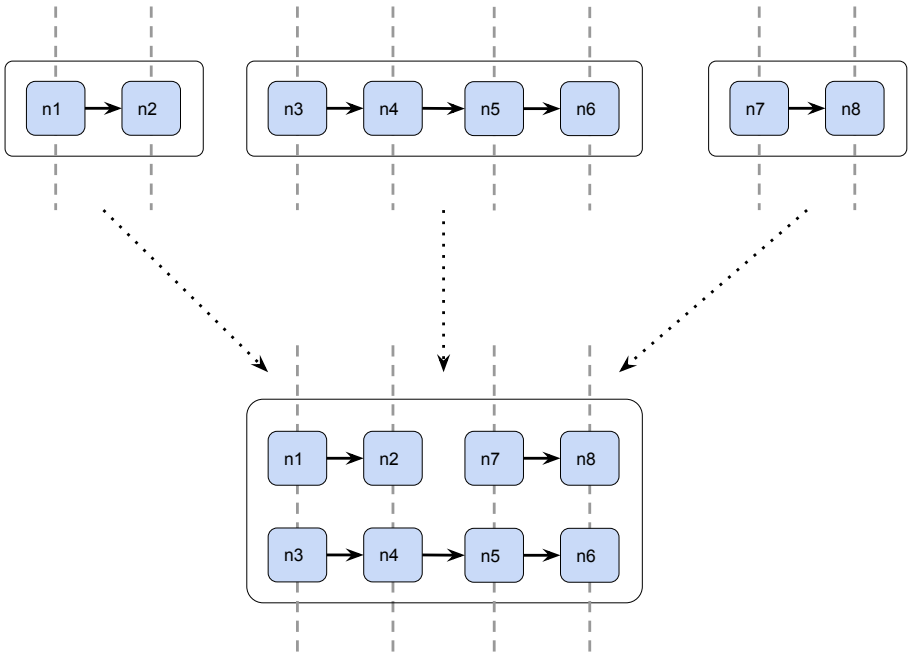


Abbildung 4.2. Schritte der Ebenenzuweisung. Zunächst wird eine Ebenenzuweisung für jede Komponente ausgeführt und danach werden die Ergebnisse in gemeinsam genutzte Ebenen zusammengeführt.

Kanten erhalten. Eine ungünstige Verteilung von vielen großen Knoten auf die Ebenen kann die Breite und Höhe des finalen Diagramms stark beeinflussen. Als weitere Annahme wird zunächst davon ausgegangen, dass die Porteinschränkung auf FIXED ORDER gesetzt ist. Als Nächstes werden die nötigen Schritte mit den genannten Annahmen erläutert.

4.3.1. Erkennen von Komponenten

Wie bereits erwähnt erhalten wir als Eingabe in dieser Phase, anders als beim Connected Components Processing, den kompletten Graphen des hierarchischen Knotens. Als Erstes müssen enthaltene Zusammenhangskomponenten erkannt werden. Hierfür wird eine Tiefensuche durchgeführt und alle erreichbaren Knoten werden zu einer Komponente zusammengefasst.

4.3.2. Reihenfolge von Komponenten mit hierarchischen Portverbindungen

Die einzelnen Portseiten haben auf Grund der Einschränkung FIXED ORDER oder FIXED POSITION eine feste Reihenfolge vorgegeben. Komponenten müssen später in Ebenen eingeteilt werden und neue Ebenen werden in horizontaler Richtung hinzugefügt. Für diese Phase ist daher eine Reihenfolge der Komponenten mit Verbindungen zu Nord- und Südports wichtig. Eine Komponente sollte möglichst so in die Ebenen eingeteilt werden, dass eine gleichmäßige Verteilung der Ports auf der Nord- und Südportseite entsteht. Falls möglich,

sollten die Komponenten für die Einschränkung FIXED POSITION so platziert werden, dass keine zurücklaufenden Kanten entstehen. Der externe Portdummyknoten sollte dabei also mindestens eine Ebene vor der Portverbindung platziert sein. Wenn eine Komponente mehrere Verbindungen auf einer Portseite hat, wird das erste Auftreten der Komponente für die Reihenfolge gewählt. In diesem Schritt wird auch für jede Komponente gespeichert, mit welcher Portseite des hierarchischen Knotens eine Verbindung besteht, denn Komponenten können mit mehreren Portseiten verbunden sein.

4.3.3. Ebenenzuweisung für jede Komponente

Damit die einzelnen Komponenten möglichst kompakt sind und keine unnötig langen Kanten innerhalb der Komponente produziert werden, wird zunächst eine Ebenenzuweisung für jede Komponente durchgeführt. Dadurch ist für jede Komponente bekannt, wie viele Ebenen und Knoten pro Ebene sie beansprucht.

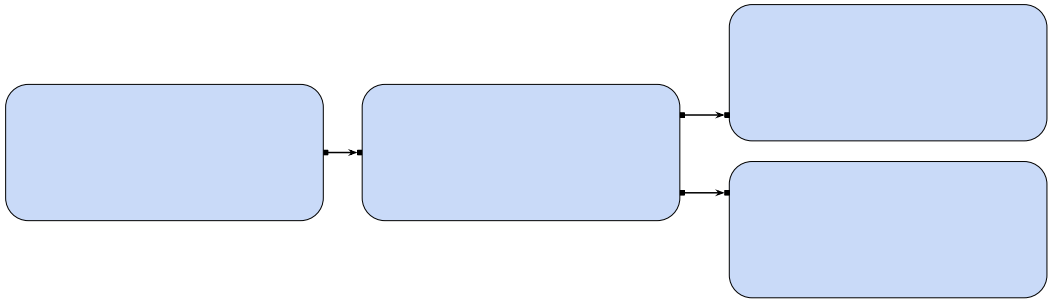
Der Algorithmus für die Ebenenzuweisung erwartet einen Graphen als Eingabe und in dieser Phase steht nur der Graph mit allen Zusammenhangskomponenten zur Verfügung. Daher muss für jede Komponente ein kopierter Graph erstellt werden. Die kopierten Graphen sind dabei genau so aufgebaut wie die Komponenten selbst. Für einen kopierten Graphen werden alle Knoten, Ports und Kanten kopiert, die der jeweiligen Komponente zugewiesen sind. Auf jeden kopierten Graphen wird dann der Network Simplex-Algorithmus angewendet. Network Simplex wird von ELK standardmäßig für die Ebenenzuweisung verwendet, da dieser die Zahl der Dummyknoten minimiert. Dadurch werden Kanten so kurz wie möglich gehalten und die Lesbarkeit des Datenflusses wird verbessert. Im weiteren Verlauf dieser Phase werden die Ergebnisse der kopierten Graphen als Grundlage genutzt. Erst beim eigentlichen Platzieren der Komponenten werden dann die Ergebnisse der kopierten Graphen auf die originalen Knoten übertragen.

4.3.4. Berechnung einer angestrebten Graphengröße

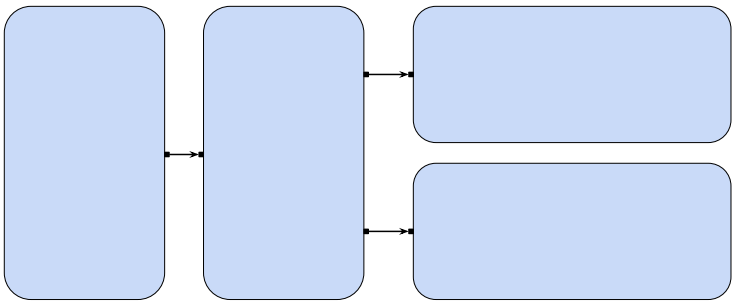
Beim Verteilen von Knoten in Ebenen benötigt der von mir entwickelte Algorithmus eine angestrebte Anzahl an Ebenen und Knoten pro Ebene, damit entschieden werden kann, wann eine Ebene die maximale Anzahl an Knoten erreicht hat und eine Komponente dementsprechend in einer anderen Ebene beginnen muss. Durch die erhaltenen Informationen aus dem vorherigen Schritt können hierfür Abschätzungen berechnet werden.

Als Erstes muss hierbei beachtet werden, dass die Anzahl der Ebenen mindestens so groß sein muss wie die Anzahl der Ebenen der Komponente mit den meisten Ebenen, und die Anzahl der Knoten pro Ebene muss mindestens so groß sein wie die der Komponente mit den meisten Knoten in einer Ebene.

Als nächstes ist hierbei zu beachten, dass ein Seitenverhältnis des hierarchischen Knotens das Seitenverhältnis des restlichen Diagramms beeinflusst. In Abbildung 4.3 sehen wir ein Beispiel hierfür, welches nur aus hierarchischen Knoten besteht. Ein gleiches Seitenverhältnis für alle hierarchischen Knoten in Abbildung 4.3a resultiert dabei in einem größeren Seitenver-



(a) In diesem Beispiel wird durch gleiche Seitenverhältnisse aller hierarchischen Knoten ein kleineres Seitenverhältnis des gesamten Diagramms erreicht.



(b) Durch zum Beispiel gemischte Seitenverhältnisse der hierarchischen Knoten, kann das Seitenverhältnis des gesamten Diagramms verkleinert werden.

Abbildung 4.3. In dieser Abbildung sind alle Knoten hierarchische Knoten. Das Seitenverhältnis von hierarchischen Knoten beeinflusst das Seitenverhältnis des gesamten Diagramms.

hältnis und mehr ungenutzter Fläche des Diagramms. In Abbildung 4.3b werden gemischte Seitenverhältnisse für die hierarchischen Knoten genutzt und wir erhalten ein kleineres Seitenverhältnis und weniger ungenutzte Fläche des Diagramms. Das Problem hierbei ist, dass wir in dieser Phase keinerlei Informationen darüber haben, wie der restliche Graph aufgebaut ist und wie das Seitenverhältnis des aktuellen hierarchischen Knotens das Seitenverhältnis des restlichen Graphen beeinflusst. Falls das standardmäßig genutzte Seitenverhältnis keine gewünschten Ergebnisse liefert, kann der Benutzer für jeden hierarchischen Knoten eine Option für ein angestrebtes Seitenverhältnis festlegen und dadurch selbst Einfluss auf das Ergebnis nehmen.

Die Idee für die eigentliche Berechnung ist, die angestrebte Zahl der Ebenen und Knoten pro Ebene so zu setzen, dass alle Knoten in der resultierenden Fläche untergebracht werden können und das angestrebte Seitenverhältnis eingehalten wird. Hierfür können wir zwei Gleichungen aufstellen: $x \cdot y = n$ und $\frac{x}{y} = AR$. Dabei ist x die gesuchte angestrebte Anzahl an Ebenen und y die gesuchte angestrebte Anzahl an Knoten in einer Ebene. AR und n sind gegeben und beschreiben das Seitenverhältnis und die Anzahl aller Knoten im Graphen.

Durch Umstellen und Einsetzen ergeben sich daraus zwei Gleichungen, mit denen sich die gesuchten Werte berechnen lassen: $x = \sqrt{AR \cdot n}$ und $y = \frac{n}{x}$.

4.3.5. Verteilen von Komponenten auf Ebenen

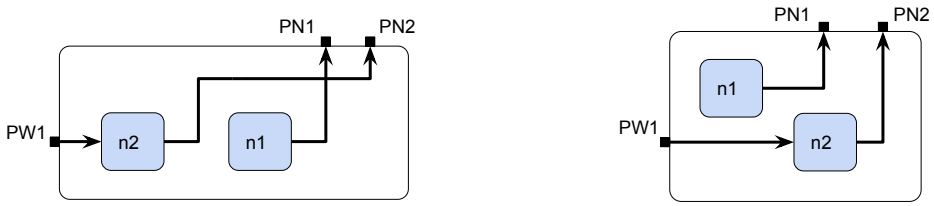
Das Verteilen von Komponenten geschieht in einer festen Reihenfolge. Die Idee hierbei ist, eine Reihenfolge festzulegen, durch welche die Kanten zu den östlichen und westlichen Ports kurz gehalten werden. Weiterhin sollen die Komponenten mit nördlichen und südlichen Portverbindungen über die Ebenen verteilt werden, um potentielle Kantenkreuzungen zu vermeiden und ein lesbareres Diagramm zu erstellen. Die Reihenfolge ist im Folgenden aufgelistet und danach wird das eigentliche Vorgehen für das Platzieren erklärt.

Westverbindungen Zunächst werden alle Komponenten platziert, die ausschließlich Verbindungen zu westlichen Ports des hierarchischen Knotens haben. Das sorgt dafür, dass diese Komponenten möglichst weit links und somit so nah wie möglich an der zugehörigen Portverbindung platziert werden.

Nordverbindungen Sobald eine Komponente mindestens eine Verbindung mit einem nördlichen Port des hierarchischen Knotens hat, wird sie als zweites platziert. Hierbei darf die Komponente auch Verbindungen zu anderen Portseiten haben. Die vorher berechnete Reihenfolge der Komponenten wird hier als Grundlage verwendet, damit keine zusätzlichen Kantenknicke und Kantenkreuzungen entstehen, wie wir sie in Abbildung 4.4a sehen. Das ist auch der Grund dafür, dass Komponenten mit einer Verbindung zur westlichen und nördlichen Portseite erst hier behandelt werden. Wie in der Abbildung kann es sonst passieren, dass eine Komponente mit einer Verbindung zu einem späteren Port zuerst platziert wird und dadurch zusätzliche Kantenknicke und Kantenkreuzungen entstehen. In Abbildung 4.4b sehen wir das Ergebnis, wenn wir die Komponenten erst in diesem Schritt platzieren.

Südverbindungen Als nächstes werden Komponenten, die eine Südportverbindung haben, platziert. Man beachte, dass Komponenten die sowohl Nord- als auch Südportverbindungen haben, hier nicht mehr betrachtet werden, da sie im vorangegangenen Schritt bereits platziert wurden. Die Komponenten mit Südportverbindungen verhalten sich so wie die Komponenten mit Nordportverbindungen und sollen somit aus denselben Gründen über die Ebenen verteilt werden. Die Komponenten mit Südverbindungen dürften auch vor den Komponenten mit Nordverbindungen behandelt werden. Es musste sich lediglich für eine der beiden Reihenfolgen entschieden werden.

Keine Portverbindungen Nun werden Komponenten ohne jegliche Portverbindungen platziert. Diese können, falls möglich, als Lückenfüller zwischen bereits platzierten Komponenten genutzt werden.



(a) In diesem Beispiel werden alle Komponenten mit Westverbindungen im ersten Schritt behandelt. Dabei können zusätzliche Kantenknicke und Kantenkreuzungen entstehen.

(b) In diesem Beispiel werden Komponenten mit Westverbindungen und zusätzlichen Nord- oder Südverbindungen in den jeweiligen späteren Schritten behandelt.

Abbildung 4.4. Probleme die entstehen können, falls alle Komponenten mit Westverbindungen im ersten Schritt behandelt werden.

Ostverbindungen Als letztes werden Komponenten, die ausschließlich Ostportverbindungen haben, platziert. Diese sollen so weit wie möglich rechts platziert werden, damit die Verbindung zu dem zugehörigen Port möglichst kurz ist.

Die in einem früheren Schritt berechnete angestrebte Graphengröße liefert beim Platzieren ein Indiz dafür, wann Komponenten in anderen Ebenen beginnen müssen, falls zum Beispiel die Anzahl der Knoten in einer Ebene überschritten wurde. Falls die angestrebten Werte nicht eingehalten werden können, kann weiterhin entschieden werden, an welcher Stelle das Platzieren am wenigsten zusätzliche Ebenen oder Knoten pro Ebene hinzufügt.

Die startende Ebene, in der eine Komponente bei der eben vorgestellten Reihenfolge beginnen soll, wird mit einer *Kostenfunktion* berechnet. Die Grundidee hierfür ist in Algorithmus 3 in Pseudocode gegeben. Zusätzlich wird in Abbildung 4.6 und Abbildung 4.7 ein Beispiel gegeben, wie Komponenten auf die Ebenen verteilt werden. Das Beispiel aus den Abbildungen wird nach dem Erklären des Algorithmus genauer beschrieben.

Im Algorithmus erhalten wir eine Liste von Komponenten, die in die vorhandenen Ebenen eingeteilt werden sollen. Dafür werden die Komponenten von links nach rechts durch die Ebenen geschoben. Als Erstes beginnt eine Komponente also in der ersten Ebene und es werden Kosten für diese Position berechnet. Als nächstes beginnt die Komponente in der zweiten Ebene und es werden erneut Kosten berechnet. Die Komponente wird so weit durch die Ebenen geschoben, bis sie eine Ebene hinter der angestrebten Anzahl an Ebenen beginnt. Die aktuellen Kosten, die für eine Komponente berechnet wird, setzen sich für die aktuelle Position aus der Anzahl der überschrittenen Knoten in einer Ebene und der Anzahl der überschrittenen Ebenen zusammen. Für jede überschrittene Ebene und für jede überschrittene Höhe wird dabei jeweils ein Kostenpunkt verteilt. Für jede Position wird zusätzlich geprüft, ob sie keine Kosten verursacht, weil die Komponente in diesem Fall direkt in der aktuellen Position platziert werden kann. Ansonsten wird überprüft, ob die aktuellen Kosten besser als die bisher besten Kosten sind. Dann wird sich die beginnende Ebene gemerkt, sodass am Ende die Position mit den wenigsten Kosten gewählt wird. Falls die angestrebten Ebenen oder

Algorithm 3: Kostenfunktion

```
Data: Liste der Komponenten einer Portseite
forall Ebene i aus angestrebten Ebenen do
    Berechne Kosten für aktuelle Komponente beginnend in Ebene i;
    if aktuelle Kosten = 0 then
        Fange direkt mit dem Platzieren der Komponente beginnend in Ebene i an.;
    else if aktuelle Kosten < beste Kosten then
        Setze aktuellen Beginn der Komponente auf Ebene i;
    else if aktuelle Kosten = beste Kosten then
        if Besseres Seitenverhältnis then
            Setze aktuellen Beginn der Komponente auf Ebene i;
Erhöhe angestrebte Ebenen um die überschrittenen Ebenen der gewählten Position;
Erhöhe angestrebte Höhe um die überschrittene Höhe der gewählten Position;
Platziere Komponente beginnend in der gespeicherten Ebene;
```

die angestrebte Höhe überschritten wird, werden auch die jeweiligen Werte entsprechend erhöht.

Beim Platzieren von Komponenten mit Verbindungen zur Nord- oder Südportseite ist eine Besonderheit zu beachten. Diese Komponenten sollen so gut wie möglich über die Ebenen verteilt werden. Hierfür gibt es zwei Verfahren, welche im Folgenden beschrieben werden.

Starkes Verteilen Bei diesem Verfahren wird zunächst geprüft, ob die Komponenten nebeneinander platziert werden können. Dafür wird die Anzahl der Ebenen von Komponenten der jeweiligen Portseite addiert und geprüft, ob diese die angestrebte Anzahl der Ebenen unterschreitet. Falls das der Fall ist, werden die Komponenten in diesem Schritt nebeneinander platziert. Ansonsten wird das *leichte Verteilen* genutzt. Ein Beispiel für das starke Verteilen sehen wir in Abbildung 4.5a. Die angestrebte Anzahl an Ebenen beträgt dort 2 und deshalb können alle Komponenten nebeneinander verteilt werden.

Bei diesem Verfahren ist zu beachten, dass wenn die Komponenten mit Verbindungen zur Nordseite verteilt wurden und eine Komponente existiert, die sowohl eine Nordverbindung, als auch eine Südverbindung hat, dann kann dieses Verfahren nicht mehr auf die Komponenten mit Südverbindungen angewendet werden, da bereits eine Komponente mit Verbindungen zur Südportseite platziert wurde. Die restlichen Komponenten der Südportseite müssen dann, falls es möglich ist, entsprechend ihrer Reihenfolge vor und hinter der bereits behandelten Komponente platziert werden, damit keine zusätzlichen Kantenkreuzungen und Kantenknicke entstehen.

Leichtes Verteilen Das zweite Verfahren prüft, wie viele Portverbindungen pro Ebene platziert werden müssen, damit alle Komponenten in das Diagramm passen. Es kann zum Beispiel wie in Abbildung 4.5b sein, dass mehr Ports auf der Nord- oder Südseite als angestrebte Ebenen vorhanden sind. Dann müssen mehrere Portverbindungen in einer Ebene platziert

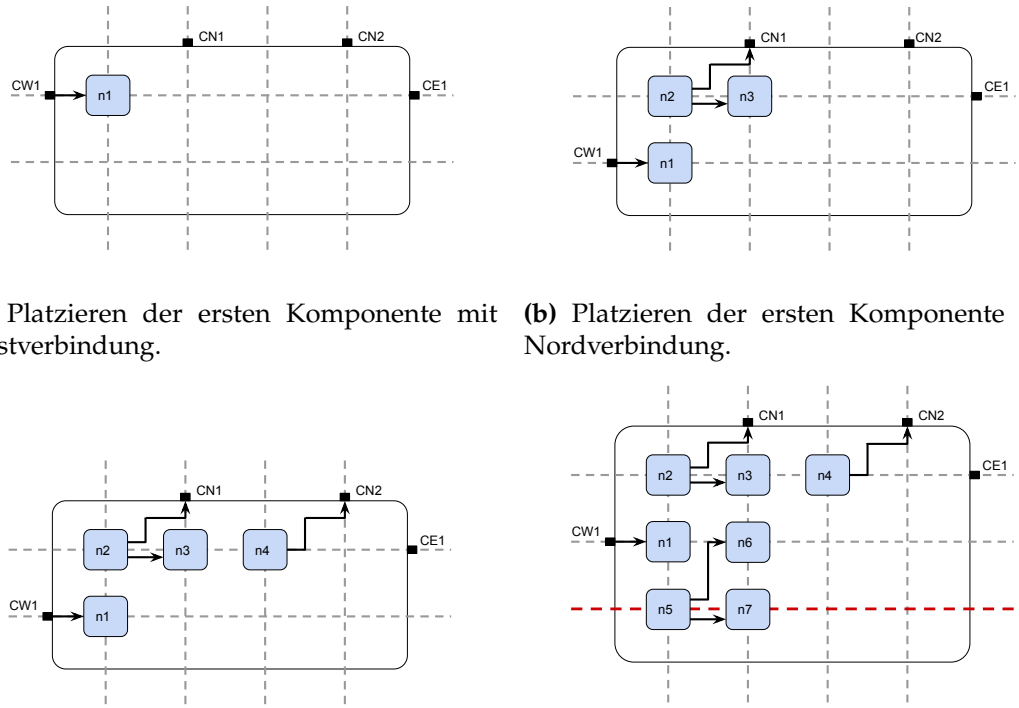
(a) Starkes Verteilen. **(b)** Leichtes Verteilen.

Abbildung 4.5. Verteilen von Nord- und Südports.

werden. In der Abbildung werden daher in der ersten Ebene zwei Komponenten mit einer Nordportverbindung platziert und erst dann wird die dritte Komponente mit einer Nordverbindung in die nächste Ebene einsortiert.

Es gibt eine weitere Besonderheit für das Platzieren von Komponenten mit ausschließlich Verbindungen zu östlichen Ports. Diese werden versucht von rechts nach links platziert zu werden und nicht von links nach rechts. Beim Berechnen der Kosten beginnt eine Komponente mit Ostportverbindungen als erstes eine Ebene rechts neben der angestrebten Anzahl an Ebenen und wird in jedem Schritt eine Ebene nach links verschoben. Weiterhin gibt es hierbei zusätzliche Kosten für die Entfernung zum Port. Das sorgt dafür, dass die Komponente so weit wie möglich rechts platziert wird und dadurch die Kantenlänge zum zugehörigen Port nicht zu lang wird.

In Abbildung 4.6 und Abbildung 4.7 sehen wir, wie das Verteilen von Komponenten an einem Beispiel funktioniert. Der Graph strebt dabei vier Ebenen und zwei Knoten pro Ebene an. Die Ebenen sind mit vertikal gestrichelten Linien und die Knoten pro Ebene mit horizontal gestrichelten Linien gekennzeichnet. Als Erstes wird die westliche Portseite behandelt und die Komponente bestehend aus dem Knoten $n1$ kann direkt ohne Kosten in der ersten Ebene platziert werden. Die Anzahl der Ebenen von Komponenten mit Verbindungen zu nördlichen Ports beträgt insgesamt drei: zwei Ebenen für die Komponente bestehend aus den Knoten $n2$ und $n3$ und eine Ebene für die Komponente bestehend aus dem Knoten $n4$. Die Komponenten mit Verbindungen zu nördlichen Ports können daher mit der Methode des *starken Verteilens* über die Ebenen verteilt werden. Zunächst wird die Komponente bestehend aus den Knoten $n2$ und $n3$ ohne Kosten in den ersten beiden Ebenen platziert. In Abbildung 4.6c wird die Komponente bestehend aus dem Knoten $n4$ auf Grund des starken Verteilens direkt in Ebene drei platziert. Da der Graph keine Komponenten mit Verbindungen zu Ports auf der südlichen Seite hat, wird im nächsten Schritt die Komponente bestehend aus $n5$, $n6$ und $n7$ platziert, welche keine Portverbindungen hat. Diese verursacht in der ersten Position in der Höhe die Kosten von 1 und wird deshalb um eine Ebene nach rechts verschoben. Dort verursacht sie weiterhin Kosten von 1 in der Höhe, weshalb sie weiter nach rechts geschoben wird. Wie wir in Abbildung 4.7b sehen, kann die Komponente nun ohne Kosten platziert werden. Im nächsten



(a) Platzieren der ersten Komponente mit Westverbindung.

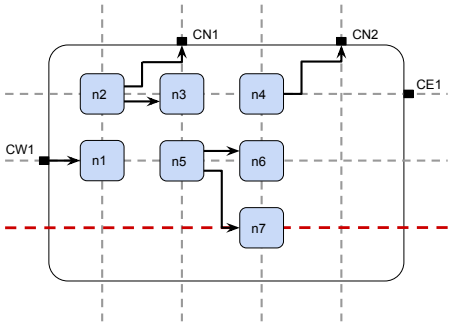
(b) Platzieren der ersten Komponente mit Nordverbindung.

(c) Platzieren der zweiten Komponente mit Nordverbindung.

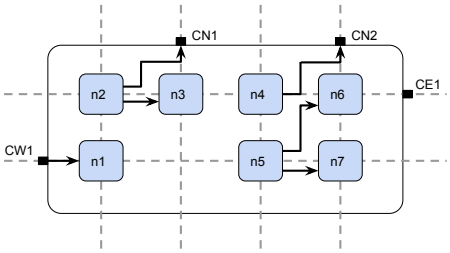
(d) Platzieren der Komponente ohne Portverbindungen. Hierbei entstehen Kosten von 1 in der Höhe.

Abbildung 4.6. Verteilen von Komponenten. Der Graph strebt vier Ebenen und zwei Knoten pro Ebene an. Fortführung in Abbildung 4.7.

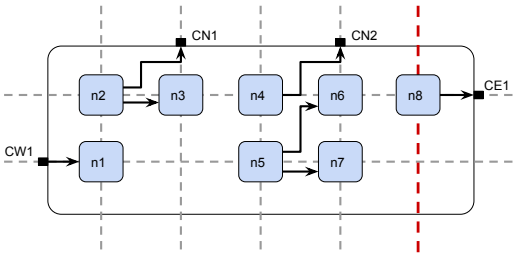
Schritt wird die Komponente bestehend aus dem Knoten $n8$ platziert. Da sie eine Verbindung zu einem östlichen Port hat, versucht der Algorithmus von rechts nach links zu platzieren. Zunächst wird die Komponente also eine Ebene rechts neben der angestrebten Zahl von Ebenen platziert. Somit erhält sie in dieser Position die Kosten von 1, da die angestrebte Zahl der Ebenen überschritten wird. Als nächstes wird die Komponente nach links verschoben und erhält die Kosten von 1, da die Anzahl der Knoten pro Ebene überschritten wird. Beim weiteren Verschieben nach links erhält die Komponente jetzt jedes mal Kosten, die sich um 1 erhöhen, da sie sich immer weiter von ihrer Portverbindung entfernt. Letztendlich wird die Position gewählt, die am weitesten rechts ist und die wenigsten Kosten verursacht. Das Ergebnis aus ELK mit dem neuen Ansatz sehen wir in Abbildung 4.7f.



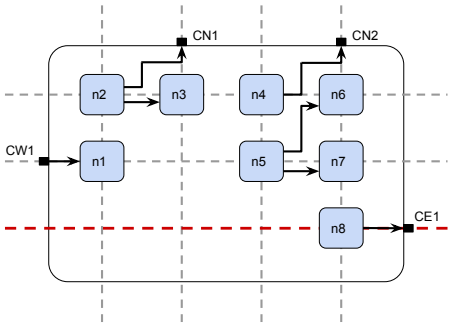
(a) Bei dieser Platzierung entstehen Kosten von 1 in der Höhe.



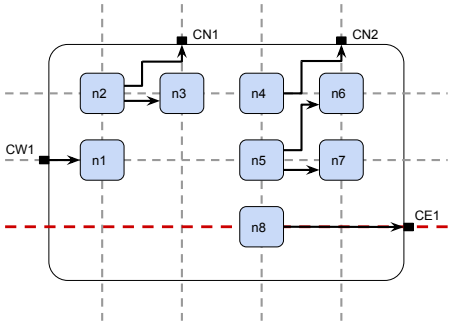
(b) Bei dieser Platzierung entstehen keine Kosten.



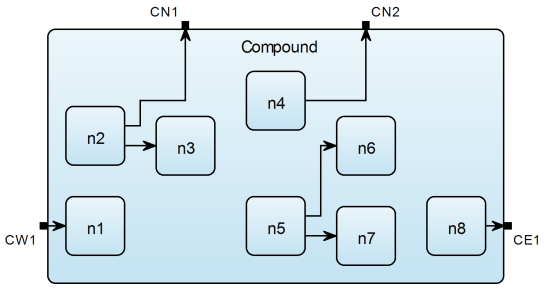
(c) Platzieren der Komponente mit Ostverbindung. Hierbei entstehen Kosten von 1 in der Zahl der Ebenen.



(d) Bei dieser Platzierung entstehen Kosten von 1 in der Höhe.



(e) Bei dieser Platzierung entstehen Kosten von 2. Einmal für die Höhe und einmal für die Entfernung zur Portverbindung.



(f) Resultierendes Diagramm aus ELK.

Abbildung 4.7. Fortführung für das Verteilen von Komponenten. Der Graph strebt vier Ebenen und zwei Knoten pro Ebene an.

Kapitel 5

Kreuzungsminimierung

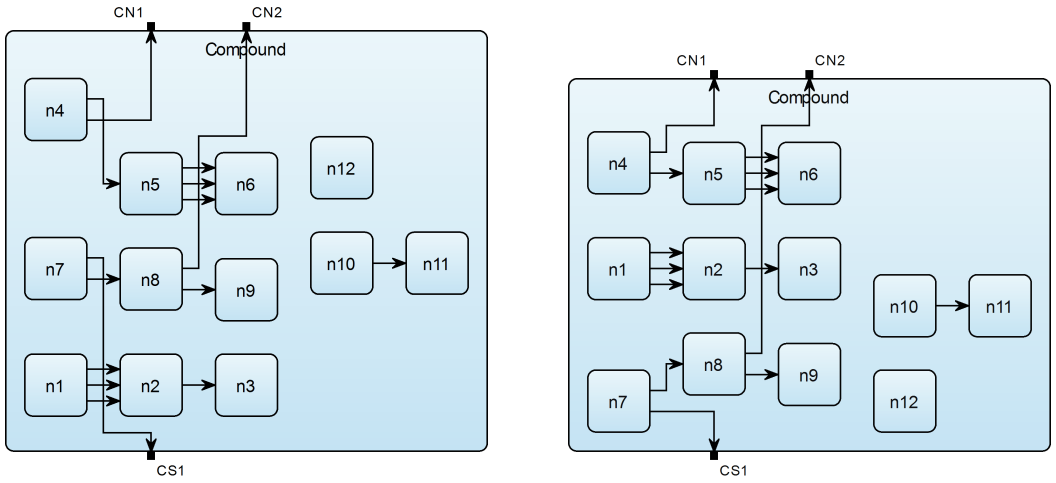
Wie in Abschnitt 2.3 erklärt wurde, ist es für die Lesbarkeit eines Diagramms wichtig, möglichst viele Kantenkreuzungen zu vermeiden. Der zweite Teil der Arbeit befasst sich daher mit diesem Thema.

Abbildung 5.1a zeigt mehrere Zusammenhangskomponenten und es gibt Verbindungen zu nördlichen und südlichen Ports des hierarchischen Knotens. In dem Beispiel kreuzen diese Verbindungen andere Zusammenhangskomponenten. Wie wir in Abbildung 5.1b sehen, kann das Vertauschen der y-Koordinaten von Komponenten in weniger Kreuzungen resultieren. Für eine Kreuzungsminimierung von Zusammenhangskomponenten müssen diese also in vertikaler Richtung vertauscht werden.

Die in Kapitel 3 vorgestellten Methoden können für die Kreuzungsminimierung von Zusammenhangskomponenten nicht genutzt werden. In Abbildung 5.1a sehen wir das Ergebnis von Layer Sweep angewandt auf einen Graphen mit Zusammenhangskomponenten. Wie wir in Abschnitt 2.2.1 gesehen haben, werden in dieser Phase die Ports des hierarchischen Knotens durch einen Dummyknoten repräsentiert. Das Problem von Layer Sweep ist, dass die nördlichen und südlichen externen Portdummies in der jeweiligen Ebene nicht am Anfang beziehungsweise am Ende platziert werden. Vielmehr werden sie ober- oder unterhalb der zugehörigen Komponente platziert, wodurch sie keine Kreuzungen mit anderen Komponenten verursachen. Erst in einem späteren Zwischenprozessor werden die Dummyknoten an den Anfang oder das Ende der Ebene verschoben, wodurch die Kreuzungen mit anderen Komponenten entstehen können. Layer Sweep vertauscht keine Komponenten in vertikaler Richtung, da die Kreuzungen mit anderen Komponenten nicht berücksichtigt werden. Als Hinweis ist hier noch zu erwähnen, dass die Knoten $n4$ und $n7$ jeweils eine Kantenkreuzung weniger haben könnten, wenn die Reihenfolge ihrer ausgehenden Kanten vertauscht wäre. Auf dieses Problem wird in Abschnitt 6.4 näher eingegangen.

Wegen der eben genannten Probleme mit Layer Sweep muss also eine neue Kreuzungsminimierung entwickelt werden, welche die externen Portverbindungen von Zusammenhangskomponenten berücksichtigen kann. In Abbildung 5.1b sehen wir den Graphen mit Berücksichtigung dieser Verbindungen und weniger Kreuzungen.

Zunächst werden im Folgenden Lösungsansätze vorgestellt und erklärt, ob es sinnvoll ist, diese Ansätze weiter zu verfolgen. Danach werden die Schritte erklärt, die für die neue Kreuzungsminimierung notwendig sind.



(a) Kreuzungen durch Layer Sweep minimiert mit acht Kreuzungen.

(b) Kreuzungen mit Berücksichtigung von Zusammenhangskomponenten minimiert mit vier Kreuzungen.

Abbildung 5.1. Auswirkung der Kreuzungsminimierung.

5.1. Lösungsansätze

Wie schon in Kapitel 4 beschrieben wurde, erhält diese Phase ebenfalls den kompletten Graphen des hierarchischen Knotens als Eingabe. Es gibt demnach wieder zwei Möglichkeiten, wie man mit dem Graphen umgehen kann.

Gesamter Graph Hierbei muss die Kreuzungsminimierung gleichzeitig sowohl innerhalb als auch zwischen den Komponenten Kreuzungen minimieren. Potentiell gute Algorithmen für diese Phase, wie zum Beispiel der Layer Sweep-Algorithmus, bieten dafür keine Möglichkeit. Generell dürfte ein gleichzeitiges Behandeln beider Probleme zu komplex sein.

In Zusammenhangskomponenten unterteilen Wenn der Graph wieder in die Zusammenhangskomponenten unterteilt wird, kann zunächst eine optimale Kreuzungsminimierung für jede Komponente berechnet werden, bevor Kreuzungen durch das Vertauschen von Komponenten weiter minimiert werden. Ein Algorithmus für diese Methode kann daher sowohl optimale Ergebnisse innerhalb der Komponenten als auch zwischen den Komponenten berechnen, weshalb diese Methode gewählt wurde.

Im Folgenden werden verschiedene Ideen erklärt, mit denen das Problem der Kreuzungsminimierung zwischen Komponenten gelöst werden könnte. Für die Kreuzungsminimierung innerhalb von Komponenten kann weiterhin Layer Sweep verwendet werden.

Permutation aller Komponenten Ein simpler Ansatz, um die Kreuzungen zwischen Komponenten zu reduzieren, ist es, jede mögliche Permutation aller Komponenten zu testen und für

jede Permutation die Kreuzungen zu zählen. Die Permutation mit der geringsten Anzahl an Kreuzungen wird dann gewählt. Hierdurch erhalten wir die bestmögliche Anordnung, da wir alle Kombinationen betrachten. Der Nachteil dieses Ansatzes ist, dass die Laufzeit in $\mathcal{O}(n!)$ liegt, wobei n der Anzahl der Komponenten entspricht.

Permutation aller überlappender Komponenten Die eben genannte Laufzeit kann verbessert werden, indem nur Permutationen zwischen Komponenten, die sich eine Ebene teilen, berechnet werden, da nur diese Komponenten Kreuzungen untereinander erzeugen können. Im schlechtesten Fall liegt die Laufzeit hierfür aber immer noch in $\mathcal{O}(n!)$, wobei n der Anzahl der Komponenten entspricht.

Hyperkanten mit Kreuzungsgraph Wird das Problem der Kreuzungsminimierung für Zusammenhangskomponenten etwas abstrakter betrachtet, kann es mit der Kreuzungsminimierung für *Hyperkanten* von Sander et al. [San04] verglichen werden. Hyperkanten sind mehrere Kanten, die dasselbe Ziel oder dieselbe Quelle haben und sich deshalb ein Kantensegment teilen. In Abbildung 5.2 sehen wir zwei gleiche Graphen mit Hyperkanten. Um die Kantenverbindungen besser unterscheiden zu können, wird die eine Kante gepunktet dargestellt. In der Abbildung fällt auf, dass das Tauschen der horizontal verlaufenden Kantensegmente eine unterschiedliche Anzahl an Kantenkreuzungen zur Folge hat.

In der Kreuzungsminimierung für Zusammenhangskomponenten sollen Kreuzungen vertikal verlaufender Kanten mit horizontal verlaufenden Kanten von Komponenten minimiert werden. Die Zusammenhangskomponenten können also als die horizontalen Kantensegmente und die Portverbindungen als vertikale Kantensegmente aus der Abbildung aufgefasst werden. Für das von Sander vorgestellte Verfahren werden die Komponenten dann paarweise verglichen. Somit ergibt sich eine Laufzeit, die in $\mathcal{O}(n^2)$ liegt, wobei n der Anzahl der Komponenten entspricht. Zusätzlich müssen, falls vorhanden, Zyklen in einem Kreuzungsgraph entfernt werden. Wie bereits in Abschnitt 2.2 erwähnt, liegt die Laufzeit hierfür in $\mathcal{O}(|E|)$, wobei E der Menge der Kanten entspricht. Der Kreuzungsgraph beinhaltet maximal $\frac{n(n-1)}{2}$ Kanten, wobei n der Anzahl der Komponenten entspricht. Auf Grund der besseren Laufzeit habe ich mich für dieses Verfahren entschieden und werde die nötigen Schritte hierfür im Folgenden genauer erläutern.

Als Ziele der neuen Kreuzungsminimierung ergibt sich als erstes das Unterstützen von Zusammenhangskomponenten. Für ein zweites Ziel schauen wir uns die Abbildung 5.3a an. Dort sehen wir zwei verschachtelte Komponenten, da in der zweiten Ebene ein Knoten aus der unteren Komponente zwischen den Knoten aus der oberen Komponente eingeordnet ist. Durch die Kreuzungsminimierung sollte dies verhindert werden. In Abbildung 5.3b sehen wir das Ergebnis des Graphen ohne solche Verschachtelungen. Durch die klarere Trennung der Komponenten, können sie einfacher identifiziert werden.

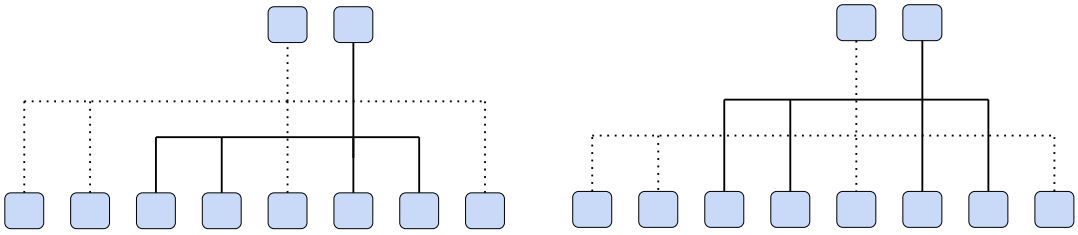
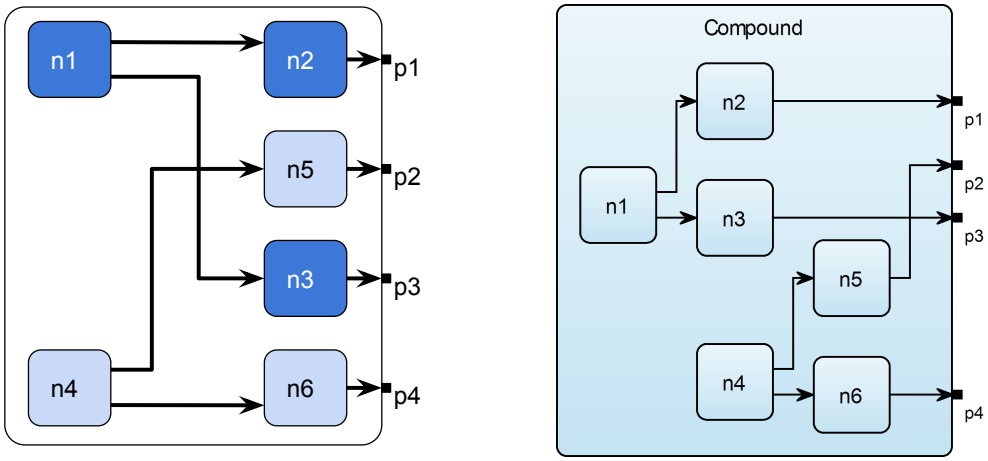


Abbildung 5.2. Zwei Repräsentationen eines Graphen mit denselben Hyperkanten-Segmenten. Links mit zwei Kreuzungen und rechts mit fünf Kreuzungen.



(a) Lesbarkeit wird verschlechtert, falls Komponenten verschachtelt sind.

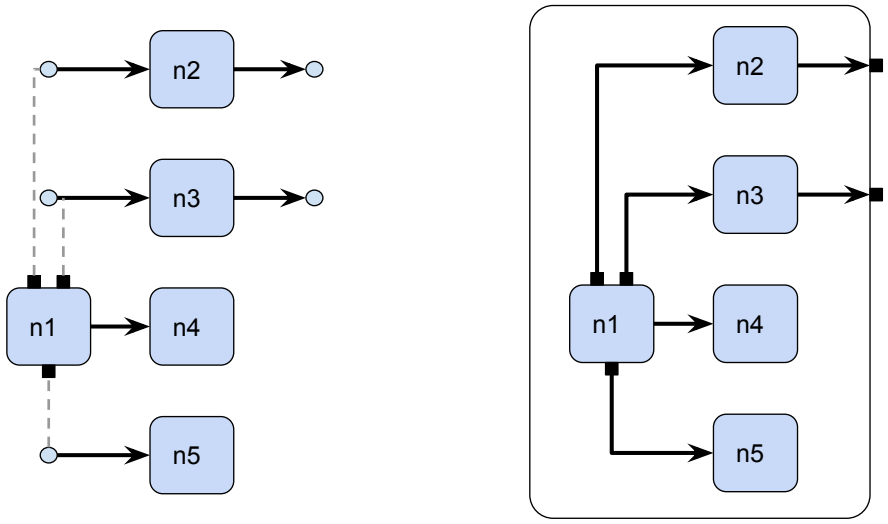
(b) Komponenten können einfacher identifiziert werden, wenn keine Verschachtelungen vorhanden sind.

Abbildung 5.3. Prävention von Verschachtelungen.

5.2. Schritte der Kreuzungsminimierung

5.2.1. Erkennen von Komponenten

Da die Phasen im ebenenbasierten Ansatz getrennt voneinander nutzbar sein sollen, müssen für diese Phase die Zusammenhangskomponenten erneut erkannt werden. Durch Zwischenprozessoren können, seit dem Erkennen von Komponenten in der vorherigen Phase, weitere Dummyknoten hinzugefügt worden sein. Interessant hierfür sind die Dummyknoten, die für Nord- und Südports von normalen Knoten erstellt wurden. Wie in Abschnitt 2.2.1 beschrieben wurde, haben diese Dummyknoten keine Verbindung zu der Komponente, mit der sie eigentlich verbunden sind. Schauen wir uns dafür Abbildung 5.4a an. Die grauen gestrichelten Linien symbolisieren die eigentliche Portverbindung. Mit einer Tiefensuche würden wir in dem Beispiel vier Zusammenhangskomponenten finden, aber das Beispiel enthält nur



(a) In dieser Phase existieren Dummyknoten für Nord- und Südports. Dadurch bestehen keine expliziten Kantenverbindungen, obwohl die Knoten später miteinander verbunden sind.

(b) In späteren Phasen werden die Dummyknoten wieder aufgelöst und die expliziten Kantenverbindungen werden somit wiederhergestellt.

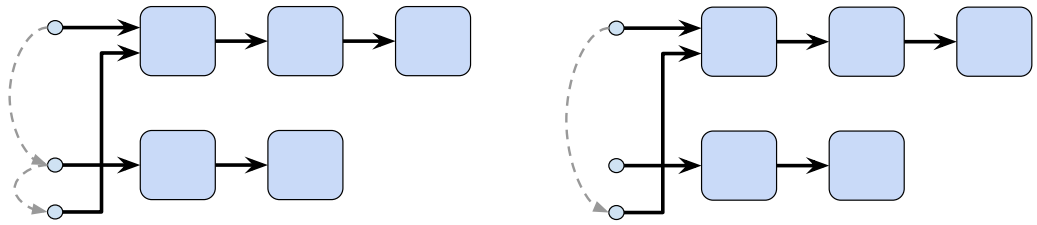
Abbildung 5.4. Erkennen von Komponenten mit Nord- und Südportdummyknoten.

eine. Während der Tiefensuche muss dieser Fall also beachtet werden. Eine entsprechende Eigenschaft speichert für die Ports des normalen Knotens die zugeordneten Dummyknoten und andersherum. Dadurch können diese Knoten ohne explizite Kantenverbindung trotzdem in die richtige Zusammenhangskomponente eingeordnet werden.

5.2.2. Kreuzungsminimierung für jede Komponente

Um eine Kreuzungsminimierung für jede einzelne Komponente ausführen zu können, muss wieder ein kopierter Graph erstellt werden. Für alle kopierten Objekte müssen auch die Eigenschaften kopiert werden, die auf den originalen Objekten gesetzt wurden. Hierbei gibt es einige Besonderheiten für zwei Dummyknoten zu beachten, die im Folgenden erklärt werden.

Externe Ports Die östlichen und westlichen externen Portdummies haben jeweils eine Eigenschaft gesetzt, durch die sich der in der Reihenfolge nächste externe Portdummy identifizieren lässt. Da wir eine feste Reihenfolge der Ports haben, sorgt diese Eigenschaft in der Kreuzungsminimierung für das Berücksichtigen dieser Reihenfolge. Das Problem hierbei ist, dass diese Eigenschaft beim Kopieren eine Referenz auf den originalen Knoten hat und nicht auf den repräsentierenden kopierten Knoten. Daher muss in diesem Schritt



(a) Die Eigenschaft zeigt auf den in der Reihenfolge nächsten Portdummy.

(b) Die Eigenschaft muss für die Kreuzungsminimierung auf den nächsten Portdummy derselben Komponente zeigen.

Abbildung 5.5. Eigenschaft für die Reihenfolge der externen Portdummyknoten.

die Eigenschaft neu gesetzt werden. In Abbildung 5.5a sehen wir, dass diese Eigenschaft auch auf Portdummies anderer Komponenten zeigen kann. Da wir für die Komponenten zunächst jeweils eine Kreuzungsminimierung durchführen, darf diese Eigenschaft nicht auf Knoten aus anderen Komponenten zeigen, da diese bei der Kreuzungsminimierung für die einzelnen Komponenten nicht bekannt sind. Nun kann, wie in dem Beispiel zu sehen ist, der Portdummy der zweiten Komponente wieder auf einen Portdummy der ersten Komponente zeigen. Diese Transitivität muss also aufgelöst werden, sodass, wie in Abbildung 5.5b zu sehen ist, ein externer Portdummy auf den jeweils nächsten Portdummy derselben Komponente zeigt.

Nord- und Südports Für Nord- und Südportdummies gibt es mehrere Eigenschaften, die auf Knoten aus dem originalen Graphen zeigen und auf deren Kopie zeigen müssen. Eine Eigenschaft zeigt von Ports eines normalen Knoten auf den zugehörigen Portdummyknoten und es existiert eine weitere Eigenschaft in die andere Richtung. Diese beiden Eigenschaften müssen auf die kopierten Objekte zeigen. Weiterhin gibt es für die Portdummies eine Eigenschaft, die eine Reihenfolge der Dummyknoten widerspiegelt. Als letztes gibt es noch eine Eigenschaft auf dem normalen Knoten, die alle Nord- und Südportdummies für die Barycenter-Berechnungen in der Kreuzungsminimierung markiert. Erst wenn die eben genannten Eigenschaften auf die richtigen Knoten zeigen, kann die Kreuzungsminimierung mit Nord- und Südports umgehen. Anderenfalls können zum Beispiel Kreuzungen zwischen Kanten und Knoten entstehen. Weitere Beispiele und Erklärungen zu diesem Thema wurden von Schulze et al. [SSH14] gegeben und können bei Bedarf nachgelesen werden.

Mit den gesetzten Eigenschaften in dem kopierten Graphen kann nun eine Kreuzungsminimierung auf jedem kopierten Graphen ausgeführt werden. Hierfür wird für jeden Graphen der Layer Sweep-Algorithmus ausgeführt. Die daraus resultierende Reihenfolge der Knoten wird zunächst in einer Liste für jede Komponente abgespeichert und später in dieser Phase berücksichtigt. Layer Sweep berechnet auch eine Reihenfolge der Ports für jeden Knoten.

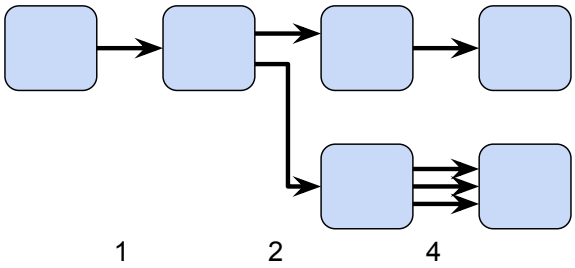


Abbildung 5.6. Zählen der Kanten zwischen Ebenen.

Diese Reihenfolge wird in diesem Schritt bereits für alle originalen Knoten übernommen. Wenn ein Knoten mehrere ein- oder ausgehende Kanten hat, dann ist erst durch eine korrekte Reihenfolge der Ports gewährleistet, dass die Kantenkreuzungen minimiert sind. Ansonsten können zusätzliche Kantenkreuzungen wie bei den Knoten $n4$ und $n7$ aus Abbildung 5.1a entstehen.

5.2.3. Vorbereitung für das Zählen der Kantenkreuzungen

Damit die Kreuzungsminimierung später Kantenkreuzungen zählen kann, muss für jede Komponente gezählt werden, wie viele Kanten von Ebene zu Ebene übergehen. In Abbildung 5.6 sehen wir eine Zusammenhangskomponente zusammen mit der Anzahl der Kanten zwischen den Ebenen. Dadurch kann später in der Kreuzungsminimierung für Kanten, die eine Verbindung zu einem nördlichen oder südlichen hierarchischen Port haben, die Zahl der entstehenden Kreuzungen leichter gezählt werden.

5.2.4. Erstellen des Kreuzungsgraphen

Im nächsten Schritt wird wie bei dem Verfahren von Sander et al. [San04] ein Kreuzungsgraph erstellt, durch den später die Kreuzungen minimiert werden können. Der Kreuzungsgraph besteht aus einem Knoten für jede Komponente und aus Kanten zwischen diesen Knoten. Die Kanten stellen eine Abhängigkeit zwischen den jeweiligen Komponenten dar und sind mit einer Nummer gewichtet. Falls der Kreuzungsgraph Zyklen enthält, müssen diese entfernt werden, damit eine Reihenfolge der Komponenten abgeleitet werden kann. Die Gewichtung einer Kante repräsentiert die Anzahl der zusätzlich entstehenden Kreuzungen, falls die Abhängigkeitskante entfernt wird. Für das Entfernen der Zyklen haben wir bereits das FAS-Problem in der ersten Phase des ebenenbasierten Ansatzes in Abschnitt 2.2 kennengelernt. In Abbildung 5.7 sehen wir den Kreuzungsgraphen für den Graphen aus Abbildung 5.1.

In Algorithmus 4 ist der Algorithmus für das Erstellen des Kreuzungsgraphen in Pseudocode dargestellt. Wie das Zählen von Kreuzungen realisiert wurde, wird im nächsten Abschnitt erklärt. Für das Erstellen des Kreuzungsgraphen werden zunächst die Komponenten paarweise miteinander verglichen. Hierbei werden aber nur die Komponenten miteinander

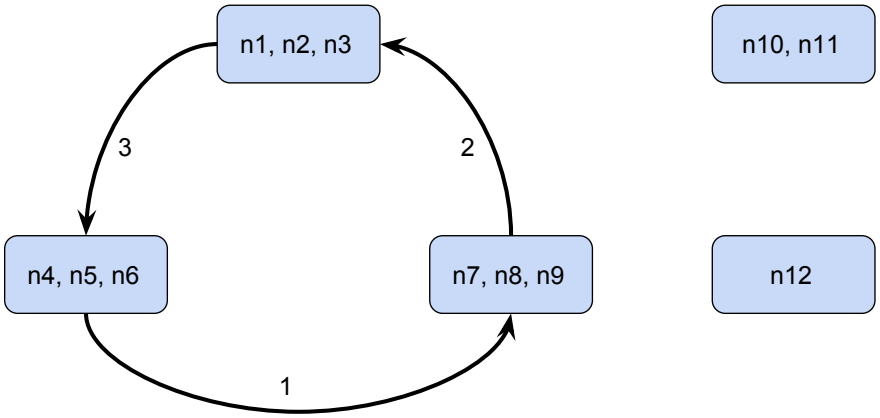


Abbildung 5.7. Resultierender Kreuzungsgraph.

verglichen, die sich mindestens eine Ebene teilen. Anderenfalls kann keine Kreuzung zwischen den Komponenten entstehen und somit besteht keine Abhängigkeit zwischen den Komponenten. Beim paarweisen Vergleichen von zwei Komponenten $C1$ und $C2$ wird zunächst $C1$ unterhalb von $C2$ platziert und die Kreuzungen werden gezählt. Anschließend werden die Komponenten vertauscht, sodass $C2$ unterhalb von $C1$ platziert wird und die Kreuzungen werden erneut gezählt. Danach wird abhängig von der berechneten Anzahl an Kreuzungen eine Kante zwischen den beiden Komponenten in den Kreuzungsgraph hinzugefügt. Wenn die Zahl der Kreuzungen in beiden Konstellationen gleich groß ist, wird zusätzlich geprüft, ob eine der Komponenten eine Verbindung zu einem nördlichen oder südlichen Port hat, damit die Komponente dementsprechend oberhalb oder unterhalb der anderen Komponente eingeordnet wird, um kürzere Kantenlängen zu erhalten.

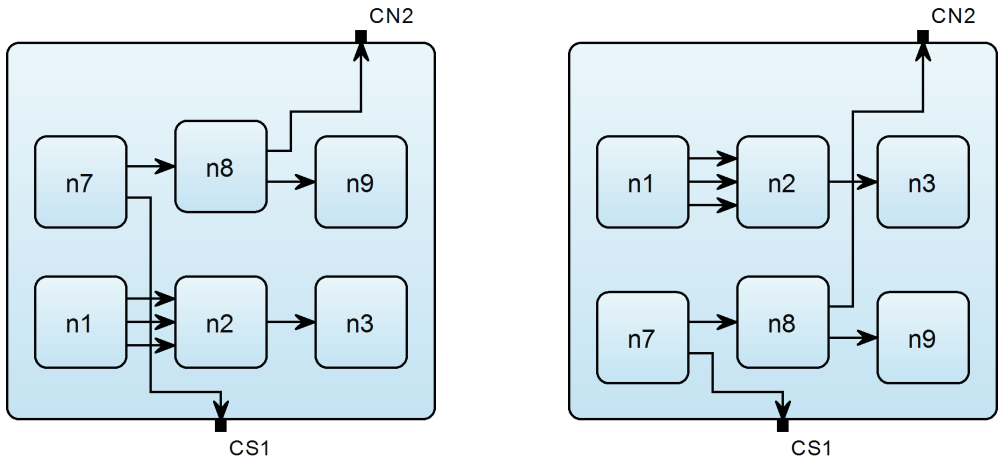
In Abbildung 5.8 sehen wir ein Beispiel von zwei Komponenten aus Abbildung 5.1, die miteinander verglichen werden. In der ersten Konstellation wird $C1$, bestehend aus den Knoten $n1$, $n2$ und $n3$, unterhalb von $C2$, bestehend aus den Knoten $n7$, $n8$ und $n9$, platziert. Hierbei können wir drei Kreuzungen erkennen, welche in der Variable $kC1Unten$ abgespeichert werden. In der zweiten Konstellation entsteht nur eine Kreuzung, welche in die Variable $kC2Unten$ geschrieben wird. Da $kC1Unten > kC2Unten$ ist, wird eine Abhängigkeit von $C2$ zu $C1$ mit den Kosten $3 - 1$ eingefügt, wie wir es auch in Abbildung 5.7 sehen können.

Kreuzungen zählen

Für das Zählen von Kreuzungen müssen zwei Fälle unterschieden werden. Diese beiden Fälle werden am Ende addiert und ergeben somit die Gesamtzahl an Kreuzungen für die jeweilige Konstellation der beiden Komponenten.

Algorithm 4: Erstellen des Kreuzungsgraphen

```
Data: Paarweise Komponenten
Result: Kreuzungsgraph
if Komponenten teilen sich eine Ebene then
    if Mindestens eine Komponenten hat Portverbindungen then
        kC1Unten = zähle Kreuzungen zwischen C1 und C2;
        kC2Unten = zähle Kreuzungen zwischen C2 und C1;
        if kC1Unten < kC2Unten then
            Abhängigkeit von C1 zu C2 mit Kosten (kC2Unten - kC1Unten);
        else if kC1Unten = kC2Unten then
            if C2 hat nur Nordverbindungen then
                Abhängigkeit von C1 zu C2 mit Kosten 0;
            else if C1 hat nur Südverbindungen then
                Abhängigkeit von C1 zu C2 mit Kosten 0;
            else
                Abhängigkeit von C2 zu C1 mit Kosten 0;
        else
            Abhängigkeit von C2 zu C1 mit Kosten (kC1Unten - kC2Unten);
```



(a) C1 ist unterhalb von C2. Hierbei entstehen drei Kreuzungen.

(b) C2 ist unterhalb von C1. Hierbei entsteht eine Kreuzung.

Abbildung 5.8. Vergleichen von zwei Komponenten.

Nord- und Südports Hierbei wird über die Nord- und Südports der beiden Komponenten iteriert und anhand des Indexes der externen Portdummies können die Kreuzungen aus den vorher gezählten Kantenverbindungen ausgelesen und addiert werden. Zuvor muss zwischen ein- und ausgehender Kante unterschieden werden, da die Kreuzung entweder

zwischen den Knoten der vorherigen oder nachfolgenden Ebene des externen Portdummies entstehen. In Abbildung 5.8b sehen wir zum Beispiel, dass $n7$ eine ausgehende Kante zum externen Portdummy hat, weshalb die Kreuzungen vor dem externen Portdummy entstehen.

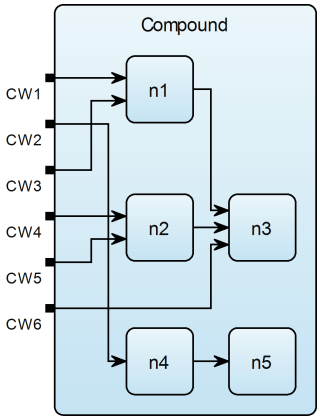
West- und Ostports Weiterhin können Komponenten Verbindungen mit hierarchischen Ports auf der West- oder Ostseite haben. Hierfür wird ein anderes Verfahren zum Kreuzungszählen verwendet, da diese Kanten horizontal und nicht vertikal verlaufen. In Abbildung 5.9a sehen wir ein Beispiel für einen solchen Graph. In Abbildung 5.9b sehen wir die Graphenrepräsentation, in der die Kreuzungen gezählt werden. Für jede Komponente wird ein repräsentierender Knoten erstellt. Hierbei reicht ein repräsentierender Knoten für jede Komponente, da die Komponenten, wie in Abbildung 5.3 beschrieben wurde, nicht verschachtelt sind. Die Ports werden ebenfalls durch einen Knoten repräsentiert und erhalten eine Kantenverbindung zu der jeweiligen Komponente, mit der sie verbunden sind. Für das Zählen der Kreuzungen werden die Kantenverbindungen zu der oberen Komponente gezählt. Danach wird durch die Ebene mit den Knoten, die die Ports repräsentieren, iteriert. Falls der Knoten eine Verbindung zu der oberen Komponente hat, wird die Zahl der Verbindungen zur oberen Komponente reduziert. Falls der Knoten allerdings eine Verbindung zur unteren Komponente hat, dann wird auf die bereits gezählten Kreuzungen die Zahl der aktuellen Verbindungen zur oberen Komponente addiert. Dadurch erhalten wir die Anzahl der Kantenkreuzungen für die westliche Portseite. Das Zählen der Kreuzungen für die östliche Portseite verhält sich analog.

Schauen wir uns für ein besseres Verständnis noch einmal Abbildung 5.9b an. Die obere Komponente hat fünf Verbindungen. Zuerst wird $CW1$ betrachtet. Da der Knoten eine Verbindung zur oberen Komponente hat, wird die Zahl der Verbindungen auf vier reduziert. Als nächstes wird $CW2$ betrachtet. Dieser hat eine Verbindung zur unteren Komponente und somit werden vier Kreuzungen gezählt. Danach werden die Verbindungen zur oberen Komponente nur noch reduziert, da wir keine zweite Verbindung zur unteren Komponente haben.

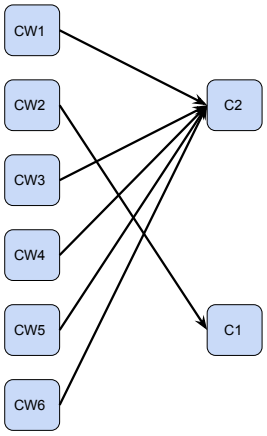
Zyklen entfernen

Wie in Abbildung 5.7 zu sehen ist, kann der erstellte Kreuzungsgraph Zyklen enthalten. Dadurch kann keine eindeutige Reihenfolge für die Komponenten festgelegt werden, da durch einen Zyklus keine topologische Reihenfolge gefunden werden kann. Der Kreuzungsgraph muss also für das Finden einer Komponentenreihenfolge azyklisch sein. Das Entfernen von Zyklen ist für gerichtete Graphen ein NP-schweres Problem und wird, wie in der ersten Phase in Abschnitt 2.2, mit einem gierigen Algorithmus, basierend auf Eades et al. [ELS93] und Battista et al. [DET+99], gelöst.

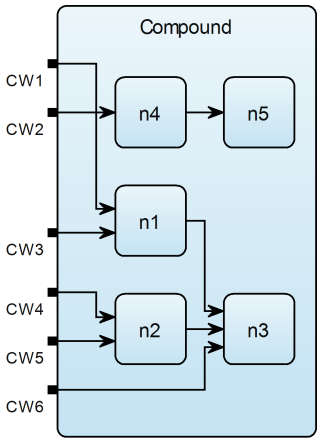
Durch den Algorithmus wird aus dem Kreuzungsgraph in Abbildung 5.7 die Kante mit den Kosten von 1 entfernt. Dies entspricht der Kante mit den geringsten Kosten, woraufhin



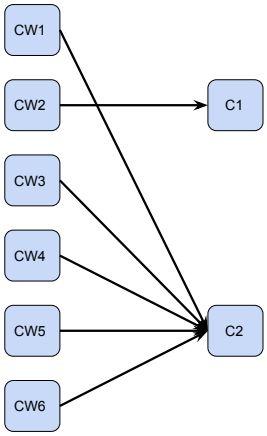
(a) In dieser Konstellation enthält der Graph vier Kreuzungen.



(b) Graph für das Zählen der Kreuzungen.



(c) In dieser Konstellation enthält der Graph eine Kreuzung.



(d) Graph für das Zählen der Kreuzungen.

Abbildung 5.9. Zählen der westlichen und östlichen Kantenkreuzungen.

die Komponentenreihenfolge berechnet werden kann. Diese Reihenfolge hat die minimale Anzahl an Kreuzungen für den erhaltenen Graph.

Komponentenreihenfolge festlegen

Wie wir in dem Kreuzungsgraph aus Abbildung 5.7 sehen, müssen nicht alle Knoten miteinander verbunden sein. Die unverbundenen Komponenten haben keine Abhängigkeit zueinander, da sie entweder in keinen gemeinsamen Ebenen sind oder keine Portverbindungen haben. Für alle Zusammenhangskomponenten aus dem Kreuzungsgraph wird zunächst für jeden Knoten

ein Rang berechnet. Die Ränge ergeben sich, indem die Zusammenhangskomponenten des Kreuzungsgraphen in umgekehrter Richtung durchlaufen werden. Danach werden die Komponenten anhand der Ränge sortiert. Dadurch ist eine finale Reihenfolge der Komponenten vorgegeben. In einem letzten Schritt wird die Reihenfolge auf die Komponenten appliziert, wobei die in Abschnitt 5.2.2 berechnete Reihenfolge der Knoten berücksichtigt wird. Dadurch, dass Komponenten nacheinander behandelt werden, wird eine Verschachtelung von Komponenten automatisch vermieden. Das Ergebnis der Kreuzungsminimierung für den Beispielgraphen wurde bereits in Abbildung 5.1b gezeigt.

Kapitel 6

Evaluation

In diesem Kapitel werden die in dieser Arbeit entstandenen Algorithmen evaluiert und mit bereits vorhandenen Algorithmen aus ELK verglichen. Mein neuer Ansatz für die Ebenenzuweisung wird in diesem Kapitel als *Components*-Algorithmus bezeichnet und die in dieser Arbeit entwickelte Kreuzungsminimierung wird *Components Crossing Minimizer* genannt. In der Evaluation werden zwei Fragestellungen analysiert:

1. Wie vergleicht sich der Components Crossing Minimizer mit Layer Sweep bei der Verwendung von Zusammenhangskomponenten?
2. Wie schneidet der Components-Algorithmus im Vergleich mit den anderen vorgestellten Algorithmen der Ebenenzuweisung aus Kapitel 3 ab? Inwiefern wird ein angestrebtes Seitenverhältnis eingehalten?

Dafür wird zunächst erklärt, mit welchen Testgraphen gearbeitet wurde und welche Analysen verwendet wurden. Danach werden die beiden Fragestellungen evaluiert und im letzten Abschnitt werden Probleme meiner Ansätze identifiziert und diskutiert. Durch die Behebung der Probleme könnten bessere Ergebnisse erzielt werden.

6.1. Testumgebung

Für das in dieser Arbeit behandelte Problem stehen nicht genügend reale Beispiele zur Verfügung, da die meisten zur Verfügung gestellten Beispiele aus zu wenigen Zusammenhangskomponenten bestehen. Deshalb wurden für die Evaluation zufällig generierte Graphen verwendet, die mit ELK erstellt wurden. Es wurden 250 Testgraphen mit 1 bis 50 Zusammenhangskomponenten generiert, welche jeweils aus 5 bis 20 Knoten bestehen. Weiterhin beinhaltet jeder Graph zwischen 0 und 50 Kanten zu hierarchischen Ports, welche über die Portseiten und Zusammenhangskomponenten verteilt sind. Eine solche hierarchische Kante hat eine Chance von 50%, eine ein- oder ausgehende Kante einer Zusammenhangskomponente zu sein.

Zur Untersuchung verschiedener Eigenschaften der Graphen, wurden für die Evaluation folgende Analysen aus GrAna verwendet.

Anzahl der Knoten Diese Analyse berechnet die Anzahl der Knoten in dem Graphen. Die Ergebnisse wurden für die x-Achse aller Diagramme in diesem Kapitel verwendet. Durch eine erhöhte Anzahl an Knoten erhöht sich in den Testgraphen auch die Anzahl der

Kanten und hierarchischen Ports. Die Diagramme werden folglich komplexer, weshalb dieser Aspekt für die x-Achse geeignet ist.

Kantenkreuzungen Der wichtigste Aspekt dieser Analyse ist die Anzahl aller Kantenkreuzungen im Graphen. Diese wird für die Evaluation der Kreuzungsminimierung benötigt.

Fläche Für die Fläche wird in der Analyse sowohl die Breite und Höhe der Zeichenfläche, als auch deren Produkt ausgegeben. Die Breite und Höhe wird jeweils in Pixeln bemessen.

Seitenverhältnis Diese Analyse berechnet das Seitenverhältnis des Graphen, indem die Breite durch die Höhe der Zeichenfläche geteilt wird.

Ungenutzte Fläche Bei dieser Analyse wird die durch alle Knoten und Kanten bedeckte Fläche in ein Verhältnis mit der Gesamtfläche gesetzt. Die ungenutzte Fläche wird dabei in Prozent angegeben.

Kantenlänge Bei dieser Analyse erhalten wir Informationen über die maximale und durchschnittliche Kantenlänge aller Kanten in dem Graph. Die Ergebnisse werden in Pixeln angegeben.

6.2. Kreuzungsminimierung

Der Vergleich von Layer Sweep und dem Components Crossing Minimizer wird durch die Betrachtung der Kantenkreuzungen bewertet. In Abbildung 6.1 sehen wir die Ergebnisse der Analyse. Wir sehen den Components-Algorithmus einmal mit Layer Sweep und einmal mit dem Components Crossing Minimizer. Ebenfalls sehen wir den Cell Packing-Algorithmus. Es ist zu erkennen, dass der Components Crossing Minimizer wesentlich mehr Kreuzungen entfernen konnte als Layer Sweep. Bei einer hohen Anzahl an Knoten konnten die Kreuzungen fast um ein Fünftel reduziert werden. In einem weiteren Diagramm, welches in der Arbeit nicht zu sehen ist, habe ich die Kreuzungen von Network Simplex mit beiden Kreuzungsminimierungen untersucht. Mit dem Components Crossing Minimizer konnten die Kreuzungen bei einer hohen Zahl an Knoten sogar um ein Fünfzehntel reduziert werden. Die Werte von Cell Packing sind als Referenzwerte zu deuten. Diese können bei der Porteinschränkung Fixed Order in der Regel nicht erreicht werden, da bei einer freien Reihenfolge von Ports zusätzliche Kreuzungen vermieden werden können.

Die Ergebnisse der Analyse beinhalten allerdings noch vier Fälle, bei denen Layer Sweep ebenso viele Kreuzungen erzeugt hat wie der Components Crossing Minimizer. Dies ist jedoch ein zu erwartendes Ergebnis, da beim Betrachten dieser Graphen aufgefallen ist, dass nur jeweils eine Zusammenhangskomponente vorhanden war. Da für jede Komponente Layer Sweep ausgeführt wird und keine Kreuzungen zwischen Komponenten minimiert werden müssen, stimmen die Ergebnisse überein.

Beim weiteren Betrachten einiger Testgraphen ist aufgefallen, dass die Diagramme mit Layer Sweep bis auf den eben genannten Fall schlechter lesbar sind. Es bestehen zu viele

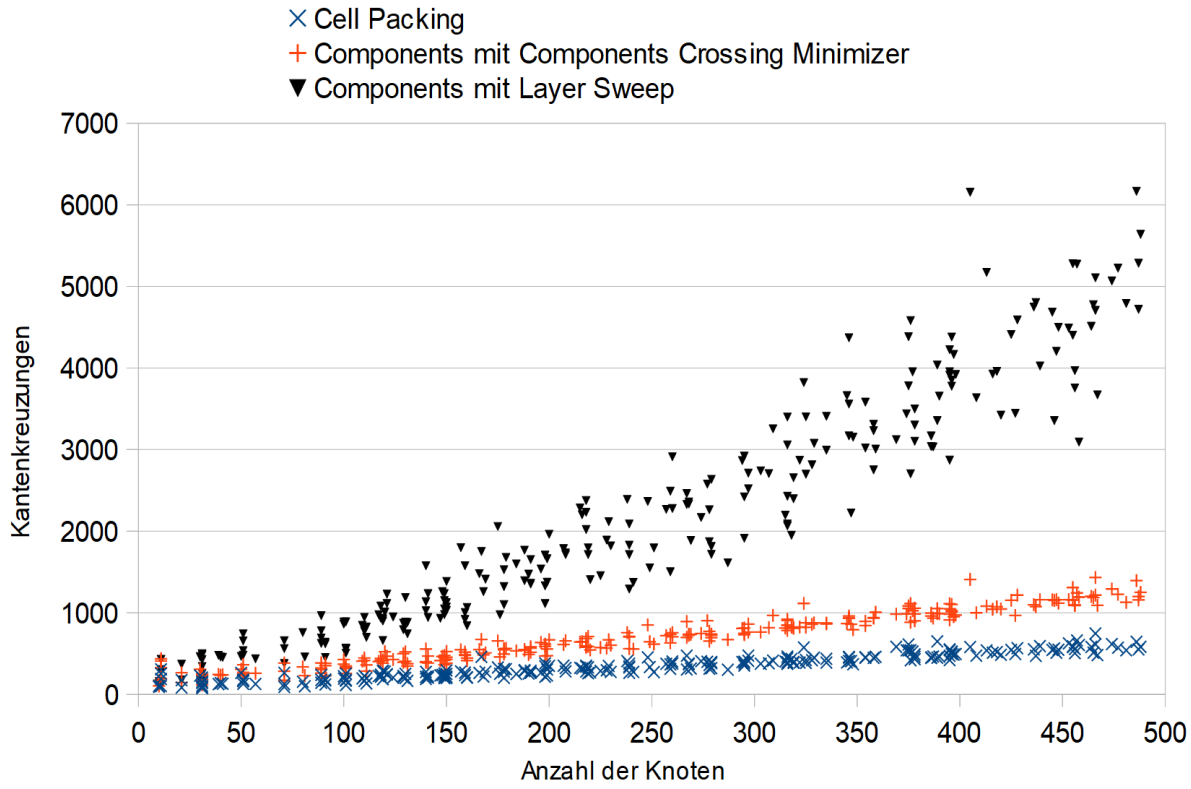


Abbildung 6.1. Evaluierung der Kreuzungsminimierung.

Kreuzungen und die Knoten der Komponenten sind in den einzelnen Ebenen miteinander verschachtelt. Komponenten können also nicht ohne Weiteres unterschieden werden. Mit dem Components Crossing Minimizer hingegen konnten Komponenten deutlicher erkannt werden und nur bei dicht nebeneinander platzierten Komponenten musste genauer betrachtet werden, ob eine Verbindung zwischen den beiden Komponenten besteht. Der Components Crossing Minimizer erfüllt in dieser Hinsicht somit seinen Zweck und wird in der weiteren Evaluation als Grundlage verwendet.

6.3. Ebenenzuweisung

Für die Evaluation der Ebenenzuweisung wurde der Components Crossing Minimizer, wie zuvor erwähnt, als Grundlage verwendet, damit die Diagramme einer lesbaren Form entsprechen. Die Werte von Cell Packing werden hier ebenfalls als Referenzwerte verwendet. Da Cell Packing verwendet wird, falls die Porteinschränkungen auf Fixed Side oder Free gesetzt sind, liefert ein Vergleich mit Cell Packing ein gutes Indiz dafür, ob der Components-Algorithmus gute Werte erreicht. Ebenfalls wichtig ist der Vergleich zwischen dem Components-Algorithmus und dem Network Simplex-Algorithmus, da Network Sim-

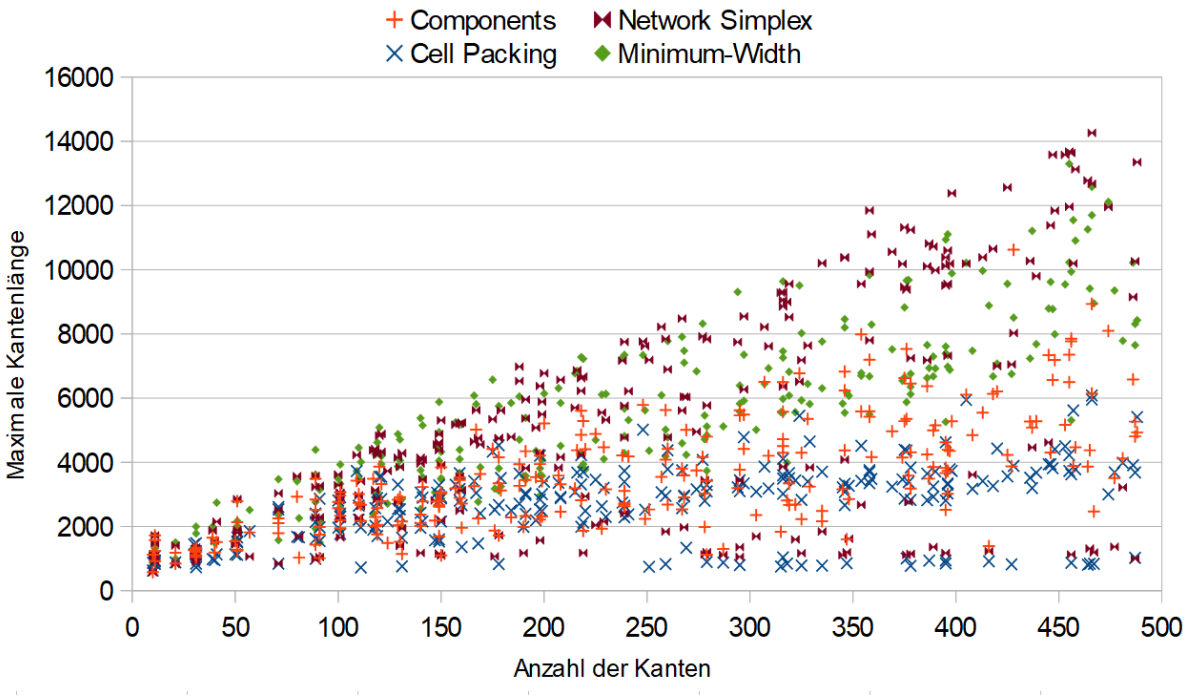


Abbildung 6.2. Evaluierung der maximalen Kantenlänge.

plex aktuell der Algorithmus ist, der verwendet wird, falls die Porteinschränkungen nicht auf FIXED SIDE oder FREE gesetzt sind. Die anderen Algorithmen aus Kapitel 3 wurden hier ebenfalls betrachtet, um deren Eignung für Graphen mit Zusammenhangskomponenten zu analysieren.

Evaluierung der Kantenlängen

Zunächst betrachten wir die durchschnittliche und maximale Kantenlänge der einzelnen Algorithmen für die Ebenenzuweisung. Für Stretch-Width und Coffman-Graham liegen die Werte der durchschnittlichen Kantenlänge deutlich oberhalb der Werte von Cell Packing, Components, Network Simplex und Minimum-Width. Die durchschnittlichen Kantenlängen erreichen dabei 2 bis 5 mal höhere Werte als bei den anderen Algorithmen. Ein ähnliches Verhalten spiegelt sich bei der maximalen Kantenlänge wieder. Wie bereits in Kapitel 4 vermutet, eignen sich diese beiden Algorithmen nicht für Zusammenhangskomponenten, da die einzelnen Zusammenhangskomponenten selbst bereits aus extrem langen Kanten bestehen, wodurch die Lesbarkeit zu sehr eingeschränkt wird. Deshalb werden die beiden Algorithmen im weiteren Verlauf dieses Kapitels nicht mehr berücksichtigt.

In Abbildung 6.2 und Abbildung 6.3 sehen wir die Ergebnisse der Kantenlängen der übrigen Algorithmen. Network Simplex hat nun die höchste maximale Kantenlänge und die durchschnittliche Kantenlänge ist auch schlechter als bei Cell Packing und Components. Dies liegt vermutlich daran, dass die Diagramme mit Network Simplex bei Zusammen-

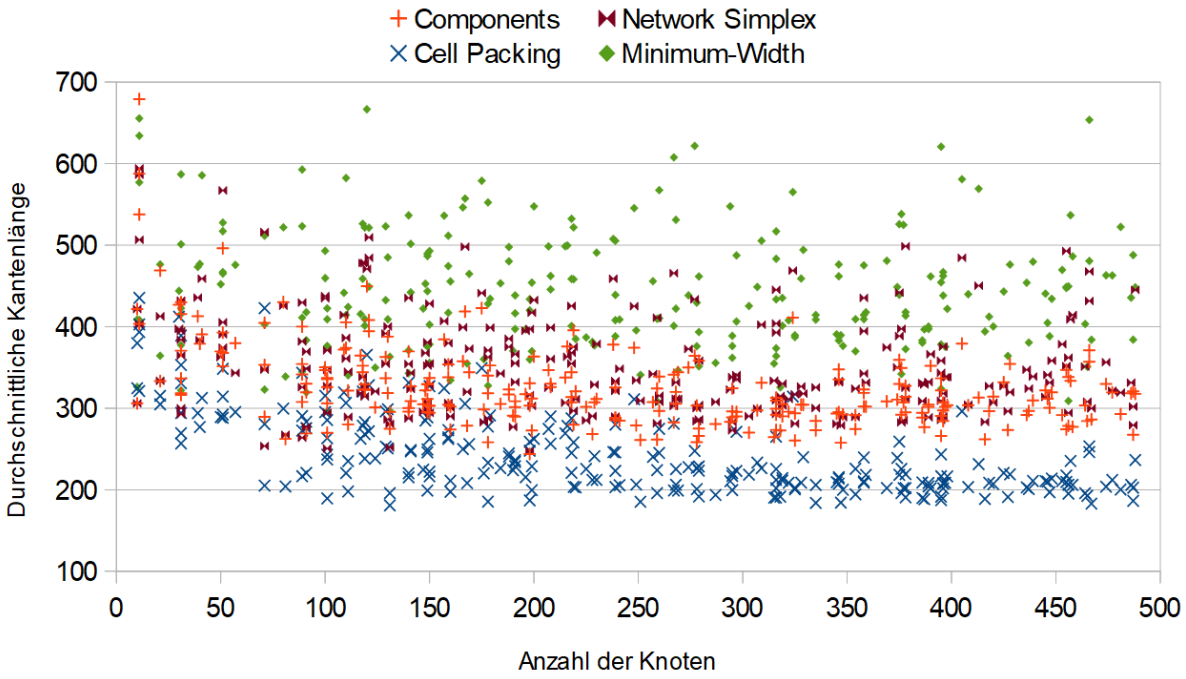


Abbildung 6.3. Evaluierung der durchschnittlichen Kantenlänge.

hangskomponenten in die Höhe wachsen und Komponenten mit sowohl Nord- als auch Südportverbindungen hierdurch extrem lange Kanten produzieren. Durch die Evaluation der Höhe der Algorithmen im Laufe dieses Kapitels wird diese Behauptung bestätigt. Für Minimum-Width sind die Werte der Kantenlängen vergleichbar mit denen von Network Simplex. Die besten Ergebnisse liefert der Cell Packing-Algorithmus und am zweitbesten sind die Werte für Components.

Evaluierung des Seitenverhältnisses

Als nächstes betrachten wir das produzierte Seitenverhältnis der Algorithmen in Abbildung 6.4. Für Network Simplex erreichen wir hierbei Niedrigstwerte von 0,07 bei Graphen mit vielen Knoten. Ein Ziel des Components-Algorithmus ist es, ein bestimmtes Seitenverhältnis einzuhalten. Für den Components-Algorithmus beträgt das angestrebte Seitenverhältnis 1,6. Die anderen Algorithmen streben kein vorgegebenes Seitenverhältnis an. Für den Components-Algorithmus sind die Werte für das Seitenverhältnis immerhin 3 bis 6 mal höher als bei Network Simplex, liegen aber deutlich unter dem angestrebten Seitenverhältnis. Die Algorithmen Cell Packing und Minimum-Width verhalten sich hierbei wesentlich besser und sind durchschnittlich bei Werten von 1,1 beziehungsweise 1,2 einzuordnen.

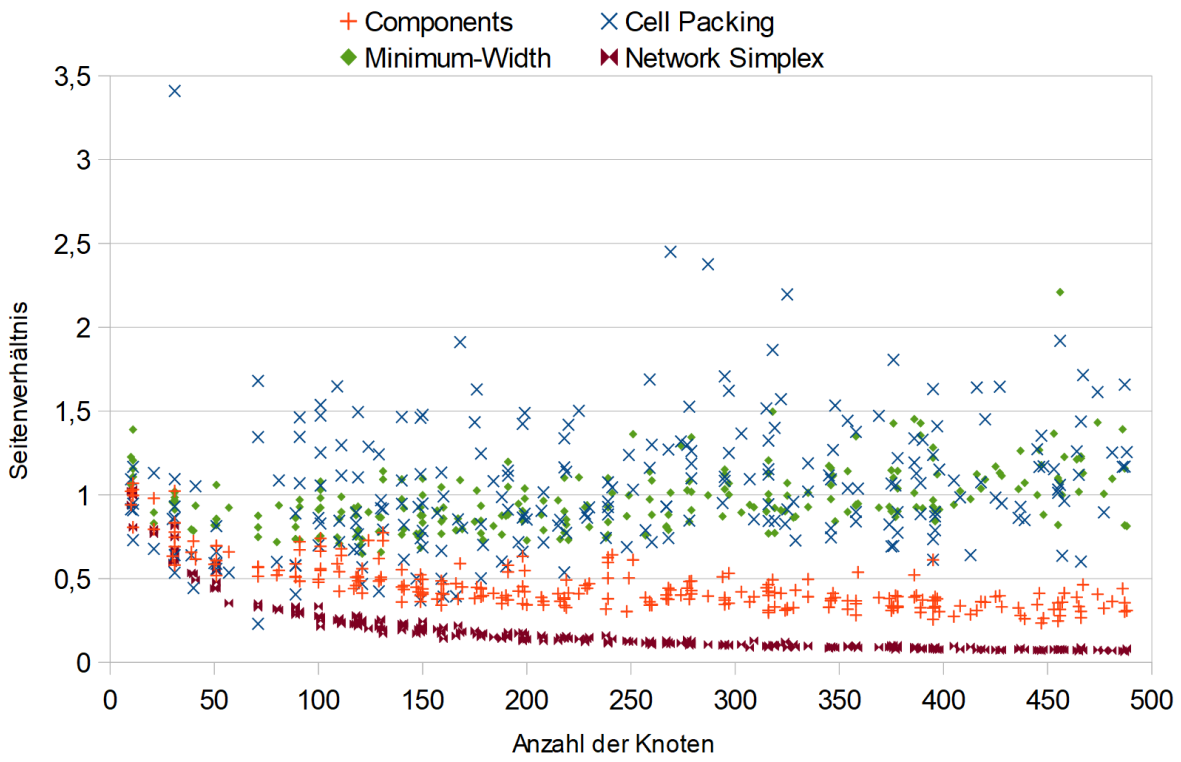


Abbildung 6.4. Evaluierung des Seitenverhältnisses.

Evaluierung der Breite und Höhe

Um den Abweichungen der erwarteten Seitenverhältnissen auf den Grund zu gehen, schauen wir uns als nächstes die Breite und Höhe der Testgraphen in Abbildung 6.5 und Abbildung 6.6 an. Während Minimum-Width zwar ein gutes Seitenverhältnis hat, produziert der Algorithmus aber mit Abstand die breitesten Diagramme und die Höhe befindet sich im Mittelfeld. Die Fläche der Testgraphen, bei denen Minimum-Width verwendet wurde, ist demnach die Größte im Vergleich zu den anderen Algorithmen. Network Simplex produziert wie erwartet die höchsten und schmalsten Diagramme. Die Breite hängt von der breitesten Komponente ab und die Komponenten werden untereinander platziert, woraus sich die Höhe ergibt. Wenn wir Cell Packing und Components vergleichen, sehen wir, dass die Breite von Cell Packing etwa doppelt so groß ist wie bei Components. Dieses Ergebnis ist zu erwarten, da Cell Packing Komponenten als Rechtecke abstrahiert, um visuelle Trennungen zwischen den Komponenten zu erzeugen und weil zwischen den einzelnen Zellen auch klare Trennungen vorhanden sind. Im Components-Algorithmus sollen die Komponenten möglichst kompakt platziert werden, was dazu führt, dass die Komponenten dichter zusammen geschoben sind als bei Cell Packing. Vor allem aber dürfen Kreuzungen von Nord- und Südportverbindungen mit anderen Komponenten entstehen, was ein dichteres Zusammenschieben weiter ermög-

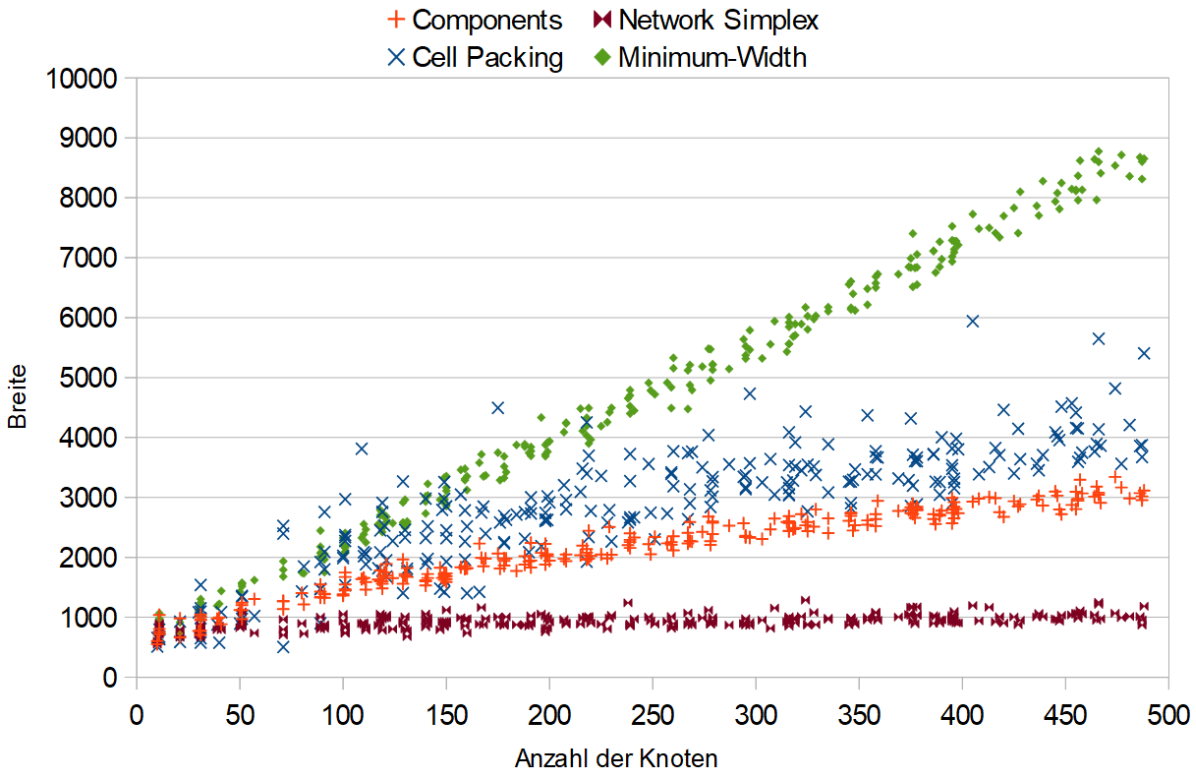


Abbildung 6.5. Evaluierung der Breite.

licht. Bei der Höhe hingegen verhält sich Cell Packing mehr als doppelt so gut wie der Components-Algorithmus. Diese Werte sind für den Components-Algorithmus unerwartet gewesen, da ursprünglich kompakte Diagramme produziert werden sollten.

Evaluierung der ungenutzten Fläche

Als Letztes schauen wir uns die Analyse für die ungenutzte Fläche in Abbildung 6.7 an. Für Minimum-Width erkennen wir, dass die ungenutzte Fläche für große Diagramme bei über 95% liegt. Dieser Algorithmus produziert die größte Fläche und hat die meiste ungenutzte Fläche. Dementsprechend eignet sich dieser Algorithmus ebenfalls nicht gut für Zusammenhangskomponenten. Die ungenutzte Fläche für Network Simplex ist am besten, da auch die Gesamtfläche am kleinsten ist und die einzelnen Zusammenhangskomponenten bei Network Simplex, Cell Packing und Components gleich groß sind. Die Werte für Cell Packing variieren in dieser Analyse stark. Es gibt Werte, die besser als bei Network Simplex sind, aber auch Werte, die wesentlich schlechter sind. Das kann daran liegen, dass einige Zellen in dem Algorithmus nicht optimal gepackt werden oder durch die Zellstruktur viel ungenutzte Fläche bei ungünstiger Belegung entstehen kann. Zum Beispiel wird durch das diagonale Platzieren bei Konflikten viel ungenutzte Fläche erzeugt. Der Components-Algorithmus

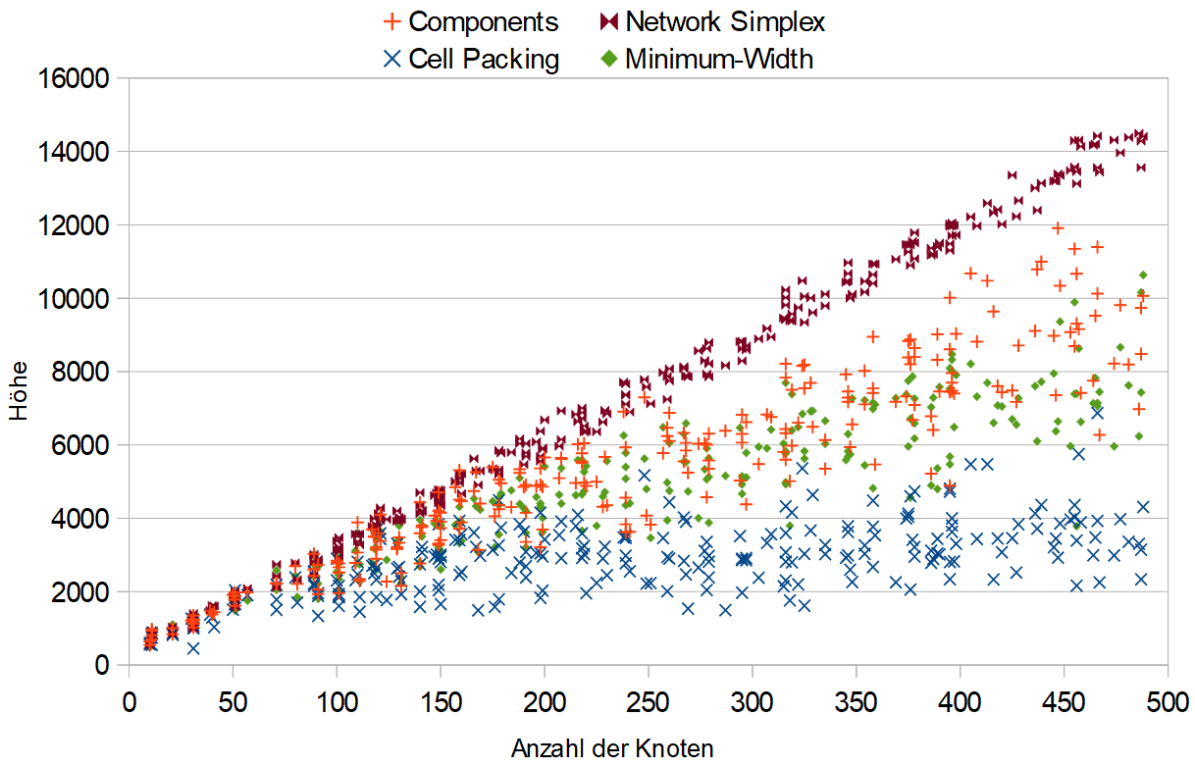


Abbildung 6.6. Evaluierung der Höhe.

schneidet hier auch nicht so gut ab, da die Gesamtfläche größer als bei Network Simplex und Cell Packing ist. Ursprünglich sollten durch den Algorithmus kompakte Diagramme erstellt werden, die ungenutzte Fläche im Vergleich mit Cell Packing und Network Simplex spricht jedoch ebenfalls dagegen.

Ergebnisse

Aus den im Vergleich nicht so optimalen Werten für den Components-Algorithmus muss aber nicht zwangsläufig geschlossen werden, dass der Algorithmus seine Ziele nicht erfüllt hat. In Abschnitt 6.4.1 werden Probleme genannt, die einen negativen Effekt auf die Höhe der erstellten Diagramme haben. Durch ein Beheben dieser Probleme kann vermutlich die Höhe für die meisten Diagramme optimiert werden, woraus sich auch eine kleinere Gesamtfläche und weniger ungenutzte Fläche ergibt. Das Seitenverhältnis sollte sich dadurch auch dem angestrebten Seitenverhältnis nähern. Ein weiteres Ziel war es, die Komponenten nah an ihre Portverbindungen zu platzieren, um kurze Kantenlängen zu erhalten. Die durchschnittliche und maximale Kantenlänge war beim Components-Algorithmus am zweitbesten. Cell Packing erreichte hierbei etwas bessere Werte. Durch die große Höhe bei Components entstehen sehr

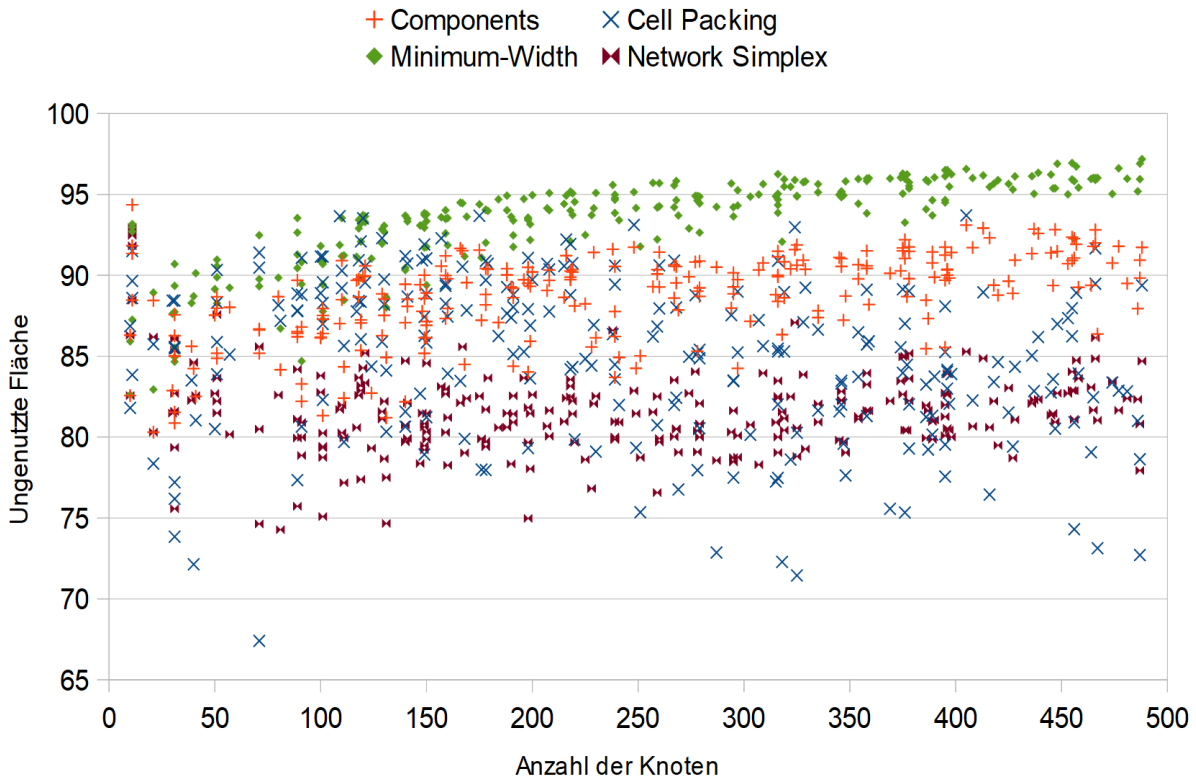


Abbildung 6.7. Evaluierung der ungenutzten Fläche.

lange Kanten, wenn eine Komponente Verbindungen zur Nord- und Südportseite hat. Durch eine Optimierung der Höhe könnten sich die Kantenlängen also auch weiter verbessern.

Für weitere Schlussfolgerungen sind weitere Analysen notwendig, die aus zeitlichen Gründen nicht mehr untergebracht werden konnten. Eine interessante Analyse wäre zum Beispiel, wie leicht sich Komponenten in den einzelnen Algorithmen identifizieren lassen. Cell Packing kann hier vermutlich am besten abschneiden, da Komponenten als Rechtecke abstrahiert werden, um visuelle Trennungen zwischen den Komponenten zu erzeugen. Ebenfalls kann bei Cell Packing der Abstand zwischen Komponenten über eine Option eingestellt werden. Bei Components und Network Simplex kann diese Option nicht verwendet werden, da die Komponenten in gemeinsam genutzte Ebenen eingeteilt werden, was bei Cell Packing nicht der Fall ist. Bei Components werden für diese diese Evaluation wahrscheinlich schlechtere Werte herauskommen, da Komponenten potentiell sehr dicht nebeneinander platziert werden können. In diesen Fällen muss genau geprüft werden, ob zwei Knoten zur selben Komponente gehören. Dieses Ergebnis wäre jedoch zu erwarten, da der Algorithmus den Fokus auf kompakte Diagramme legt.

Eine weitere interessante Analyse wäre eine Evaluation der Lesbarkeit und Verständlichkeit der erstellten Diagramme. Hierfür müsste eine Nutzerstudie durchgeführt werden, in der für die Ebenenzuweisung die Algorithmen Components, Cell Packing und Network Simplex

sowie für die Kreuzungsminimierung die Algorithmen Layer Sweep und Components Crossing Minimizer verglichen werden. Dadurch könnten bessere Aussagen darüber getroffen werden, inwiefern ein Algorithmus ein anschauliches Diagramm für den Benutzer erstellt.

6.4. Problemfälle

6.4.1. Höhe des hierarchischen Knotens

Wie im vorherigen Abschnitt beschrieben, ist die Höhe der Diagramme vom Components-Algorithmus zu groß, was in einem unerwartetem Seitenverhältnis und zu viel ungenutzter Fläche resultiert. Die Betrachtung einiger Testgraphen hat für die Höhe folgende mögliche Gründe ergeben:

- Für den ersten Grund schauen wir uns zunächst Abbildung 6.8 an. Hier sehen wir unter anderem mehrere Komponenten ohne Portverbindungen. Diese Komponenten haben keine Abhängigkeiten zueinander, da ihr Vertauschen keine zusätzlichen Kreuzungen einführen würde. In solchen Fällen kann eine ungünstige Reihenfolge der Komponenten zu einer schlechteren Höhe führen. In der oberen Hälfte der Abbildung ist genug Platz, um zumindest drei der vier unteren Komponenten zu platzieren. Das Problem ist, dass während der Phase der Kreuzungsminimierung noch keine Informationen gegeben sind, wie viel Platz die einzelnen Komponenten letztendlich benötigen, da erst in den späteren Phasen Positionen berechnet werden, aus denen sich die Größe einer Komponente ableitet. Es ist somit schwierig, eine Reihenfolge der unabhängigen Komponenten untereinander festzulegen, welche im finalen Diagramm zu einer minimierten Höhe führen könnte. Ein Ansatz hierfür wäre ein Nachbearbeitungsschritt. Nachdem die Phasen des ebenenbasierten Ansatzes durchgelaufen sind und wir dadurch Größen für die einzelnen Komponenten vorgegeben haben, könnten die unabhängigen Komponenten frei verschoben werden, mit dem Ziel andere Lücken zu füllen und die Gesamtgröße des hierarchischen Knotens zu optimieren.
- Der zweite mögliche Grund, der die Höhe der Diagramme beeinflusst, ist in Abbildung 6.9a dargestellt. Dort sehen wir drei Knoten, die jeweils eine Zusammenhangskomponente repräsentieren sollen. Gehen wir in dem Beispiel davon aus, dass die Ebenenzuweisung diese Komponenten nebeneinander eingeordnet hat. Die Kreuzungsminimierung berechnet nun Abhängigkeiten auf Grund der langen Kanten zu den östlichen Ports. Um Kreuzungen zu verhindern, werden die Zusammenhangskomponenten untereinander platziert und somit wird die Höhe des Diagramms beeinflusst. Durch das Erlauben von zusätzlichen Kreuzungen könnte die Höhe des Diagramms wieder verbessert werden, wie wir in Abbildung 6.9b sehen. Die Frage, die sich hierbei aber stellen würde, wäre, wie viele zusätzliche Kantenkreuzungen man erlauben möchte und wie sich dies entscheiden ließe. Alternativ könnten die Kanten wie in Abbildung 6.9c auch oberhalb der hierarchischen Knoten verlaufen und würden sogar keine Kreuzungen verursachen. Dafür müssen allerdings Kantenknicke und sogar etwas längere Kanten in Kauf genommen werden.

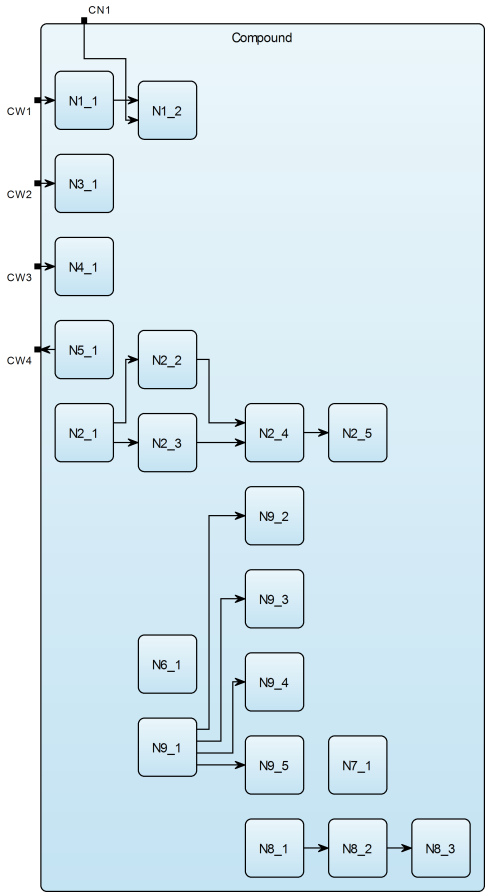
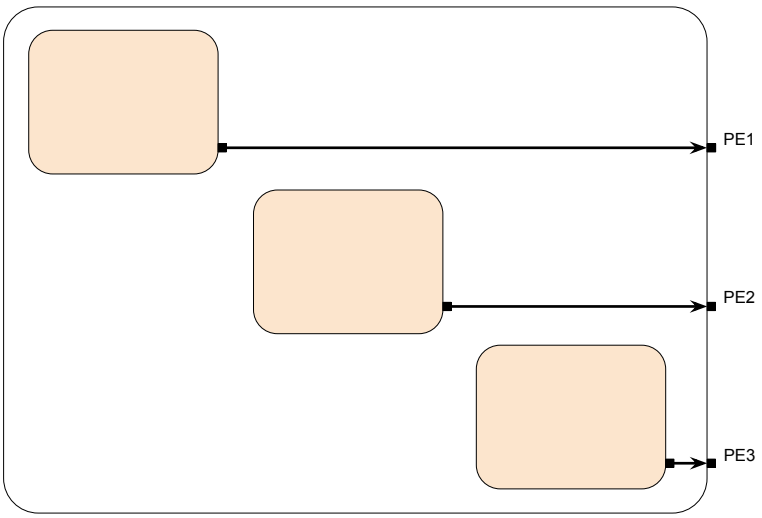


Abbildung 6.8. Schlechte Sortierung von unabhängigen Komponenten.

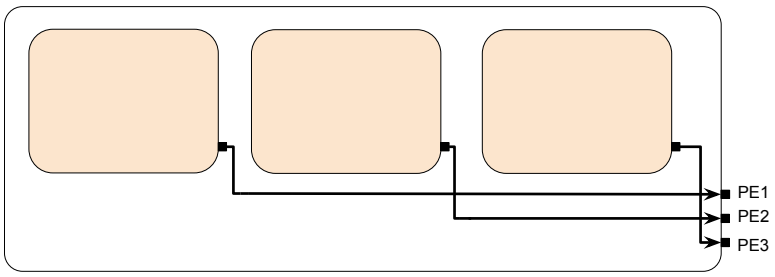
Wenn diese beiden Probleme ausreichend gelöst wurden, muss eine neue Evaluation durchgeführt werden, in der das Erreichen eines angestrebten Seitenverhältnisses noch einmal genauer getestet wird. Bisher haben wir gesehen, dass ein angestrebtes Seitenverhältnis nicht eingehalten wird.

6.4.2. Unterschiedliche Knotengrößen

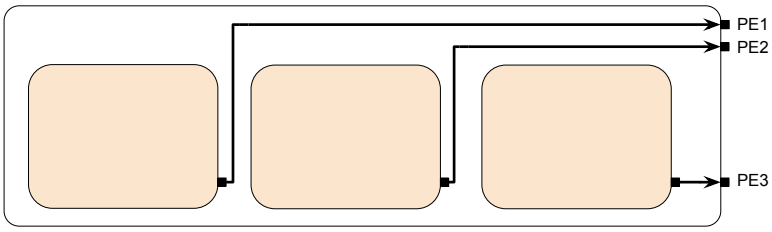
Beim Berechnen einer angestrebten Breite und Höhe in Abschnitt 4.3.4 sind wir bisher davon ausgegangen, dass alle Knoten die gleiche Größen haben. In realen Diagrammen können Knoten aber verschiedene Größen haben, welche ebenfalls die Breite und Höhe des Diagramms beeinflussen. Die Breite einer Ebene ergibt sich nämlich aus dem breitesten Knoten in der jeweiligen Ebene. In Abbildung 6.10 sehen wir, dass es für große Knoten nicht ausreicht, sie in einer Ebene zu sammeln. Die Graphen in Abbildung 6.10a und Abbildung 6.10b sind jeweils äquivalent, aber das Sammeln von großen Knoten in derselben Ebene führt zu einem größeren Seitenverhältnis. Bei dem Graph aus Abbildung 6.10c und Abbildung 6.10d ist



(a) Ungenutzte Fläche kann durch Komponentenreihenfolge und langen Kanten erhöht werden.



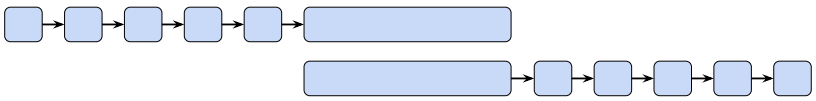
(b) Durch zusätzliche Kantenknicke und Kreuzungen kann der Platz besser genutzt werden.



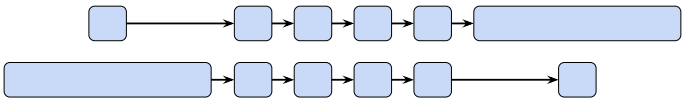
(c) Durch zusätzliche Kantenknicke und längere Kanten kann der Platz besser genutzt werden.

Abbildung 6.9. Ungenutzte Fläche durch vorgegebene Komponentenreihenfolge.

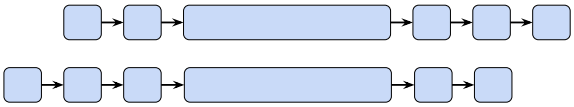
hingegen das Gegenteil der Fall. Eine Lösungsansatz für dieses Problem könnte sein, dass die großen Knoten in eine Reihe mehrerer kleiner Knoten aufgeteilt werden, damit die Ebenen schmaler werden. Eine Problem hierbei ist, dass dadurch Kreuzungen zwischen Kanten und Knoten entstehen können, wenn der große Knoten wiederhergestellt wird [FS04].



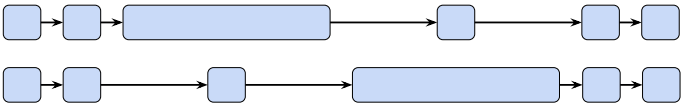
(a) Hier werden große Knoten in einer Ebene gesammelt.



(b) Hier werden große Knoten in unterschiedliche Ebenen eingeteilt.



(c) Hier werden große Knoten in einer Ebene gesammelt.

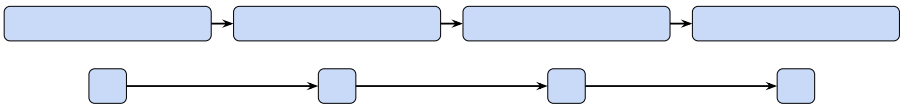


(d) Hier werden große Knoten in unterschiedliche Ebenen eingeteilt.

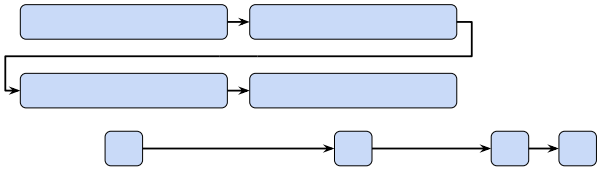
Abbildung 6.10. Probleme mit unterschiedlich großen Knoten. Das Platzieren von großen Knoten in gleiche oder unterschiedliche Ebenen kann jeweils zu verschiedenen Seitenverhältnissen führen.

Weiterhin kann die Breite des Diagramms von einer Komponente, bestehend aus mehreren großen Knoten, beeinflusst werden. In Abbildung 6.11a sehen wir ein Beispiel hierfür. Dieses Problem könnte dadurch gelöst werden, dass die Komponente wie in Abbildung 6.11b umgebrochen wird. Hierfür muss eine lange rücklaufende Kante eingeführt werden und die Komponente wird in einer früheren Ebene fortgesetzt [RES+15].

Auf Grund der eben genannten Beobachtungen sollte die Berechnung einer angestrebten Graphengröße aus Abschnitt 4.3.4 überarbeitet werden. Für unterschiedlich große Knoten muss entweder die Knotengröße in die Abschätzungen mit einfließen. Ansonsten muss eine Lösung gefunden werden, welche die möglichen Kreuzungen zwischen Knoten und Kanten beim Wiederherstellen eines großen Knotens verhindert. Falls große Knoten bei der Berechnung nicht berücksichtigt werden, kann sich einem angestrebten Seitenverhältnis nicht zuverlässig genähert werden. Der Einfluss großer Knoten muss in einer neuen Evaluation getestet werden. Dabei sollte zum Einen getestet werden, wie viel Einfluss große Knoten aktuell auf das Seitenverhältnis haben, und diese Werte sollten mit einem neuen Verfahren, welches große Knoten berücksichtigt, verglichen werden.



(a) Hier hängt die Breite des Diagramms von einer Zusammenhangskomponente ab.



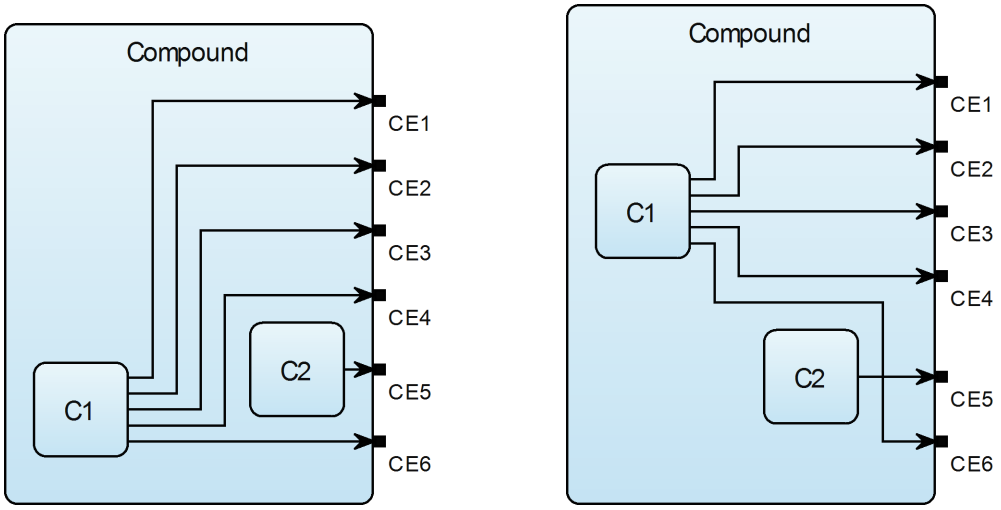
(b) Durch ein Umbrechen der breiten Komponente wird die Breite des Diagramms verringert.

Abbildung 6.11. Weitere Probleme mit unterschiedlich großen Knoten, die das Seitenverhältnis beeinflussen.

6.4.3. Unnötige Kreuzungen

Ein weiteres Problem tritt beim Applizieren der Komponentenreihenfolgen auf. Die Komponenten werden dabei zusammen mit allen zugehörigen Dummyknoten behandelt. Wie bereits in Kapitel 5 erwähnt wurde, sollen Komponenten nicht verschachtelt sein, damit sie besser identifiziert werden können. Schauen wir uns aber eine Konstellation von zwei Komponenten wie in Abbildung 6.12 an. $C1$ hat hierbei Verbindungen zu mehreren Ports auf der östlichen Portseite. $C2$ hat ebenfalls eine Verbindung zu einem östlichen Port, der in der Reihenfolge dazwischen auftaucht. Es würden keine Kreuzungen auftreten, wenn die Dummyknoten der langen Kante von $C1$ in Abbildung 6.12a mit $C2$ verschachtelt sein dürften. Das Verhindern von Verschachtelungen ergibt in diesem Fall, wie in Abbildung 6.12b zu sehen ist, eine zusätzliche Kreuzung. Für lange Kanten müsste also zusätzlich differenziert werden, ob sie im Falle einer Verschachtelung weniger Kreuzungen verursachen würden.

Ein weiteres Problem mit unnötigen Kreuzungen entsteht bei Layer Sweep mit externen Portdummies. Wenn diese einen Nord- oder Südport repräsentieren, beinhalten sie Metainformationen, ob sie ober- oder unterhalb von anderen Knoten platziert werden müssen. Diese Information wird allerdings momentan von Layer Sweep nicht berücksichtigt und somit können zusätzliche Kantenkreuzungen entstehen, wie es in Abbildung 6.13 zu sehen ist. Durch einen späteren Zwischenprozessor wird der externe Portdummy an die richtige Position platziert, aber die Ports an dem zugehörigen Knoten werden dadurch nicht in die richtige Reihenfolge sortiert. Dieses Problem muss im Layer Sweep-Algorithmus in ELK behoben werden und ist dementsprechend kein Problem, welches durch meinen Ansatz hervorgerufen wird.



(a) Keine Kreuzungen, aber Dummyknoten sind mit anderer Komponente verschachtelt. Ober- und unterhalb von *C*2 sind Dummyknoten für lange Kanten von *C*1 platziert.

(b) Eine zusätzliche Kreuzung wurde eingefügt, aber Dummyknoten sind nicht mit anderen Komponenten verschachtelt.

Abbildung 6.12. Zusätzliche Kreuzungen, wenn Dummyknoten von langen Kanten nicht mit anderen Komponenten verschachtelt sein dürfen.

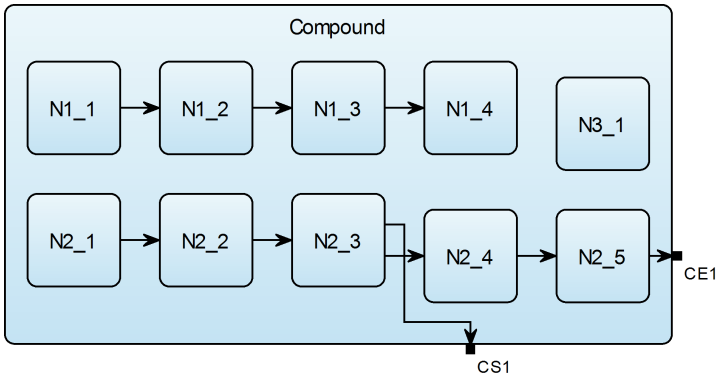


Abbildung 6.13. Unnötige Kantenkreuzungen bei Layer Sweep.

Kapitel 7

Schlussfolgerung

In diesem Kapitel wird noch einmal zusammengefasst, welche Schritte für den in dieser Arbeit entwickelten Ansatz nötig sind und inwiefern die gesetzten Ziele eingehalten wurden. Danach werden weitere offene Probleme genannt, deren Behebung die vorgestellte Lösung weiter verbessern könnte.

7.1. Zusammenfassung

Für diese Arbeit mussten zwei Phasen aus dem ebenenbasierten Ansatz von Sugiyama et al. [STT81] angepasst werden. Das Verfahren Cell Packing, welches aktuell in ELK bei hierarchischen Ports mit den Porteinschränkungen FREE oder FIXED SIDE genutzt wird, kann für die anderen Porteinschränkungen nicht verwendet werden. Das Ziel meiner Arbeit war es, für die Porteinschränkungen FIXED ORDER oder FIXED POSITION einen neuen Algorithmus zu entwerfen, da Cell Packing nicht verwendet werden kann und die üblichen Algorithmen für die Ebenenzuweisung nicht gut mit Zusammenhangskomponenten umgehen können.

Als Erstes wurde die Phase der Ebenenzuweisung angepasst. Hierfür wurde ein Algorithmus entwickelt, der die Kantenverbindungen von Komponenten mit Verbindungen zu ausschließlich den westlichen oder östlichen Ports des hierarchischen Knotens möglichst kurz hält, damit das Diagramm durch kurze Kanten besser lesbar ist. Weiterhin werden die Komponenten mit Verbindungen zu nördlichen und südlichen hierarchischen Ports über die Ebenen verteilt. Das sorgt dafür, dass durch diese Komponenten potentiell weniger Kreuzungen mit anderen Komponenten entstehen und somit die Lesbarkeit der Diagramme weiter verbessert wird.

Als Zweites musste die Phase der Kreuzungsminimierung angepasst werden. Ein wichtiges Ästhetikkriterium ist es, die Zahl der Kreuzungen in einem Graphen zu verringern, damit die Lesbarkeit für den Betrachter erhöht wird. Die üblichen Algorithmen für diese Phase eignen sich nur dafür, die Kreuzungen innerhalb von Zusammenhangskomponenten zu minimieren, aber nicht dafür, die Kreuzungen zwischen den Komponenten zu minimieren. Hierfür werden die Komponenten paarweise miteinander verglichen und mit Hilfe eines Kreuzungsgraphen wird eine Reihenfolge der Komponenten mit möglichst wenig Kreuzungen abgeleitet.

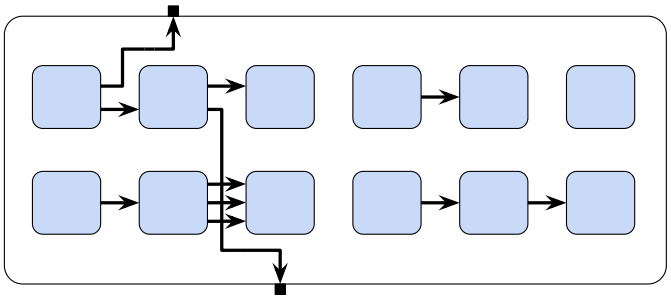
Als Letztes wurden die Ergebnisse der vorgestellten Lösungen evaluiert. Dabei wurde festgestellt, dass nicht alle erwarteten Werte erreicht wurden. Ein Ziel war es, möglichst kompakte Diagramme zu erzeugen. Die resultierende ungenutzte Fläche der Diagramme

konnte die Werte von Cell Packing nicht erreichen. Diese Werte dienten als Referenzwerte, die erreicht werden sollten. Es ist aufgefallen, dass die Höhe der Testgraphen zwar besser als beim Network Simplex Algorithmus ist, aber trotzdem noch zu hoch ist. Mögliche Ursachen hierfür sind, dass durch eine ungünstig gewählte Reihenfolge von Komponenten, die keine Kreuzungen untereinander verursachen, die Höhe der Diagramme wächst. Ebenfalls wird die Höhe der Diagramme durch gerade lange Kanten beeinflusst. Ein weiteres Ziel war es, die Kantenlängen möglichst kurz zu halten, um lesbarere Diagramme zu erzeugen. Für die Testgraphen waren die Ergebnisse hierfür am zweitbesten, knapp hinter den Ergebnissen des Cell Packing. Durch das Beheben der Probleme mit der Höhe könnten diese Werte jedoch weiter verbessert werden.

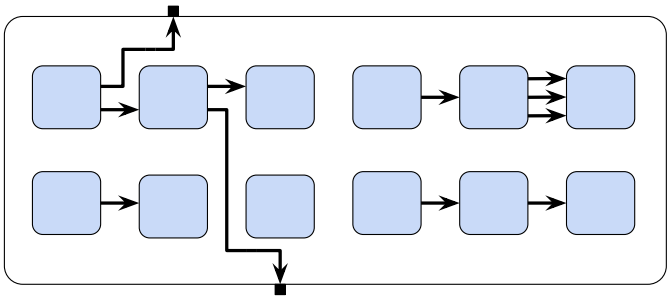
7.2. Ausblick

Wie bereits erwähnt konnten manche der gesetzten Ziele der Arbeit nicht eingehalten werden. Ein Teil zukünftiger Arbeiten wäre es demnach, die bereits in Abschnitt 6.4 dargestellten Probleme zu lösen und eine neue Evaluation durchzuführen. Für weitere zukünftige Arbeiten in dem Themengebiet schauen wir uns noch zwei Beispiele an.

- Durch eine komplexere Ebenenzuweisung können weitere Kreuzungen wie in Abbildung 7.1 aus dem Graphen entfernt werden. Das Erkennen solcher möglichen Kreuzungen während der Ebenenzuweisung ist jedoch sehr komplex. Eine Idee hierfür wäre es, die Phasen der Ebenenzuweisung mit der Kreuzungsminimierung zu kombinieren und somit während des Einordnens in Ebenen direkt zu prüfen, inwiefern sich die Kreuzungen durch die aktuelle Einordnung der Knoten verändern.
- Als Zweites kann man sich überlegen, meine entwickelten Algorithmen auch für die Porteinschränkungen Fixed Side und Free zu verwenden, falls zum Beispiel der Cell Packing-Algorithmus mindestens einen Konflikt verursacht. Wie wir in Abbildung 7.2a sehen, wird durch einen Konflikt in dem Algorithmus viel ungenutzte Fläche erzeugt. Durch meine entwickelten Algorithmen ist es für Zusammenhangskomponenten allerdings erlaubt, andere Zusammenhangskomponenten zu kreuzen, wodurch das Diagramm in Abbildung 7.2b wesentlich kompakter ist.

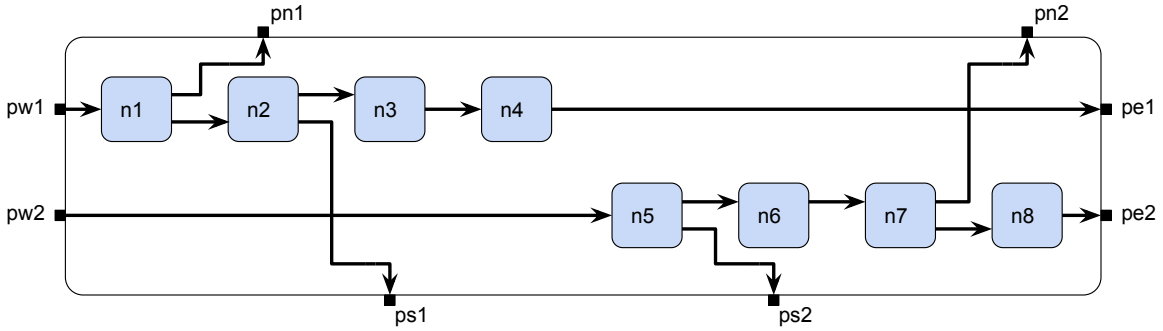


(a) Bei dieser Anordnung der Komponenten entstehen drei Kreuzungen.

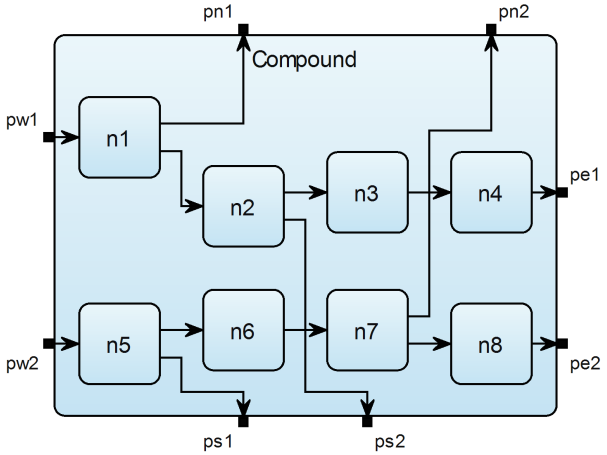


(b) Bei dieser Anordnung der Komponenten entstehen keine Kreuzung.

Abbildung 7.1. Durch die Ebenenzuweisung können bereits Kreuzungen vermieden werden. Die Platzierung von Komponenten beeinflusst die späteren Kreuzungen.



(a) Bei Konflikten erzeugt Cell Packing viel ungenutzte Fläche.



(b) Durch meinen Algorithmus können bei Konflikten kompaktere Diagramme entstehen.

Abbildung 7.2. Falls Konflikte beim Cell Packing entstehen, kann mein Ansatz auch für die Einschränkungen FIXED SIDE oder FREE verwendet werden und somit kompaktere Diagramme erstellen.

Literatur

[All] Allicorn. Colorful Natural Tree. https://www.goodfreephotos.com/vector-images/colorful-natural-tree-vector-clipart.png.php. [Online; accessed 04-Dezember-2017].

[BJM02] Wilhelm Barth, Michael Jünger und Petra Mutzel. „‚Simple and Efficient Bilayer Cross Counting'". In: *Proceedings of the 10th International Symposium on Graph Drawing (GD'02)*. Hrsg. von Stephen G. Kobourov und Michael T. Goodrich. Bd. 2528. LNCS. Springer, 2002, S. 331–360.

[BK02] Ulrik Brandes und Boris Köpf. „‚Fast and Simple Horizontal Coordinate Assignment'". In: *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*. Hrsg. von Petra Mutzel, Michael Jünger und Sebastian Leipert. Bd. 2265. LNCS. Springer, 2002, S. 33–36. ISBN: 978-3-540-43309-5. DOI: 10.1007/3-540-45848-4.

[CG72] Edward G. Coffman. und Ronald L. Graham. „‚Optimal scheduling for two-processor systems'". In: *Acta Informatica* 1.3 (1972), S. 200–213. ISSN: 0001-5903. DOI: 10.1007/BF00288685.

[DET+99] Giuseppe Di Battista, Peter Eades, Roberto Tamassia und Ioannis G. Tollis. Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, 1999. ISBN: 0-13-301615-3.

[ELS93] Peter Eades, Xuemin Lin und W. F. Smyth. „‚A fast and effective heuristic for the feedback arc set problem'". In: *Information Processing Letters* 47.6 (1993), S. 319–323. ISSN: 0020-0190. DOI: 10.1016/0020-0190(93)90079-O.

[FH10] Hauke Fuhrmann und Reinhard von Hanxleden. Taming Graphical Modeling. Technical Report 1003. Christian-Albrechts-Universität zu Kiel, Department of Computer Science, Mai 2010.

[FR91] Thomas M. J. Fruchterman und Edward M. Reingold. „‚Graph drawing by force-directed placement'". In: *Software—Practice & Experience* 21.11 (1991), S. 1129–1164. ISSN: 0038-0644. DOI: http://dx.doi.org/10.1002/spe.4380211102.

[FS04] Carsten Friedrich und Falk Schreiber. „‚Flexible Layering in Hierarchical Drawings with Nodes of Arbitrary Size'". In: *Proceedings of the 27th Australasian Conference on Computer Science (ACSC'04)*. Australian Computer Society, Inc., 2004, S. 369–376.

[GKN+93] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North und Kiem-Phong Vo. „‚A Technique for Drawing Directed Graphs'". In: *Software Engineering* 19.3 (1993), S. 214–230.

[HB05] Jeffrey Heer und Danah Boyd. „‚Vizster: Visualizing online social networks‘“. In: *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*. IEEE. 2005, S. 32–39.

[HDM+13] Reinhard von Hanxleden, Björn Duderstadt, Christian Motika, Steven Smyth, Michael Mendler, Joaquín Aguado, Stephen Mercer und Owen O'Brien. SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications. Technical Report 1311. ISSN 2192-6247. Christian-Albrechts-Universität zu Kiel, Department of Computer Science, Dez. 2013.

[HF10] Reinhard von Hanxleden und Hauke Fuhrmann. Taming Graphical Modeling. Presentation at the 17th International Open Workshop on Synchronous Programming (SYNCHRON'10), Frejus, France. Dez. 2010.

[HMA+13] Reinhard von Hanxleden, Michael Mendler, Joaquín Aguado, Björn Duderstadt, Insa Fuhrmann, Christian Motika, Stephen Mercer, Owen O'Brien und Partha Roop. Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation. Technical Report 1308. ISSN 2192-6247. Christian-Albrechts-Universität zu Kiel, Department of Computer Science, Aug. 2013.

[Ker] Eclipse Layout Kernel. Documentation of ELK. `http://www.eclipse.org/elk/documentation.html`. [Online; accessed 04-Dezember-2017].

[Kla12] Lars Kristian Klauske. „‚Effizientes Bearbeiten von Simulink Modellen mit Hilfe eines spezifisch angepassten Layoutalgorithmus‘“. Diss. Technische Universität Berlin, 2012.

[NTB05] Nikola S. Nikolov, Alexandre Tarassov und Jürgen Branke. „‚In search for efficient heuristics for minimum-width graph layering with consideration of dummy nodes‘“. In: *Journal of Experimental Algorithmics* 10 (2005). ISSN: 1084-6654. DOI: 10.1145/1064546.1180618.

[Pet95] Marian Petre. „‚Why looking isn't always seeing: Readership skills and graphical programming‘“. In: *Communications of the ACM* 38.6 (Juni 1995), S. 33–44.

[Pur02] Helen C. Purchase. „‚Metrics for Graph Drawing Aesthetics‘“. In: *Journal of Visual Languages and Computing* 13.5 (2002), S. 501–516.

[Pur97] Helen C. Purchase. „‚Which aesthetic has the greatest effect on human understanding?‘“ In: *Proceedings of the 5th International Symposium on Graph Drawing (GD'97)*. Bd. 1353. LNCS. Springer, 1997, S. 248–261.

[RDM+87] Lawrence Rowe, Michael Davis, Eli Messinger, Carl Meyer, Charles Spirakis und Allen Tuan. „‚A Browser for Directed Graphs‘“. In: *Software – Practice and Experience* 17.1 (Jan. 1987), S. 61–76.

[RES+15] Ulf Rüegg, Thorsten Ehlers, Miro Spönemann und Reinhard von Hanxleden. A Generalization of the Directed Graph Layering Problem. Technical Report 1501. ISSN 2192-6247. Kiel University, Department of Computer Science, Feb. 2015.

[RSG+16] Ulf Rüegg, Christoph Daniel Schulze, Daniel Grevismühl und Reinhard von Hanxleden. Using One-Dimensional Compaction for Smaller Graph Drawings. Technical Report 1601. ISSN 2192-6247. Kiel University, Department of Computer Science, Apr. 2016.

[RT81] E. M. Reingold und J. S. Tilford. „‚Tidier Drawing of Trees'". In: *IEEE Transactions on Software Engineering* 7 (März 1981), S. 223–228.

[San04] Georg Sander. „‚Layout of Directed Hypergraphs with Orthogonal Hyperedges'". In: *Proceedings of the 11th International Symposium on Graph Drawing (GD'03)*. Hrsg. von Giuseppe Liotta. Bd. 2912. LNCS. Springer, 2004, S. 381–386. ISBN: 978-3-540-20831-0.

[San99] Georg Sander. „‚Graph Layout for Applications in Compiler Construction'". In: *Theoretical Computer Science* 217.2 (1999), S. 175–214.

[Sch11] Christoph Daniel Schulze. „‚Optimizing Automatic Layout for Data Flow Diagrams'". Diploma Thesis. Kiel University, Department of Computer Science, Juli 2011.

[SSH13] Christian Schneider, Miro Spönemann und Reinhard von Hanxleden. „‚Just Model! – Putting Automatic Synthesis of Node-Link-Diagrams into Practice'". In: *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'13)*. San Jose, CA, USA, Sep. 2013, S. 75–82. DOI: 10.1109/VLHCC.2013.6645246.

[SSH14] Christoph Daniel Schulze, Miro Spönemann und Reinhard von Hanxleden. „‚Drawing Layered Graphs with Port Constraints'". In: *Journal of Visual Languages and Computing, Special Issue on Diagram Aesthetics and Layout* 25.2 (2014), S. 89–106. ISSN: 1045-926X. DOI: 10.1016/j.jvlc.2013.11.005.

[STT81] Kozo Sugiyama, Shojiro Tagawa und Mitsuhiko Toda. „‚Methods for visual understanding of hierarchical system structures'". In: *IEEE Transactions on Systems, Man and Cybernetics* 11.2 (Feb. 1981), S. 109–125.