Masterarbeit

Eine Variante des Sugiyama-Algorithmus für ungerichtete Graphen mit Portconstraints

Julian Walter

Abgabedatum:	16. Juni 2020
Betreuer:	Prof. Dr. Alexander Wolff
	Johannes Zink, M. Sc.

Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

Zusammenfassung

Der Ansatz von Sugiyama et al. [STT81] ist einer der bekanntesten zur Visualisierung hierarchischer Graphen – er wird auch Sugiyama-Algorithmus genannt. Eine darauf aufbauende Arbeit von Schulze et al. [SSvH14] erweitert den Sugiyama-Algorithmus um sogenannte *Portconstraints*. Das bedeutet, dass Kanten nicht mit Knoten, sondern mit Ports verbunden sind. Die Ports wiederum sind einem Knoten zugehörig und werden auf dem Rand dieses Knotens positioniert. Es wird die Möglichkeit geboten, eine feste Portreihenfolge von vornherein zu wählen.

In dieser Arbeit wird der Ansatz von Schulze et al. um drei Möglichkeiten erweitert: um die Unterstützung von Portgruppen, Portpaaren sowie ungerichteter Graphen als Eingabe. Portgruppen sind eine Sammlung von Ports, welche nebeneinander gezeichnet werden sollen, deren Reihenfolge untereinander jedoch frei wählbar ist. Portpaare sind Ports, die innerhalb eines Knotens verbunden sind. Zur Unterstützung von ungerichteten Graphen wird der Algorithmus um den neuen Schritt Richtungszuweisung erweitert.

Ziel ist es, durch Unterstützung von Portgruppen und Portpaaren mehr Freiheiten nutzen und damit Visualisierungen mit weniger Kreuzungen erstellen zu können. Bisher konnten diese nur durch eine fest vorgegebene Portreihenfolge unterstützt werden. Des Weiteren werden drei Möglichkeiten zur Richtungszuweisung für ungerichtete Kanten verglichen – eine rein zufällige, kreisfreie, eine entlang einer Breitensuche und eine Richtungszuweisung anhand einer kräftebasierten Darstellung des Graphen. Die neuen Zeichnungen werden analysiert und die getesteten Verfahren anhand einiger Parameter, die die Übersichtlichkeit fördern, verglichen. Auch wird ein visueller Vergleich der Resultate des bereits bekannten und des neuen Verfahrens durchgeführt.

Inhaltsverzeichnis

1 Einleitung

Der Sugiyama-Algorithmus [STT81] ist einer der bekanntesten Ansätze zur Visualisierung hierarchischer Graphen basierend auf den Ideen von Warfield [War77]. Er ist, wie viele andere Ansätze Visualisierungen von allgemeinen Graphen zu erzeugen, heuristisch, da schon Teilprobleme der Visualisierung, wie z. B. Kreuzungsminimierung, NP-schwer sind [TDB88]. Sugiyamas Ansatz stellt nach wie vor die Grundlage für aktuelle Algorithmen dar, wie unter anderem für die Arbeit von Schulze et al. [SSvH14]. Nicht nur hierarchische Graphen lassen sich mit dem Sugiyama-Algorithmus übersichtlich darstellen. Auch bei anderen Graphen kann die Einteilung der Knoten in einzelne Ebenen (oder Lagen) zu einer besseren Übersichtlichkeit beitragen.

Es liegt nahe, ihn zur Visualisierung elektrischer Schaltpläne einzusetzen, wobei auch schon gute Ergebnisse erzielt wurden. Ein Beispiel hierzu findet sich in Abbildung 1.1. Zur Berechnung der Ansicht wurde der Algorithmus „ELK Layered“ in eclipse.elk [elk20] verwendet, welcher dem „KLay Layered“ Algorithmus von Schulze et al. [SSvH14] aus dem KIELER-Projekt [kie20] entspricht. Sie bietet unter anderem die Möglichkeit der Verwendung von Ports. Das bedeutet, dass Kanten, welche zu einem Knoten führen, nicht alle an derselben Stelle mit dem Knoten verbunden werden. Jeder Knoten wird durch ein Rechteck dargestellt und jeder Port hat eine bestimmte Position auf dem Rand des Rechtecks seines zugehörigen Knotens. Kanten starten und enden an Ports und verbinden so die den Ports zugehörigen Knoten miteinander. Auch berechnet *KLay Layered* eine orthogonale Zeichnung der Kanten, was bei elektrischen Schaltplänen üblich und für das Ergebnis gewünscht ist.

Als Eingabe wird bei uns ein ungerichteter Graph übergeben, bei welchem Kanten elektrische Verbindungen und Knoten verschiedene elektrische Komponenten darstellen – wie Sensoren, Steuereinheiten oder Aktuatoren. Auf dem Eingabegraphen gibt es daher nur wenige Einschränkungen, was einen sehr allgemein gehaltenen Lösungsansatz notwendig macht. Auch gibt es einige Besonderheiten, wie Multikanten, Steckverbindungen und Portgruppen, welche von *KLay Layered* nicht unterstützt werden und nur durch zusätzliche Einschränkungen umgesetzt werden können. Kantenrichtungen könnten durch Analyse der Metadaten, wie zum Beispiel der Richtung des Stromflusses, festgelegt werden; jedoch stehen diese Daten nicht zur Verfügung und es besteht die Hoffnung, auch allgemeine Verbesserungen durch die gezielte Unterstützung der Sonderfälle erreichen zu können – insbesondere bei der Anzahl der Kreuzungen und damit der Übersichtlichkeit der Darstellung sowie der Verringerung der benötigten Zeichenfläche.

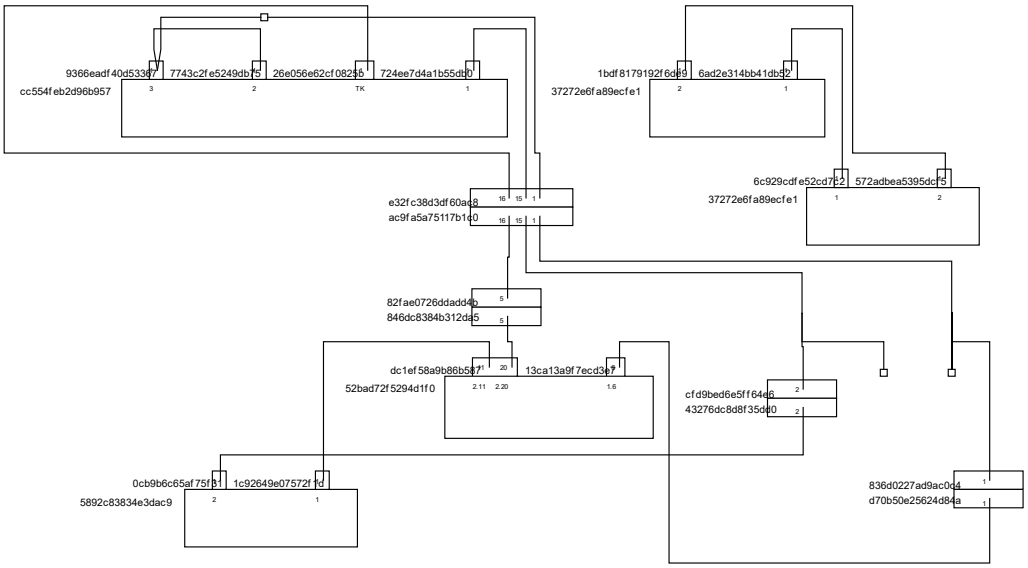


Abb. 1.1: Visualisierung eines Beispielgraphen nach dem bisherigen Verfahren

1.1 Problemstellung

Ziel dieser Arbeit ist es, einen Visualisierungsansatz zur Verfügung zu stellen, welcher die gegebenen Sonderfälle berücksichtigt. Hierdurch soll eine Verbesserung der bisherigen Ergebnisse – unter Verwendung von *KLay Layered* – erreicht werden. Hierzu werden die folgenden Erweiterungsmöglichkeiten untersucht.

Eine Optimierungsmöglichkeit stellt die Vorverarbeitung dar. Da ein ungerichteter Graph zugrunde liegt, jedoch ein gerichteter für den Algorithmus benötigt wird, müssen den Kanten Richtungen zugewiesen werden. Sollten diese nicht aus den Metadaten hervorgehen oder keine passenden Informationen vorhanden sein, müssen Richtungen anderweitig festgelegt werden. Es besteht die Hoffnung, bei einer **geschickten Wahl der Kantenrichtung**, eine Verbesserung der Ergebnisse im Vergleich zu einer beliebigen Wahl erreichen zu können.

Die aktuelle Implementierung bietet die Möglichkeit, Ports zu konfigurieren und deren Reihenfolge entweder vom Algorithmus bestimmen zu lassen oder von Hand festzulegen. Es fehlt bisher dabei die Möglichkeit **Ports zu Gruppen zusammenzufassen** – was z.B. Kabelsträngen oder Steckverbindungen entspricht. Die Ports innerhalb der Gruppen müssen nebeneinander gezeichnet werden, wobei deren Reihenfolge untereinander variieren darf. Durch Schaffung dieser Möglichkeit können unnötige Kreuzungen, welche aufgrund einer ungeschickt gewählten, festen Eingabereihenfolge der Ports entstehen können, vermieden werden.

Innerhalb der Schaltpläne kann noch eine weitere Besonderheit auftreten: **Steckverbindungen**. Hierbei handelt es sich um Knoten, bei welchen eine Veränderung der Reihenfolge der Ports auf der einen Seite auch eine Änderung der Reihenfolge auf der anderen verursacht. Genauer haben diese Knoten auf der Unterseite und Oberseite Ports. Jeder

Port der Oberseite ist mit genau einem der Unterseite gekoppelt. Wird nun die Reihenfolge auf der einen Seite festgelegt, werden die jeweiligen Partner auf der anderen Seite ebenfalls entsprechend der festgelegten Reihenfolge angeordnet, sodass jeder Port die selbe Position hat, wie sein Partner auf der anderen Seite. Hierdurch sollen zur besseren Übersichtlichkeit Kreuzungen innerhalb der Knoten verhindert werden. Dies führt dazu, dass die Anordnung der Ports nicht komplett unabhängig von der Knotenpositionierung gehandhabt werden kann, wie es bei normalen Knoten, welche keine Steckverbindungen darstellen, der Fall ist.

1.2 Literaturüberblick

Tamassia et al. [TDB88] stellen einen Algorithmus zur Berechnung einer orthogonalen Zeichnung vor. Sie beschreiben in ihrer Arbeit den „Grid-Standard“ zur Zeichnung von Graphen, welcher beinhaltet, dass Knoten auf einem festen Raster angeordnet sind und Kanten immer orthogonal verlaufen. Bei ihrem Ansatz können verschiedene Constraints wie Minimierung der Kreuzungen oder der notwendigen Fläche berücksichtigt werden. Tamassias Algorithmus arbeitet in drei Schritten: Planar machen – durch Ersetzung von Kreuzungen durch Knoten; Orthogonal machen; Kompakt machen. Dieser Ansatz hat den Nachteil, dass Kreuzungen durch Knoten ersetzt werden. Da wir keine Beschränkungen bei der Eingabe haben, kann das bei unserem Anwendungsfall zu einer sehr großen Anzahl von Knoten führen.

Beim Grid-Standard wird ein großes Gitter mit quadratischen Zellen zugrunde gelegt und eine Zelle mit höchstens einem Bauteil belegt. Hierdurch entstehen immer orthogonale Zeichnungen.

Auch Papakostas et al. [PST97] stellen einen Layout-Algorithmus vor, welcher einen Graphen in einem Grid-Standard visualisiert. Sie beschäftigen sich ausführlich mit orthogonalen Zeichnungen von Graphen mit maximalem Knotengrad von vier und Beschränkungen der Größe der Fläche, welche die Visualisierung einnehmen wird. Diese Vorgehensweisen lassen sich jedoch nicht auf einfache Art und Weise auf Graphen mit unbeschränktem Knotengrad übertragen.

Batini et al. [BNT86] erweitern den Grid-Standard, sodass Knoten mehrere Gitterzellen einnehmen können. Hierdurch erreichen sie kompaktere Darstellungen. Auf diese Weise können auch Graphen mit Knoten mit einem höheren Grad als vier visualisiert werden können.

Für unsere Vorverarbeitung könnte sich eine Ansicht eignen, wie sie in Sanders Arbeit [San96] vor Ausführung des letzten Algorithmusschrittes angegeben ist; d. h. Knoten werden auf einem Gitter angeordnet, während Kanten nur vertikal, nicht aber horizontal eigene Gitterzellen benötigen. Hieraus könnte eine Lagenzuordnung abgeleitet werden. Sanders Algorithmus stellt gleichzeitig eine Alternative zu Sugiyama [STT81] dar. Aber auch Sander spricht Probleme bei Knoten mit zu hohem Grad an.

Zu ungerichteten Graphen finden sich im Allgemeinen hauptsächlich kräftebasierte Algorithmen wie z. B. von Kamada und Kawai [KK89] oder Fruchterman und Reingold [FR91] basierend auf den Ideen von Eades [Ead84]. Welche Möglichkeiten kräftebasierte

Verfahren für die Vorverarbeitung bieten wird in Abschnitt 3.1 näher betrachtet.

Einen anderen Ansatz verfolgt Tunkelang [Tun94]. Er beschreibt in seinem Paper basierend auf seiner Masterarbeit einen nicht-kräftebasierten Algorithmus zum Zeichnen ungerichteter Graphen. Hierbei wird zuerst das Zentrum des Graphen mithilfe von Breitensuche ermittelt und anschließend der Graph um den Knoten im Zentrum herum gezeichnet, wobei eine Kostenfunktion die Positionen der Knoten ermittelt.

Schulze et al. [SSvH14] verweisen in ihrem Paper für den Schritt der Zuordnung der Knoten zu Lagen auf die Arbeit von Gansner et al. [GKNV93]. In dieser stellen Gansner et al. aufbauend auf der Arbeit von Warfield [War77] einen Algorithmus vor, mit welchem gerichtete Graphen hierarchisch dargestellt werden können. Der Algorithmus arbeitet wie Sugiyama in mehreren Schritten. In dem für die Zuordnung der Knoten zu Lagen verantwortlichen Schritt werden zunächst Multikanten zu einzelnen Kanten zusammengefasst und entsprechend gewichtet. Mithilfe der Gewichtung soll eine Minimierung der Gesamtkantenlänge im Endergebnis erreicht werden. Ihr Hauptaugenmerk liegt hierbei auf der Berechnungsgeschwindigkeit mit dem Ziel, einen Algorithmus für interaktive Nutzung zu finden. Die Einteilung der Knoten zu Lagen erfolgt bei einem gerichteten Graphen unter Einbezug der Kantenrichtung.

Ebenfalls empfehlen Schulze et al. [SSvH14] für die Berechnung der Knotenkoordinaten den Algorithmus von Brandes und Köpf [BK02]. In ihrer Arbeit stellen die Autoren einen Ansatz vor, mit dem bei gegebener Einteilung der Knoten in Lagen und gegebener Knotenreihenfolge genaue Koordinaten ermittelt werden können. Hierbei wird vor allem darauf geachtet Knicke bei langen Kanten zu vermeiden sowie möglichst viele Kanten vertikal zu zeichnen. Diese Kriterien sind höher priorisiert als die Größe der Zeichnung, was mitunter zu sehr breiten Zeichnungen führen kann.

2 Grundlagen der Arbeit

Der von uns gewählte Ansatz orientiert sich im Wesentlichen an der Arbeit von Schulze et al. [SSvH14], welche wiederum auf dem Sugiyama-Algorithmus [STT81] basiert. Wir haben die einzelnen Schritte von Schulze et al. übernommen und an unseren Anwendungsfall angepasst und beziehen uns daher häufig auf diese beiden Arbeiten. Daher sollen im Folgenden der Visualisierungsalgorithmus für hierarchische Graphen von Sugiyama et al. sowie der Algorithmus von Schulze et al. näher betrachtet werden. Zusätzlich wird noch einmal näher auf die Arbeit von Brandes und Köpf [BK02] eingegangen, deren Algorithmus wir im Schritt der Knotenpositionierung verwenden.

2.1 Sugiyama-Algorithmus

Der Sugiyama-Algorithmus vorgestellt von Sugiyama et al. [STT81] in deren Arbeit „Methods for Visual Understanding of Hierarchical System Structures“ erhält als Eingabe einen zusammenhängenden gerichteten Graphen und arbeitet in vier Schritten. Die Schritte werden hier am Beispiel einer Hierarchie mit Kantenrichtung von unten nach oben bzw. in positive y-Richtung vorgestellt.

Schritt 1: Vorverarbeitung Für die weiteren Schritte wird eine „proper hierarchy“ benötigt. Das bedeutet, dass der Graph kreisfrei sein muss und alle Knoten in Ebenen eingeteilt sein müssen. Kanten verlaufen immer in dieselbe y-Richtung und dürfen nur Knoten verbinden, die in benachbarten Ebenen liegen. Kanten dürfen keine ebenenüberspannen. Um das zu erreichen, werden Kreise zu einem Knoten komprimiert und ebenenüberspannende Kanten erhalten in jeder Ebene einen Dummyknoten, durch welchen sie unterbrochen werden und in mehrere Dummykanten aufgesplittet werden. Wie eine Lageneinteilung vorgenommen werden kann ist nicht näher ausgeführt.

Schritt 2: Kreuzungsreduzierung Die Reihenfolge der Knoten innerhalb jeder Ebene wird angepasst, um möglichst viele Kantenkreuzungen zu vermeiden. Hierzu arbeiten die Autoren mit Matrizen, welche die Verbindungen zwischen zwei Ebenen darstellen, da sich aus ihnen die Anzahl der Kreuzungen ermitteln lässt. Das ist nur möglich, da Verbindungen über mehrere Ebenen ausgeschlossen wurden.

Da bereits die kreuzungsminimierende Anordnung der Knoten zweier Ebenen ein NP-schweres Problem ist [War77], verwenden sie eine Heuristik – die *Barycenter-Heuristik* – und zusätzlich einen *Layer-Sweep-Algorithmus*. Es wird also von unten nach oben über alle Paare von benachbarten Ebenen iteriert und wieder zurück, bis ein Abbruchkriterium

erfüllt ist; z. B. keine Änderungen nach einer Iteration. Hierbei wird die Ordnung der ersten Ebene als fest angenommen und die Knoten der nächsten Ebene werden entsprechend der Barycenter-Heuristik umsortiert. Die Barycenter-Heuristik weist jedem Knoten einen ganzzahligen Wert anhand seiner Position in der Reihenfolge der Knoten seiner Ebene zu. Beim Umsortieren der Knoten einer Ebene wird für jeden Knoten der Durchschnitt der Positionen der Knoten auf der vorherigen Ebene berechnet, zu welchen er eine Kanten hat. Dieser Durchschnittswert wird auch *Barycenter* genannt. Anhand dieser Barycenter wird die Knotenreihenfolge neu bestimmt.

Schritt 3: Knotenpositionierung Die genauen x-Koordinaten der Knoten werden ermittelt, wobei deren Reihenfolge aus Schritt 2 nicht verändert werden darf. Hierbei wird darauf geachtet, dass Kanten möglichst kurz gezeichnet werden, also verbundene Knoten möglichst ähnliche x-Koordinaten erhalten. Auch wird versucht, Knoten mit mehreren eingehenden bzw. ausgehenden Kanten möglichst mittig zwischen diesen zu positionieren, um Kanten besser auseinander halten zu können.

Schritt 4: Ausgabe Der Graph wird visualisiert, indem die Graphstruktur aus Schritt 3 wieder in den ursprünglichen Graph umgewandelt wird. Die Knotenpositionen bleiben wie berechnet; die Dummyknoten und -kanten werden durch die ursprünglichen, längeren Kanten ersetzt.

2.2 KLay Layered

Schulze et al. [SSvH14] stellen einen Algorithmus basierend auf dem von Sugiyama et al. vor. Dieser wird in der Implementierung im KIELER-Projekt unter der Bezeichnung „KLay Layered“ verwendet [kie20]. Die Ergebnisse dieser Implementierung dienen als Referenz für die Ergebnisse dieser Arbeit. Da Sugiyamas Algorithmus bereits ausführlich beschrieben wurde, soll hier auf die wesentlichen Unterschiede zu den Schritten von Sugiyama eingegangen werden.

Schritt 1: Vorverarbeitung Dieser Schritt wurde in zwei Teilschritte mit folgenden Aufgaben aufgeteilt:

Schritt 1a: Kreisfrei machen Um eine Hierarchie finden zu können, dürfen keine Kreise vorhanden sein, da alle Kanten in dieselbe Richtung gerichtet sein sollen. Hierzu werden einige Kanten umgedreht, um einen gerichteten azyklischen Graphen (DAG „directed acyclic graph“) zu erhalten. Da diese Kanten am Ende entgegen der Hierarchie gerichtet gezeichnet werden müssen, ist es von Vorteil deren Anzahl zu minimieren. Dies ist ein NP-schweres Problem und nennt sich *Feedback Arc Set*. Daher werden in der Praxis meist Heuristiken für diesen Schritt verwendet. Schulze et al. verweisen hierzu auf die Arbeit von Eades et al. [ELS93].

Schritt 1b: Lagenzuordnung Alle Knoten erhalten feste, ganzzahlige y-Koordinaten. Anhand dieser werden sie anschließend einzelnen Lagen zugeordnet. Kanten, welche mehrere Lagen überspannen würden, werden durch Einsetzen von Dummyknoten aufgesplit-

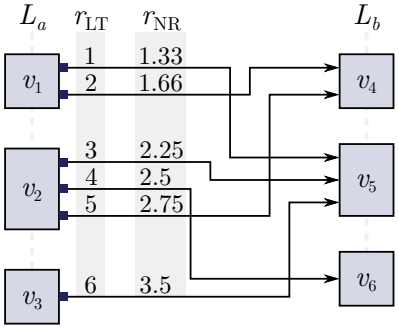


Abb. 2.1: Graphik aus dem Paper „Drawing Layered Graphs with Port Constraints“ von Schulze et al. [SSvH14]; Methoden zum Festlegen der Portkoordinaten für die Barycenterberechnung im Vergleich: r_{LT} *layer-total*-Methode; r_{NR} *node-relative*-Methode

tet. Hierzu wird auf die Arbeit von Gansner et al. [GKNV93] verwiesen. Darin wird der *Network-Simplex-Algorithmus* für die Lagenzuordnung verwendet.

Schritt 2: Kreuzungsreduzierung Dieser Schritt entspricht im Wesentlichen Schritt 2 von Sugiyama und es wird auch auf dessen Arbeit und die Verwendung der Barycenter-Heuristik verwiesen. Er wird um die Verwendung von Ports erweitert. Hierzu werden zwei Möglichkeiten zur Einberechnung von gegebenen Portreihenfolgen vorgestellt und verglichen. Entweder bekommt jeder Port – wie bei Sugiyama jeder Knoten – einen eigenen ganzzahligen Wert (*layer-total*-Methode) oder die Ports eines Knotens teilen sich dessen Wertebereich und erhalten einen Bruch als Wert für die Barycenter-Berechnung (*node-relative*-Methode). Die beiden Methoden sind in Abbildung 2.1 abgebildet, welche die Erklärende Graphik aus der Arbeit von Schulze et al. zeigt. Dabei ist keine der beiden Methoden generell besser. Laut Schulze et al. hängt es vom Anwendungsfall ab, welche Methode zu besseren Ergebnissen führt.

Sollten keine Reihenfolgen vorgegeben sein, kann Sugiyama wie gewohnt verwendet werden. Die Portreihenfolge kann dann frei gewählt und daher erst nach der Anordnung der Knoten berechnet werden. Die Reihenfolge der Ports wird also zwischen Schritt 2 und 3 berechnet sofern sie nicht vorgegeben wurde.

Schritt 3: Knotenpositionierung Die Positionierung der Knoten entspricht ebenfalls im Wesentlichen Schritt 3 von Sugiyama. Zur Implementierung wird hier auf die Algorithmen von Sander [San94] sowie Brandes und Köpf [BK02] verwiesen. Zur Unterstützung von Ports wird die Berechnung der Port-Koordinate nach der Berechnung der Knoten-Koordinaten durchgeführt. Im Normalfall werden diese einfach gleichmäßig auf die zugehörige Seite des Knotens verteilt. Bei der Berechnung der Knotenkoordinaten gilt es die Knotengrößen groß genug entsprechend der Anzahl der Ports zu wählen, sofern keine anderen Vorgaben für die Knotengröße vorliegen.

Schritt 4: Kantenführung Schulze et al. arbeiten nicht einfach nur mit geraden Verbindungen sondern mit einer orthogonalen Kantenführung. Hierzu müssen für die Kantenführung hauptsächlich passende Koordinaten für Knicke berechnet werden. Des Weiteren sollen auch Multikanten unterstützt werden. Hierzu wird auf die Arbeiten von Eschbach [EGB06] und Sander [San03] verwiesen.

Zusätzlich zu diesen Abweichungen stellen die Autoren Möglichkeiten vor, mit Ports auf allen Seiten der Knoten zu arbeiten. Wir werden diese hier nicht weiter betrachten und verweisen daher für Details auf ihren Artikel [SSvH14].

2.3 Algorithmus von Brandes und Köpf

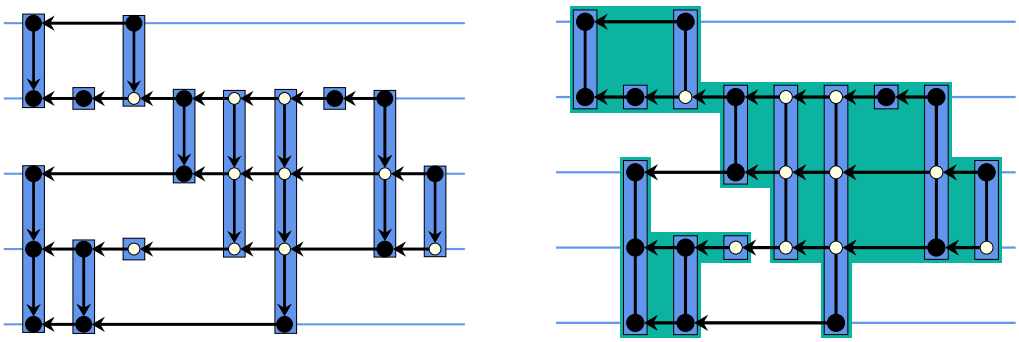
Die Arbeit von Brandes und Köpf [BK02] stellt einen passenden Ansatz für Schritt 3 von *KLay Layered* dar. Hierbei sollen Knotenpositionen bestimmt werden mit dem Ziel, möglichst viele der Kanten vertikal zu zeichnen um eine bessere Übersichtlichkeit zu gewährleisten. Es werden zunächst vorübergehend Kanten entfernt um Kreuzungen zu beseitigen – da von zwei sich kreuzenden Kanten nicht beide vertikal gezeichnet werden können – und die Knoten anschließend entlang ihrer verbliebenen Kanten untereinander angeordnet.

Entfernt werden vorwiegend Kanten, welche originale Graphknoten verbinden. Dadurch werden lange Kanten bevorzugt gerade gezeichnet, da Kanten, welche zwei Dummyknoten verbinden, immer Teilstücke einer mehrere ebenenüberspannenden Kante sind. Anschließend werden bei Knoten mit mehreren Kanten weiterhin Kanten entfernt, bis jeder Knoten nur noch eine Kante hat.

Um bei Kreuzungen zwischen zwei bevorzugten Kanten, sowie beim Entfernen von Kanten von Knoten mit mehreren Kanten zu entscheiden, werden immer die Kanten auf einer Seite bevorzugt. Hierbei kann vorab entschieden werden, ob immer die Kante vom linken oder rechten Startknoten bevorzugt wird, sowie ob zunächst die Start- oder Endknoten verglichen werden sollen.

Nachdem jeder Knoten jeweils maximal eine Kante nach oben und unten hat, werden zusammenhängende Knoten in Blöcke zusammengefasst. Die Aufteilung in Blöcke und Klassen ist in Abbildung 2.2 zu sehen. Blöcke orientieren sich immer an deren obersten Knoten – der Wurzel eines Blocks. Klassen orientieren sich immer an dem Block, der am weitesten links ist – die Wurzel dieses Blocks ist die Senke der Klasse. In Summe führt das zu einer Ausrichtung in Richtung einer Ecke, in diesem Fall oben links.

Danach werden direkt benachbarte Blöcke zu Klassen zusammengefasst – zwei Knoten sind direkt benachbart, wenn sie auf der selben Ebene direkt nebeneinander liegen, z. B. der zweite und dritte Knoten der ersten Ebene. Bei der Zusammenfassung von Blöcken entscheidet für einen Block immer, zu welcher Klasse der direkte linke Nachbar der Wurzel dieses Blocks gehört. Der Klasse dieses Nachbarknotens wird auch dieser Block zugeteilt. Existiert kein solcher direkter linker Nachbar, handelt es sich um eine Senke. Diese Klassen werden nun möglichst nah zusammengeschoben, sodass schlussendlich auch alle Klassen direkt benachbart sind.



(a) Aufteilung in Blöcke – blau **(b)** Einteilung in Klassen – grün

Abb. 2.2: Graphiken aus dem Paper „Fast and Simple Horizontal Coordinate Assignment“ [BK02]; Graph nach dem Entfernen sich kreuzender Kanten

Dieses Verfahren wird vier Mal ausgeführt. Jeweils einmal mit Ausrichtung oben links, unten links, oben rechts und unten rechts. Aus den errechneten Positionen wird dann der Mittelwert gebildet und die Knoten dort positioniert.

3 Algorithmus

Es wird zunächst darauf eingegangen, wie die Spezialfälle, die sich aus der Aufgabenstellung ergeben, angegangen werden. Anschließend wird ein Überblick über die verwendete Datenstruktur gegeben und der komplette Algorithmus im Detail beschrieben.

3.1 Umsetzung der Problemstellung

Grundsätzlich orientiert sich der Algorithmus an dem von Schulze et al. [SSvH14]. Es werden also die zwei Vorverarbeitungsschritte *kreisfrei machen* und *Lagenzuordnung* durchlaufen sowie die Hauptverarbeitungsschritte *Kreuzungsreduzierung*, *Knotenpositionierung* und *Kantenführung*. Die drei eingangs genannten Problemstellungen sollen auf folgende Art und Weise behandelt werden:

Die **Vorverarbeitung** beinhaltet insbesondere die Zuweisung von Richtungen zu den Kanten. Hierzu haben wir zunächst die Möglichkeit betrachtet, die Knoten Ebenen mit fester Kapazität zuzuordnen. Um die Zeichnung möglichst kompakt zu halten sollte die Anzahl der notwendigen Ebenen minimiert werden. Die Kanten könnten dann von der ersten Ebene aus nach oben gerichtet werden. Damit eine möglichst gleichmäßige Verteilung und somit Visualisierung der Knoten zustande kommt, müssten für diesen Vorverarbeitungsschritt folgende Bedingungen gelten.

Jede Ebene beinhaltet Knoten und Platz für ebenenüberspannende Kanten bzw. Dummyknoten hierzu. Jede Ebene hat eine vorgegebene Kapazität, welche nicht überschritten werden darf. Alle Knoten haben eine vorgegebene Größe in Form eines Wertes, welcher von der Kapazität einer Ebene abgezogen wird, wenn der Knoten ihr zugeordnet wird. Die erste und letzte Ebene beinhalten nur Knoten, da sie nicht überspannt werden können. Knoten welche durch eine Kante verbunden sind dürfen nicht derselben Ebene zugeordnet werden. Existiert somit eine ausgehende Kante eines Knotens der ersten Ebene zu einem Knoten, welcher nicht in der zweiten Ebene angeordnet ist, muss hierzu in der zweiten Ebene der Platz für einen Dummyknoten für diese Kante reserviert werden.

Es gibt in der Literatur, wie in Kapitel 2 dargestellt, einige Arbeiten in ähnliche Richtungen. Jedoch ist uns kein passender Algorithmus für die Vorverarbeitung bekannt, mit welchem die zusätzlichen Freiheiten genutzt werden können, die durch eine Unabhängigkeit von Kantenrichtungen sowie einer Nichtberücksichtigung der Kantenführung gegeben sind. Da keinerlei Beschränkungen auf den Eingabegraphen gegeben sind, stellt es sich ebenfalls als schwierig dar eine feste Kapazität der einzelnen Ebenen vorzugeben, da ohne eine aufwändige Analyse des Graphen nicht sichergestellt werden kann, dass eine Lösung auch möglich ist.

Eine alternative Idee ist die Nutzung eines kräftebasierten Layouts des Eingabegraphen um Kantenrichtungen zu bestimmen. Hierzu können aus einer solchen Darstellung

beispielsweise die Lagen der Endknoten der Kanten genommen werden, um deren Richtung zu bestimmen. So könnten sie jeweils in positive x-Richtung gerichtet werden. Eine hierzu passende Darstellung könnte dahingehend beeinflusst werden, dass bei der Layoutberechnung ein Rahmen der gewünschten Form hinzugefügt wird, welcher die Knoten nach innen abstößt. Die Effektivität dieser Vorverarbeitung soll in dieser Arbeit getestet werden.

Portgruppierungen sollen realisiert werden, indem der Zwischenschritt zwischen Schritt 2 und 3 aus Abschnitt 2.2 – in welchem die Reihenfolge der Ports mithilfe der Barycenter-Heuristik berechnet wird – erweitert wird. Für Portgruppen in unseren Daten gilt folgende Beschränkung: Ein Port ist entweder genau einer Gruppe oder genau einem Knoten zugeordnet und eine Gruppe ist ebenfalls entweder genau einer anderen Gruppe oder genau einem Knoten zugeordnet. Da nur solche Gruppen, die eine Baumstruktur bilden, erlaubt sind, können die Verschachtelungen mit einer Tiefensuche der Reihe nach aufgelöst werden. Es wird also für jede Gruppe zunächst die Reihenfolge ihrer Elemente ermittelt. Aus deren Barycentern wird dann das Barycenter der Gruppe bestimmt. Hierbei zählt jeder Port gleichwertig. Das bedeutet, enthält eine Gruppe einzelne Ports und eine andere Gruppe mit zwei Ports, fließt das Barycenter der untergeordneten Gruppe doppelt in die Berechnung des Barycenters der übergeordneten Gruppe ein, da sie zwei Ports repräsentiert.

Steckverbindungen können unterstützt werden, indem der *Sweep-Line-Algorithmus* im Schritt der Kreuzungsreduzierung angepasst wird. Das ist möglich, indem der Stecker beim Umsortieren der Ebene, in welcher er sich befindet, als Knoten gehandhabt wird. Anschließend werden in einem Zwischenschritt seine Ports passend zu der Ebene sortiert, von der der Algorithmus gekommen ist. Im nächsten regulären Schritt werden die umsortierten Ports als einzelne Knoten für die *Barycenter-Heuristik* betrachtet. Hierzu kann eines der beiden von Schulze et al. [SSvH14] hierfür entwickelten Verfahren verwendet werden.

Grundsätzlich sind **Portgruppierungen** und **Steckverbindungen** komplexere Strukturen als die Knoten zweier Ebenen, wofür bei Sugiyama die Barycenter-Heuristik eingesetzt wird. Die eben beschriebenen Herangehensweisen lassen sich nur problemlos umsetzten, wenn gewährleistet ist, dass keine Ports auf der Unterseite eines Knotens Verbindungen zu Knoten oberhalb haben und umgekehrt. Dies kann jedoch nicht ausgeschlossen werden und auch bei der Richtungszuweisung nicht erzwungen werden ohne eventuell Kreise zu erzeugen. Wir haben daher verschiedene Heuristiken entwickelt, diese Sonderfälle in den allgemeinen Sugiyama-Algorithmus einzubauen.

Die erste Möglichkeit ist die Knoten in den ersten Schritten des Algorithmus wie normale Knoten zu behandeln und die Anordnung der Ports zwischen Schritt 2 und 3 festzulegen. Hierbei würde sich durch die Anordnung der anderen Knoten eine Reihenfolge der eingehenden Kanten um den Knoten herum ergeben, an welche nun die Reihenfolge der Ports mit möglichst wenigen Kreuzungen anzupassen versucht wird.

Die Idee ist dargestellt in Abbildung 3.1. Links zu sehen ist ein Knoten mit Ports, welche – gekennzeichnet durch die farbigen Rechtecke – zum Teil in Portgruppen untergliedert sind. Die Linien, die auf dem Kreis enden, stellen die eingehenden Kanten dar. Daraus ergibt sich eine Reihenfolge, wenn man den Kreis im oder gegen den Uhrzeiger-

(a) Knoten mit gegebener Reihenfolge eingehender Kanten

(b) Ports sortiert entsprechend der Reihenfolge der eingehenden Kanten

Abb. 3.1: Problem des Findens der Portreihenfolge, welche die wenigsten Kreuzungen verursacht – Ein Knoten mit Ports (Kreise) und Portgruppen (farbige Rechtecke); links: Ausgangslage eines einzelnen Problems, gegeben Reihenfolge eingehender Kanten auf dem Kreis – gesucht Reihenfolge der Ports; rechts: Kanten durch gestrichelte Linien mit zugehörigen Ports verbunden, nachdem deren Reihenfolge an die der Kanten angepasst wurde

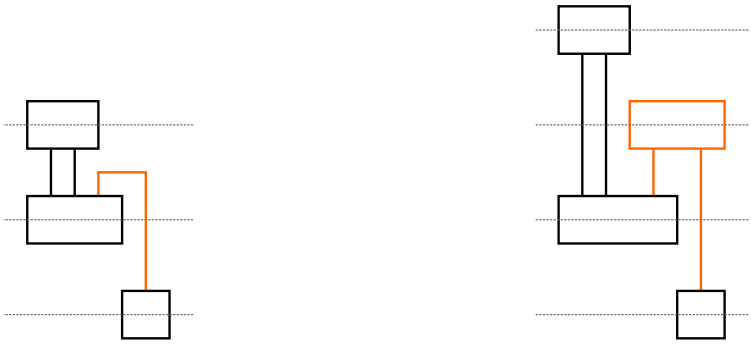
sinn abläuft. Wenn die Ports in genau dieser Reihenfolge angeordnet werden können, kann der Knoten ohne zusätzliche Kreuzungen angeschlossen werden. In diesem Fall ist durch die Gruppen kein kreuzungsfreier Anschluss möglich – daher sollte eine Möglichkeit mit minimaler Anzahl an Kreuzungen gefunden werden, wie in Abbildung 3.1b.

Hierbei ist eine Methode gefragt, welche sowohl Portgruppen als auch PortPairings berücksichtigen kann, da durch diese Vorgaben nicht alle Portreihenfolgen möglich sind. Mit einer geeigneten Methode die Portreihenfolge zu bestimmen, könnte diese gegebenenfalls abgewandelt werden, sodass sie wie oben beschrieben auch als Zwischenschritt während Schritt 2 eingesetzt werden kann; dass also die Ports sortiert werden, nachdem die Ebene, in welcher sich der Knoten selbst befindet, sortiert wurde. Anschließend kann deren Reihenfolge in die Sortierung der nächsten Ebene einfließen.

Im Rahmen dieser Arbeit wurde zunächst versucht, das Problem von Ports auf der falschen Seite eines Knotens durch zusätzliche Dummyknoten zu lösen. Hierzu werden über und unter jeder Ebene – sofern benötigt – je eine Zwischenebene eingefügt. Für jeden Port, der eine Kante zur Ebene auf der anderen Seite des Knotens hat, wird ein Dummyknoten eingefügt. Sitzt also ein Port auf der Unterseite eines Knotens und hat eine Kante zu einem Knoten in der Ebene darüber, wird ein Dummyknoten in der unteren Zwischenebene eingefügt und diese Kante über den Dummyknoten geleitet. Somit hat der Port nur noch Kanten zur Ebene darunter.

Ein Beispiel hierzu findet sich in Abbildung 3.2. Abbildung 3.2a zeigt einen Knoten, bei dem durch eine Portgruppe alle Ports auf einer Seite – der Oberseite – sind. Um zu verhindern, dass er eine Kante (orange markiert) von der Oberseite in die Ebene darunter hat, wird ein Dummyknoten erstellt (ebenfalls orange), zu sehen in Abbildung 3.2b. Für diesen wird eine neue Ebene direkt oberhalb des mittleren Knotens angelegt.

Nach Schritt 1 steht die Anzahl der Kanten jeder Portgruppe zu den Ebenen darüber und darunter fest. Die Portgruppen werden nun so platziert, dass die Anzahl der



(a) Vorher **(b)** Nachher

Abb. 3.2: Erstellung von Dummyknoten zur Vermeidung von Kanten mit Verankerung des zugehörigen Ports auf der falschen Seite eines Knotens; die orange Kante links wird rechts durch einen Dummyknoten und zwei Dummykanten ersetzt; hierzu wird eine Zwischenebene für Dummyknoten angelegt

Dummyknoten in der Zwischenebene minimiert wird. D. h. hat eine Portgruppe zwei Verbindungen zur nächsthöheren Ebene und fünf Kanten zu der Ebene darunter, wird sie auf die Unterseite des Knotens gelegt, da hierdurch nur zwei Dummyknoten entstehen, anderenfalls würden fünf benötigt.

Nach Anwendung diese Verfahrens kann die Barycenter-Heuristik wie angedacht zum behandeln von Portgruppen und Steckverbindungen verwendet werden.

3.2 Datenstruktur

Die Eingabedaten liegen im Format entsprechend der Datenstruktur vor, welches im UML-Diagramm in Abbildung 3.3 dargestellt ist. Sie beinhaltet die folgenden Sonderfälle, welche nicht unterstützt werden und in Schritt 0 des Algorithmus umgewandelt werden: Kanten mit mehreren Ports (`Edge`), Knotengruppen (`VertexGroup`), Ports mit mehreren Kanten (`Port`), Kanten, welche ein und denselben Knoten verbinden sowie nicht zusammenhängende Graphen.

Bei `PortGroup`s sind folgende Fälle möglich: Es sollen beliebig tiefe Verschachtelungen von Gruppen von `Port`s erlaubt sein, jedoch keine überlappenden. Das bedeutet, hat ein Knoten (`Vertex`) wie in Abbildung 3.4 fünf `Port`s und die ersten drei bilden eine Gruppe (`PortGroup`), dann können die ersten zwei ebenfalls eine Gruppe bilden, was einer Gruppe in einer Gruppe entspricht, wie in Abbildung 3.4b. Der dritte und vierte Port dürfen keine Gruppe bilden – zu sehen in Abbildung 3.4c – da sonst Port 3 (rot markiert) direkt zwei Portgruppen zugeordnet wäre. Für `PortGroup` (sowie auch für `VertexGroup`) gilt demnach immer, dass die Schnittmenge zweier Gruppen entweder leer ist oder alle Ports bzw. Knoten der Gruppe enthält, welche weniger Knoten beinhaltet.

Eine `PortGroup` kann demnach beliebig viele Ports und andere Portgruppen enthalten. Jeder `Port` und jede `PortGroup` ist genau einer Portgruppe oder einem Knoten direkt

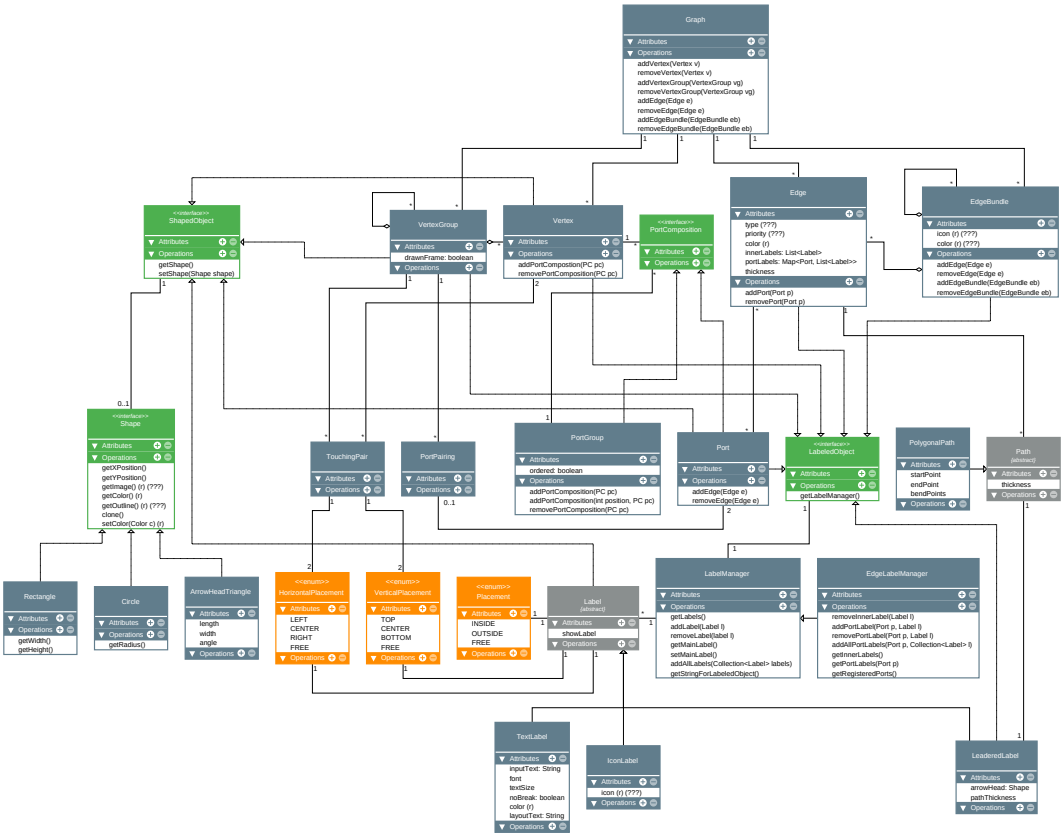
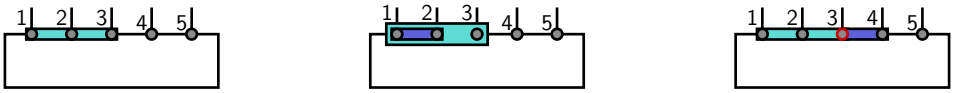


Abb. 3.3: UML-Diagramm der zugrunde liegenden Datenstruktur



(a) Knoten mit `PortGroup` **(b)** zulässige `PortGroup` **(c)** unzulässige `PortGroup`

Abb. 3.4: Beispiel für mögliche Portgruppen; links Knoten mit fünf Ports und bestehender `PortGroup` (hellblau); mittig Knoten mit zulässiger weiterer `PortGroup` (blau) – Portgruppen dürfen weitere Portgruppen enthalten; rechts aufgrund der bestehenden `PortGroup` nicht erlaubte, weitere `PortGroup`, da Port 3 (rot) direkt zwei Gruppen zugeordnet wäre (blau und hellblau)

zugeordnet. Jede `VertexGroup` kann beliebig viele Knoten und Knotengruppen enthalten. Jeder `Vertex` und jede `VertexGroup` kann maximal einer `VertexGroup` zugeordnet sein. Es ergibt sich also eine Baumstruktur, welche in linearer Zeit durchlaufen werden kann.

Die Sonderfälle der Steckverbindungen sind als `VertexGroup` bestehend aus genau zwei Knoten mit genau einem `TouchingPair` gegeben. Beide Knoten haben genau gleich viele Ports und für jeden Port des einen Knotens existiert genau ein `PortPairing` zu genau einem Port des anderen Knotens.

3.3 Beschreibung

Wir habend die fünf Schritte wie im Algorithmus von Schulze et al. [SSvH14] verwendet. Im Gegensatz zu ihnen haben wir ihn nicht in vier Hauptschritte unterteilt, sondern alle Schritte einzeln durchnummeriert. Auch haben wir den Algorithmus um den Schritt *Dummyknotenerstellung* erweitert, welcher an die *Lagenzuordnung* anschließt. Demnach besteht unser Algorithmus aus insgesamt 7 Schritten. Diese finden sich zunächst in einer kurzen Übersicht und werden im Folgenden im Detail vorgestellt.

0. *Vorverarbeitung* – der Eingabegraph wird so umgewandelt, dass eine Kante immer genau zwei Knoten über jeweils genau einen Port pro Knoten verbindet und dass jeder Port mit genau einer Kante verbunden ist.

1. *Richtungszuweisung* – der ungerichtete Eingabegraph wird in einen gerichteten, kreisfreien Graphen umgewandelt.

2. *Lagenzuordnung* – jeder Knoten wird einer Lage zugeordnet (jedem Knoten wird ein Rang entsprechend seiner Lage zugewiesen), sodass bei allen Kanten der Endknoten einen höheren Rang hat als der Startknoten.

3. *Dummyknotenerstellung* – alle notwendigen Dummyknoten werden erstellt um zu gewährleisten, dass jede Kante zwei Knoten verbindet, welche genau einen Rang auseinander liegen und dass alle Ports auf der Unterseite eines Knotens nur Kanten eingehende Kanten von unten sowie Ports auf der Oberseite nur ausgehende Kanten nach oben haben.

4. *Kreuzungsreduzierung* – eine Reihenfolge der Knoten jeder Ebene wird mithilfe der Barycenter-Heuristik festgelegt.

5. *Knotenpositionierung* – endgültige Positionen der Knoten werden festgelegt, wobei möglichst viele Kanten vertikal verlaufen sollen und genügend Platz für Beschriftungen innerhalb der Knoten bereitgestellt wird.

6. *Kantenführung* – genaue Pfade, wie die Kanten gezeichnet werden sollen werden festgelegt.

Schritt 0: Vorverarbeitung Der Eingabegraph beinhaltet einige Besonderheiten, die vom restlichen Algorithmus nicht unterstützt werden. Hierzu gehören Kanten mit mehreren Ports, Knotengruppen, Ports mit mehreren Kanten, Kanten, welche ein und denselben Knoten verbinden sowie nicht zusammenhängende Graphen.

Kanten, welche mehr als einen Port haben, werden durch einen Dummyknoten ersetzt. Dieser Dummyknoten wird dann mit allen Ports der ursprünglichen Kante über jeweils eine neue Kante verbunden.

Eine Knotengruppe wird durch einen Dummyknoten ersetzt. Sofern sich alle Knoten über `TouchingPairs` berühren, wird für jeden Knoten eine Portgruppe angelegt und ggf. die Information über `PortPairings` erhalten. Der Dummyknoten wird nun mit allen Knoten bzw. deren Ports verbunden, zu denen Knoten der Gruppe Verbindungen hatten.

Ports mit mehreren Kanten werden durch eine Portgruppe ersetzt; dieser wird für jede Kante ein Port hinzugefügt und die Kanten zu diesen Ports umgeleitet.

Verbindet eine Kante ein und denselben Knoten über unterschiedliche Ports, werden diese zunächst entfernt. Im Schritt *Knotenpositionierung* wird dann darauf geachtet, an einer Seite des Knotens Platz für diese beiden Ports zu lassen. Deren Kante kann sehr nah am Knoten gezeichnet werden und führt daher zu keinen vermeidbaren Kreuzungen. Unvermeidbare Kreuzungen können aufgrund ungünstiger Portgruppen entstehen.

Der Graph wird immer in seine Zusammenhangkomponenten zerteilt, sodass der Algorithmus einen zusammenhängenden Graph als Eingabe bekommt. Bei der aktuellen Implementierung wird der Algorithmus dann für die größte Komponente ausgeführt. Dieser Schritt lässt sich erweitern, sodass der Algorithmus für jede Komponente einmal ausgeführt wird. Weiter Überlegungen, was hier noch umgesetzt werden könnte, finden sich in Kapitel 5.

Schritt 1: Richtungszuweisung Da wir einen ungerichteten Graphen als Eingabe bekommen, können wir die Kantenrichtungen direkt so wählen, dass der Graph kreisfrei ist. Da nur die Richtungen festgelegt werden müssen, haben wir den Schritt *kreisfrei machen* durch den Schritt *Richtungszuweisung* ersetzt. Hierfür haben wir drei intuitiv naheliegende Möglichkeiten untersucht. Eine Richtungszuweisung durch Breitensuche, eine Richtungszuweisung durch ein kräftebasiertes Layout sowie eine komplett zufällige.

Für erstere wird eine Breitensuche von einem zufälligen Knoten aus durchgeführt. Während dieser werden die Kanten entsprechend der Reihenfolge gerichtet, in welcher die Knoten gefunden werden, sodass sie immer von dem Knoten, welcher früher gefunden wurde, zu dem gerichtet sind, welcher später gefunden wurde. Der Startknoten stellt also die einzige Quelle im Graphen dar.

Für die zweite Möglichkeit wird zunächst ein kräftebasiertes Layout des Graphen erzeugt. Hierbei wird, um ein möglichst gleichmäßiges Ergebnis zu erreichen, zusätzlich ein rechteckiger Rahmen eingefügt, der das gewünschte Seitenverhältnis hat. Die Knoten werden nun initial innerhalb des Rahmens platziert. Der Rahmen hat – genau wie die Knoten untereinander – eine abstoßende Kraft auf die Knoten, sodass sie sich möglichst gleichmäßig auf den zur Verfügung stehenden Raum verteilen und nicht alle an den Rand geschoben werden.

Die Richtungsbestimmung folgt dann aus der fertigen Zeichnung, die mit dem kräftebasierten Verfahren erstellt wurde, indem eine Seite des Rahmens als unterer Rand gewählt wird. Alle Knoten werden anhand ihres Abstands zu dieser Seite sortiert und die Kanten entsprechend dieser Reihenfolge so gerichtet, dass sie vom unteren Rand weg zeigen. In der aktuellen Implementierung wird hierzu die JUNG-Library [jun20] verwendet.

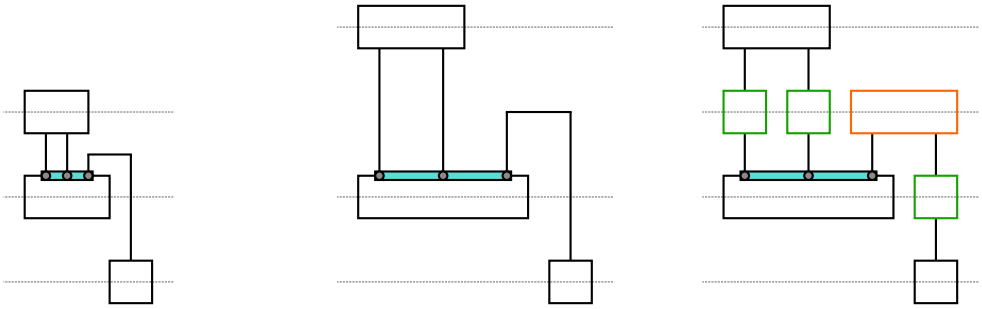
Für die zufällige Richtungsbestimmung werden allen Knoten zufällige, unterschiedliche, ganzzahlige Werte zugewiesen. Kanten werden immer so gerichtet, dass sie zu dem Endknoten mit größerem Zufallswert zeigen. Hierdurch wird verhindert, dass gerichtete Kreise erzeugt werden können. Es wird also zufällig eine totale Ordnung über den Knoten festgelegt und die Kanten dementsprechend gerichtet.

Schritt 2: Lagenzuordnung Dieser Schritt wird entsprechend der Beschreibung in der Arbeit von Gansner et al. [GKNV93] mithilfe eines *Network-Simplex-Algorithmus* durchgeführt. Es wird mit einem beliebigen Spannbaum des Graphen begonnen. Anschließend werden die *Schnittwerte* berechnet. Hierzu wird eine Kante aus dem Spannbaum entfernt, was diesen in zwei Teile aufteilt. Der Schnittwert ergibt sich dann durch die Anzahl der Kanten, welche den Schnitt kreuzen – also die beiden getrennten Teilmengen von Knoten verbinden. Die Anzahl der Kanten, welche in dieselbe Richtung gerichtet sind wie die ausgewählte Baumkante, geht positiv in den Schnittwert ein – die der entgegen gerichteten Kanten negativ. Ist die Summe negativ, wird eine alternative Kante der Schnittkanten mit positivem Schnittwert gesucht und die ausgewählte Kante durch diese ersetzt. Anschließend werden alle Schnittwerte neu berechnet.

Dieses Verfahren wird so lange ausgeführt, bis ein lokales Minimum erreicht wird, was daran zu Erkennen ist, dass keine Kante einen negativen Schnittwert hat. Danach werden alle Knoten Ebenen zugewiesen, indem jeder Knoten genau eine Ebene höher, als seine Nachbarn von eingehenden Baumkanten eingeordnet wird. Umgekehrt wird jeder Knoten genau eine Ebene tiefer als seine Nachbarn von ausgehenden Baumkanten eingeordnet. Eine solche Anordnung sorgt für eine Minimierung der Gesamtkantenlänge.

Schritt 3: Dummyknotenerstellung Vor Ausführung der Kreuzungsreduzierung muss sichergestellt werden, dass keine ebenenüberspannenden Kanten vorhanden sind. Hierzu werden bei allen Kanten, deren Knoten weiter als eine Ebene auseinander liegen, in allen Ebenen dazwischen Dummyknoten eingefügt und eine lange Kante somit durch mehrere kurze Kanten ersetzt.

Um sicherzustellen, dass Portgruppen wie geplant verarbeitet werden können, müssen diese auf einer Seite – oben oder unten – des Knotens liegen. Hat eine Portgruppe jedoch sowohl Ports mit Verbindungen nach unten als auch nach oben, werden auch hier Dummyknoten eingefügt, wie in Abschnitt 3.1 beschrieben. Hierzu wird die Anzahl der Kanten der Portgruppe nach oben und die nach unten berechnet. Die Portgruppe wird nun auf die Seite des Knotens platziert, in deren Richtung sie mehr Verbindungen hat. Wurde eine Portgruppe auf der Oberseite eines Knotens platziert, werden für alle Kanten, welche zu Knoten in niedrigeren Ebenen führen, Dummyknoten in einer Zwischenebene oberhalb des Knotens erstellt. Somit führen diese Kanten nach oben aus dem Knoten



(a) Ursprungsgraph **(b)** auseinander geschoben **(c)** mit Dummyknoten

Abb. 3.5: Erstellung von Dummyknoten; links Ursprungsgraph, in der Mitte Ursprungsgraph mit originalen Knoten an neuen Positionen, rechts Graph nach Schritt 3 des Algorithmus; Dummyknoten aufgrund von Portgruppen orange, Dummyknoten aufgrund von ebenenüberspannenden Kanten grün, originale Knoten schwarz, Ports der Portgruppe Kreise, Portgruppe hellblaues Rechteck

heraus jeweils zu einem eigenen Dummyknoten, welcher dann eine Kante nach unten zum anderen Knoten der ursprünglichen Kante hat.

Abbildung 3.5 zeigt einen kleinen Beispielgraphen vor und nach dem Einfügen der Dummyknoten. In Abbildung 3.5a ist der Beispielgraph abgebildet. Er hat drei Knoten in drei verschiedenen Ebenen – gekennzeichnet durch die gestrichelten Linien. Der Knoten der mittleren Ebene besitzt eine Portgruppe – zugehörige Ports sind gekennzeichnet durch Kreise. Der Port der rechten Kante befindet sich trotz der Verbindung nach unten auf der Oberseite des Knotens, da sich alle Ports einer Portgruppe auf derselben Seite befinden müssen und die Verbindungen der anderen beiden Ports nach oben gehen. Zur besseren Übersichtlichkeit sind zunächst in Abbildung 3.5b die Knoten des Beispielgraphen an die Positionen geschoben, an denen sie sich nach Ausführung von Schritt 3 befinden werden. Abbildung 3.5c zeigt den Beispielgraph nach Ausführung von Schritt 3. Dummyknoten aufgrund ebenenüberspannender Kanten sind grün markiert, der aus Abbildung 3.2 bekannte Dummyknoten aufgrund der Portgruppe orange. Aufgrund der Notwendigkeit des orangen Dummyknotens ist das Einfügen einer Zwischenebene nötig. Da die beiden linken Kanten diese nun überspannen, müssen sie auf dieser Ebene durch einen Dummyknoten unterbrochen werden. Auf der zuvor mittleren Ebene ist nun ebenfalls ein Dummyknoten nötig, da die Kante vom orangen Dummyknoten zum Knoten in der untersten Ebene nun diese Ebene überspannt.

Schritt 4: Kreuzungsreduzierung Vor diesem Schritt ist jeder Knoten einer Ebene zugeordnet und Kanten verlaufen nur zwischen benachbarten Ebenen. Die Anzahl der Kantenkreuzungen hängt also nur noch von der Permutation der Knoten und Ports auf den Ebenen ab. In diesem Schritt soll eine möglichst gut Permutation festgelegt werden.

Hierzu wird der übliche Ansatz von Sugiyama et al. verwendet – ein *Layer-Sweep-Algorithmus* und die *Barycenter-Heuristik*. Es wird mit zufälligen Permutationen der

Knoten auf den Ebenen und einer zufälligen Reihenfolge ihrer Ports begonnen. Anschließend wird, beginnend mit der zweiten Ebene, die Reihenfolge der Knoten entsprechend der Barycenter-Heuristik neu berechnet. Das heißt, jeder Knoten erhält einen Wert entsprechend dem arithmetischen Mittel der Positionen seiner adjazenten Knoten aus der vorherigen Ebene. Hierbei gehen die Positionswerte der Knoten zu denen mehrere Kanten existieren auch mehrfach ein.

Eine Besonderheit sind – aufgrund von Schritt 3 sehr häufig auftretende – Knoten, welche nur Verbindungen zu einer Ebene haben. Da hier kein Barycenter durch Kanten bestimmt werden kann, wird aus der aktuellen Position des Knotens ein Barycenter berechnet, welches ihn theoretisch an seiner Position hält. Es wird einfach das maximal mögliche Barycenter durch die aktuelle Position des Knotens geteilt. Auf diese Art werden alle Ebenen in aufsteigender Reihenfolge, nacheinander sortiert.

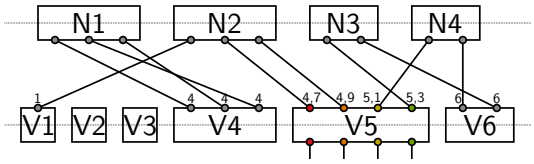
Nachdem die oberste Ebene sortiert ist, werden alle Ebenen nochmals in umgekehrter Reihenfolge sortiert. Das heißt, die oberste Ebene wird als Start-Ebene genommen und ihre Reihenfolge als fest angesehen; von ihr aus werden nach und nach die Knotenreihenfolgen der unteren Ebenen neu berechnet. Diese Schritte werden wiederholt, bis keine Verbesserung mehr erreicht wird.

Der gesamte Schritt wird mehrmals mit unterschiedlichen zufälligen Startpermutationen der Knoten ausgeführt. Unter allen Ergebnissen wird das beste – also das mit der geringsten Anzahl an Kreuzungen – ausgewählt.

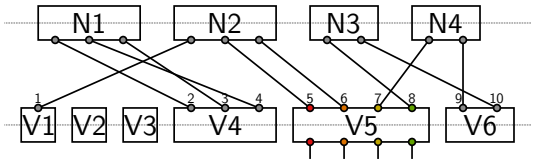
Dasselbe Verfahren kann ebenfalls verwendet werden um die Positionen der Ports zu bestimmen. Hierbei sind zunächst zwei mögliche Verfahren zum Vergleich vorgesehen: Die Bestimmung der Portreihenfolge nach der Sortierung aller Knoten-Ebenen und alternativ die Bestimmung der Portreihenfolge jeweils direkt nach der Sortierung der Knoten einer Ebene. Ersteres wird nach Abbruch der Schleife, in welcher die Knotenpositionen bestimmt werden, durchgeführt. Es werden also zunächst alle Knoten auf ihren Ebenen sortiert, wobei für alle Kanten eines Knotens derselbe Wert – nämlich die aktuelle Position des Knotens in seiner Ebene – in die Berechnung des Barycenters eingeht. Der wesentliche Unterschied ist, dass im zweiten Verfahren auch die Reihenfolge der Ports genutzt wird um die Knoten und Ports der nächsten Ebene zu sortieren und hiermit nur indirekt die Reihenfolge der Knoten. Die einzige Ausnahme bilden Knoten mit `PortPairing`. Da hier die Reihenfolge der Ports auf der einen Seite eines Knotens die Reihenfolge der Ports auf der anderen Seite beeinflusst, haben wir uns entschlossen deren Reihenfolge auch in die Berechnung auf Basis der Knotenpositionen einfließen zu lassen.

Die Berechnung im Detail lässt sich am besten anhand eines Beispiels erklären. In Abbildung 3.6 sind zwei Ebenen eines Graphen abgebildet. Die Reihenfolge der unteren Ebene (Knoten $V1$ – $V6$) steht fest, die der oberen Ebene (Knoten $N1$ – $N4$) soll berechnet werden. Bei $V5$ handelt es sich um einen Knoten mit `PortPairing`, gepaarte Ports sind in derselben Farbe markiert und genau gegenüber voneinander angeordnet.

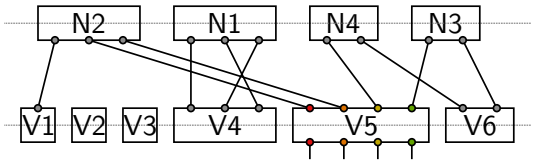
Die Berechnung unter Verwendung der Knotenpositionen ist in Abbildung 3.6a und Abbildung 3.6c abgebildet. Abbildung 3.6a zeigt die Ausgangslage und die Werte zur Berechnung der Barycenter als kleine Zahlen. Da die Knotenpositionen die Reihenfolge bestimmen, haben alle Ports eines Knotens denselben Wert. Bei $V5$ haben die Ports unterschiedliche Werte, da es sich um einen Knoten mit `PortPairing` handelt. Hier sind



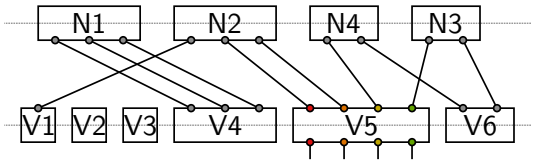
(a) Ausgangssituation mit Werten für die Barycenterberechnung als kleine Zahlen – unter Verwendung der Knotenpositionen



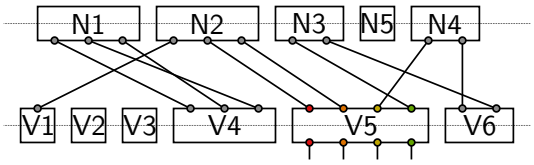
(b) Ausgangssituation mit Werten für die Barycenterberechnung als kleine Zahlen – unter Verwendung der Portpositionen



(c) Anordnung nach Sortierung mit Knotenpositionen



(d) Anordnung nach Sortierung mit Portpositionen



(e) Ausgangssituation mit zusätzlichem Knoten N5 ohne Kanten

Abb. 3.6: Barycenter-Sortierung (Sortierung der oberen Ebene anhand der Kanten zur unteren Ebene): oben vor Ausführung, unten nach Umsortierung, links unter Verwendung der Knotenpositionen, rechts unter Verwendung der Portpositionen; unten Mitte Ausgangslage mit zusätzlichem Knoten N5 ohne Kanten zur Lage darunter

Werte entsprechend der *node-relative*-Methode (Abbildung 2.1) gewählt. Für $N1$ ergibt sich also ein Barycenter von 4 ($= \frac{4+4+4}{3}$), da er drei Kanten zu $V4$ hat, welche alle mit Wert 4 in die Berechnung eingehen. $N2$ erhält somit ein Barycenter von 3,53 ($= \frac{1+4,7+4,9}{3}$) welches kleiner ist als das von $N1$. Daher sind deren Positionen in Abbildung 3.6c getauscht.

Bei striktem Vorgehen nach den Knotenkoordinaten würden $N3$ und $N4$ an ihren Positionen bleiben, da sie dasselbe Barycenter ($\frac{5+6}{2} = 5{,}5$) hätten. Bei einer beliebigen Reihenfolge der Ports würde es auch keinen Unterschied in der Anzahl der Kreuzungen machen. Da wir aber die Ports schon im vorherigen Schritt so angeordnet haben, dass auf der Unterseite des Knotens möglichst wenige Kreuzungen entstehen, wollen wir bei der aktuellen Portreihenfolge möglichst wenige Kreuzungen erzeugen, da bei einer Änderung in der Ebene darunter neue Kreuzungen entstehen können. Daher haben wir hier den Ports leichte Unterschiede gegeben, damit bei einem gleichen Barycenter zwei Knoten so verschoben werden, dass beim Beibehalten der Portreihenfolge weniger Kreuzungen entstehen.

Da wir aber auch noch andere Vorgaben wie Portgruppen haben, welche eine beliebige Reihenfolge der Ports verbieten, haben wir noch eine zweite Barycenterberechnung verwendet. Hier werden direkt die aktuellen Positionen der Ports verwendet, unabhängig davon, wie viele Ports ein Knoten hat und an welcher Position der Knoten ist. Die entsprechenden Werte sind in Abbildung 3.6b eingetragen; sie zeigt denselben Ausgangsgraphen wie Abbildung 3.6a aber mit anderen Werten für die Barycenterberechnung. Hier ergeben sich die Werte entsprechend der *layer-total*-Methode. Abbildung 3.6d zeigt eine andere Sortierung als Abbildung 3.6c. $N1$ erhält hier ein Barycenter von 3 ($= \frac{2+3+4}{3}$) das in diesem Fall kleiner ist als das von $N2$ ($4 = \frac{1+5+6}{3}$). Daher ändern sich die Positionen dieser beiden Knoten nicht.

Ein zweiter Unterschied besteht darin, dass nach der Sortierung der Knoten die Ports entsprechend ihrer Barycenter sortiert werden. Dieser Schritt erfolgt bei Verwendung der Portkoordinaten direkt im Anschluss an die Sortierung der Knoten, da beim Durchlauf von oben nach unten die Reihenfolge der Ports auf der Unterseite der oberen Ebene für die Berechnung der Barycenter der Knoten und Ports der unteren Ebene genutzt werden. Daher ist die Kreuzung der beiden Kanten zwischen $N1$ und $V4$ in Abbildung 3.6d bereits aufgelöst und bleibt in Abbildung 3.6c zunächst bestehen. Derselbe Schritt wird bei Sortierung unter Verwendung der Knotenreihenfolge auch ausgeführt, allerdings erst nachdem die Positionen der Knoten aller Ebenen feststehen. Eine Sortierung der Ports schon während der Sortierung der Knoten wäre auch möglich, hätte jedoch keinen Einfluss auf das Ergebnis und kann daher im Anschluss gemacht werden um Rechenzeit zu sparen.

Sofern Portgruppen vorhanden sind, werden zunächst diese sortiert und im Anschluss deren Ports. Das funktioniert genau wie die Sortierung der Knoten. Es werden also rekursiv zunächst die Knoten, dann deren Portgruppen, und zum Schluss die Ports sortiert. Da Portgruppen - da sie Portgruppen enthalten können - beliebig tief geschachtelt sein können, wird, um zu gewährleisten, dass ein Sortierschritt linear in der Anzahl der Ports plus der Anzahl der Portgruppen verläuft, beim rekursiven Aufruf in der Implementierung zunächst die Reihenfolge der Ports der untersten Hierarchie berechnet. Die Barycenter der Gruppenmitglieder werden dann mit einem Gewicht entsprechend der Anzahl der enthaltenen Ports zurückgegeben. Daher müssen nicht bei jeder Berechnung eines Barycenters für eine Portgruppe alle enthaltenen Ports durchlaufen werden.

Abbildung 3.6e zeigt den Sonderfall, dass ein Knoten keine Verbindungen zur vorherigen Ebene hat. Bei Knoten ohne Kanten wird versucht diese an ihrer aktuellen Position zu halten, indem sie den Barycenterwert erhalten, das sie theoretisch an ihre aktuelle Position einordnen würde. Für $N5$, der auf der oberen Ebene zwischen $N3$ und $N4$ eingefügt wurde, ergeben sich folgende Barycenter. Im Fall der Verwendung der Knoten gibt es 6 Knoten in der unteren Ebene. Daher kann ein Knoten mit Kanten maximal ein Barycenter von 6 haben und wird mindestens eines von 1 haben. Wird der Bereich der Barycenterwerte gleichmäßig auf alle Knoten verteilt, erhält $N5$ als vierter Knoten ein Barycenter von $\left[\frac{[6-1]\cdot[4-1]}{5-1}\right] + 1 = 4{,}75$. Diesen Wert weisen wir $N5$ als Barycenter zu. Im Falle der Barycenterberechnung anhand der Ports erhät $N5$ also ein Barycenter von

$\left[\frac{[10-1]\cdot[4-1]}{5-1}\right] + 1 = 7{,}75$. Wir berechnen Barycenter für diese Knoten also mit der Formel

$$barycenter = \left[\frac{[b_{max} - 1] \cdot [pos - 1]}{\#n - 1}\right] + 1 \tag{3.1}$$

wobei b_{max} das größtmögliche Barycenter ist, $\#n$ die Anzahl der Knoten der zu sortierenden Ebene und *pos* die Position des zu sortierenden Knotens.

Schritt 5: Knotenpositionierung Die Knotenpositionierung wird nach dem Ansatz von Brandes und Köpf [BK02] umgesetzt. Hierbei werden zunächst vorübergehend Kanten entfernt um Kreuzungen zu unterbinden und die Knoten anschließend entlang ihrer verbliebenen Kanten untereinander angeordnet.

In unserem Fall haben wir ebenfalls berücksichtigt, die Knoten breit genug für eine Beschriftung zu machen sowie Positionen für die Ports zu finden. Als Knoten für dieses Verfahren verwenden wir daher nicht die Knoten des Graphen, sondern deren Ports. Da jeder Knoten sowohl auf der Oberseite als auch auf der Unterseite Ports haben kann, wird jede Ebene von Knoten in zwei Ebenen von Ports aufgeteilt. Um aus den Portpositionen später auch Knotenpositionen errechnen zu können, werden zusätzliche Dummyportpaare eingefügt. Es wird also links von alle Ports auf der Oberseite eines Knotens ein Dummyport eingefügt, welcher mit einem weiteren Dummyport links von allen Ports auf der Unterseite desselben Knotens verbunden ist. Ein weiteres solches Paar wird auch rechts eingefügt, zwischen zwei Knoten aber in Summe nur eines. Da zwischen Ports auf der Ober- und Unterseite ein und desselben Knotens keine sich kreuzenden Kanten existieren können, ist gewährleistet, dass diese beiden Dummyports dieselben x-Koordinaten erhalten. Da die Reihenfolge der Ports nicht mehr geändert werden kann, ist damit ebenfalls gewährleistet, dass alle anderen Ports eines Kontens zwischen den beiden zugehörigen Dummyportpaaren angeordnet werden. Im Bereich eines Knotens werden nun noch weitere Dummyports ohne Kanten entsprechend der Breite der Beschriftung des Knotens hinzugefügt. Hierdurch wird gewährleistet, dass der Knoten breit genug für seine Beschriftung sein wird.

– Im Folgenden ist mit *Knoten* ein Knoten aus Sicht des Ansatzes gemeint, welcher einem Port des originalen Graphen entspricht; *Knoten* aus Sicht des Ansatzes sind zur bessern Übersicht kursiv, Knoten des originalen Graphen normal geschrieben. –

Die Knoten einer Ebene werden wie in Abbildung 3.7 in Ports und Kanten aufgeteilt. In Abbildung 3.7a sind zwei Knoten aus Ebene 3 zu sehen. Da es sich um Ebene 3 handelt, werden die unteren Ports in eine neue Ebene 6 eingefügt und die oberen in Ebene 7 – zu sehen in Abbildung 3.7b. (Die Ports der Knoten aus Ebene 0 belegen die neuen Ebenen 0 und 1, die der Knoten aus Ebene 1 entsprechend die neuen Ebenen 2 und 3 usw.) Von links nach rechts wird zunächst ein trennendes *Knoten*paar in Ebene 6 und 7 eingefügt aus dessen Position sich später der linke Rand des Knotens $V1$ errechnen lässt. Dann kommen die Ports von $V1$ wie beschrieben. Nach diesen kommt wieder ein trennendes *Knoten*paar und dann die Ports von $V2$.

Angenommen ein Kantenabstand von 1cm ist gewünscht, die Beschriftung von $V2$ nimmt jedoch eine Breite von 3cm in Anspruch. Um genügend Platz für die Beschriftung

(a) Knoten aus Ebene 3 **(b)** Knoten umgewandelt

Abb. 3.7: Umwandlung der Knoten in Ports und Kanten; links die Knoten $V1$ und $V2$, rechts repräsentiert durch Ports und Kanten, was als Eingabe für Schritt 5 dient

von $V2$ zu haben, wird auf der Unterseite zusätzlich zu dem einen *Knoten* für den einen Port noch links und rechts daneben jeweils ein Dummy*knoten* ohne Kante hinzugefügt. Hiermit wird $V2$ mindestens eine Breite von 3cm einnehmen und die Beschriftung wird komplett in den Knoten passen. Abgeschlossen wird alles durch ein drittes trennendes *Knoten*paar.

Durch diese Aufteilung ist ebenfalls sichergestellt, dass ein *Knoten* maximal eine Kante in die Ebene darüber und eine in die darunter hat – also maximal zwei Kanten insgesamt. Das erleichtert den ersten Schritt, da nur bei Kreuzungen eine Kante entfernt werden muss und nicht aus allen Kanten eines *Knotens* eine ausgewählt werden muss, da ja immer nur eine existiert. Auch werden hierdurch direkt die Positionen der Ports so gewählt, dass Kanten immer gerade verlaufen; Knoten können dabei unter Umständen deutlich in die Breite gezogen werden.

Um den geraden Verlauf von langen Kanten sowie von Kanten innerhalb eines Knotens mit `PortPairing` zu gewährleisten, werden hierfür ebenfalls Kanten innerhalb solcher Knoten eingefügt – zu sehen in Abbildung 3.8. $V1$ ist in Abbildung 3.8a ein Knoten mit `PortPairing` – gepairte Ports sind mit gleicher Farbe markiert – und $V2$ ist ein Dummyknoten für eine ebenenüberspannende Kante. Abbildung 3.8b zeigt deren Repräsentation, wobei die beiden Ports von $V2$ eine Verbindungskante erhalten, sowie jedes

(a) Knoten aus Ebene 3 **(b)** Knoten umgewandelt

Abb. 3.8: Umwandlung von Dummyknoten und Knoten mit `Portpairing` in Ports und Kanten; links die Knoten $V1$ mit `Portpairing` (gepairte Ports haben die gleiche Farbe) und der Dummyknoten für eine Ebenen-Überspannende Kante $V2$, rechts Repräsentation der Knoten, wobei sowohl gepairte Ports als auch die Ports des Dummyknotens durch Kanten verbunden sind

der Beiden Port-Paare von $V1$. Auch bei den Kanten für `PortPairing`s ist gewährleistet, dass keine Kreuzungen durch diese Kanten entstehen können, da diese Fälle in Schritt 4 bereits berücksichtigt wurden.

Schritt 6: Kantenführung Geplant ist eine orthogonale Kantenführung wie in Schritt 4 von „KLay Layered“ (Abschnitt 2.2). Der Fokus dieser Arbeit liegt vor allem auf der Unterstützung der Sonderfälle und hierbei insbesondere auf der Richtungszuweisung sowie der Kreuzungsreduzierung. Um hier mehr Möglichkeiten testen zu können, wurde nur eine direkte, geradlinige Verbindung umgesetzt und eine orthogonale Kantenführung aus Zeitgründen zunächst weggelassen. Die Implementierung kann jedoch jederzeit um einen passenden Schritt erweitert werden.

Die geradlinige Verbindung nutzt die in Schritt 5 errechneten Portkoordinaten. Eine Kante wird immer als gerade Linie von ihrem Start- zu ihrem Endport gezeichnet. Dummyknoten werden hierzu als waagerechte bzw. senkrechte Striche visualisiert. Eine Auflösung von Ersatzknoten für Kanten mit mehr als zwei Ports sowie für Portgruppen gibt es noch nicht.

4 Versuche

Alle Schritte des Algorithmus wurden implementiert. Für die Schritte 1 und 4 stehen mehrere Möglichkeiten zur Verfügung, die wir hier experimentell vergleichen. Da das Ziel war, eine übersichtliche Darstellung zu berechnen, Übersichtlichkeit als solche aber ein subjektives Kriterium ist, wurden einige andere Eigenschaften, welche zu einer besseren Übersichtlichkeit beitragen, zum Vergleich herangezogen. Die Eigenschaften sind die Anzahl der Kreuzungen und die Größe der Visualisierung sowie das Seitenverhältnis. Hierbei sind offensichtlich eine geringe Anzahl an Kreuzungen sowie eine kleine Fläche der Zeichnung als auch ein möglichst ausgeglichenes Seitenverhältnis positiv. Ebenfalls gemessen und hier eingebracht werden die Anzahl der Dummyknoten sowie die Breite und Höhe der Zeichnung.

Im Folgenden finden sich einige Diagramme – sogenannte Violindiagramme – welche die Ergebnisse im Vergleich zeigen. Zur besseren Übersichtlichkeit beschränken sich diese auf die ersten zehn Beispielgraphen. Eine Übersicht der Ergebnisse in dieser Form für alle Graphen befindet sich in Abschnitt A.1 im Anhang. Die Diagramme sind aus den Messwerten unter Anwendung der Methode von Scott („Scott's normal reference rule“) errechnet. In ihrer Mitte findet sich zusätzlich die Darstellung als Boxplot.

Der Testkorps besteht aus einem Datensatz von 377 Graphen, welche echte Kabelpläne repräsentieren. Beschriftungen wurden geändert aber die Struktur aus Knoten, Kanten und Ports entspricht den Originalen. Die Pläne haben sehr unterschiedliche Dimensionen; sie liegen im Bereich von 2 bis 354 Knoten sowie von 1 bis 572 Kanten.

4.1 Test 1: Richtungszuweisung

Hier stehen aktuell drei Möglichkeiten zur Verfügung: eine komplett zufällige, kreisfreie Richtungszuweisung, eine Richtungszuweisung per Breitensuche von einem zufälligen Startknoten aus sowie eine Richtungszuweisung basierend auf den Koordinaten der Knoten in einer durch einen kräftebasierten Algorithmus errechneten Darstellung. Es wurden für jeden Testgraphen 50 Visualisierungen mit jedem Verfahren berechnet, mit je 50 Durchläufen der Kreuzungsreduzierung, von welchen dann das Minimum gewählt wurde. Zu den verschiedenen Eigenschaften waren hier die Erwartungen wie folgt.

Bei Verwendung des kräftebasierten Ansatzes kommt eine Visualisierung mit wenigen Kreuzungen sowie eines ausgewogenen Seitenverhältnisses heraus. Die Breitensuche sorgt für eine Visualisierung die wenig Platz benötigt dafür aber deutlich breiter als hoch wird. Es war die Hoffnung, einen Unterschied zu einer zufälligen Richtungsbestimmung aufzeigen zu können und im Vergleich zu dieser bessere Ergebnisse erzielen zu können. Die Ergebnisse haben hier größtenteils unsere Erwartungen erfüllt.

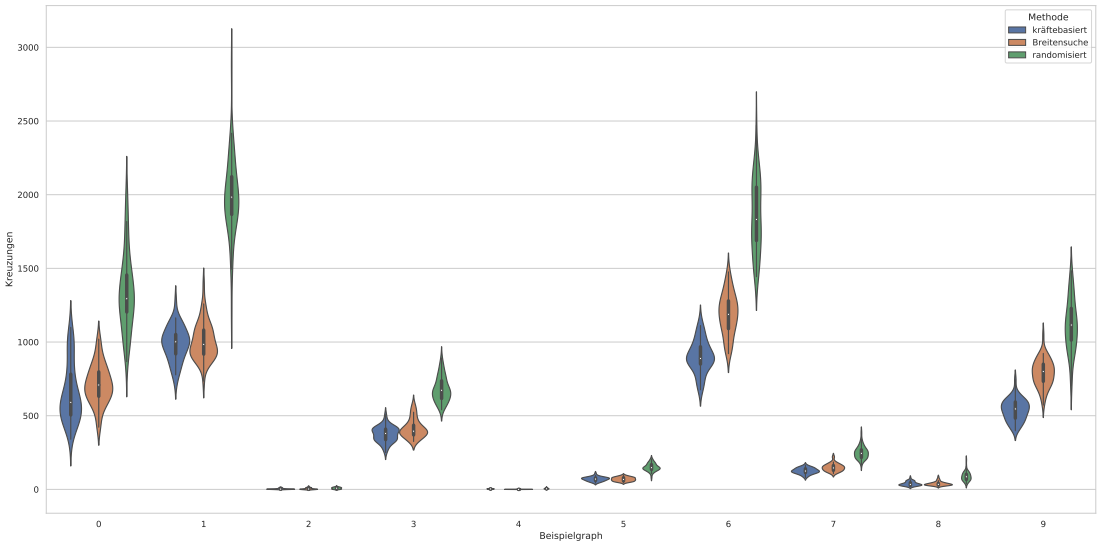


Abb. 4.1: Ergebnisse Test 1; Anzahl der Kreuzungen für alle Beispielgraphen in Abhängigkeit der Methode zur Richtungszuweisung; blau kräftebasiert; orange Breitensuche; grün zufällig

Anzahl der Kreuzungen Bei der Anzahl der Kreuzungen, was aus unserer Sicht die wichtigste Eigenschaft ist, sind eindeutige Unterschiede erkennbar. Hier schneidet die rein zufällig Richtungszuweisung deutlich schlechter ab als die anderen beiden Verfahren, was zeigt, dass es sinnvoll ist, geeignete Methoden zur Vorverarbeitung zu suchen. Im Vergleich des kräftebasierten Verfahrens zur Breitensuche ist meist der kräftebasierte Ansatz besser, jedoch gibt es auch Beispiele, bei denen die Breitensuche besser abgeschnitten hat. In den Fällen in denen die Breitensuche besser war, ist der kräftebasierte Ansatz nie deutlich schlechter, sondern beide sind sehr nah beieinander. Genauer war in 84% der Fälle der kräftebasierte Ansatz im Mittel besser oder gleich gut, gemessen am arithmetischen Mittel. Betrachtet man die Extremen, also vergleicht die Minima, wurde in 86% unserer Tests mit dem kräftebasierten Ansatz bei 50-facher Ausführung ein besseres Ergebnis erreicht. In nur einem der 377 Fälle konnte mit dem zufälligen Verfahren ein besseres Ergebnis gefunden werden.

Die Ergebnisse für die ersten 10 Graphen sind in Form von Violindiagrammen in Abbildung 4.1 abgebildet – durchnummeriert von 0 bis 9. Es ist schon hier erkennbar, dass sich die Graphen deutlich in Größe und Komplexität unterscheiden. Einige Graphen wie der Graph 4 sind kreuzungsfrei darstellbar, bei anderen wie Graph 1 wurde mit keinem Verfahren eine Visualisierung mit unter 500 Kreuzungen gefunden.

Größe der Zeichenfläche Bei der benötigten Zeichenfläche ergibt sich ein ganz anderes Bild. Positiv hier wäre eine kleine Zeichenfläche, was einer kompakten Darstellung entspricht. Die gemessenen Werte hierzu sind in Abbildung 4.2 abgebildet. Sie zeigt die Fläche der Bounding Box der Zeichnung. Es fällt zunächst auf, dass die Minima aller

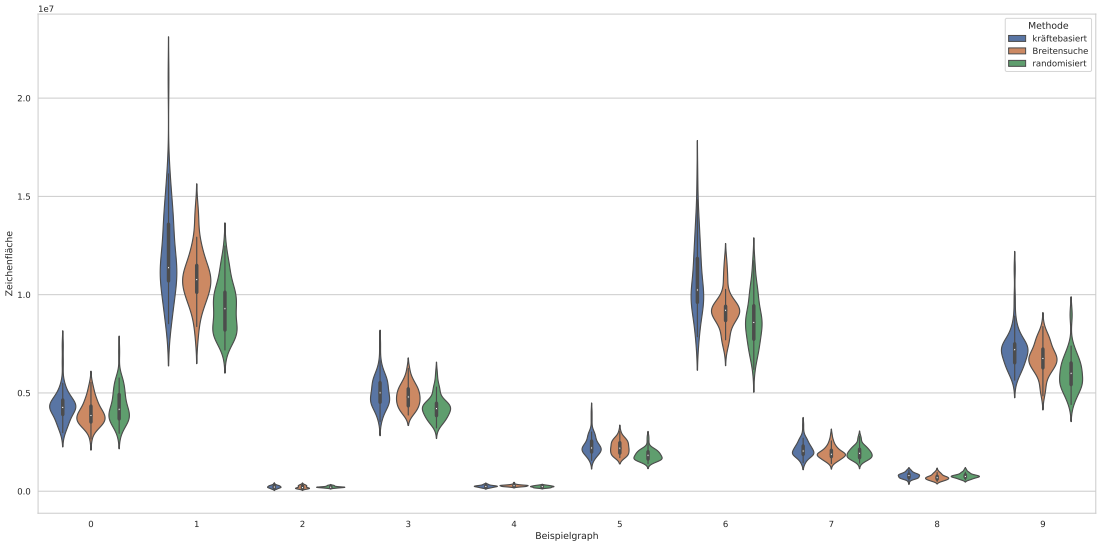


Abb. 4.2: Ergebnisse Test 1; benötigte Zeichenfläche für alle Beispielgraphen in Abhängigkeit der Methode zur Richtungszuweisung; Fläche der Bounding Box in pt^2; blau kräftebasiert; orange Breitensuche; grün zufällig

Zeichnungen recht nah beieinander liegen, jedoch hier meistens und insbesondere im Vergleich der Mittelwerte die zufällige Vorverarbeitung deutlich besser abschneidet. Nur in 8% der Fälle schneidet hier der kräftebasierte Ansatz im Mittel am besten ab und in 14% betrachtet man das Minimum.

Eine mögliche Erklärung hierzu sind die beiden letzten Algorithmusschritte. Aufgrund der größeren Anzahl an Kreuzungen werden weniger Kantenstücke gerade gezeichnet, da von sich kreuzenden Kanten immer nur eine gerade gezeichnet werden kann. Je mehr gerade gezeichnet werden, desto breiter wird die Zeichnung im gesamten, was auch ein Problem ist, das Brandes und Köpf [BK02] in Ihrer Arbeit ansprechen. Zudem steht für alle Kanten zwischen zwei Ebenen Raum derselben Höhe zur Verfügung. Genauer ist momentan die Höhe ausschließlich von der Anzahl der Lagen abhängig. Fließt hier noch der Platz zum Kantenzeichnen mit ein, wird sich vermutlich zeigen, dass all die sich kreuzenden Kanten orthogonal zu Zeichnen deutlich mehr Platz in der Höhe in Anspruch nehmen wird.

Seitenverhältnis Demnach wäre zu erwarten, dass die Zeichnungen mit kräftebasierter Richtungsbestimmung oder die mit Breitensuche am breitesten ausfallen. Das Gegenteil ist hier der Fall. In Abbildung 4.3, welche die Breite der Zeichnung im Vergleich abbildet, ist deutlich zu erkennen, dass die bei randomisierter Richtungszuweisung eine breitere Zeichnung entsteht. Interessant ist hier die Breite im Vergleich zur Höhe – zu sehen in Abbildung 4.4. Hier zeigt sich genau das gegenteilige Bild, bei Verwendung des kräftebasierten Verfahrens ist die Zeichnung meist deutlich höher bzw. werden die Knoten in mehr Lagen verteilt als bei den anderen Verfahren. Die Werte für den Ansatz mit Breitensuche

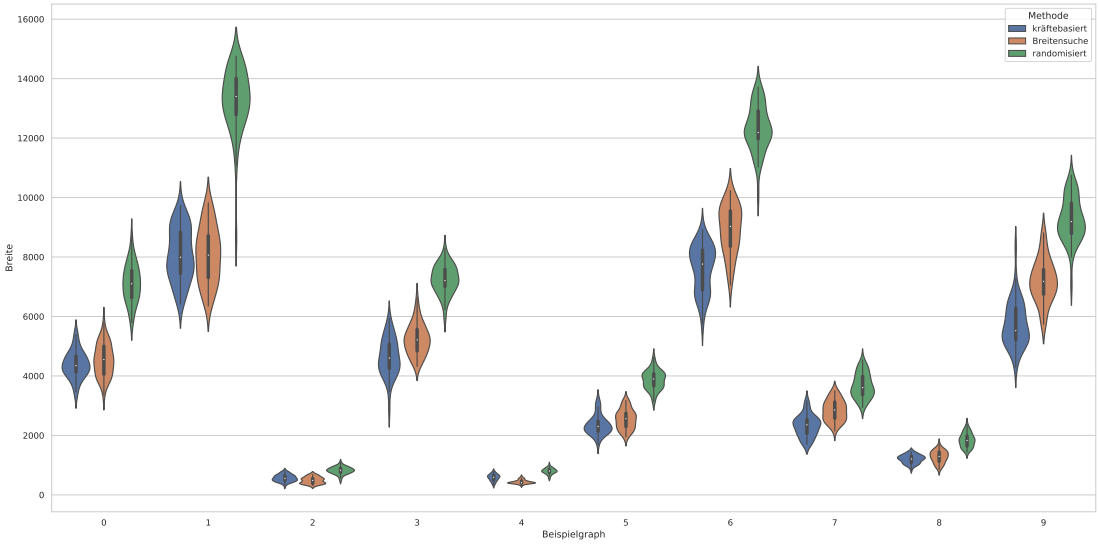


Abb. 4.3: Ergebnisse Test 1; Breite der Zeichnung für alle Beispielgraphen in Abhängigkeit der Methode zur Richtungszuweisung; blau kräftebasiert; orange Breitensuche; grün zufällig

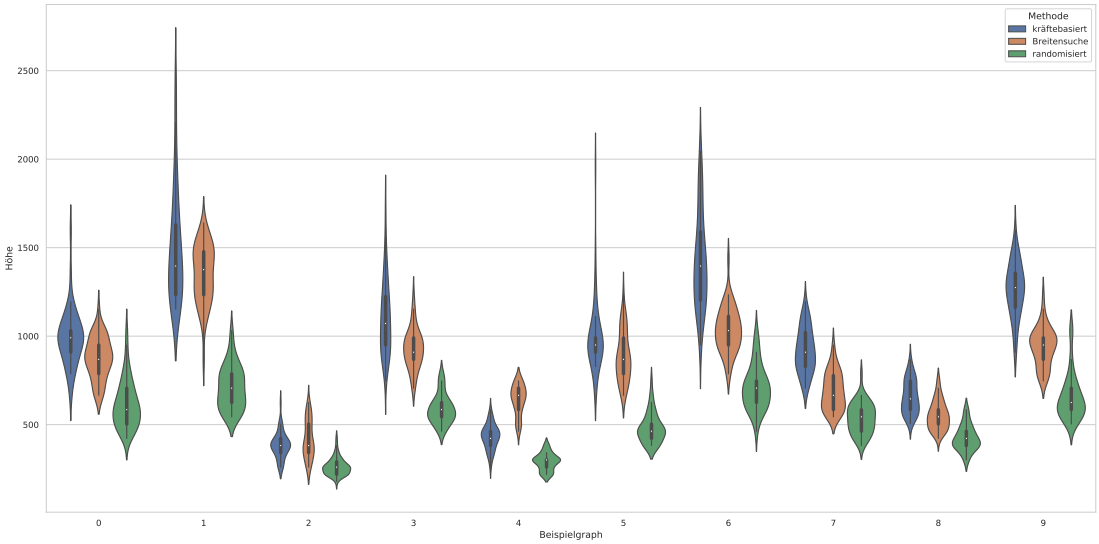


Abb. 4.4: Ergebnisse Test 1; Höhe der Zeichnung für alle Beispielgraphen in Abhängigkeit der Methode zur Richtungszuweisung; blau kräftebasiert; orange Breitensuche; grün zufällig

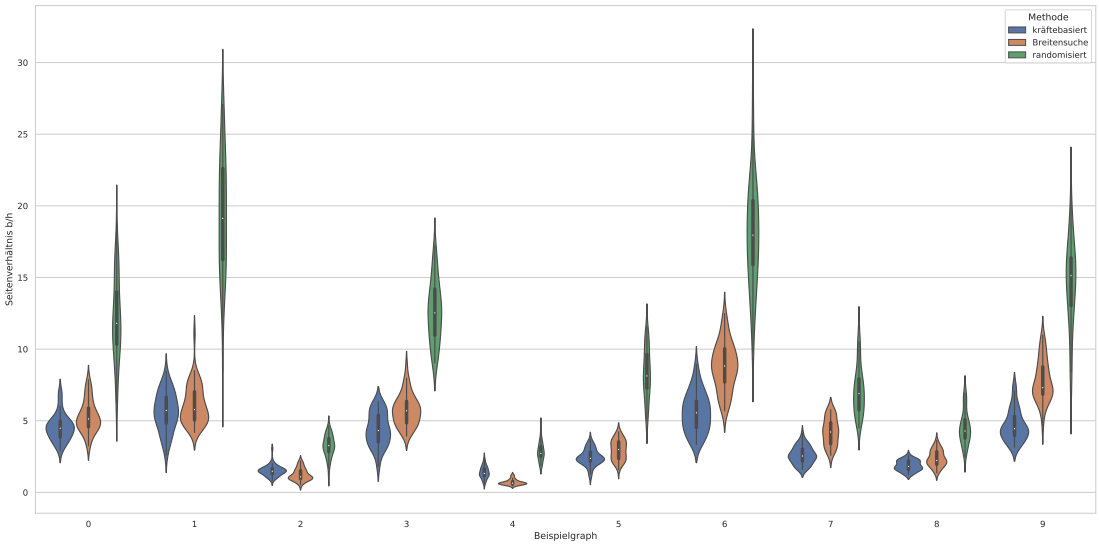


Abb. 4.5: Ergebnisse Test 1; Seitenverhältnis der Zeichnung für alle Beispielgraphen in Abhängigkeit der Methode zur Richtungszuweisung; berechnet durch Breite ÷ Höhe; blau kräftebasiert; orange Breitensuche; grün zufällig

befinden sich auch hier meist zwischen den anderen beiden, wobei sie tendenziell näher an den Ergebnissen des kräftebasierten Ansatzes sind.

Die beiden Diagramme an sich sind nur schwer zu bewerten, daher haben wir die beiden Werte nochmal in Abhängigkeit als Seitenverhältnis der Zeichnungen dargestellt – zu sehen in Abbildung 4.5, die die Breite geteilt durch die Höhe anzeigt. Hier zeigt sich, dass bei Verwendung des kräftebasierten Ansatzes die ausgeglichensten Zeichnung erreicht werden. Bei der Breitensuche etwas breitere und beim zufälligen Verfahren kommen deutlich breitere als hohe Visualisierungen heraus.

Da dieses Verhältnis für alle Graphen unabhängig von ihrer Größe gut vergleichbar ist, haben wir hier einmal alle Werte mit folgendem Ergebnis verglichen. Im Durchschnitt über alle Visualisierungen schneidet der kräftebasierte Ansatz mit einem Seitenverhältnis von 3,5 am besten ab, gefolgt von Breitensuche mit 5,2 und randomisiert mit 10,5.

Anzahl der Dummyknoten Einen ebenfalls interessanten Wert, welcher aber nur sehr indirekten Einfluss auf die Güte des Ergebnisses hat, dafür aber einen deutlich stärkeren auf die Laufzeit, ist die Anzahl der Dummyknoten. Die Ergebnisse dieses Tests sind in Abbildung 4.6 zu finden. Hier zeigt sich ein interessantes Ergebnis: bei Verwendung der Breitensuche ist die Zahl der Dummyknoten immer in einem engen Bereich und variiert, verglichen mit den anderen, nicht sehr stark. Bei der zufälligen Richtungsbestimmung finden sich stärkere Variationen und meist deutlich mehr Dummyknoten und beim kräftebasierten Ansatz gibt es extreme Ausschläge in beide Richtungen.

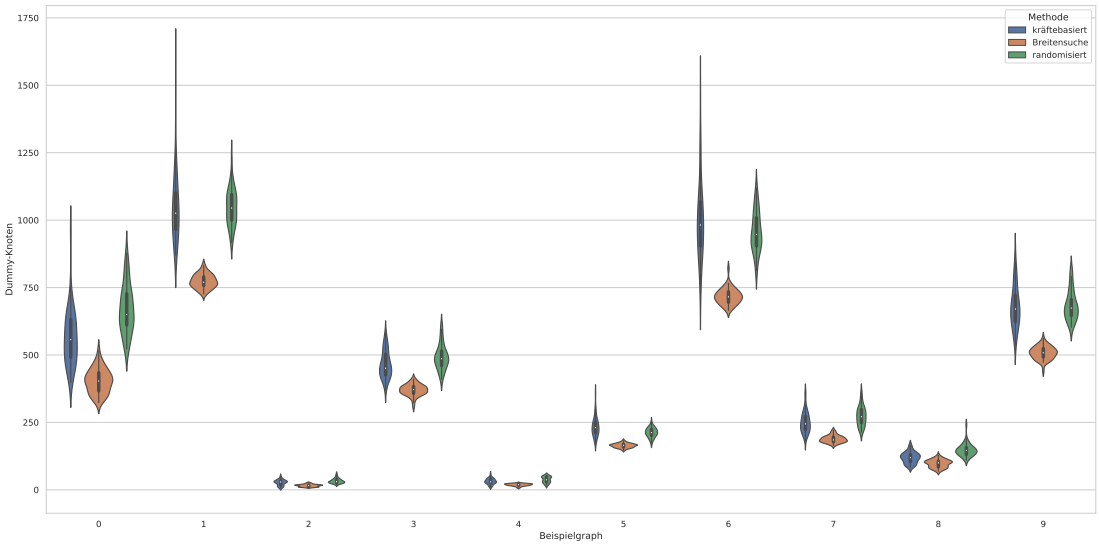


Abb. 4.6: Ergebnisse Test 1; Anzahl der benötigten Dummyknoten für alle Beispielgraphen in Abhängigkeit der Methode zur Richtungszuweisung; blau kräftebasiert; orange Breitensuche; grün zufällig

In 86% der Fälle wurde bei der Verwendung der Breitensuche die geringste Menge an Dummyknoten gezählt. Im Durchschnitt war die Breitensuche sogar in $94,5\%$ der Ansatz mit den wenigsten Dummyknoten.

Es werden immer dann besonders viele Dummyknoten angelegt, wenn Zwischenebenen notwendig sind. Auch bei sehr langen Kanten werden diese benötigt um Ebenen zu überbrücken. Das legt die Vermutung nahe, dass bei Verwendung der Breitensuche weniger Zwischenebenen vonnöten sind. Die Anzahl der Ebenen insgesamt spiegelt sich direkt in der Höhe der Zeichnung wieder; wir haben jedoch die Anzahl der Zwischenebenen nicht gesondert gemessen und können daher hierzu nichts eindeutiges sagen.

Es lässt sich hier abschließend sagen, dass bei Verwendung des kräftebasierten Ansatzes zur Vorverarbeitung bei gleicher Anzahl an Durchläufen mit höherer Wahrscheinlichkeit eine Zeichnung mit wenigen Kreuzungen und ausgeglichener Größe der Zeichenfläche zu erreichen ist. Breitensuche zur Richtungsbestimmung einzusetzen ist eine einfach umsetzbare Alternative mit meist ebenfalls guten Ergebnissen. Das Verfahren, welches für die Richtungszuweisung gewählt wird, hat große Auswirkungen auf die endgültige Zeichnung und es ist somit zu empfehlen hier ein geeignetes Verfahren zu wählen. Um bei einer rein zufälligen Richtungsbestimmung ebenso gute Ergebnisse zu erzielen wie mit angepassten Verfahren sind deutlich mehr Iterationen notwendig und es ist daher nicht zu empfehlen.

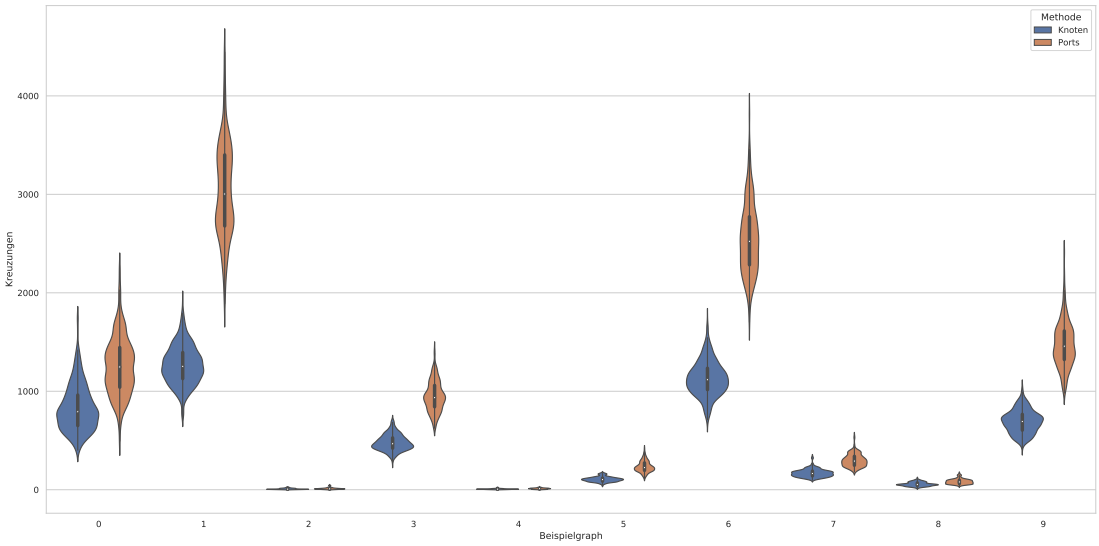


Abb. 4.7: Ergebnisse Test 2; Anzahl der Kreuzungen für alle Beispielgraphen in Abhängigkeit der Methode zur Barycenterberechnung; blau Knotenkoordinaten; orange Portkoordinaten

4.2 Test 2: Kreuzungsreduzierung

Für diesen Schritt wurden zwei verschiedene Möglichkeiten zur Berechnung der Barycenter umgesetzt. Die erste ist die reine Verwendung der Knotenpositionen mit Ausnahme der Steckverbindungen. Da bei Steckverbindungen die Reihenfolge der Ports auf der einen, die Reihenfolge derer auf der anderen Seite beeinflusst und damit auch die Anzahl der Kreuzungen, wurden hier die Portkoordinaten nach der *node-relative*-Methode (Abbildung 2.1) verwendet. Die zweite ist die Verwendung der Portkoordinaten nach der *layer-total*-Methode. Bei diesem Test wurden für jeden Graphen die Algorithmusschritte 0 bis 3 insgesamt 200 Mal ausgeführt – wobei immer der kräftebasierte Ansatz zur Richtungsbestimmung genommen wurde. Anschließend wurde eine zufällige Knotenreihenfolge festgelegt und mit jedem der beiden zu testenden Verfahren 5 Mal ein Durchlauf der Kreuzungsreduzierung dafür ausgeführt. Da hier immer beide Verfahren gleich oft mit derselben Startkonfiguration ausgeführt wurden, sind die Ergebnisse besser vergleichbar. Nur die Startreihenfolge der Ports war noch komplett dem Zufall überlassen.

Hier war unsere Erwartung, dass sich die *node-relative*-Methode als besser erweisen würde, da von vornherein die durch Portgruppen gegebenen Einschränkungen bei allen Knoten und nicht nur bei Steckverbindungen berücksichtigt werden. Überraschenderweise stellt sich hier die zweite Methode als deutlich besser heraus. Im direkten Vergleich sind die Ergebnisse in Abbildung 4.7 zu sehen. Hier entspricht orange der *layer-total*-Methode und blau der Verwendung der Knotenkoordinaten. Da die blauen Graphen hier deutlich bessere Ergebnisse zeigen, ist eindeutig, dass diese Methode für unsere Beispielgraphen besser geeignet ist.

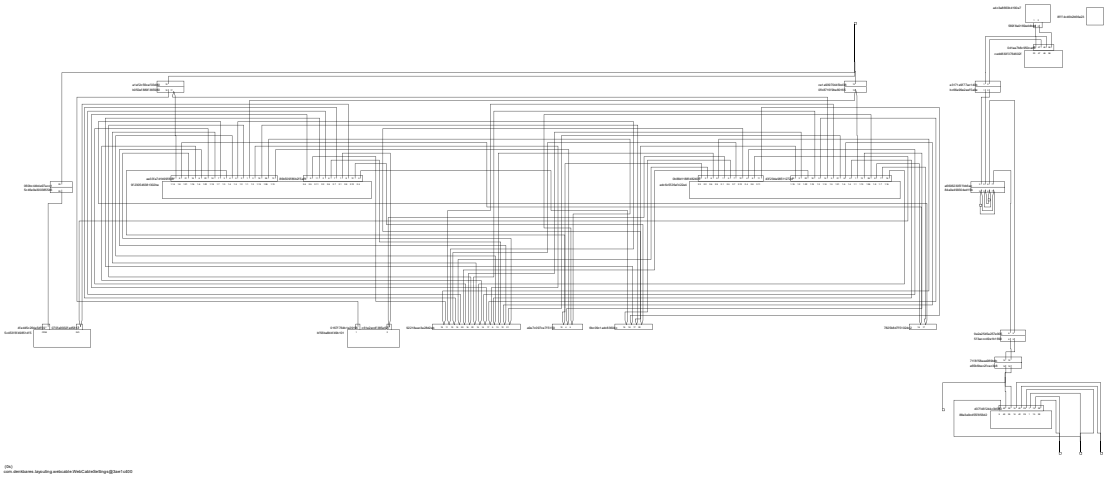


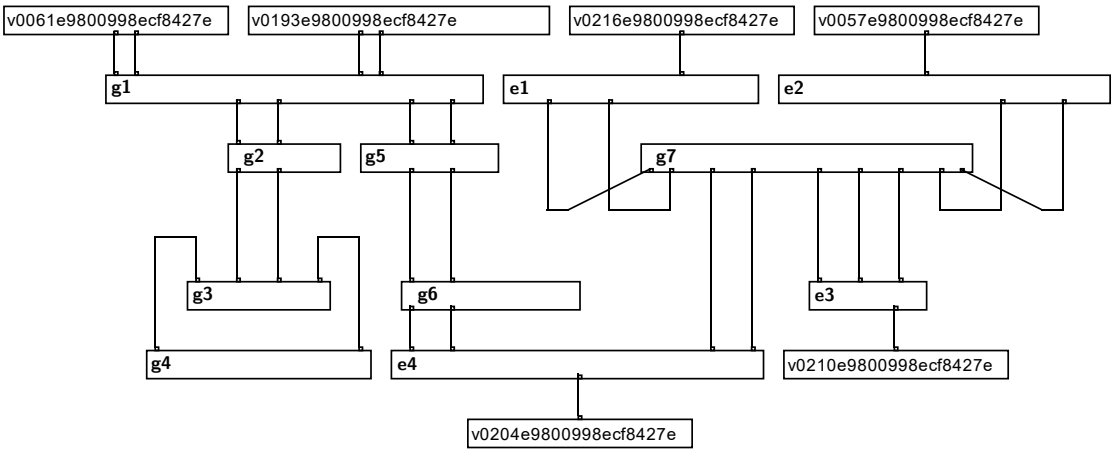
Abb. 4.8: Visualisierung eines Beispielgraphen mit „KLay Layered“[SSvH14]

4.3 Test 3: Visuelle Analyse

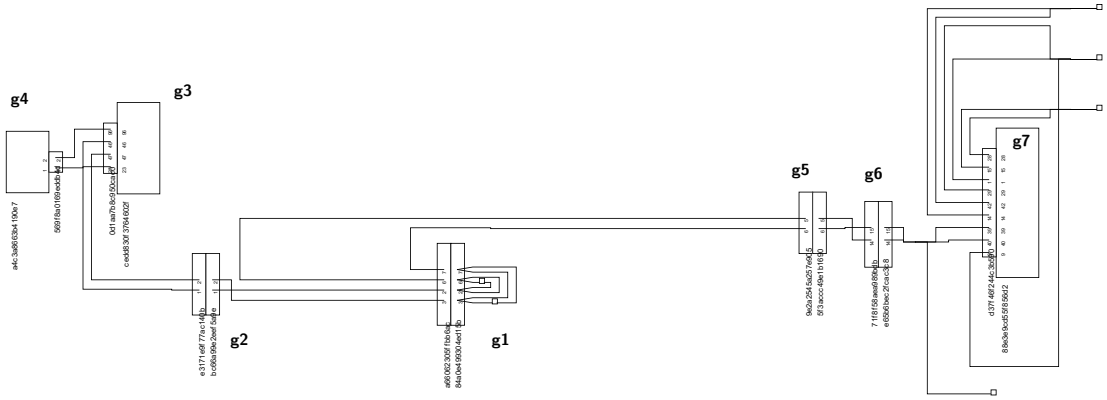
Zunächst soll eine Zeichnung mit dem bisher verwendeten Verfahren (mit „KLay Layered“) mit einer unserer Zeichnungen verglichen werden. Anschließend gehen wir nochmal näher auf momentane Schwächen unserer Implementierung ein und wie die Zeichnung noch verbessert werden kann.

Visueller Vergleich der Verfahren Abbildung 4.8 zeigt einen Graphen nach dem alten Verfahren und Abbildung 4.9a die neue Visualisierung. Hierbei fällt als erstes auf, dass die alte Visualisierung deutlich größer ist. Das liegt daran, dass der Graph nicht zusammenhängend ist und unsere Implementierung nur die größte Zusammenhangkomponente zeigt, gemessen an der Anzahl der Knoten. Daher ist in Abbildung 4.9b der Teil des Graphen abgebildet, der dem in Abbildung 4.9a entspricht. Natürlich sind diese dahingehend nicht direkt vergleichbar, dass der ausgeschnittene Teil recht lang gezogen ist, da er am rechten Rand der eigentlichen Zeichnung untergebracht war. Auch fehlt bei unserer Implementierung noch die Kantenführung sowie die Auflösung der Ersatzknoten für Knoten mit `PortPairing` sowie derer für Multikanten. Zur besseren Übersicht sind daher Knoten, welche Gruppen repräsentieren, mit gx auf beiden Zeichnungen durchnummeriert, wobei x eine Nummer ist. Knoten, welche zu einer Multikante aufgelöst werden müssen, sind mit ex beschriftet.

Vernachlässigen wir aber das Seitenverhältnis und die Auflösung der Knoten, sind dennoch einige, deutliche Unterschiede zwischen Abbildung 4.9b und Abbildung 4.9a erkennbar. Zum einen fällt auf, dass unsere Zeichnung weniger Kreuzungen hat, nämlich zwei, wohingegen die alte Zeichnung fünf Kreuzungen beinhaltet. Auch sind Beschriftungen der Knoten außerhalb dieser gezeichnet, was dazu führt, dass diese, wie bei $g4$ zu beobachten, schlechter lesbar sind, da sie sich zum Teil mit anderen Elementen der Zeichnung überschneiden. Durch die Einführung der Ersatzknoten für Multikanten ist



(a) Visualisierung mit dem, in dieser Arbeit beschriebenen Algorithmus



(b) Visualisierung mit „KLay Layered“[SSvH14]

Abb. 4.9: Zeichnungen eines Beispielgraphen im Vergleich; oben Ergebnis des hier beschriebenen Ansatzes; unten Visualisierung mit dem alten Verfahren mit „KLay Layered“

auch einfacher zu erkennen, wo sich Kanten kreuzen und wo Kanten eine Multikante bilden, da Multikanten immer auf einer Knotenebene zusammengeführt werden.

Bei einigen Kanten in Abbildung 4.9b finden sich auch unnötige Knicke, wie bei der Kante von $g1$ nach $g5$. Solche kleinen Knicke, wo die Kante um wenige Pixel versetzt weiter führt, treten bei den neuen Zeichnungen nicht auf. Daher ist auch zwischen $g3$ und $g4$ nicht eindeutig ersichtlich, welcher Port mit welchem verbunden ist und ob es sich um eine Multikante oder eine Kreuzung handelt. Dieses Problem, dass sich Teile von Kanten überschneiden, beruht in erster Linie auf der Kantenführung. Es tritt bei uns nicht auf, da ausgeschlossen ist, dass sich vertikale Kantenteile überschneiden und die restlichen Teile Direktverbindungen auf einer Ebene sind. Das kann ohne eine geeignete Kantenführung, die in unserer Implementierung noch fehlt, sehr unübersichtlich werden. Daher wird hierauf auch nicht weiter eingegangen.

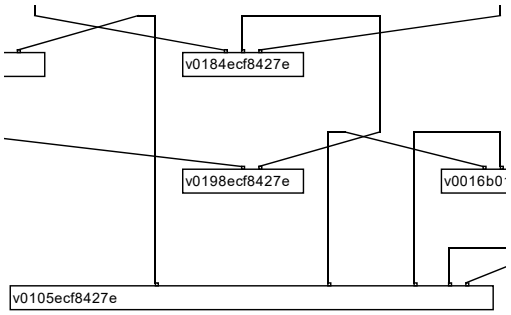
Verbesserungsmöglichkeiten Abgesehen von den eben beschriebenen, noch fehlenden Elementen des Algorithmus – der Kantenführung sowie der Auflösung der Ersatzknoten – gibt es ein paar Schwächen. Beispiele hierzu sind in Abbildung 4.10 abgebildet. Im Folgenden werden diese Schwächen aufgezeigt und diskutiert, wie die Implementierung verbessert werden kann.

Einige Probleme treten bei Knoten auf, welche keine Verbindung in eine Richtung (oben oder unten) haben und auf einer der mittleren Lagen platziert sind. Beispiele hierzu finden sich in Abbildung 4.10a und 4.10b.

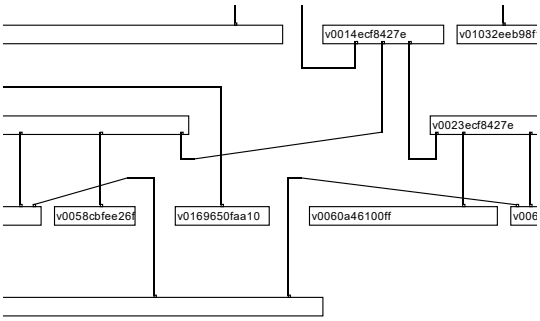
Oben links in Abbildung 4.10a findet sich eine offensichtlich unnötige Kreuzung zweier Kanten, die schon bei einer geschickten Implementierung der Kantenführung abgefangen und aufgelöst werden könnte. Derselbe Fall tritt rechts darunter nochmals auf. Macht man sich jedoch klar, dass es sich hier um einen Dummyknoten und keine Kante handelt, fällt auf, dass die Kreuzung eigentlich im Schritt der Kreuzungsreduzierung aufgelöst hätte werden sollen. Beim Fall oben links hätten eigentlich der Dummyknoten zur Richtungsänderung oben rechts mit dem Dummyknoten für die Ebenen-überspannende Kante oben links getauscht werden sollen. Dann wäre diese Kreuzung nicht entstanden. Für die Kreuzung in der Mitte gilt dasselbe, auch hier hätte durch einen Tausch der Dummyknoten die Kreuzung vermieden werden können.

Zusätzlich fällt auf, dass diese Fälle nur bei Dummyknoten zu finden sind, die beide Verbindungen nach unten haben. In die andere Richtung tritt der Fall nicht auf. Das liegt daran, dass die Kreuzungsreduzierung die Ebenen immer von unten nach oben nach unten durchläuft. Sie endet also immer unten. Das Problem ist demnach darauf zurückführbar, wie Knoten behandelt werden, die keine Verbindungen in die Ebene haben, von der der Algorithmus kommt. Kreuzungen, wie in Abbildung 4.10a oben links, können nur auftreten, wenn der Algorithmus von oben kommt und dabei den Dummyknoten ohne Verbindung rechts von dem anordnet, der die Ebenen-überspannende Kante repräsentiert.

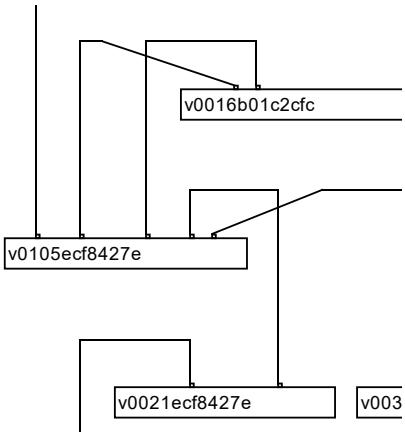
Eine mögliche Alternative wäre, diese Knoten an festen Positionen zu belassen. Das heißt, sind von fünf Knoten der dritte und vierte ein Dummyknoten, bleiben diese an



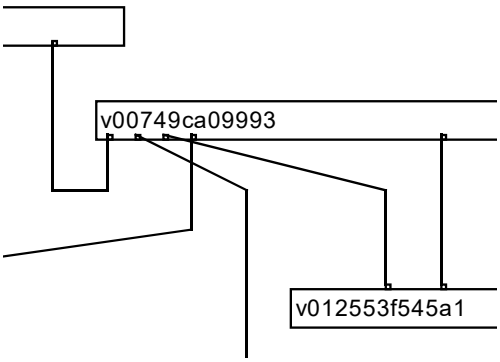
(a) Problem bei Anordnung der Knoten ohne Verbindung nach oben/unten



(b) Problem bei Anordnung der Knoten ohne Verbindung nach oben/unten



(c) Problem bei mehreren Dummyknoten nach Abbildung 3.2 nebeneinander



(d) Problem beim Sortieren der Ports

Abb. 4.10: Ausschnitte aus einem Graphen, visualisiert mit unserer Implementierung; oben Probleme resultierend aus dem aktuellen Umgang mit Knoten ohne Verbindungen in eine Richtung im Schritt Kreuzungsreduzierung; unten links Probleme bei aktueller Implementierung der Dummyknoten; unten rechts Fehler beim Sortieren der Ports

Position 3 und 4. Die anderen drei Knoten werden nach ihren Barycentern aufsteigend auf die Positionen 1, 2 und 5 verteilt.

Auch Abbildung 4.10b zeigt Probleme, die aus demselben Grund entstanden sind. Hier könnte man meinen, die Kreuzungen ließen sich vermeiden, wenn die Knoten eine Ebene höher über die jeweiligen Kanten verschoben werden. Aber auch hier sollten diese Kreuzungen gar nicht erst während der Kreuzungsreduzierung entstehen. Der Unterschied ist nur, dass es in diesem Fall nur zu einer und nicht mehreren zusätzlichen Kreuzungen kommt. Auch die Kreuzung der Kante des Knotens $v0169650faa10$ könnte vermieden werden, indem dieser weiter links angeordnet wird - der entsprechende Bereich ist jedoch in diesem Ausschnitt nicht zu sehen.

Ein Problem mit Dummyknoten zur Richtungsänderung, welches sich nicht allein im Schritt der Kreuzungsreduzierung lösen lässt, ist in Abbildung 4.10c abgebildet. Hier gibt es gleich zwei Mal nebeneinander den Fall, dass zwei Kanten sich kreuzen, die parallel verlaufen könnten. Das passiert, da sie jeweils über einen eigenen Dummyknoten gleitet werden und sich diese Dummyknoten auf derselben Ebene befinden. Der Algorithmus hat bei dieser Art der Repräsentation keine andere Wahl, als eine Kreuzung einzufügen, da er nicht weiß, dass es sich bei diesen Dummyknoten theoretisch um Kantenstücke handelt.

Bei einer Vertauschung der Ports an einem der beiden Endknoten, könnte in einem Nachbearbeitungsschritt die Situation in eine kreuzungsfreie Kantenführung aufgelöst werden. Das hilft aber nur bei zwei parallelen Kanten. Eine bessere Alternative wäre, alle Kanten, die auf der einen Seite des Knotens vorbeigeführt werden, über einen gemeinsamen Dummyknoten zu leiten, der mehrere Kanten repräsentiert. Dabei kann bei der Portsortierung darauf geachtet werden, dass er sich, sofern die Portgruppen es zulassen, kreuzungsfrei auflösen lässt. Hierbei ist es jedoch nicht möglich, von vornherein zu wissen, welche Kanten auf welcher Seite des Knotens vorbeigeführt werden. Daher könnte man einfach alle Kanten über einen Dummyknoten leiten und somit auf einer Seite des Knotens vorbeizwingen. Das Hilft auch der Übersichtlichkeit. Jedoch nimmt man damit auch dem Algorithmus die Möglichkeit, Kanten an beiden Seite vorbeizuführen und hierdurch gegebenenfalls Kreuzungen zu Vermeiden. Das kann insbesondere bei besonders ungeschickter Lage von Portgruppen zu Problemen führen. Es ist also nicht einfach, hier einen Weg zu finden, der ohne die Möglichkeiten des Algorithmus bei der Kreuzungsreduzierung zu beschneiden, alle Möglichkeiten der Kantenführung offen hält.

Das letzte Problem, welches beobachtet wurde, zu sehen in Abbildung 4.10d, ist auf relativ einfache Art und Weise lösbar. Bei der Kreuzungsreduzierung wird momentan im Fall, dass Knotenpositionen für die Barycenterberechnung genutzt wurden, die Portsortierung erst im Anschluss ausgeführt. Hierzu wird nur ein Mal über den Graphen iteriert und die Ports sortiert. Das funktioniert aber nur, wenn alle Ports frei positionierbar sind. Da aber Portgruppen auftreten, muss auch hier der Schritt erweitert werden. Es sollte also zunächst die Reihenfolge der Ports und Portgruppen festgelegt werden, welche direkte Kinder eines Knotens sind. Danach die, welche einer der schon positionierten Portgruppen angehören und so weiter, bis alle eine feste Position haben. Da der Fall nur sehr selten auftritt, da in unseren Beispielgraphen meist keine verschachtelten Portgruppen auftreten, wurde dieser Fehler erst nach Ausführung der Tests entdeckt.

5 Zusammenfassung und Ausblick

Durch die zeitliche Begrenzung der Arbeitszeit war es nicht möglich alle Ideen umzusetzen. Daher wird hier ein Überblick gegeben, was noch fehlt, um den Algorithmus zu vervollständigen. Im Anschluss werden noch einige Ideen aufgezeigt, die alternativ noch getestet werden können.

Die Ergebnisse dieser Arbeit wurden nachträglich um eine neue Art von Dummyknoten, eine neue Möglichkeit bei der Kreuzungsreduzierung und eine orthogonale Kantenführung erweitert und beim „28th International Symposium on Graph Drawing and Network Visualization“ unter dem Titel „Layered Drawing of Undirected Graphs with Generalized Port Constraints“ [WZBW20] eingereicht und angenommen. Die dabei vorgenommenen Änderungen am Code, welche nicht mehr im Rahmen dieser Arbeit durchgeführt wurden, sind in Abschnitt A.2 aufgeführt.

Bei der Implementierung des Algorithmus von Brandes und Köpf [BK02] im Rahmen dieser Arbeit sind wir auf Fehler in deren Algorithmus gestoßen, welche wir lösen konnten. Diese Korrektur wurde mit dem Titel „Erratum: Fast and Simple Horizontal Coordinate Assignment“ [BWZ20] ebenfalls dort eingereicht, allerdings nicht angenommen. Der Artikel ist auf arXiv verfügbar.

Zusammenfassung In dieser Arbeit wurde eine Erweiterung des Algorithmus von Sugiyama et al. zur Visualisierung hierarchischer Graphen vorgestellt. Sie baut direkt auf der Arbeit von Schulze et al. auf, die besagten Algorithmus bereits um die Verwendung von Ports erweitert haben [SSvH14]. Wir haben diesen Ansatz um die Unterstützung von Portgruppen und Portpaaren erweitert. Außerdem wurden Möglichkeiten diskutiert, wie bei ungerichteten Graphen Kantenrichtungen bestimmt werden können, um möglichst übersichtliche Visualisierungen erzeugen zu können.

Hierzu wurde der Algorithmus auf insgesamt 7 Schritte erweitert: Vorverarbeitung, Richtungszuweisung, Lagenzuordnung, Dummyknotenerstellung, Kreuzungsreduzierung, Knotenpositionierung und Kantenführung, die in Abschnitt 3.3 im Detail vorgestellt werden. Im Anschluss wurden Zeichnungen erstellt und anhand der Parameter Anzahl der Kreuzungen, Anzahl der Dummyknoten, Größe der Zeichenfläche und Seitenverhältnis der Zeichnung analysiert. Hierbei wurden verschiedene Möglichkeiten zur Richtungszuweisung sowie zur Kreuzungsreduzierung verglichen. Zuletzt wurde ein visueller Vergleich von erstellten Zeichnungen unter Verwendung unseres Ansatzes sowie dem von Schulze et al. durchgeführt.

Ausblick Der Implementierung fehlt noch der letzte Schritt: die orthogonale Kantenführung. Er ist insbesondere bei großen Graphen für eine übersichtliche Darstellung unverzichtbar. Des Weiteren können noch einige Dinge der endgültigen Zeichnung verbessert

werden. Das umfasst eine Anpassung der Knotenbreiten nach der Knotenpositionierung. Hier können einige Knoten schmaler gezeichnet werden. Zum anderen können Dummyknotenebenen komplett aufgelöst werden und dabei die Zeichnung zusammengeschoben werden, wodurch auch Platz in der Höhe eingespart werden kann.

In den Beispielgraphen sind Knotengruppen nur in Form von sich berührenden Knoten vorgekommen. Die Datenstruktur sieht aber auch die Möglichkeit vor, dass Knotengruppen so eingesetzt werden, dass bestimmte Knoten nahe beieinander gezeichnet werden, ohne sich zu berühren. In diesem Fall wird momentan nur ein Ersatzknoten erstellt, welcher nicht aufgelöst wird. Im Allgemeinen fehlt noch die Auflösung aller Ersatzknoten, diese ist im Fall solcher Knotengruppen nicht trivial.

Auch fehlt noch eine Unterstützung von nicht zusammenhängenden Graphen. Momentan wird nur die größte Zusammenhangkomponente dargestellt, alle anderen werden ignoriert. Hier müssen entweder alle gezeichnet und und dann die Zeichnungen geschickt nebeneinander platziert werden, oder eine gemeinsame Zeichnung für alle in einem Durchlauf erstellt werden. Bei Umsetzung des zweiten Vorschlags müssen die einzelnen Algorithmusschritte um eine solche Unterstützung erweitert werden.

Wie in Abschnitt 4.3 schon ausführlich diskutiert, sollte der Umgang mit Knoten, die in eine Richtung nicht verbunden sind, überarbeitet werden. Eine ausführliche Beschreibung, wie damit umgegangen werden kann, findet sich dort unter dem Stichpunkt „Verbesserungsmöglichkeiten“.

Zuletzt fehlt noch, ebenfalls in Abschnitt 4.3 beschrieben, eine Sortierung der Ports bei der Kreuzungsreduzierung in mehreren Schritten. Dabei sollten nach Feststehen der Knotenpositionen die Positionen ihrer `PortCompositions` entsprechend der Barycenter-Heuristik angepasst werden, bis diese sich nicht mehr ändern. Danach, wenn es sich um `PortGroups` gehandelt hat, die Positionen deren `PortCompositions` bis irgendwann alle `PortGroups` und Ports feste Positionen haben.

Alternativen Zunächst wollen wir einige weitere Möglichkeiten bei der Berechnung der kräftebasierten Ansicht im Schritt der Richtungszuweisung betrachten. Hier wurde aus Zeitgründen nur ein vorimplementiertes Verfahren getestet. Es könnte aber auch ein eigener kräftebasierter Ansatz entwickelt werden, der auch einige Besonderheiten unserer Graphen berücksichtigt. Wenn das besonders gut funktioniert, kann eventuell schon diese Ansicht als Visualisierung genommen werden.

Hierbei könnten Knoten bereits so angelegt werden, dass sie nicht nur ein Punkt sind, sondern selbst schon eine rechteckige Fläche einnehmen, welche durch abstoßende Kräfte frei gehalten wird. Auch kann bei Knoten mit `PortPairings` versucht werden, die Anschlusspunkte der Kanten auf eine bestimmte Seite des Knotens zu legen. Sollte das funktionieren, könnte es die Anzahl der Dummyknoten zur Richtungsänderung reduzieren. Um noch weiterzugehen, kann versucht werden, auch hier bereits Ports und Portgruppen einzubauen, welche dann Kräfte verursachen, die schon hier versuchen Kreuzungen zu vermeiden. Diese müssten dann aber auch während der Berechnung umsortiert werden, ähnlich wie bei der Kreuzungsreduzierung. Das kann auch zu deutlich längeren Laufzeiten führen.

Eine alternative Möglichkeit zur Verarbeitung des Ergebnisses des kräftebasierten Algorithmus bei der Richtungszuweisung ist, einfach die untersten m Knoten (ein geeigneter Wert für m muss dazu gefunden werden) der ersten Lage zuzuweisen. Die nächsten m Knoten dann der zweiten Lage etc., bis alle Knoten zugeordnet sind. Hierbei könnte auch berücksichtigt werden, dass Knoten mit mehr Ports oder längerer Beschriftung mehr Platz beanspruchen. Auch kann bereits die Anzahl der Dummyknoten für jede dieser Lagen ermittelt werden. Der Vorteil wäre eine gleichmäßige Breite über den ganzen Graphen. Das funktioniert jedoch nur, wenn Kanten zwischen Knoten derselben Lage zugelassen werden. Solche Kanten könnten z.B. über Dummyknoten in einer Zwischenebene geführt werden.

Es wäre aber auf alle Fälle interessant zu testen, welchen Einfluss unterschiedliche kräftebasierte Algorithmen auf das Ergebnis haben. Hierbei sollten auch die Ergebnisse der kräftebasierten Algorithmen vor Ausführung des restlichen Algorithmus visuell verglichen werden.

Für die Behandlung der Portgruppen könnte alternativ zur Richtungsänderung über Dummyknoten auch versucht werden, das Zuordnungsproblem, wie in Abbildung 3.1 zu lösen. Hierdurch können aber auch neue Knicke in Kanten entstehen, wenn diese im Kreis um den Knoten herum geführt werden müssen. Eine Behandlung dieser Probleme ohne Dummyknoten bietet die Möglichkeit der Einsparung von Zusatzebenen und damit einer Beschleunigung des Algorithmus.

Zuletzt wollen wir noch alternative Möglichkeiten im Schritt der Kreuzungsreduzierung vorstellen. Zur Berechnung der Barycenter nutzen wir bisher entweder die Knotenpositionen in Kombination mit der Portreihenfolge bei Knoten mit `PortPairings` nach der *node-relative*-Methode von Schulze et al. oder nur die Portreihenfolge nach der *layer-total*-Methode. Hierbei wäre noch der Vergleich mit einer Berechnung rein nach den Knotenpositionen, ohne Berücksichtigung der Ports interessant, um zu prüfen, ob wir durch die Sonderbehandlung von Knoten mit `PortPairings` eine Verbesserung erreichen. Ebenfalls kann die Berechnung komplett nach der *node-relative*-Methode getestet werden. Als drittes könnte eine geschachtelte Berechnung nach der *node-relative*-Methode getestet werden. Das bedeutet, dass der Wertebereich eines Knotens gleichmäßig nach der *node-relative*-Methode auf seine `PortCompositions` aufgeteilt wird. Wenn eine dieser `PortCompositions` eine `PortGroup` ist, wird ihr Bereich nach demselben Verfahren auf alle zugehörigen `PortCompositions` aufgeteilt.

Literaturverzeichnis

[BK02] Ulrik Brandes und Boris Köpf: Fast and simple horizontal coordinate assignment. In: Petra Mutzel, Michael Jünger und Sebastian Leipert (Herausgeber): *Graph Drawing*, Band 2265 der Reihe *Lecture Notes in Computer Science*, Seiten 31–44. Springer Berlin Heidelberg, 2002. https://doi.org/10.1007/3-540-45848-4_3.

[BNT86] Carlo Batini, Enrico Nardelli und Roberto Tamassia: A layout algorithm for data flow diagrams. *IEEE Transactions on Software Engineering*, SE-12(4):538–546, 1986. https://doi.org/10.1109/TSE.1986.6312901.

[BWZ20] Ulrik Brandes, Julian Walter und Johannes Zink: Erratum: Fast and Simple Horizontal Coordinate Assignment. *arXiv preprint arXiv:2008.01252*, 2020. https://arxiv.org/pdf/2008.01252.pdf.

[Ead84] Peter Eades: A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984. http://www.cs.usyd.edu.au/~peter/old_spring_paper.pdf.

[EGB06] Thomas Eschbach, Wolfgang Guenther und Bernd Becker: Orthogonal Hypergraph Drawing for Improved Visibility. *Journal of Graph Algorithms and Applications*, 10(2):141–157, 2006. https://doi.org/10.7155/jgaa.00122.

[elk20] ELK Layered, 2020. https://www.eclipse.org/elk/reference/algorithms/org-eclipse-elk-layered.html, besucht: 2020-09-22.

[ELS93] Peter Eades, Xuemin Lin und William F. Smyth: A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319–323, 1993. https://doi.org/10.1016/0020-0190(93)90079-O.

[FR91] Thomas M.J. Fruchterman und Edward M. Reingold: Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991. https://doi.org/10.1002/spe.4380211102.

[GKNV93] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North und Kiem-Phong Vo: A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993. https://doi.org/10.1109/32.221135.

[jun20] JUNG-Library, 2020. http://jung.sourceforge.net/, besucht: 2020-09-22.

[kie20] KIELER Project, 2020. https://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/KLay+Layered, besucht: 2020-09-22.

[KK89] Tomihisa Kamada und Satoru Kawai: An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989. https://doi.org/10.1016/0020-0190(89)90102-6.

[pra20] Praline Project, 2020. https://github.com/j-zink-wuerzburg/pseudo-praline-plan-generation, besucht: 2020-09-22.

[PST97] Achilleas Papakostas, Janet M. Six und Ioannis G. Tollis: Experimental and theoretical results in interactive orthogonal graph drawing. In: Stephen North (Herausgeber): *Graph Drawing*, Band 1190 der Reihe *Lecture Notes in Computer Science*, Seiten 371–386. Springer Berlin Heidelberg, 1997. https://doi.org/10.1007/3-540-62495-3_61.

[San94] Georg Sander: Graph layout through the VCG tool. In: Roberto Tamassia und Ioannis G. Tollis (Herausgeber): *Graph Drawing*, Band 894 der Reihe *Lecture Notes in Computer Science*, Seiten 194–205. Springer Berlin Heidelberg, 1994. https://doi.org/10.1007/3-540-58950-3_371.

[San96] Georg Sander: A fast heuristic for hierarchical Manhattan layout. In: Franz J. Brandenburg (Herausgeber): *Graph Drawing*, Band 1027 der Reihe *Lecture Notes in Computer Science*, Seiten 447–458. Springer Berlin Heidelberg, 1996. https://doi.org/10.1007/BFb0021828.

[San03] Georg Sander: Layout of directed hypergraphs with orthogonal hyperedges. In: Giuseppe Liotta (Herausgeber): *Graph Drawing*, Band 2912 der Reihe *Lecture Notes in Computer Science*, Seiten 381–386. Springer Berlin Heidelberg, 2003. https://doi.org/10.1007/978-3-540-24595-7_35.

[SSvH14] Christoph Daniel Schulze, Miro Spönemann und Reinhard von Hanxleden: Drawing layered graphs with port constraints. *Journal of Visual Languages & Computing*, 25(2):89–106, 2014. https://doi.org/10.1016/j.jvlc.2013.11.005.

[STT81] Kozo Sugiyama, Shojiro Tagawa und Mitsuhiko Toda: Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981. https://doi.org/10.1109/TSMC.1981.4308636.

[TDB88] Roberto Tamassia, Giuseppe Di Battista und Carlo Batini: Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):61–79, 1988. https://doi.org/10.1109/21.87055.

[Tun94] Daniel Tunkelang: A practical approach to drawing undirected graphs. Technischer Bericht CMU-CS-94-161, School of Computer Science, Carnegie Mel-

lon University, 1994. http://reports-archive.adm.cs.cmu.edu/anon/1994/CMU-CS-94-161.ps.

[War77] John N. Warfield: Crossing theory and hierarchy mapping. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(7):505–523, 1977. https://doi.org/10.1109/TSMC.1977.4309760.

[WZBW20] Julian Walter, Johannes Zink, Joachim Baumeister und Alexander Wolff: Layered Drawing of Undirected Graphs with Generalized Port Constraints. In: *International Symposium on Graph Drawing and Network Visualization*. Springer, 2020. https://arxiv.org/pdf/2008.10583.pdf.

A Anhang

A.1 Ergebnisse

In den folgenden Abbildungen und Tabellen sind alle Ergebnisse der Tests aus Kapitel 4 in folgender Reihenfolge dargestellt. Zunächst die Ergebnisse des ersten Tests, bei welchem die unterschiedlichen Methoden zur Vorverarbeitung getestet wurden; in der Reihenfolge Anzahl an Dummyknoten, Anzahl an Kreuzungen, Größe der Zeichenfläche, Seitenverhältnis, Höhe der Zeichnung, Breite der Zeichnung. Anschließend die Ergebnisse des zweiten Tests; die Anzahl der Kreuzungen in Abhängigkeit der Methode zur Barycenterberechnung.

Alle Tests wurden mit demselben Set an Testgraphen durchgeführt. Die Ergebnisse für jeden Testgraph sind separat gezeichnet, da sie sich in der Größe und Komplexität (Anzahl an Knoten und Kanten) deutlich unterscheiden. Um alle Ergebnisse übersichtlich genug darstellen zu können, wurde auf Achsenbeschriftungen für die einzelnen Diagramme verzichtet. Wichtig ist vor allem, die Unterschiede zwischen den verschiedenen Methoden sehen zu können. Die Ergebnisse sind in Form von Violindiagrammen, berechnet nach „Scott's normal reference rule“. In der Mitte einer jeden Violine ist ein Boxplot über die Ergebnisse abgebildet.

Der Aufbau der Übersichten ist immer gleich: die Ergebnisse der ersten 20 Graphen sind in der obersten Zeile immer in gleicher Reihenfolge abgebildet; die der anderen zeilenweise darunter. D. h. für alle Schaubilder gilt, die Diagramme an derselben Position entsprechen demselben Beispielgraphen. (Beispielsweise befinden sich alle Testergebnisse der unterschiedlichen Tests für den dritten Beispielgraphen immer in der ersten Zeile an dritter Stelle.)

Für jedes Diagramm für den ersten Test gilt: von links nach rechts entspricht die erste Violine (blau) dem kräftebasierten Ansatz zur Vorverarbeitung, die zweite (orange) der Breitensuche und die dritte (grün) der zufälligen Richtungsbestimmung.

Für die Diagramme des zweiten Tests gilt, links (blau) entspricht dem Ansatz der Verwendung der Knotenkoordinaten zur Barycenterberechnung und rechts (orange) dem Ansatz die Portkoordinaten zu verwenden.

Es gibt einige Besonderheiten. Auffällig sind insbesondere zunächst die leeren Diagramme. Sie entsprechen Testgraphen, welche nur aus einem Knoten oder einer Knotengruppe bestehen, daher gibt es hier keine sinnvollen Ergebnisse zu sammeln. Sind eine oder mehrere der Violinen nur ein Strich, waren die Ergebnisse für alle Durchläufe exakt gleich – dieser Fall tritt häufiger bei der Anzahl der Kreuzungen bei besonders kleinen Graphen oder solchen mit wenigen Kanten auf, da diese dann immer kreuzungsfrei gezeichnet werden können.

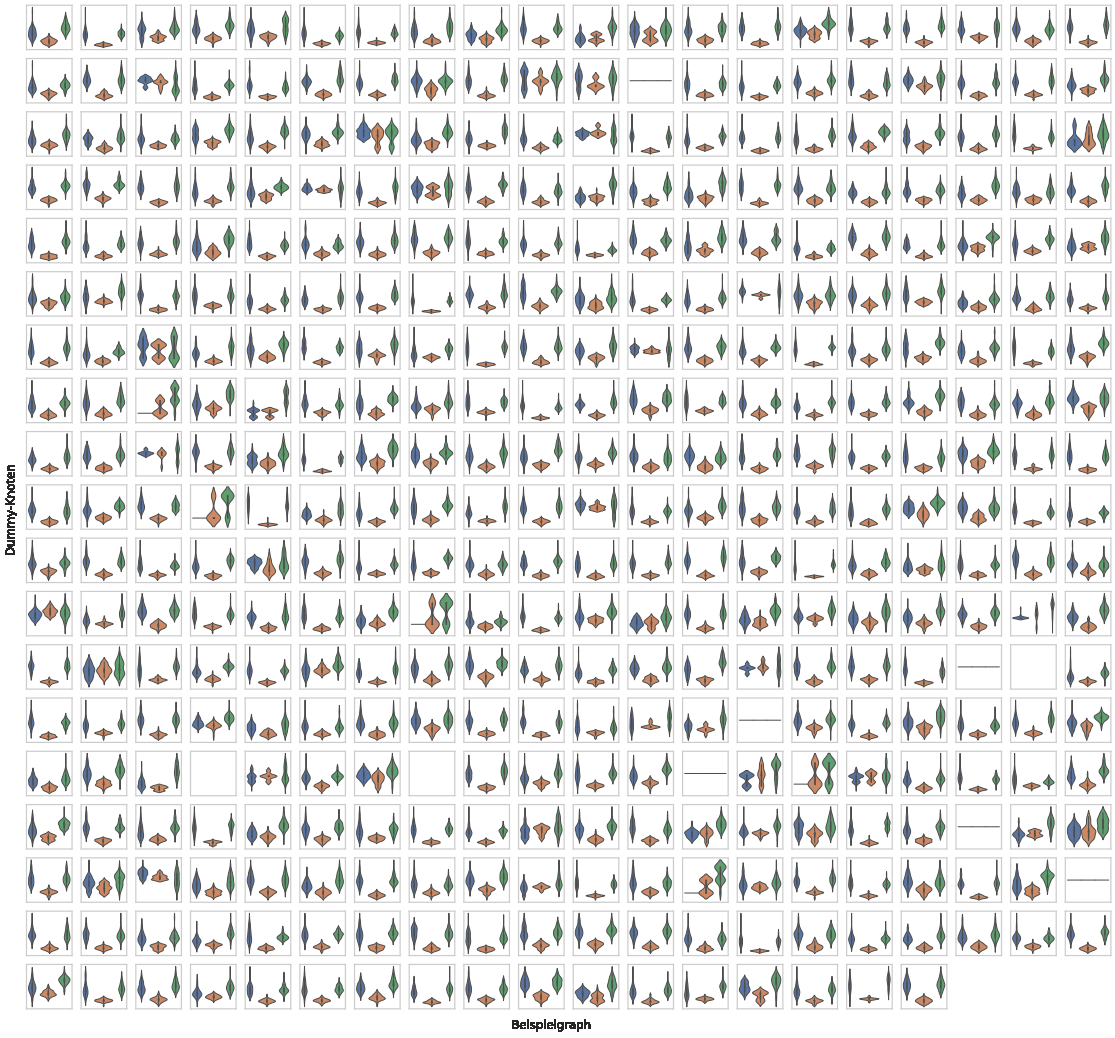


Abb. A.1: Ergebnisse Test 1; Anzahl der benötigten Dummyknoten für alle Beispielgraphen in Abhängigkeit der Methode zur Richtungszuweisung; blau kräftebasiert; orange Breitensuche; grün zufällig

Abb. A.2: Ergebnisse Test 1; Anzahl der Kreuzungen für alle Beispielgraphen in Abhängigkeit der Methode zur Richtungszuweisung; blau kräftebasiert; orange Breitensuche; grün zufällig

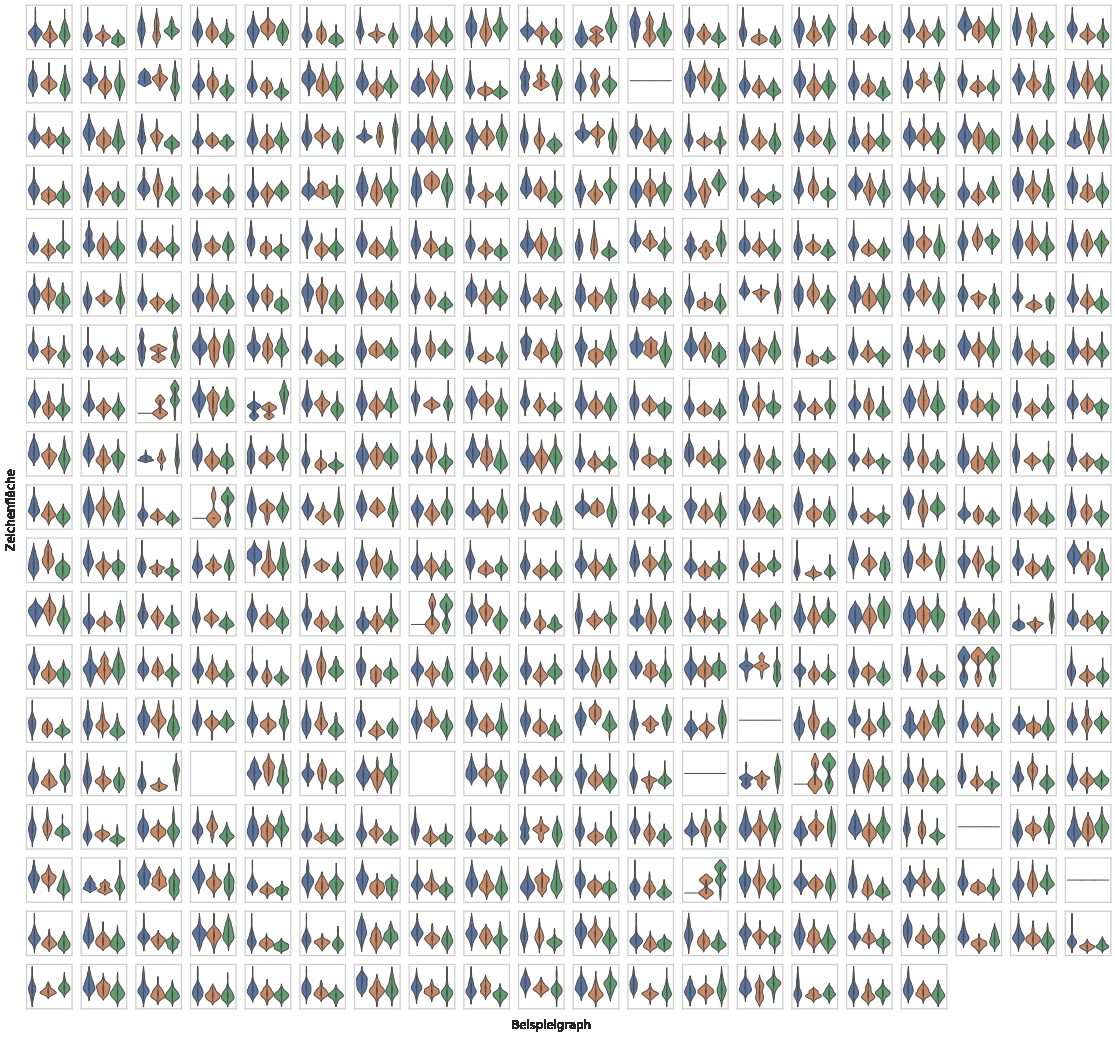


Abb. A.3: Ergebnisse Test 1; benötigte Zeichenfläche für alle Beispielgraphen in Abhängigkeit der Methode zur Richtungszuweisung; blau kräftebasiert; orange Breitensuche; grün zufällig

Abb. A.4: Ergebnisse Test 1; Seitenverhältnis der Zeichnung für alle Beispielgraphen in Abhängigkeit der Methode zur Richtungszuweisung; blau kräftebasiert; orange Breitensuche; grün zufällig

Abb. A.5: Ergebnisse Test 1; Höhe der Zeichnung für alle Beispielgraphen in Abhängigkeit der Methode zur Richtungszuweisung; blau kräftebasiert; orange Breitensuche; grün zufällig

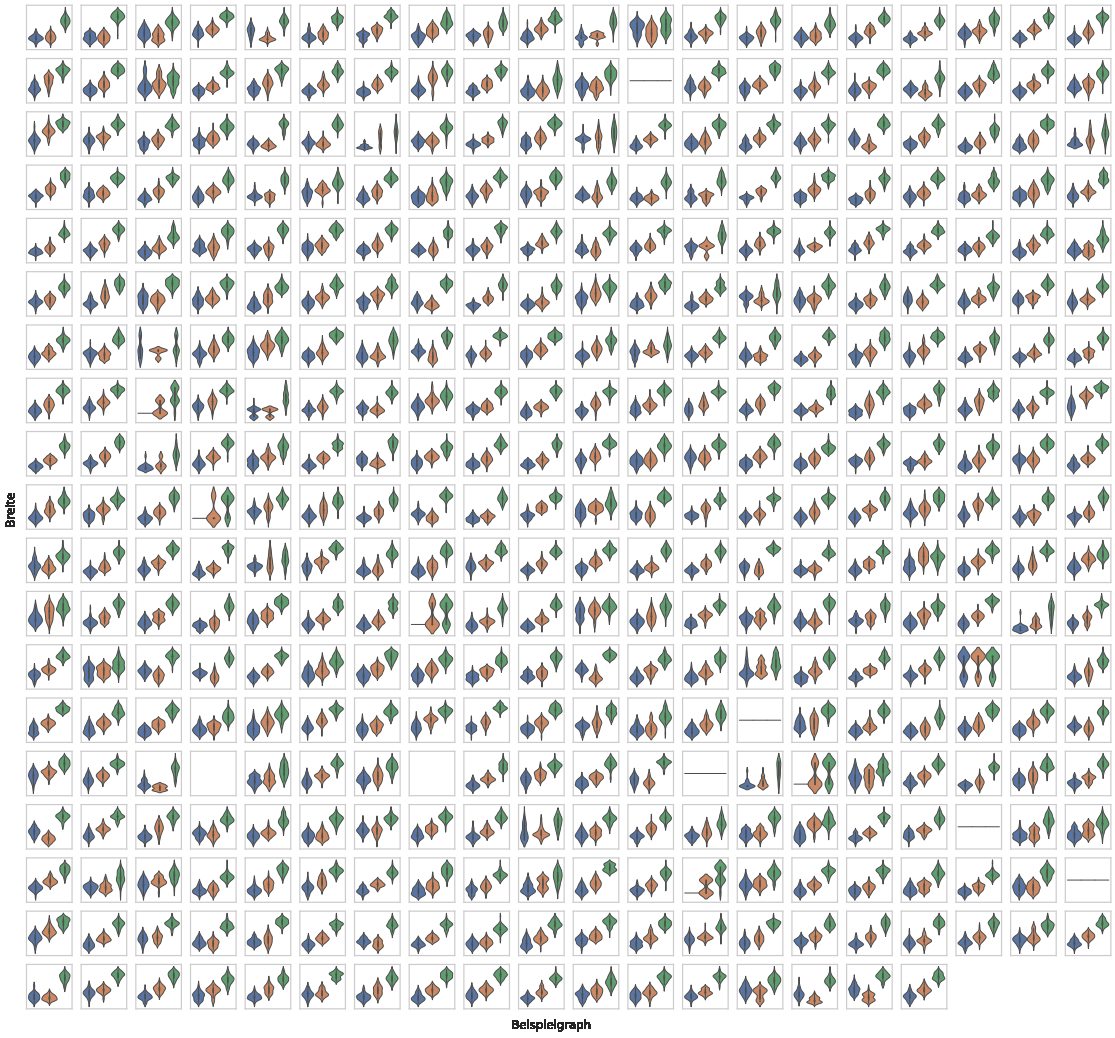


Abb. A.6: Ergebnisse Test 1; Breite der Zeichnung für alle Beispielgraphen in Abhängigkeit der Methode zur Richtungszuweisung; blau kräftebasiert; orange Breitensuche; grün zufällig

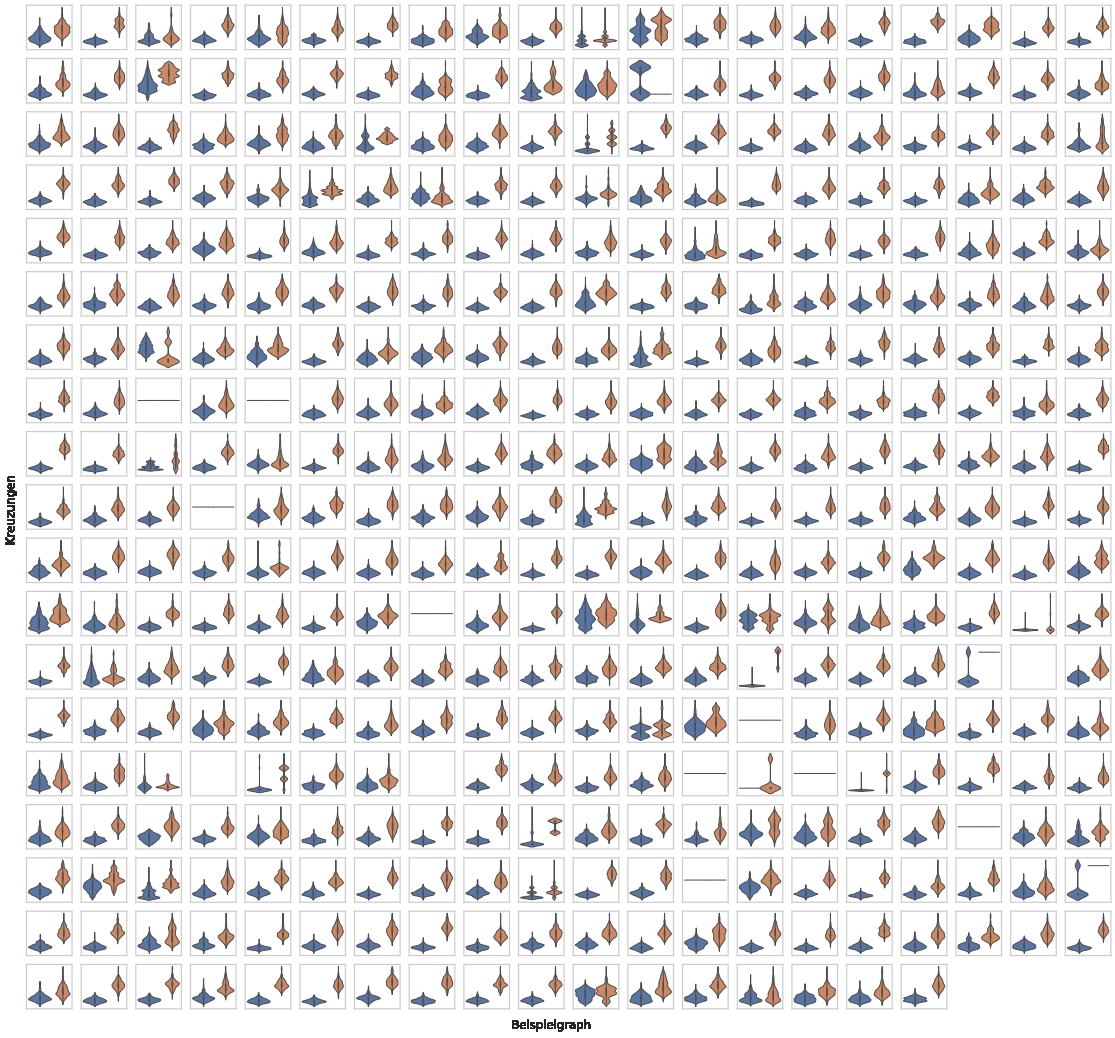


Abb. A.7: Ergebnisse Test 2; Anzahl der Kreuzungen für alle Beispielgraphen in Abhängigkeit der Methode zur Barycenterberechnung; blau Knotenkoordinaten; orange Portkoordinaten

A.2 Implementierung

Zur Implementierung wurde die Datenstruktur des Praline-Projekts [pra20] verwendet. Diese liefert Methoden zum Einlesen von Graphen aus json-files. Gegeben wird also eine Praline-Graph-Instanz (`Graph`), bestehend aus Knoten (`Vertex`), Knotengruppen (`VertexGroup`), Kanten (`Edge`), Kantenbündeln (`EdgeBundle`), Ports (`Port`) und Portgruppen (`PortGroup`). Jede Instanz einer dieser Klassen hat eine ID gespeichert als `MainLabel`. Zusätzliche Informationen liefern `TouchingPairs` (Knoten welche sich berühren z.B. bei Steckverbindungen) und `PortPairings` (Portverbindungen ohne Kanten, können bei TouchingPairs auftreten).

Beschreibung der Klassen

Der Algorithmus ist vollständig in der Klasse `Sugiyama` implementiert, der bei Erstellung ein Praline-Graph-Objekt übergeben werden muss. Zusätzlich können Informationen zur Darstellung in Form eines `DrawingInformation` Objekts übergeben werden. `Sugiyama` beinhaltet die Methode `computeLayout()`, welche dann entsprechend des in dieser Arbeit beschriebenen Algorithmus Knotenpositionen und Pfade für die Kantenführung berechnet. Die einzelnen Schritte des Algorithmus sind jeweils in einzelnen Klassen implementiert, sodass sie leicht ausgetauscht werden können. Jede dieser Klassen benötigt bei Erstellung ein `Sugiyama`-Objekt und ggf. zusätzliche Informationen bei Ausführung des entsprechenden Algorithmusschrittes. Die Klassen der Algorithmusschritte sowie `Sugiyama` sind im Folgenden jeweils einzeln näher beschrieben.

Sugiyama Klasse, welche den `Graph` hält und über `getGraph()` zur Verfügung stellt. Hier werden ebenfalls alle Informationen gespeichert und Methoden zum Abfragen und Abspeichern der Ergebnisse der Algorithmusschritte bereit gestellt.

Enthält auch die Methode `prepareGraph()` die im Konstruktor aufgerufen wird. Sie sorgt für ein eindeutiges `MainLable` aller Objekte des Graphen, welche in Schritt 1 Richtungszuweisung für die Erstellung eines `edu.uci.ics.jung.graph.UndirectedSparseGraph` benötigt werden.

Die Hauptmethode `computeLayout()` ruft als erstes die Methode `construct()` auf, welche den Graphen durch Dummyknoten und Dummykanten in einen Graphen mit den geforderten Voraussetzungen umbaut. Sie stellt also im Wesentlichen den Schritt 0 Vorverarbeitung dar – der einzige Schritt der nicht in eine eigene Klasse ausgelagert ist.

DirectionAssignment Stellt Schritt 1 dar. enthält die Methoden `forceDirected()` und `breathFirstSearch()` welche bei Aufruf Kantenrichtung mit der entsprechenden Methode errechnen. Diese werden in `Sugiyama` in den `HashMaps` `edgeToStart`, `edgeToEnd`, `nodeToOutgoingEdges` und `nodeToIncomingEdges` über die Methode `assignDirection()` gespeichert.

LayerAssignment Stellt Schritt 2 dar. enthält die Methode `NetworkSimplex()` welche ein `HashMap<Vertex,Integer>` zurück gibt. Diese `HashMap` ordnet Knoten einen

Rang zu. Sie wird in `Sugiyama` in `nodeToRank` gespeichert und anschließend wird von `Sugiyama` mit der Methode `createRankToNodes()` `rankToNodes` mit den Informationen aus `nodeToRank` befüllt.

DummyNodeCreation Stellt Schritt 3 dar. Stellt die Methode `createDummyNodes()` zur Verfügung, welche eine HashMap mit allen erstellten Dummyknoten zurück gibt. Diese wird in `Sugiyama` in `dummyNodes` gespeichert und Anhand dieser können alle Dummyknoten erkannt und wieder durch die ursprünglichen Kanten ersetzt werden. Sie dient ebenfalls zur Kennzeichnung der Dummyknoten was insbesondere für die Knotenpositionierung wichtig ist.

CrossingMinimization Stellt Schritt 4 dar. Liefert die Methode `layerSweepWithBarycenterHeuristic()` welcher die Information übergeben werden muss, welche Methode für die Berechnung der Barycenter verwendet werden soll. Sie gibt ein `CMResult` zurück welches die Informationen über die errechnete Reihenfolge der Knoten und Ports enthält. Dieses Objekt wird als `orders` in `Sugiyama` abgespeichert.

`CrossingMinimization` ist so gehalten, dass das Objekt nur einmal erstellt werden muss und `layerSweepWithBarycenterHeuristic()` mehrfach aufgerufen werden kann um unterschiedliche Ergebnisse zu erhalten. Bei jedem Aufruf kann die Methode zur Barycenter-Berechnung frei gewählt werden. Auch wird bei jedem Aufruf eine neue, komplett zufällige Startreihenfolge der Knoten und Ports gewählt.

NodePlacement Stellt Schritt 5 dar. Erwartet bei Erstellung zusätzlich zum `Sugiyama`-Objekt ein `CMResult` sowie eine `DrawingInformation`. In der `DrawingInformation` sind alle relevanten Informationen zur Zeichnung enthalten, wie Kantenabstand oder Schriftart und -größe.

`NodePlacement` leifert die Methode `placeNodes()` welche im `Graph` des `Sugiyama`-Objects die `Shape`s der Knoten und Ports in Form von `Rectangle`s setzt. Hierbei wird zunächst aus den Informationen des `CMResult` die in Kapitel 3 Schritt 5 beschriebene Datenstruktur der Ports im Objekt `structure` erstellt. Anschließend werden die Schritte, wie von Brandes und Köpfe [BK02] beschrieben, ausgeführt. Der Code hält sich größtenteils an den gegebenen Pseudocode des Papers und die benötigten Werte für die Knoten – in unserem Fall `Ports` – werden in der `HashMap portValues` in Form von `PortValue`-Objekten abgespeichert.

EdgeRouting Stellt Schritt 6 dar. Ist momentan leer. Sollte jedoch um eine Methode erweitert werden, welche die `Path`-Objekte der Kanten setzt.

Nachträgliche Änderungen

Für die Arbeit „Layered Drawing of Undirected Graphs with Generalized Port Constraints“ [WZBW20] wurden einige Änderungen am Code vorgenommen, welche nicht Teil dieser Arbeit sind. Da sich die Abgabe dieser Arbeit jedoch durch die Arbeit an

„Layered Drawing of Undirected Graphs with Generalized Port Constraints“ verschoben hat, Ist der Code auf einem aktuelleren Stand und beinhaltet einige Neuerungen, die hier aufgeführt sind. Diese sind ausdrücklich nicht mehr im Rahmen der Masterarbeit entstanden und zum Teil von Johannes Zink umgesetzt.

Sortierung Einige der Klassen haben mehrere Rückgabeobjekte, welche in einer Rückgabeklasse zusammengeführt sind. Dies trifft insbesondere auf die Klasse `CrossingMinimization` zu, wo die Klasse `CMResult` erstellt wurde, welche die Reihenfolge der Knoten und Ports beinhaltet. Es wurden daher für jeden Algorithmusschritt ein eigenes Paket erstellt, welches alle diesem Schritt zugehörigen Klassen beinhaltet.

DummyTurningPoints Der Schritt der Dummyknoten-Erstellung wurde um eine Neue Form von Dummyknoten erweitert, „DummyTurningPoints“. Über einen solchen Umkehrpunkt bzw. Umkehrknoten werden alle Kanten geleitet, die von der Oberseite eines Knotens zu einer Ebene darunter führen oder umgekehrt. Es werden hierdurch Dummyknoten nach Abbildung 3.2 ersetzt und es existiert immer maximal ein Knoten dieser Art für jeden ursprünglichen Knoten über den alle solchen Kanten geführt werden. Die restlichen Schritte des Algorithmus wurden um die Unterstützung dieser neuen Knoten erweitert.

CrossingMinimization Es existieren aktuell 2 Klassen zu diesem Schritt. `CrossingMinimization` entspricht der oben beschriebenen Implementierung. `CrossingMinimization2` ist eine Erweiterung um eine zusätzliche Möglichkeit der zur Barycenterberechnung sowie um eine Unterstützung der neuen Dummyknoten. Hier ist geplant, dass in der Kreuzungsreduzierung darauf geachtet wird, dass innerhalb der neuen DummyTurningPoints bei Auflösung keine neuen Kreuzungen entstehen können. Diese Unterstützung ist momentan noch in Arbeit. `CrossingMinimization` funktioniert natürlich auch noch, hat aber keine besondere Berücksichtigung der DummyTurningPoints.

EdgeRouting Es wurde ebenfalls nachträglich ein Algorithmus zur orthogonalen Kantenführung implementiert. Dieser sorgt für eine Kantenführung ohne zusätzliche Kreuzungen und läuft in linearer Zeit. Auch Überschneidungen von Kanten sind ausgeschlossen. Details hierzu finden sich in der Arbeit „Layered Drawing of Undirected Graphs with Generalized Port Constraints“ [WZBW20].

DrawingPreparation Die Klasse `DrawingPreparation` wurde hinzugefügt um die Zeichnung zu optimieren. In diesem Schritt werden Dummyknoten zu Kanten aufgelöst, sowie normale Knoten verkleinert. Hier sollen ebenfalls Dummyknoten-Ebenen komplett aufgelöst und die Zeichnung dabei soweit möglich verkleinert werden. Teile hiervon sind noch nicht fertig umgesetzt.

Erklärung

Hiermit versichere ich die vorliegende Abschlussarbeit selbstständig verfasst zu haben, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 16. Juni 2020

. . . . . .
Julian Walter