Karlsruhe Institute of Technology

Kräftebasierter Layoutalgorithmus für Argumentkarten mit Gruppenstruktur

Seminararbeit zur Veranstalltung Visualisierung komplexer Argumentation

Christian, Jonathan, Sven, Michael, Sebastian, Mira

Betreut durch:

Jun.-Prof. Gregor Betz, Diplom Inform. Andreas Gemsa und Dr. Ignaz Rutter

Abstract

Argumentkarten werden zur Visualisierung von Debatten verwendet. Eine Problemstellung hierbei ist das automatische Berechnen von Layouts. In dieser Arbeit präsentieren wir einen Ansatz zur Layoutberechnung von Argumentkarten welche eine Gruppenstruktur enthalten. Der Ansatz basiert auf kräftebasierten Algorithmen und der Kapselungen von Gruppen zur Trennung der Layouts pro Stufe. Darüberhinaus werden zwei alternative Ansätze beschrieben und mit der präsentierten Lösung verglichen.

Inhaltsverzeichnis

1 Motivation

Diese Arbeit beschäftigt sich mit Argumentkarten, manchmal auch Argumentationskarten genannt. Diese dienen als graphische Darstellung von Debatten. Auf einer solchen Karte werden die Thesen und Argumente der Debatte als einzelstehende Elemente herausgestellt. Um ihre Zusammenhänge zu verstehen, ist es auch nötig ihre Beziehungen, welche von unterstützenden oder angreifender Natur sein können, ebenfalls zu visualisieren. Eine Argumentkarte ist also die Darstellung einer Debatte in Form eines Graphen.

Die Argumentkarten sind vor allem ein Werkzeug zum besseren Verständnis von der Debatte. Über die Visualisierung der Elemente und ihre Beziehungen hinaus, existieren weitere Möglichkeiten um das Verstehen der Debatte zu verbessern und zu erleichtern. So können beispielsweise thematisch verwandte Knoten gruppiert werden. Auch durch die Option, Gruppen zunächst geschlossen zu visualisieren und Details zu verbergen, könnnen Gruppen einen schnelleren Überblick über die Debatte ermöglichen. Das Hervorheben einzelner Agrumentationsstrukturen durch die Wahl des Layouts ist eine weiterer wichtiger Aspekt bei der Visualisierung von Debatten.

Damit sich die von einer Argumentkarte bereitgestellten Informationen sinnvoll verwenden lassen muss die Argumentkarte folglich auf eine geeignete Weise dargestellt werden. Es existieren bereits einige Werkzeuge mit denen Argumentkarte angelegt und ein Layout automatisch erzeugt werden kann. Allerdings müssen die resultierenden Layouts der Argumentkarten oftmals von Hand nachbearbeitet werden, um eine befriedigende Darstellung der Strukturen zu erhalten. Darüber hinaus ist eine Unterstützung von Gruppen nicht immer gegeben.

In dieser Arbeit geht es um die automatisierte Darstellung von Argumentkarten mit Gruppen, sowie die Interaktion mit diesen Gruppen. Es geht also um die Frage, wie sich Gruppen und Gruppenhierarchien den Anforderungen an Argumentkarten entsprechend visualisieren lassen und wie das Layout auf das Öffnen und Schließen von Gruppen reagiert.

Im nächsten Kapitel 2 präzisieren und abstrahieren wir diese Problemstellung. In Kapitel 3 präsentieren wir unseren Lösungsansatz, welchen wir dann in Kapitel 4 mit alternativen Ansätzen vergleichen und bewerten. Kapitel 5 schließt mit einer Zusammenfassung.

2 Problemstellung

2.1 Layoutproblem

Das Zeichnen eines Graphen setzt sich aus dem Finden eines Layouts sowie des Renders zusammen. Für uns von Interesse ist nur ersteres, das Layoutproblem. Ziel hierbei ist es für einen gegebenen Graphen und gegebenenfalls weitere Informationen algorithmisch ein Layout zu berechnen, welches gewisse Zeichenkonventionen erfüllt, erwünschte Ästhetikkriteren optimiert und gegebenenfalls weiteren lokalen Nebenbedingungen genügt.

Im Fall der Argumentkarten abstrahieren wir für das Layoutproblem einige Informationen. So spielen die Texte der einzelnen These oder Argumente für uns keine Rolle. Sie werden lediglich durch achsenparallele Rechtecke als Knoten des Graphen repräsentiert. Des Weiteren ignorieren wir bei den Beziehungen zwischen Elementen den Typ sowie die Richtung. Es ist also lediglich von Relevanz ob zwei Elemente in einer Beziehung stehen. Dies wird dann durch eine Kante repräsentiert.

Unter Einbeziehung von Gruppen kann die Eingabe des Algorithmus dann als der Form $G = (V, V_g, E, \sigma)$ dargestellt werden. Die Mengen und Abbildungen dieses Graphen werden im folgenden Beschrieben:

- Knoten V, Knoten die eine Aussage repräsentieren.
- Gruppenknoten V_g,Knoten die eine Gruppe repräsentieren.
- Kanten $E \subseteq \bar{V} \times \bar{V}$, wobei $\bar{V}$ die Menge aller Knoten ($\bar{V} = V \cup V_g$) ist.
- Rechtseindeutige Abbildung $\sigma(v) : \bar{V} \to V_g$, Die Knoten auf ihre Gruppe abbildet (Gruppen können weitere Gruppen enthalten).

Bei Gruppen ist außerdem zu beachten, dass sie entweder in einem geöffneten oder geschlossen Zustand sein können.

2.1.1 Zeichenkonventionen

Zeichenkonventionen beschreiben wie Knoten und Kanten gezeichnet beziehungsweise gelayoutet werden sollen. Sie müssen erfüllt werden. In un-

serem Fall sind diese Anforderungen im einzelnen:

- Knoten werden als achsenparallel Rechtecke dargestellt 
- Überschneidungsfreiheit von Knoten
- Überschneidungsfreiheit von Kanten mit Knoten
- Eine geschlossene Gruppe enthält keine Knoten

Für unseren Lösungsansatz haben wir für Gruppen weitere Zeichenkonvention hinzugefügt:

- Gruppen werden als Kreise dargestellt
- Eine geöffnete Gruppe beinhaltet alle Kindknoten sowie Kindgruppen
- Überschneidungsfreiheit von Gruppen mit nicht Kindknoten oder -gruppen, sowie Kanten zwischen solchen

2.1.2 Ästhetikriterien

Ästhetikkriterien sollen vom berechneten Layout optimiert werden. Auch wenn es für Argumentkarten im Allgemeinen wünschenswert ist, haben wir in unserem Lösungsansatz die Kreuzungsminimierung nicht beachtet.

- Größenminimierung
- Kreuzungsminimierung

Lokale Nebenbedingungen haben wir für das Layoutproblem nicht definiert.

2.2 Interaktion mit Gruppen

Da Gruppen sowohl geöffnet als auch geschlossen sein können, existieren eine Vielzahl von verschiedenen Layouts für eine Argumenkarte. Außerdem sollte in einer Implementierung das Wechseln zwischen den verschiedenen Zuständen möglich sein, also das Öffnen und Schließen von Gruppen. Dadurch soll es zum einen einfach sein sich einen Überblick zu verschaffen und zum anderen eine detailliertere Ansicht zu haben. Hierbei ist höchst wünschenswert, dass bei einer solchen Interaktion das alte und neue Layout nur so wenig wie möglich und nur so viel wie nötig unterscheiden. Der Nutzer soll sich mit seiner mentalen Karte der Argumentkarte auch nach der Interaktion zurechtfinden können.

Eine weitere wünschenswerte Anforderung ist, dass das Layout nach wiederholten Auf- und Zuklappzyklen immer wieder zu dem gleichen Anfangs-

layout oder dazu ähnlichem Layout zurückkehrt.

Wir erhalten für eine Interaktion und das Verhältnis der einzelnen berechneten Layouts also weitere Kriterien:

- Relative Positionen von Elementen verändert sich nur so viel wie nötig
- Gesamtlayout ändert sich nur so viel wie nötig
- Layout eines Zustandes ist auch nach mehreren Interaktionen das gleiche oder sehr ähnlich

Hierbei sei auch angemerkt das im Folgenden die Beziehungen der einzelnen Layouts, also die einzelnen Layouts von verschiednen Konfigurationen von auf- bzw. zugeklappten Gruppen, so aufgefasst werden, dass wir ein einziges Layout haben und dieses durch eine Interaktion in einen anderen Zustand wechselt.

Aus diesen und den in Abschnitt 2.1 vorgestellten Anforderungen ergibt sich allerdings eine Reihe von Konflikten. Beispielsweise steht die Größenminimierung der Zeichnung im Konflikt mit der Anforderung, dass sich das Gesamtlayout nur so viel wie nötig ändert, da größenminimale Layouts nach dem öffnen bzw. schließen einer Gruppe möglicherweise nur mit einer großen Veränderung des Layouts möglich sind.

Der im Folgenden präsentierte Ansatz gibt jedoch Lösungen zuf den Problemen oder präferiert eine der Optionen.

3 Algorithmus

Für das in Kapitel 2 definierte Layoutproblem haben wir uns einen Lösungsansatz überlegt. Die wichtigste Designentscheidung und damit Grundidee des Layouts ist es, jede Gruppen in einer Kugel zu kapseln. Sie wird also innerhalb eines Kreise dargestellt und Kanten über Gruppengrenzen werden nur über feste Ports zugeführt. Dadurch werden die Layouts pro hierarchische Stufe und Gruppe getrennt. Ist ein Layout einer höhere Stufe berechnet, hat dies nur durch das Festlegen der Portpositionen Einfluss auf das Layout von Gruppen eine Stufe tiefer. Layouts von niedrigeren Stufen beeinflussen übergeordnete Layouts nur durch ihren benötigten Platz. Abbildung 3.1 zeigt so ein Layout.

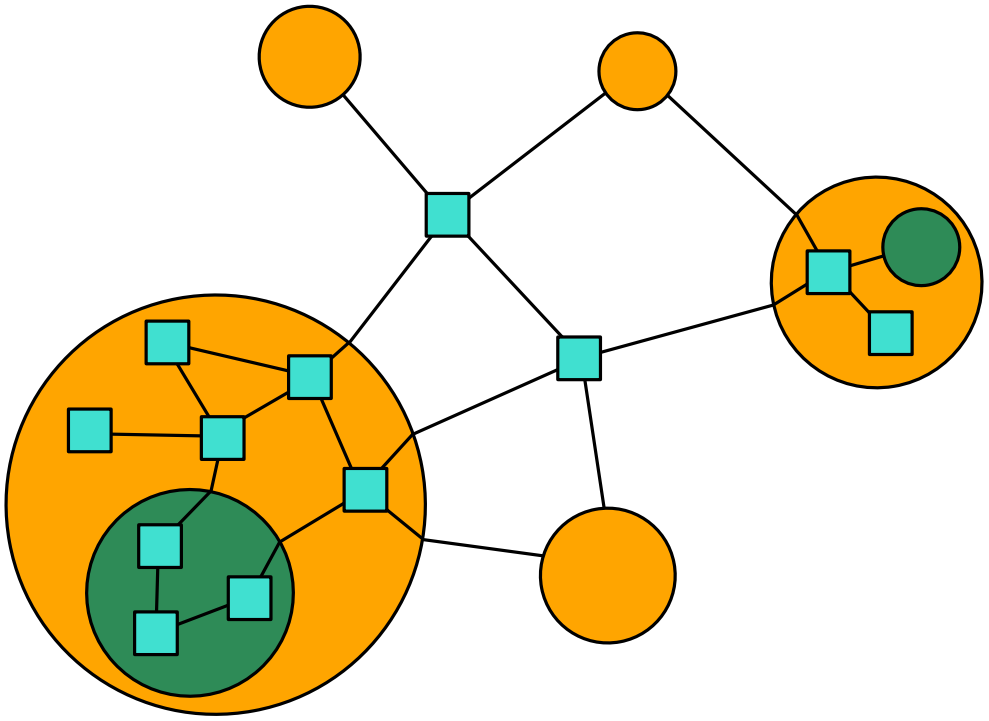

Abbildung 3.1: Beispiellayout einer abstrahierten Argumentkarte nach unserem Konzept.

Im Folgenden beschreiben wir einen Algorithmus um so ein Layout zu berechnen, sowie das Konzept um Layoutänderungen beim Öffnen oder Schlie-

ßen von Gruppen gering zu halten. Zunächst widmen wir uns jedoch einigen weiteren Aspekten von Relevanz für unseren Algorithmus.

Kräftebasierter Algorithmus

Grundlage für unseren Algorithmus ist die Verwendung eines kräftebasierten Algorithmus, welcher auch die Mächtigkeit besitzt, Knoten von bestimmter Größe, repräsentiert durch geometrischer Formen, überlappungsfrei zu layouten, sowie Knoten-Kanten-Überschneidungen zu verhindern. Mit diesem Problem hat sich zum Beispiel bereits [HK02] beschäftigt.

Gruppengrößen

Wie die Größe einer geöffneten Gruppe berechnet werden kann ist in Abschnitt 3 genauer beschrieben. Eine jedoch vom beschriebenen Algorithmus unabhängige Designentscheidung ist die Größe von geschlossenen Gruppen. Während eine Möglichkeit wäre, jede geschlossen Gruppe mit einem gleich großen Kreis darzustellen, schlagen wir einen anderen Ansatz vor.

Um direkt zu sehen, ob sich hinter einer geschlossenen Gruppe eine große oder kleine Gruppe bzw. eine mit vielen oder wenigen Kindelemente befindet, empfiehlt es sich den Kreisradius der geschlossenen Gruppe in Beziehung zu der Anzahl der Kindelemente sowie der Größe der Gruppe im offenen Zustand zu setzen. empfiehlt es sich den Kreisradius der geschlossenen Gruppe in Beziehung zu der Größe der Gruppe im offenen Zustand zu setzen. Da die Größe einer offenen Gruppe stark mit den Anzahl der Kindelemente korreliert, wird dadurch auch diese Eigenschaft einer Gruppe durch die geschlossene Repräsentation wiedergegeben. Die Größe der geschlossenen Gruppe könnte also dadurch berechnet werden, dass sie auf einer Skala von einer minimalen bis zu einer maximalen Größe abgebildet wird. Die minimale Größe könnte hierbei zum Beispiel durch die Größe des am größten dargstellten Arguments gegeben sein und die maximale Größe durch γ % der größten Gruppe.

Da der Unterschied zwischen kleinen Gruppen, wobei zum Beispiel eine 4 Kindelemente Besitzt und einer etwas größere die 5 besitzt, stärker verdeutlicht werden soll, als der zwischen relativ großen Gruppen, schlagen wir eine logoarithmische Abbildung wie in Abbildung 3.2 vor.

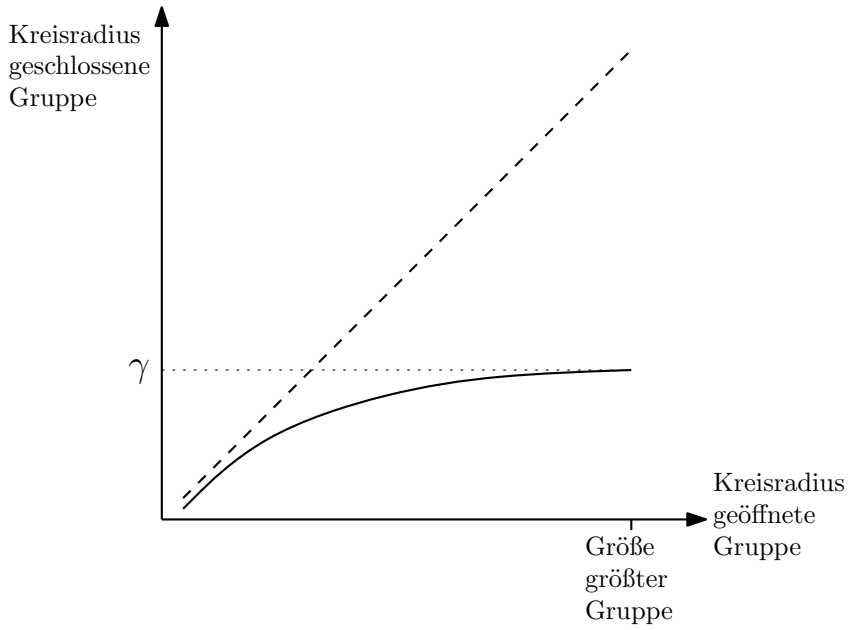

Abbildung 3.2: Vorgeschlagene Wahl der Gruppengröße einer geschlossenen Gruppe in Abhängigkeit der Gruppe in geöffnetem Zustand mit $\gamma = 25$.

Anker

Da wir in der genaueren Beschreibung des Algorithmus öfters sogenannte *Anker* verwenden werden, wollen wir diese kurz definieren. Ein Anker für einen Knoten ist ein Pseudoknoten, ein Ankerpunkt, welcher also nich gerendert wird, sowie eine Kante zu diesem Knoten. Anders als normale Kanten ist die optimale Kantenlänge eines Ankers 0. Außerdem hat sein Ankerpunkt eine feste Position im Layout. In einem kräftebasierten Algorithmus wirkt der Anker auf seinen Knoten nun wie eine weitere Feder, die ihn zum Ankerpunkt hinzieht. Die Kraft der Feder kann wie eine normale Kante gehandhabt werden oder bei Bedarf auch anders spezifiziert werden.

3.1 Konzept und Idee

Wenn alle Gruppen beim Öffnen einer Karte geschlossen sind, sollte es auf großen Karten einfach sein direkt einen Überblick gewinnen zu können. Deshalb liefert der folgendes Algorithmus ein Layout bei dem alle Gruppen geschlossen sind.

Unser in Algorithmus 1 grob beschriebene Layoutalgorithmus ist in 2 Phasen unterteilt. In der ersten Phase, Zeilen 1 -8, werden die in einem bottem-up Ansatz die Größen der Gruppen berechnet. Dies beginnt auf der tiefsten Stufe und geht dann iterativ nach oben, da die Größe einer Gruppe jeweils für die Größe der übergordneten Gruppe bzw obersten Stufe notwendig ist. In einem top-down Ansatz wird in der zweiten Phase, Zeilen 9-17, werden nun die Layouts pro Stufe berechnet und die Ports festgelegt.

Eingabe : Graph $G = (V, E)$ mit Gruppen S
Ausgabe : Gruppen-hierarchisches Layout von G

```
1  i = niedrigste Stufe einer Gruppe ;
2  solange i ≥ 0 tue
3      für Jede Gruppe H auf Stufe i tue
4          berechnete Layout L'_H der Gruppe H;
5          berechne benötigte Fläche des Gruppenlayouts;
6      Ende
7      i = i + 1;
8  Ende
9  Lege Ports für Gruppen auf Stufe 1 fest;
10 i = AnzahlStufen;
11 solange i ≤ Anzahl Stufen tue
12     für Jede Gruppe H auf Stufe i tue
13         berechnete Layout L_H der Gruppe H unter Berücksichtigung der
           Ports;
14         Lege Ports für Gruppen in H auf Stufe i − 1 fest;
15     Ende
16     i = i − 1;
17 Ende
```

Algorithmus 1 : Layoutalgorithmus

3.1.1 Bottom-up Anteil des Algorithmus

Schauen wir uns den bottom-up Teil des Algorithmus genauer an. Ziel ist es zunächst für jede Gruppe ein benötigte Größe zu approximieren. Da für die Gruppen der Positionen die Positionen der Ports jedoch noch nicht bekannt sind, werden die Gruppen zunächst ohne Ports gelayoutet. Hierfür verwenden wir dann den oben beschrieben benötigten kräftebasierten Algorithmus. Zusätzlich wird jedoch für jeden Knoten noch ein Anker zu einer relativen Mitte der Gruppe gesetzt. Auf Grund der Verwendung eines kräftebasierten Algorithmus hält dies die Gruppe, die nicht zwingend eine Zusammenhangskomponente sein muss, zusammen und konzentriert sie einem eher kreisförmigen Bereich. Anderenfalls könnte sich zum Beispiel eine lange Kette bilden.

Da die Größe von Kindgruppen einer Elterngruppe für deren Layout benötigt wird, beginnt der Algorithmus auf der tiefsten Stufe, also bei den Gruppen, die in den meisten Gruppen enthalten sind. Wurde ein Layout berechnet, kann ein Kreis darum gelegt werden und somit die Größe approximiert werden. Mit der oben genannten Rechnung für die Größe von geschlossenen Gruppen, kann nun also die Größe für das Layout eine Stufe höher verwendet werden. Dies wiederholt sich bis zur obersten Stufe. Hier sind jedoch kein Anker zu einem Mittelpunkt mehr zwingend nötig. Außer man möchte die gesamte Karte eventuell in einen keisförmigen Bereich ziehen.

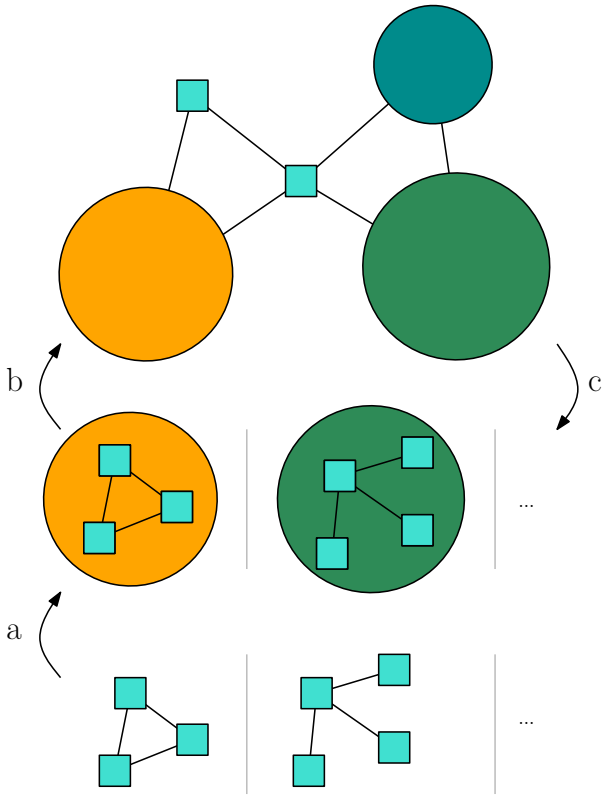

Abbildung 3.3: Bottom-up Anteil des Algorithmus zur Berechnung der benötigten Größen. Nachdem Layouts auf der untersten Stufe berechnet wurden (Zeile 4, Algorithmus 1) wir in Schritt *a* die benötigt Größe der Gruppe berechnet (Zeile 5). Nun kann dieser Prozess mit *b* und *c* bis zur obersten Stufe wiederholt werden.

3.1.2 Top-down Anteil des Algorithmus

Der Übergang vom bottom-up zum top-down Anteil erfolgt, wenn auf der obersten Stufe das Layout berechnet wird. Wenn zu Beginn, wie bei uns gewählt, alle Gruppen geschlossen sind, dann ist dieses berechnete Layout auch das Layout, dass beim öffnen der Karte zu sehen ist. Auf der obersten Stufe beginnend und durch das berechnete Layout vorgegeben, legen wir nun die Ports der Gruppen nächste-niedrigen Stufe fest und berechnen deren Layout. Wie ein Layout für eine Guppe berechnet wird und die Ports festgelegt werden, erläutern wir im folgenden genauer.

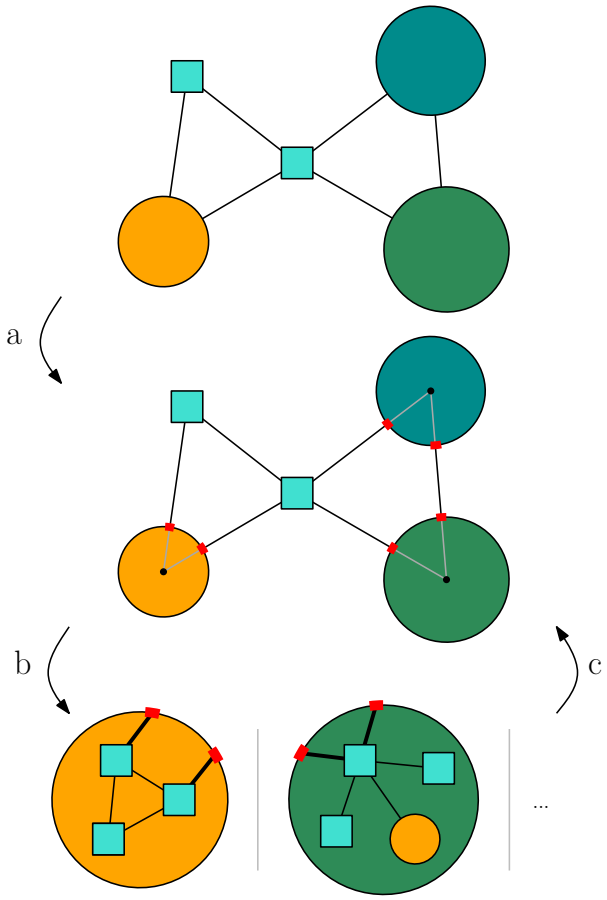

Abbildung 3.4: Top-down Anteil des Algorithmus zur Festlegung der Ports und Berechnung der Layouts. Mit dem Layout der höchsten Stufe können in Schritt *a* die Ports für die niedrigere Stufe berechnet werden (Zeile 9, Algorithmus 1). In Schritt *b* wird dann das Layout der nächsten Stufe mit Ports berechnet (Zeile 13). Dieser Prozess mit *b* und *c* bis zur untersten Stufe wiederholt werden.

3.1.3 Layout in Gruppen in Abhängigkeit von Ports

In diesem Abschnitt betrachten wir Zeile 13 des Layoutalgorithmus 1 genauer. Hier ist das Ziel ein Layout für eine Gruppe mit bereits gegebenen Ports, also mit gegebenen Winkel für Kanten zur Gruppe, zu finden. Das heißt, wir suchen nun einen Layout für den durch die Gruppe induzierten Teilgraphen sowie aus der Gruppe herausgehende Kanten. Dieses soll, wie oben beschrieben, ein Layout innerhalb eines Kreises sein und die Gruppengrenze-überschreitenden Kanten zu den festgelegten Ports führen. Eine der Herausforderung hierbei ist, eine geeignete Größe für den Kreis um die Gruppe zu finden. Auch zum finden dieses Layouts benutzen wir wieder einen kräftebasierten Algorithmus.

Gegeben sei nun also eine Gruppe H. Vom Layouts der nächsthöheren Stufe wurden bereits die Ports der Gruppe festgelegt, d.h. wir haben die Winkel an denen Gruppengrenze-überschreitenden Kanten den Kreis schneiden sollen. Diese Ports modellieren wir als feste Knoten auf dem Kreis der Gruppe an ihren jeweiligen Winkeln. Des weiteren haben wir bereits ein Layout $\mathcal{L}_{\mathcal{H}}$ der Gruppe H ohne Gruppengrenze-überschreitenden Kanten in Zeile 4 des Layoutalgorithmus 1 berechnet.

Aus Zeile 5 haben wir daher auch eine erste Größe für den Kreis der Gruppe, welche durch den Radius beschreiben und durch R'_H gegeben sei.

Der grobe Ablauf des Algorithmus ist in Algorithmus 2 beschrieben. Die essentiellen Schritte sind also die Wahl eines Anfangslayouts (Zeile 1), Finden eines optimalen Radius (2) und das Berechnen eines Layouts (3).

Eingabe : Gruppe H, sowie Ports und Kanten zu Ports
Ausgabe : Gruppenlayout $\mathcal{L}_H$ von H
1 Wähle Anfangslayout;
2 Finde Radius R für Kreis;
3 Berechne Gruppenlayout $\mathcal{L}_H$ mit kräftebasiertem Algorithmus;

Algorithmus 2 : Gruppenlayoutalgorithmus

Die Wahl eines Anfangslayouts und das finden eines optimalen Radius beschrieben wir in den folgenden Abschnitten genauer. Für die Berechnung eines Gruppenlayouts verwenden wir wieder den oben geforderten kräftebasierten Algorithmus, welcher mit Knoten von bestimmter Größer umgehen kann. Jedoch schlagen wir auch hier wieder das hinzunehmen eines Ankers vor, welcher alle Elemente zur Mitte des Kreises ziehen soll.

Dies soll auch verhindern, dass eventuell nicht zusammenhängende Komponenten der Gruppe auseinander driften. Die Kraft dieses Ankers sei mit dem Faktor α beschrieben. Solange der Kreis der Gruppe groß genug ist, sollte es für Knoten auch nicht möglich sein, aus dem Kreis herausgetragen zu werden. Andernfalls müssten man hierfür weitere Kräfte im Algorithmus

berücksichtigen.

Anfangslayout Für die Wahl eines Anfangslayouts, Zeile 1 in Algorithmus 2, schlagen wir folgenden Ansatz vor.

In Zeile 4 von Algorithmus 1 wurde für H bereits ein Layout $\mathcal{L}'_H$ berechnet. Dieses lässt sich sowohl horizontal als auch vertikal spiegeln, ohne die Ausrichtung der einzelnen Knoten, welche ja nach den Achsen ausgerichtete Rechtecke sind, verändert wird. Das heißt wir haben vier verschiedene Layouts, nämlich das original, vertikal oder horizontal gespiegelt sowie vertikal und horizontal gespiegelt. Um möglichst lange Kanten von Knoten zu Ports durch die ganze Gruppe zu vermeiden, wählt man nun jenes Layout, bei dem die Summe der Kantenlängen dieser Kanten am kleinsten ist. Dies ist nun das gewählte Anfangslayout $\mathcal{L}'_H$. Abbildung 3.5 gibt hierfür ein Beispiel.

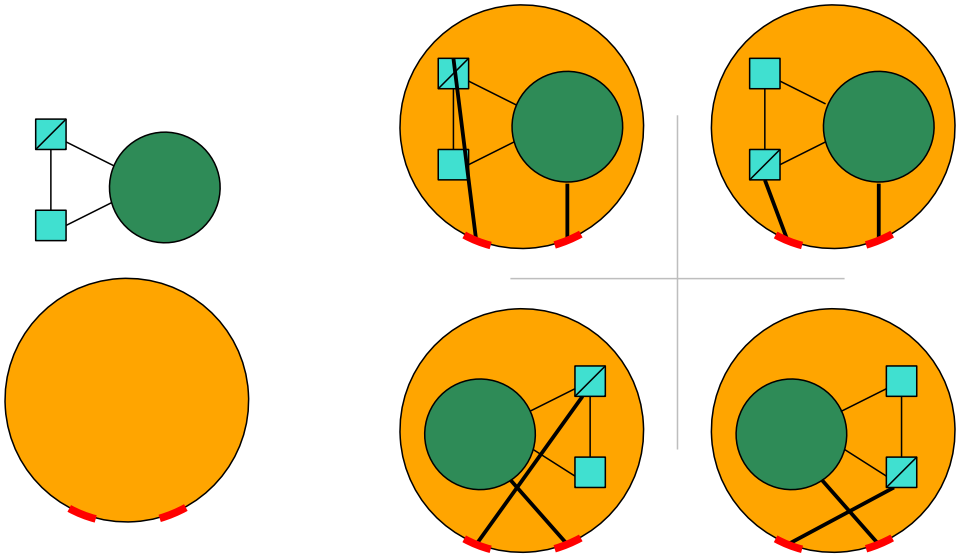

Abbildung 3.5: Ansatz zur Wahl des Anfangslayouts. Links das vorberechnete innere Layout und die Elterngruppe mit Ports. Rechts die vier verschiedene Spiegelungen des Layouts sowie die Kanten zu den Ports. In diesem Fall würde das rechte obere, also horizontal gespiegelte, Layout als Anfangslayout gewählt werden, da die Gesamtlänge der Kanten zu den Ports hier am kürzesten ist.

Natürlich gibt es auch noch weitere Ansätze zum finden eines Anfangslayouts. Einfache Ansätze wären zum Beispiel eine zufällige Platzierung der Knoten oder die Platzierung aller Knoten in die Mitte des Kreises, um dann den Rest dem kräftebasierten Layoutalgorithmus zu überlassen. Man könn-

te sich jedoch auch einen anspruchsvolleren Ansatz überlegen, bei dem man das Gruppenlayout ausgehend von den Ports konstruiert.

Bestimmen des Kreisradius einer geöffneten Gruppe Für die Berechnung eines angemessenen Kreisradius schlagen wir einen iterativen Algorithmus vor. Dessen Idee ist es, für einen gegeben Radius ein Layout zu berechnen und dann zu testen, ob der Radius verkleinert werden kann. Ist dies der Fall, so wiederhole das Vorgehen. Ansonsten erhöhe entweder den Faktor α für die Kraft des Ankers zum Mittelpunkt oder breche ab. Wichtig ist dabei anzumerken, dass die Gruppen innerhalb dieser Gruppe geöffnet sind. Als Anfangsradius kann der Radius R'_H aus Algorithmus 1 Zeile 5 mal einem Faktor β verwendet werden. Algorithmus 3 spezifiziert also Zeile 2 von Algorithmus 2 und setzt auch dessen Zeile 3 um.

Eingabe : Gruppe H, Ports, Kanten zu Ports, Anfangslayout $\mathcal{L}'_H$, Radius R'_H
Ausgabe : Gruppenlayout $\mathcal{L}_H$ von H, Radius R_H

```
1  R = β · R'_H = Anfangsradius für Kreis;
2  L_H = L'_H ∪ Ports ∪ Kanten zu Ports;
3  Führe kräftebasiertem Algorithmus auf L_H aus;
4  wenn R verkleinert werden kann dann
5  |  Passe R an;
6  sonst
7  |  wenn α nicht zu groß dann
8  |  |  erhöhe α;
9  |  |  gehe zu 3;
10 |  sonst
11 |  |  Algorithmus fertig;
12 |  Ende
13 Ende
```

Algorithmus 3 : Kreisradiusalgorithmus

Für Algorithmus 3 muss natürlich noch spezifiziert werden, was es heißt, dass der Radius in Zeile 4 verkleinert werden kann oder dass α nicht zu groß ist. Dass der Radius R verkleinert werden kann, soll bedeuten, dass bei kleinerem R aber gleichem Layout der Kreis keine Knoten schneidet. Typisch für Parameter bei kräftebasierten Algorithmen, wird um eine geeignte maximale Größe von α zu finden, wohl eine Implementierung und verschiedene Tests benötigt. Das selbe gilt für β um einen geeigneten Anfangsradius zu finden.

Eine weitere Variante zur Berechnung des Kreisradius könnte durch erwartete Größe von Gruppen umgesetzt werden. Falls eine gutes β gefunden werden kann, dass für jede Gruppe eine gute Größe approximiert und dabei nie zu klein ist, kann der iterative Prozess auch ausgelassen werden

und der Radius der Gruppe für jeden Zustand direkt berechnet werden.

Setzen der Ports für Kindgruppen Wenn ein Layout berechnet wird, werden die Kanten zu einer Gruppe so gelayoutet, dass sie eine inzidente Gruppe nur radial schneiden. Der erhaltene Schnittpunkt legt so die Position für den Port fest, welche einfach durch den Winkel beschrieben werden kann. Dies ist auch in Abbildung Abbildung 3.4 dargestellt.

Da davon auszugehen ist, dass sich bei Veränderungen im Layout die Winkel nicht zu sehr ändern werden, bleiben die Winkel für jede Gruppengröße fest. Dadurch bleibt das Layout in einer Gruppe unabhängig von den Änderungen außerhalb. Jedoch müssen die verschiedenen Ansätze für das Kantenrouting im verwendeten Algorithmus für die Interaktion beachtet werden.

3.2 Layout-Anpassung beim Öffnen oder Schließen einer Gruppe

Nachdem ein Anfangslayout für die ganze Argumentkarte gefunden wurde, bei der alle Gruppen geschlossen sind, möchte man nun auch Gruppen öffnen. Das Layout für diese Gruppe wurde ebenfalls bereits berechnet. Zur Erinnerung, wenn eine Gruppe geschlossen ist, hat sie eine Größe, die Abhängig von der Größe im offenen Zustand kleiner ist. Da diese Gruppe im geöffneten Zustand nun jedoch mehr Platz einnimmt ergibt sich folgende Problemstellung: Wie verändert sich das Layout auf höheren Stufen, wenn eine Gruppe geöffnet oder geschlossen wird? Hierbei wird gefordert, dass die Änderungen im Layout nicht zu stark sind, damit man sich mit seiner mentalen Karte von vor der Änderung auch danach noch zurechtfindet.

Der von uns vorgeschlagene Lösungsansatz basiert erneut auf einem kräftebasierten Algorithmus. Auch hier verweisen wir auf den oben beschriebenen Algorithmus. Es gibt also zwei Probleme zu lösen. Zum einen, wie groß ist eine Gruppe in ihrem jetzigen Zustand. Zum anderen, wie man dafür sorgt, dass sich das Layout nach Öffnen oder Schließen einer Gruppe nicht zu sehr verändert.

Um die Größe einer Gruppe in ihrem jetzigen Zustand zu berechnen, könnte man verschieden vorgehen. Zum Beispiel könnte man ihn aus den bekannten Radien, für den Zustand alle Untergruppen geöffnet und der Gruppe selbst geschlossen, berechnen. Oder aber man benutzt den Algorithmus aus dem vorherigen Abschnitt, um ihn entweder im Vorraus oder wenn benötigt zu ermitteln. Wird nun als eine Gruppe geöffnet oder geschlos-

sen, wird diese durch eine größere bzw. kleinere ersetzt. Da eine geöffnete Gruppe mehr Platz einnimmt, muss natürlich auch die sie beinhaltende Gruppe größer werden. Die Veränderung zieht sich also bis zur obersten Stufe durch.

Um das Verhalten beim Öffnen oder Schließen einer Gruppe zu kontrollieren erweitern wir den verwendeten kräftebasierten Layoutalgorithmus um mehrere Anker. Jedes gelayoutet Element, was sich also nicht in einer geschlossenen Gruppe befindet, bekommt einen Anker gesetzt und zwar an der Position, an der es sich im Anfangslayout der Gruppe bzw. obersten Stufe befindet. Diese Anker bezeichnen wir als Grundanker. Wenn nun also eine Gruppe geöffnet oder geschlossen wird und die dadurch verursachten Größenänderungen pro Stufe durchgeführt wurden, wird in jeder veränderten Gruppe sowie der obersten Stufe der Layoutalgorithmus mit den zuvor gesetzten Grundankern gestartet. Dies wird durch Abbildung 3.6 veranschaulicht. Die Grundanker sorgen nur dafür, dass die Elemente nicht zu stark von ihrer Position vor der Veränderung verschoben werden.

Genauer zu spezifizieren ist, wie stark die Anker sein sollten sowie ob sie von der Gruppen- bzw. Elementgröße abhängig sein sollten. Hierfür sind erneut eine Implementierung sowie Tests nötig. Dies Wahl könnte natürlich auch sehr von der Argumentkartenstruktur sowie der eigenen Vorliebe abhängen.

Ein weitere Punkte bei diesem Lösungsansatz ist, dass falls die Kräfte der Anker zu stark werden sollten, da durch das öffnen vieler Gruppen eine große Veränderung zum Grundlayout zwingend ist, dass dann die Grundanker ignoriert werden und neue Anker vor der letzten Änderung gesetzt werden. Es wird also nur versucht zum letzten Layout wenig Veränderung zu erhalten, nicht jedoch direkt zu den Anfangslayouts. Dies kann natürlich auch pro Stufe unterschiedlich gehandhabt werden.

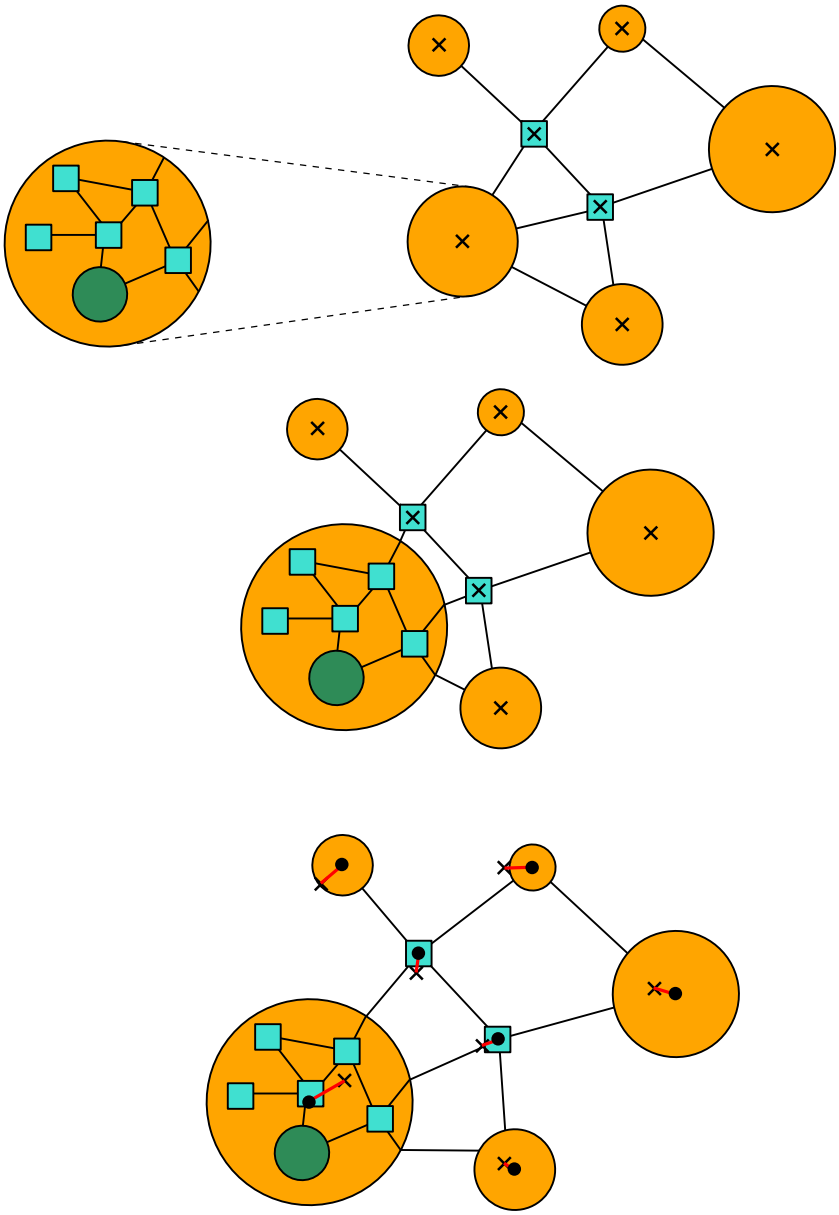

Abbildung 3.6: Veranschaulichung des Ankerprinzips beim Öffnen einer Gruppe.

4 Evaluation und Vergleich zu anderen Lösungsansätzen

Bei der Entwicklung unserer Lösung sind noch andere Ansätze entstanden, welche nach Evaluation mit unserem Ansatz aus Kapitel 3, aber nicht weiter ausgearbeitet oder ausgewertet wurden. Für die Vollständigkeit stellen wir nun noch kurz einen alternativen kräftebasierten Ansatz (siehe Abschnitt 4.1) und einen hierarchischen Ansatz (siehe Abschnitt 4.2) vor. Ein kurzer Vergleich wird in Abschnitt 4.3 gezeigt.

4.1 Uniform kräftebasiertes Layout

Dieser Lösungsansatz stellt eine einfache Erweiterung zu vorhandenen kräftebasierten Algorithmen dar: Knoten können durch zusammenfassende Pseudoknoten in diesem Fall, Gruppenknoten, repräsentiert werden und es werden zusätzlich Anker verwendet.

Konkret bedeutet dies, dass Knoten zunächst nur durch ihre zusammengefassten Pseudoknoten darstellt werden und hierfür mit einem einfachen kräftebasierten Verfahren ein Layout bestimmen, wobei die abstoßende Kraft der Pseudoknoten bzw. die Anziehungskraft zusammengelegter Kanten logarithmisch mit der Anzahl zusammengefassten Elemente skaliert wird.

Wenn nun die Elemente einer oder mehrerer Gruppen konkret dargestellt werden sollen, so werden die Pseudoknoten zunächst ähnlich wie bei der finalen Lösung verankert. Zusätzlich werden die Pseudoknoten deren Elemente dargestellt werden sollen aus dem Layout entfernt und stattdessen die Knoten der Gruppe, die ebenfalls Pseudoknoten für innere Gruppen sein können, eingefügt. Schließlich wird nun noch einmal der kräftebasierte Algorithmus verwendet mit der Besonderheit, dass jeder Knoten eine zusätzliche Anziehung zu seinem Anker bzw. dem Anker des übergeordneten Pseudoknotens.

Dieser Ablauf wird in Abbildung 4.1 skizziert.

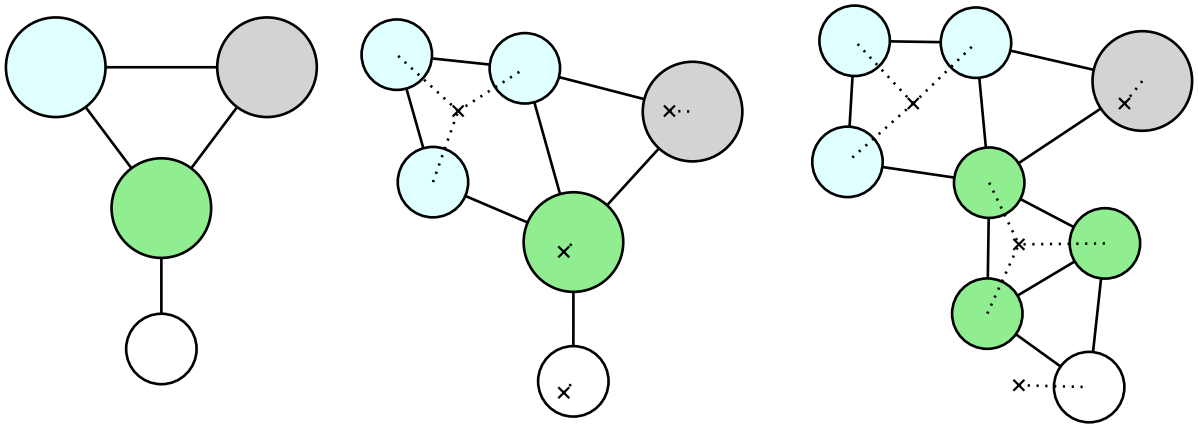

Abbildung 4.1: Skizze eines Layouts durch den uniform krqftebasierten Ansatz. Links zunächst das Layout mit alle Gruppen geschlossen. In der Mitte das Layout nach Öffnen der hellblauen Gruppe. Rechts das Layout mit hellblauer und grüner Gruppe geöffnet. Kreuze markieren hierbei die Anker der Gruppen.

4.2 Hierarchisches Layout

Häufig auftretende Eigenschaften in Argumentkarten wie Zyklenfreiheit legen andere Graphenlayouts, insbesondere hierarchische Layouts, nahe.

Wenn wir die Richtung der Kanten, entgegen der bisherigen Ansätze, für das Layout beachten wollen, wird es dadurch möglich bei einem hierarchischen Layout alle eingehenden Kanten eines Knotens an der Oberseite des Knotens und alle ausgehenden Kanten an der Unterseite des Knotens zu zeichnen. Dadurch wird ebenfalls das „einpacken “ der Knoten in eine Box zur Repräsentation bzw. Ordnung der Gruppe einfacher, da auch die Box der Gruppe die ein- und ausgehenden Kanten entsprechend ordnen kann ohne die Komplexität des Layouts innerhalb der Box zu erhöhen, wodurch das innere und äußere Layout von Gruppen getrennt berechnen werden können.

Ein grobe Skizze zu einem solchen Layout ist in Abbildung 4.2 dargestellt.

4.3 Vergleich

Da keiner der Ansätze konkret implementiert wurde und die alternativen Ansätze auch stückweise ausgearbeitet wurden, besteht der Vergleich die-

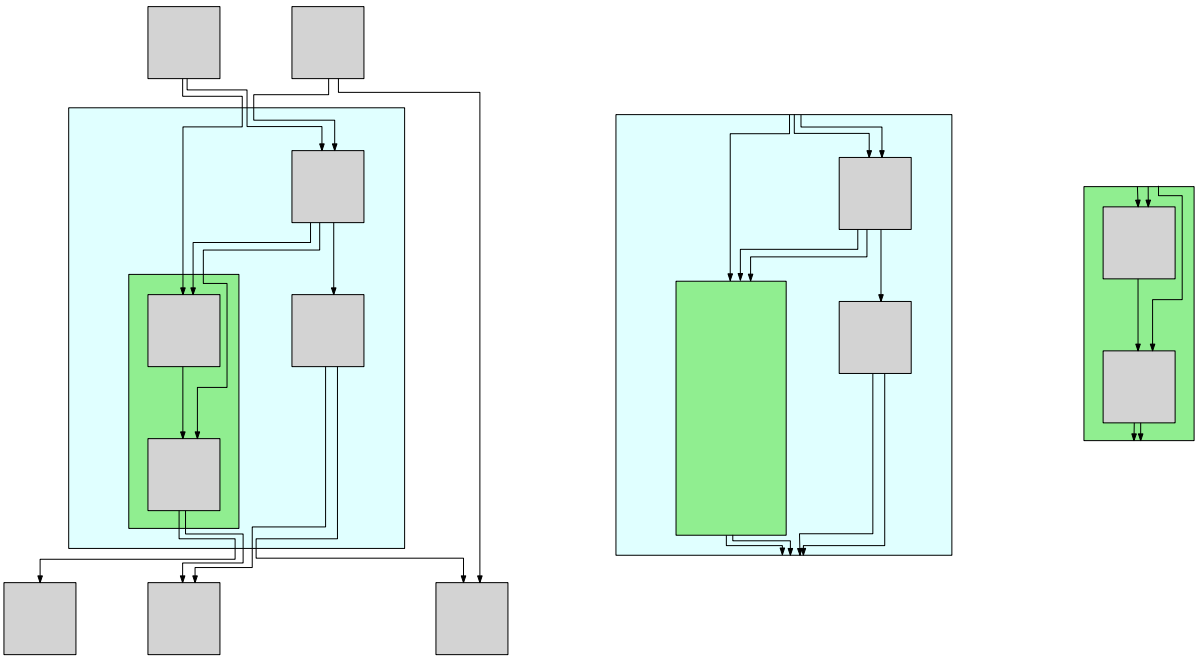

Abbildung 4.2: Skizze eines Layouts durch den hierarchischen Ansatz. Links das Gesamtlayout, Mitte und Rechts jeweils das Layout der jeweiligen Gruppe.

ser Ansätze größtenteils aus Abschätzungen und nicht tatsächlichen Messungen. Es werden folgende Abkürzungen verwendet:

KBK: Unser Lösungsansatz, der kräftebasierte Algorithmus mit Kapselung

HL: Der hierarchische Layout-Ansatz

UKB: Der uniform kräftebasierte Layout-Algorithmus

Fokus des Layouts Bei KBK und HL sind die Layouts der einzelnen Gruppenebenen größtenteils unabhängig, daher liegt der Fokus der Darstellung eher auf der Semantik der Gruppen, d.h. das Layout stellt besonders heraus, zu welcher Gruppe ein Knoten gehört.

Das durch UKB erzeugte Layout basiert dagegen größtenteils auf den Knotenzusammenhängen ab. Durch die Anker bleiben Gruppenelemente zwar größtenteils zusammen, aber das Layout wird dennoch über die Kanten der Knoten in und zwischen Gruppen festgelegt. Bei entsprechenden Zusammenhängen können Gruppen auch deformiert oder getrennt werden (siehe Abbildung 4.3).

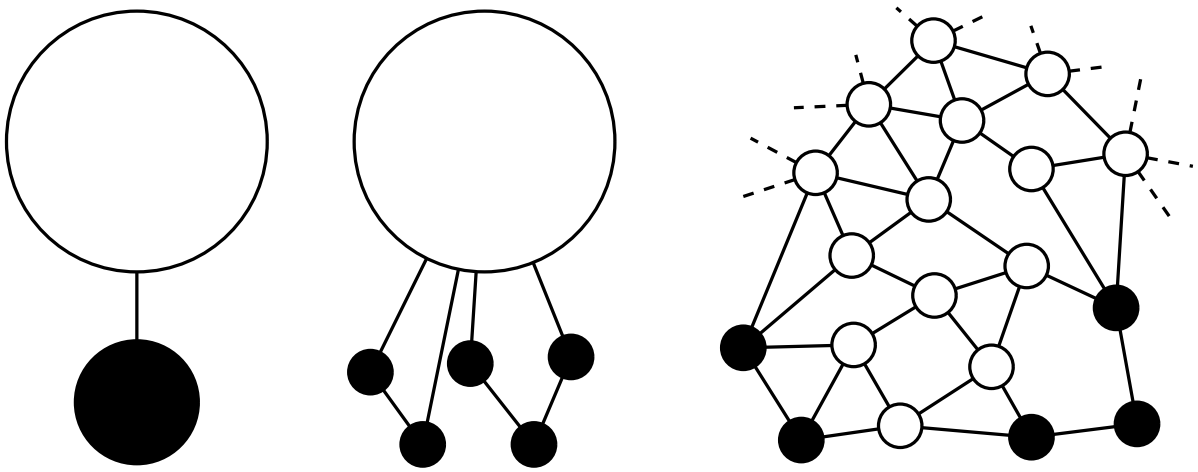

Abbildung 4.3: Beispielhafter Ablauf der zur Trennung einer Gruppe führt: Durch ungünstige Kantenzusammenhänge und einen entscheidenden Größenunterschied drängt sich die weiße Gruppe zwischen die Knoten der schwarzen Gruppe.

Änderungskonstanz Durch die unabhängigen internen und externen Gruppenlayouts bei KBK und HL, sind die Änderungen beim Öffnen und Schließen von Gruppen eher gering: Die Layouts in den Gruppen ist unabhängig davon ob andere Gruppen geöffnet oder geschlossen sind und die relativen Positionen der Gruppen zueinander ändern sich selten.

Da bei UKB die Gruppen nicht „geschirmt “ sind, ist das Layout geöffneter Gruppen sehr stark vom Zustand anderer Gruppen abhängig. Die relativen Positionen bleiben durch die Anker zwar meist erhalten, aber kleine Gruppen sind durchaus anfällig von großen Gruppen deformiert bzw. verdrängt zu werden.

Komplexität des Algorithmus Wie bereits in Kapitel 3 dargestellt, ist der für KBK nötige Algorithmus recht komplex und enthält einige Parameter, die noch bestimmt werden müssen.

Der für HL benötigt Algorithmus benötigt etwas weniger Komplexität als KBK, da die Port für Kanten stets fest sind, wodurch die Größe von geöffneten Gruppen einmalig „Bottom-Up “ bestimmt werden kann ohne Approximationen und Korrekturen. Allerdings besitzt der zugrundeliegende Algorithmus für hierarchisches Layouts eine höhere Komplexität als kräftebasierende Verfahren.

Der UKB-Algorithmus stellt schließlich eine algorithmisch weniger komplexe Erweiterung von einfachen kräftebasierten Verfahren dar, was vermut-

lich zu der geringsten Komplexität unter den hier verglichenen Ansätzen führt.

Anforderung an Graphen Während alle hier vorgestellten Ansätze davon ausgehen, dass jeder Knoten und jede Gruppe maximal einer übergeordneten Gruppe zugehört, werden nur bei HL tatsächliche Anforderungen an die eigentliche Graphenstruktur (Zyklenfreiheit) gestellt um problemlos zu funktionieren.

Kompaktheit Sowohl bei KBK als auch bei HL verliert man durch die Kapselung von Gruppen viel Kompaktheit, da hierbei oft Räume in den Gruppen entstehen, die nicht von anderen Gruppen und Knoten genutzt werden können. Das Gesamtlayout bei KBK ist allerdings bei allen Graphen eher zentriert und rund, da es kräftebasiert bestimmt wird, während HL bei unausgeglichenen Graphen zu besonders hohen bzw. stellenweise sehr breiten Graphen führt.

Das Layout des UKB-Ansatzes führt zu ähnlich kompakten Graphen wie herkömmliche kräftebasierte Algorithmen, da die einzigen zusätzlichen Kräfte, die Anker, an den Positionen der Gruppenknoten gesetzt werden, also in der Hüllkurve des Graphen liegen, und ausschließlich anziehend wirken.

Fazit

Wie man an den Einzelpunkten erkennt, vereint der Ansatz für den wir uns entschieden haben viele der positiven Eigenschaften der beiden anderen Ansätze bzw. stellt einen Kompromiss zwischen den beiden dar.

5 Zusammenfassung und Ausblick

In dieser Arbeit haben wir mit dem Layoutproblem für Argumentkarten beschäftigt, insbesondere bezüglich vordefinierten Gruppen von Argumenten, d.h. Gruppenzugehörigkeiten sind ein Teil des zu zeichnenden Graphen im Gegensatz zu Gruppen, die als Teil des Layouts bestimmt werden (wie bspw. Graph-Clustering).

Unser Lösungsansatz stellt ein Konstrukt dar, dass aufbauend auf gebräuchlichen kräftebasierte Algorithmen, Layouts erstellt, die vordefinierte Gruppen herausstellen. Zusätzlich erlaubt der vorgestellte Ansatz das Anzeigen von Gruppen in geöffneten oder geschlossenen Zuständen. Besonderer Fokus wurde zusätzlich auf das Ästhetikkriterium gelegt, dass das Layout sich durch Öffnen und Schließen der Gruppen nur gering verändert.

Eine konkrete Implementierung und entsprechende Auswertung wurde in dieser Arbeit nicht durchgeführt. Bei einer groben Abschätzung scheint unser Algorithmus keine Operation zu beinhalten, die asymptotisch aufwändiger ist als die quadratische Laufzeit eines gewöhnlichen kräftebasierenden Algorithmus.

Zukünftige Arbeiten können sich mit der in dieser Arbeit unterlassenen Implementation und Auswertung beschäftigen. Ein Teil einer solchen Implementierung sollte es sein, die Parameter, welche bei der Beschreibung des Algorithmus bewusste nicht festgelegt wurden, konkret abzuschätzen.

Alternativ sind auch Ausarbeitungen und Untersuchungen der alternativen Ansätze, die diese Arbeit nur grob umrissen hat, möglich.

Literaturverzeichnis

[HK02] David Harel und Yehuda Koren: *Drawing Graphs with Non-uniform Vertices*. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '02, Seiten 157–166, New York, NY, USA, 2002. ACM, ISBN 1-58113-537-8. http://doi.acm.org/10.1145/1556262.1556288.

[QE01] Aaron Quigley und Peter Eades: *FADE: Graph Drawing, Clustering, and Visual Abstraction*, 2001. http://gdea.informatik.uni-koeln.de/362/, 10.1007/3-540-44541-2_19.