Bachelorarbeit

im Studiengang
Bachelor of Computer Science

Implementierung eines Force-Directed Layouts zur Visualisierung von Graphen in SEE

von

Nina Unterberg

Erstprüfer:	Prof. Dr. rer.nat. Rainer Koschke
Zweitprüfer:	Dr. Sabine Kuske
Betreuer:	Prof. Dr. rer.nat. Rainer Koschke
Eingereicht am:	16. Juni 2020

Erklärung

Ich versichere, die Bachelorarbeit ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Bremen, den 16. Juni 2020

. . . . . .

(Nina Unterberg)

Inhaltsverzeichnis

1 Einleitung

Datengesteuerte, auf Graphen basierende, Informationsvisualisierungssysteme fordern eine Art automatisierten Mechanismus zur Erzeugung geometrischer Darstellungen [1]. Die Vielzahl der denkbar möglichen Visualisierungen werden die Informationen nicht so repräsentieren, dass die Darstellung ihren Zweck erfüllt und den Leser unterstützt das System zu verstehen. Die Herausforderung liegt nun darin, ein automatisiertes Verfahren für den erforderlichen Nutzen zu finden, welches eine geeignete Darstellung generiert und die Lesbarkeit der Graphvisualisierung, sowie die Informationen, die an den Leser übermittelt werden, erhöht.

Die Lesbarkeit eines Graphen wird durch Einhaltung ästhetischer Kriterien positiv beeinflusst. Dazu gehören unter anderem wenige Überkreuzungen der Kanten, die Symmetrie der Zeichnung und wenig Platzverschwendung (vgl. [2, 3]). Ebenso wie die ästhetischen Kriterien müssen auch zuvor festgelegte Konventionen zu der Darstellung des Graphen eingehalten werden. Diese Regeln sind abhängig von der gewählten Umgebung in welcher die Zeichnung genutzt wird, sowie von der Datenstruktur und dem Zweck der Visualisierung (vgl. [2]). Um die aufgeführten Kriterien zu erfüllen, wurden eine Vielzahl von Ansätzen entwickelt. Einer davon ist die *Force-Directed* Visualisierung von Graphen.

Diese Arbeit zielt darauf ab, ein neues Layout für Graphen in das Programm SEE-Software Engineering Experience zu integrieren, welches eine Codecity auf Basis eines Force-Directed Algorithmus generiert. Die durch den Algorithmus entstehende Codecity soll dem Benutzer dabei behilflich sein die Zusammenhänge zwischen den einzelnen Softwarekomponenten besser zu verstehen, symmetrische Strukturen aufzuzeigen und ein ästhetisch ansprechendes Layout zu generieren.

2 Grundlagen

In diesem Abschnitt der Arbeit werden benötigte Grundlagen beschrieben. In Abschnitt 2.1 wird auf Software Visualisierung im Allgemeinen eingegangen. Abschnitt 2.2 gibt eine detailliertere Beschreibung darüber, was eine Graphzeichnung ist. Die Grundsätze einer Graphzeichnung werden in dem folgenden Abschnitt 2.3 erläutert. Da Graphen das grundlegende Konzept in der Arbeit darstellen, wird auf diese in Abschnitt 2.4 näher eingegangen. Ein möglicher Visualisierungsansatz für Graphen ist der Force-Directed-Ansatz, welcher die Grundlage für diese Ausarbeitung darstellt und in Abschnitt 2.5 erklärt wird. Das Layout wird für das Programm SEE implementiert und in dieses integriert. SEE wird in Abschnitt 2.6 und Abschnitt 2.7 genauer erläutert.

2.1 Software Visualisierung

Die Terminologie des Computers und der Software Entwicklung ist bereits reich an Metaphern. Programmiersprachen enthalten Datenstrukturen mit den Bezeichnungen *trees* (dt. Bäume), *queues* (dt. Warteschlangen), *leaves* (dt. Blätter) and *dictionaries* (dt. Wörterbücher) (vgl. [4, S. 2]). Ziel bei dem Einsatz von Metaphern ist es, Assoziationen hervorzurufen, um Konzepte besser zu veranschaulichen und Analogien zu nutzen, damit Strukturen oder Funktionen besser verstanden werden. Menschen nehmen 75% aller Informationen visuell auf (vgl. [4]). Die Darstellung von Daten durch Abbildungen kann von essentiellem Wert für das Verständnis sein.

In der Softwarevisualisierung bedeutet das, die Software visuell durch Abbildungen und Metaphern darzustellen. Dies kann beispielsweise durch ein Knoten-Kanten-Diagramm erfolgen, welches durch Indikatoren für spezifisch auftretende Phänomene innerhalb des Softwaresystems erweitert wird. Die Knoten repräsentieren einzelne Komponenten des Systems, während die Kanten Zusammenhänge aufzeigen.

In der Visualisierung eines Softwaresystems können Merkmale wahrgenommen werden, die in den immateriellen und unsichtbaren Daten verborgen sind oder verloren gehen. Ebenso kann die Darstellung die Art und Weise verbessern, wie über das System gedacht wird (vgl. [4]).

Software Visualisierung soll nicht nur das Verständnis eines Benutzers für die Software fördern, sondern auch die Komplexität eines vorhandenen Softwaresystems reduzieren und eine Steigerung der Produktivität und Qualität anstoßen

(vgl. [4, S. 11]). Um ein gutes Gesamtverständnis zu erzielen, sollten demzufolge alle relevanten Informationen über ein Softwaresystem dargestellt werden.

Stephan Diehl definiert Software Visualisierung als *«die Visualisierung von Artefakten im Zusammenhang mit Software und deren Entwicklungsprozess»* [4, S. 3]. Zu diesen Artefakten gehören neben dem Programmcode beispielsweise Anforderungen und Konstruktionsdokumentation, Änderungen am Quellcode und Fehlerberichte. Diese Vielzahl an Aspekten einer Software hat Diehl in drei Kategorien unterteilt:

Struktur: Die Struktur bezieht sich auf die statischen Teile und Beziehungen des Systems.

Verhalten: Das Verhalten bezieht sich auf die Ausführung des Programms mit realen und abstrakten Daten.

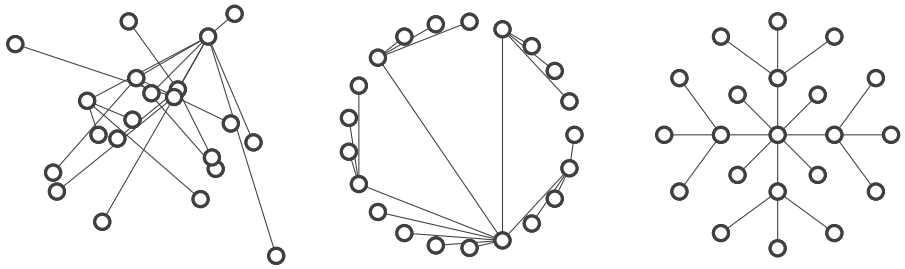
Evolution: Die Evolution bezieht sich auf den Entwicklungsprozess des Softwaresystems.

Dieses Spektrum an Software Artefakten kann keine Visualisierung abdecken. Daher ist es notwendig, eine geeignete Technik für die Darstellung zu wählen. Folgende Fragen können über die Wahl der geeigneten Visualisierung Aufschluss geben: *Warum wird die Visualisierung benötigt? Wer verwendet die Visualisierung? Welche Datenquelle soll dargestellt werden? Wie soll die Datenquelle dargestellt werden? Wo soll die Visualisierung genutzt werden?* (vgl. [5])

2.2 Graphzeichnen

«Graphzeichnen adressiert das Problem der Erstellung geometrischer Darstellungen von Graphen, Netzwerken und verwandten kombinatorischen Strukturen.» [2]. Dabei leitet sich die Verwendbarkeit einer Zeichnung aus der Lesbarkeit ab, beziehungsweise aus der schnellen und klaren Vermittlung ihres Inhalts. Graphzeichnungen sind nur so lange nützlich, wie sie effektiv Informationen an den Leser übermitteln. Für einen Graphen existieren unendlich viele verschiedene Visualisierungen. Dabei ist zu beachten, dass nicht jede Visualisierung für jeden Anwendungsbereich geeignet ist und die gewünschten Informationen übermittelt.

In Abbildung 1 sind drei verschiedene Repräsentationen desselben Graphen abgebildet. Das Layout der Knoten in Abbildung 1b und Abbildung 1c wurden mithilfe des ChiLay Frameworks [6] erstellt. Alle Visualisierungen des Graphen in Abbildung 1 werden durch einen Knoten-Kanten Graphen dargestellt. Knoten-Kanten



(a) (b) (c)

Abbildung 1: Drei verschiedene Visualisierungen eines Graphen als Knoten-Kanten-Diagramm: (a) Zufällige Platzierung der Knoten; (b) kreisförmiges Layout; (c) Spring-Layout

Graphen sind weit verbreitete Graph-Repräsentationen. Der Vorteil dieser Darstellung ist das intuitive Verständnis bei dem Interpretieren der Verbindungen, welche durch gradlinige oder gekrümmt, gerichtete oder ungerichtete Linien repräsentiert werden. In Abbildung 1a wurden die Knoten des Graphen zufällig positioniert. Die Lesbarkeit des Graphen ist durch viele überlappende Knoten und Kanten stark beeinträchtigt, da keine grundlegende Struktur zu erkennen ist. Diese Problematik umgeht das Circular Layout, dargestellt in Abbildung 1b. Knoten werden kreisförmig angeordnet wodurch eine Überlappung von Knoten, sowie die komplette Überdeckung zweier Kanten, vermieden werden kann. Manchmal können globale Eigenschaften des Graphen, wie Symmetrien und Muster des kollektiven Verhaltens, sichtbar werden. Ein Nachteil ist, dass diese Graphen sehr dicht und das Verfolgen von Pfaden schwierig sein kann [7]. In Abbildung 1c wurde ein Spring-Layout verwendet. Hier sollen die Knoten, welche über eine Kante verbunden sind, möglichst nah beieinander platziert, eine einheitliche Kantenlänge anstreben und die symmetrischen Eigenschaften des Graphen hervorgehoben werden. Für den in Abbildung 1 dargestellten Graphen ist das Spring-Layout die intuitiv beste Darstellung. Je nach Anwendungsbereich und Ziel der Visualisierung kann hier jedoch eine andere Visualisierungsart von größerem Vorteil sein.

2.3 Paradigmen für Graphzeichnungen

2.3.1 Parameter für Methoden zur Zeichnung von Graphen

Die erste Eingabe für einen Algorithmus zur Visualisierung eines Graphen ist der Graph selbst. Ebenso kann der Typ des Graphen mit in Bezug genommen werden (vgl. [2]). Der Typ wird durch die kombinatorischen Eigenschaften des Graphen

bestimmt (zum Beispiel gerichtet, azyklisch oder planar). Dieses Wissen kann erforderlich sein, da

- bestimmte Algorithmen zum Graphzeichnen nur auf einer bestimmten Klasse von Graphen angewendet werden können.
- die kombinatorischen Eigenschaften des Graphen hervorgehoben werden sollen.

Der zweite Parameter für den Algorithmus bestimmt sich aus der Beobachtung, dass es *die beste* Graphzeichnung nicht gibt. Menschen unterscheiden sich in ihrer Wahrnehmung und empfinden unterschiedliche Zeichnungen als *die Beste* (vgl. [2]). Hinzu kommt, dass unterschiedliche Anwendungsumgebungen verschiedene Anforderungen haben und somit auch hier keine Graphzeichnung gefunden werden kann, die alle Ansprüche erfüllt. Da dieses genannte Verhalten zu komplex ist, als es mit einem Parameter umschreiben zu können, wurden von Giuseppe di Battista et. in [2] drei Konzepte vorgestellt, die die Anforderungen an eine schöne Graphzeichnung verständlicher beschreiben.

2.3.1.1 Konventionen zur Zeichnung Konventionen zur Zeichnung eines Graphen sind einfache Regeln, die erfüllt werden müssen, damit die Darstellung zulässig ist. Beispiele für Regeln sind:

- C1: *Gradlinige Zeichnung*: Jede Kante wird als eine gerade Linie gezeichnet.
- C2: *Planare Zeichnung*: Keine Kanten dürfen sich überkreuzen.

2.3.1.2 Ästhetik Ästhetische Kriterien beschreiben graphische Eigenschaften der Zeichnung. Durch eine gute Umsetzung der Kriterien wird die Lesbarkeit eines Graphen positiv beeinflusst. Verbreitete ästhetische Kriterien (vgl. [2, 4]) sind:

- A1: Minimierung der *Überkreuzungen* von Kanten
- A2: Minimierung des *Bereichs*, den die Zeichnung einnimmt
- A3: Minimierung der *Summe der Länge* aller Kanten
- A4: Minimierung der *maximalen Länge* einer Kante
- A5: Minimierung der *Varianz der Kantenlängen*
- A6: Darstellung der *symmetrischen Eigenschaften* eines Graphen

- A7: Minimierung von *Kantenverkrümmungen*

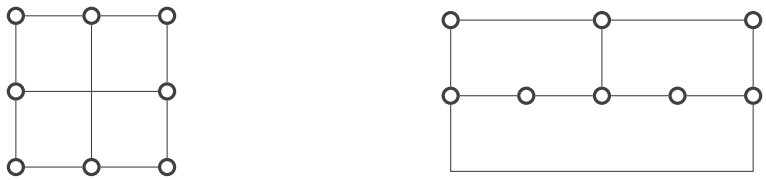
Den positiven Einfluss von einer verringerten Anzahl von Kantenüberkreuzungen, sowie der Minimierung von Kantenkrümmungen konnte von H. Purchase et al. in [3] gezeigt werden.

2.3.1.3 Einschränkungen Während sich die Konventionen zur Zeichnung und die Ästhetik auf den gesamten Graphen und die gesamte Zeichnung beziehen, gelten die Einschränkungen nur für spezifische Teilgraphen und Teilzeichnungen. Beispiele:

- R1: *Zentrum:* Platziere einen gegebenen Knoten im Zentrum der Zeichnung.
- R2: *Cluster:* Platziere eine Teilmenge der Knoten nah beieinander.
- R3: *Form:* Zeichen einen Teilgraph als eine festgelegte Form.

2.3.2 Vorrang zwischen ästhetischen Kriterien

Ästhetische Kriterien geraten oft in Konflikt miteinander. Selbst, wenn die Kriterien nicht in Konflikt geraten würden, wäre es algorithmisch in den seltensten Fällen möglich alle Kriterien gleichzeitig einzuhalten. Deswegen ist es notwendig festzulegen, welche ästhetische Kriterien Vorrang haben.



(a) (b)

Abbildung 2: Zwei orthogonale Zeichnungen des gleichen Graphen auf einer Gitterstruktur: (a) mit einer minimalen Anzahl an Kantenkrümmungen; (b) mit einer minimalen Anzahl an Kantenüberkreuzungen [2, S. 18]

In dem in Abbildung 2a dargestellten Graphen wurde dem Kriterium der minimalen Anzahl von Kantenkrümmungen Vorrang vor dem Kriterium der minimalen Anzahl von Kantenüberkreuzungen gewährt, in Abbildung 2b andersherum. Keine Zeichnung kann beide Kriterien gleichermaßen erfüllen. Verschiedene Methodiken, die entwickelt wurden, um Graphen zu zeichnen, geben verschiedenen ästhetischen Kriterien Vorrang. Eine Vorrangsbeziehung ist für eine bestimmte Anwendung passend, während sie für eine andere weniger geeignet ist.

2.4 Graphen

Graphen repräsentieren Informationen, welche als Objekte und als Beziehungen zwischen den Objekten modelliert werden können (vgl. [2]). Formal ist ein Graph G ein Paar $(V(G), E(G))$, bestehend aus einer endlichen Menge von Knoten $V(G)$ (engl. vertices) und einer von $V(G)$ abgetrennten Menge $E(G)$, welche die endliche Menge der Kanten (engl. edges) beschreibt (vgl. [8, S. 2]). Eine Kante wird als $e \in \{\{u, v\} \mid u \in V, v \in V\}$ definiert. Ein Knoten $v \in V$ ist ein Mitglied (engl. member) von Graph G, falls $G = (V, E)$. Der Graph G ist der Besitzer (engl. owner) von Knoten $v \in V$ oder Kante $e \in E$. Ein Knoten $v \in V$ wird als inzident mit einer Kante e bezeichnet, falls $e = \{u, v\}$ und zwei Knoten v, v' werden adjazent bezeichnet, falls $\{v, v'\} \in E$.

2.4.1 Compound Graphen

Ein Compound Graph (dt. zusammengesetzter Graph) ist ein Tripel $C = (V, E, F)$. Das Paar (V, E) beschreibt einen Adjazenzgraphen bestehend aus einer Menge von Knoten V und einer Menge von Kanten E (vgl. Abschnitt Graphen). Das Paar $T = (V, F)$ beschreibt den Inklusionsgraphen von G. Es ist erforderlich, dass T ein gewurzelter Baum (engl. rooted tree) ist (vgl [9]). Beispiel:

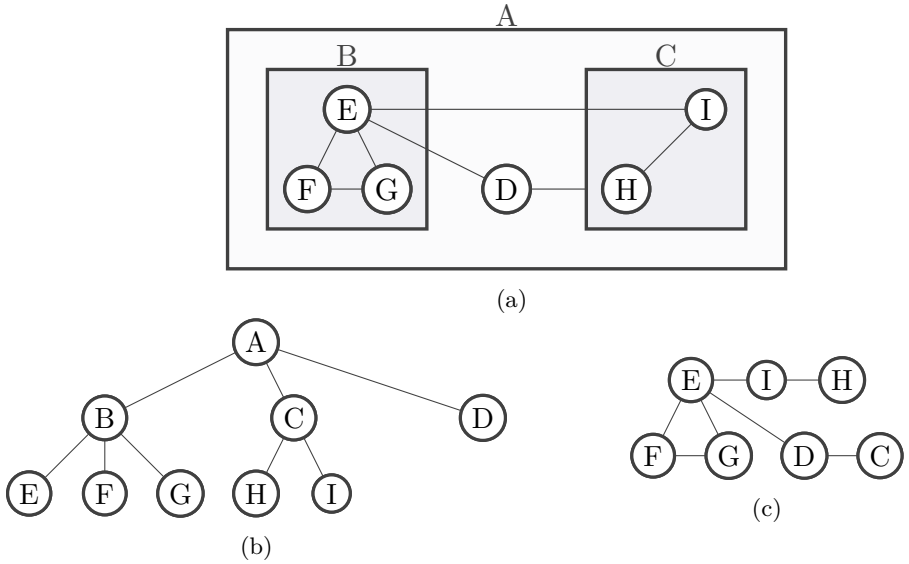


Abbildung 3: Ein Compound Graph: (a) der Compound Graph $C = (V, E, F)$ [9]; (b) der Inklusionsgraph $T = (V, F)$; (c) der Adjazenzgraphen $G = (V, E)$

Die erläuterten Definitionen sind in Abbildung 3 anhand eines Beispiels erklärt. Abbildung 3a zeigt einen Compound Graphen, Abbildung 3b den dazu gehö-

renden Inklusionsgraph und Abbildung 3c visualisiert den Adjazenzgraphen. Die formale Definitionen für den in Abbildung 3 dargestellten Compound Graphen lauten [9]:

$$V = \{a, b, ..., i\}$$

$$E = \{\{c, d\}, \{e, d\}, \{e, i\}, \{e, f\}, \{e, g\}, \{f, g\}, \{i, h\}\}$$

$$F = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, e\}, \{b, f\}, \{b, g\}, \{c, h\}, \{c, i\}\}$$

2.4.1.1 Konventionen, Ästhetik und Einschränkungen für Compound Graphen

Es wird folgend davon ausgegangen, dass ein Compound Graph durch ein Knoten-Kanten-Diagramm dargestellt wird. Die Konventionen, ästhetischen Kriterien und Einschränkungen orientieren sich an den Ausarbeitungen [10,11].

Konventionen zur Zeichnung:

- C3: *Inklusion*: Eine Inklusionskante $(u, v) \in F$ wird durch den vollständigen geometrischen Einschluss der zugehörigen Form von u in die Form von v gekennzeichnet.
- C4: *keine Inklusion*: Knoten ohne eine Inklusionsbeziehung überlappen sich nicht.

Regeln einer Zeichnung: Zusätzlich zu den angestrebten ästhetischen Kriterien für Graphzeichnungen, sollte bei der Visualisierung von Compound Graphen beachtet werden:

- A8: *Positionierung*: Knoten, welche durch Kanten verbunden sind, sollten so nah wie möglich beieinander platziert werden.

Wie in Vorrang zwischen ästhetischen Kriterien erklärt, können nicht alle ästhetischen Kriterien gleichermaßen berücksichtigt werden und eine Vorrangsbeziehung muss während des Visualisierungsprozesses bestimmt werden.

Einschränkungen

- R4: *Größe*: Die Größe von Knoten, welche andere Knoten enthalten, sollte an den Inhalt angepasst werden.

2.5 Force-Directed Visualisierung

Force-Directed Ansätze nutzen physikalische Analogien um Graphen zu zeichnen. Ein Graph wird als ein System von Körpern und Kräften betrachtet, die zwischen diesen Körpern wirken. Der Algorithmus sucht daraufhin eine Konfiguration der Körper mit minimaler lokaler Energie, also eine Position für jeden Körper, sodass die Summe der Kräfte, die auf jeden Körper wirken, gleich null ist (vgl. [2]). Es gibt verschiedene Methoden eine Force-Directed Visualisierung zu erzeugen. Grundsätzlich besteht der Ansatz aus zwei Komponenten: einem physikalischen Modell und einem Algorithmus, der nach einem ausgeglichenen Zustand des Kräftesystems sucht. Das Modell codiert die ästhetischen Kriterien. Die Kräfte sind so definiert, dass die Gleichgewichtskonfiguration eine ansprechende Zeichnung erzeugt (vgl. [2]).

2.5.1 Fruchterman/ Reingold Spring Embedder

Fruchterman und Reingold entwickelten einen Force-Directed Algorithmus (kurz FR) mit dem Ziel eine ästhetisch ansprechende, zweidimensionale Zeichnung zu generieren. Diese Zeichnung ist auf ungerichtete Graphen mit geraden Kanten ausgelegt. Der Algorithmus strebt danach die folgenden ästhetischen Kriterien zu erfüllen: eine einheitliche Kantenlänge, eine gleichmäßige Verteilung der Knoten und die symmetrischen Eigenschaften eines Graphen hervorzuheben (vgl. [12]).

Der Spring Embedder basiert auf dem Spring Embedder Modell von Eades (vgl. [12, 13]). Die Grundidee von Eades war es, alle Knoten eines Graphen durch Stahlringe and alle Kanten durch Sprungfedern zu ersetzen, um ein physikalisches System zu simulieren. Die Knoten werden mit einem initialen Layout ausgelegt und daraufhin von den Kräften der Sprungfedern bewegt. Das System gelangt dadurch zu einem Zustand von minimaler Energie. Dabei stoßen sich die Stahlringe ab, während die Sprungfedern Anziehungskräfte auf die Ringe ausüben.
Eine andere Vorarbeit stammt von Kamada und Kawai (vgl. [12, 14]), welche ebenfalls auf Eades basiert. Eades berücksichtigte in seinem Modell nur den idealen Abstand zwischen zwei adjazenten Knoten. In Erweiterung zu Eades fügten Kamada und Kawai das Konzept eines idealen Abstandes zwischen zwei nicht benachbarten Knoten hinzu: Der ideale Abstand zwischen zwei nicht adjazenten Knoten ist proportional zu dem kürzesten Weg zwischen ihnen (vgl. [15]).

Wie Kamada und Kawai berücksichtigen Fruchterman und Reingold den idealen Abstand zwischen jedem Paar aus Knoten während des Layoutprozesses. Sei k

die optimale Distanz zwischen jedem Knotenpaar. Der ideale Abstand wird wie folgt berechnet:

$$k = C * \sqrt{\frac{area}{|V|}} \tag{1}$$

Die Konstante C wurde durch Fruchterman und Reingold experimentell ermittelt, $area$ ist der Bereich des kleinsten Rechtecks, das die Zeichnung abdeckt und $|V|$ ist die Anzahl der Knoten. Somit gibt k den idealen Radius eines leeren Kreises um einen Knoten an.
Sei d_p der Abstand zwischen dem Knotenpaar $P = (u, v)$ und $f_a/\ f_r$ die Anziehungs- (engl. attractive force) und Abstoßungskräfte (engl. repulsion force) berechnet auf den Knoten u und v. f_a und f_r werden für P wie folgt berechnet:

$$\begin{aligned} f_a(d_p) &= (d_p)^2/k \\ f_r(d_p) &= -k^2/d_p \end{aligned} \tag{2}$$

Die Anziehungskraft f_a zwischen zwei benachbarten Knoten ist proportional zu dem Abstand zwischen ihnen. Andersherum ist die Abstoßungskraft f_r antiproportional zu dem Quadrat der Distanz.
Diese Formeln wurden auf Grundlage von Experimenten aufgestellt (vgl. [12]). Ebenso ähneln sie den von Hooke's Law (vgl. [16, S. 81]).

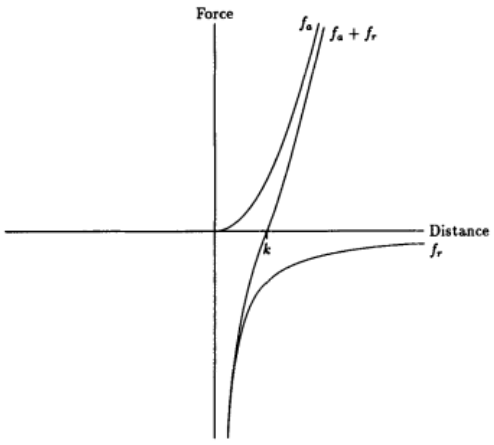


Abbildung 4: Fruchterman und Reingold: Abstoßungs- und Anziehungskräfte im Vergleich zur Distanz [12]

Der Punkt k, an dem die Summe der Anziehungskräfte und der Abstoßungskräfte die x-Achse auf dem Funktionsgraphen in Abbildung 4 überkreuzen, ist exakt der Punkt, an dem die beiden Kräfte sich gegenseitig aufheben. Der ideale Abstand zwischen zwei Knoten ist ebenfalls durch den Punkt k gegeben.

Der Algorithmus basiert auf zwei Prinzipien: Knoten, die durch eine Kante verbunden sind, sollen nah beieinander platziert werden. Knoten sollten dennoch

nicht zu nah an anderen Knoten platziert werden (vgl. [12]).

Die Position der Knoten wird iterativ verändert. Jede Iteration besteht aus drei Schritten (vgl. [15]). Zu Beginn wird der Effekt der Abstoßungskräfte für jedes Knotenpaar bestimmt. Daraufhin wird der Effekt der gegenseitigen Anziehung unter den Knoten berechnet, die durch eine Kante verbunden sind (vgl. [15]). Der Spring Embedder versucht nun das globale Energieniveau durch Bewegung der Knoten in die Richtung der Kräfte zu minimieren. Die Knoten werden, unter Berücksichtigung beider berechneter Effekte, neu positioniert. Die Bewegung der Knoten ist durch einen maximalen Wert begrenzt. Dieser Wert nimmt ab, wenn die Temperatur des Systems abkühlt. Die Knoten bewegen sich langsamer, wenn das System einen stabileren Zustand erreicht. Die Repositionierung erfolgt solange, bis die Gesamtenergie unter einen festgelegten Schwellwert fällt, an dem alle Knoten so platziert wurden, dass sie einen nahezu idealen Abstand zueinander haben.

2.5.2 Fruchterman/ Reingold Spring Embedder Varianten

Um die Laufzeit des vorgestellten Algorithmus zu verbessern, variierten Fruchterman und Reingold den Spring Embedder. Diese Variante unterteilt den Bereich der Zeichnung in ein quadratisches Raster [12]. In jeder Iteration wird jeder Knoten in einem Feld des Rasters platziert und die Abstoßungskräfte zwischen diesem und jedem Knoten in benachbarten Feldern berechnet:

$$\begin{aligned} f_r &= \frac{k^2}{d} u(2k - d) \text{ where} \\ u(x) &= \begin{cases} 1, & \text{if } x > 0. \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{3}$$

Um eine Verfälschung der Berechnung durch ungleichmäßige Abstände zu den Rändern der quadratischen Felder zu umgehen, wurde eine gleichbleibende Distanz in alle Richtungen eingeführt. Die Abstoßungskräfte werden für alle Knoten innerhalb eines Kreises mit Radius $2k$, zentriert am Knoten, berechnet.

Wie in Abbildung 5 zu sehen, wird die Abstoßungskraft zwischen v und q berechnet, von v und r/s jedoch nicht, da diese Knoten nicht innerhalb des Kreises liegen. Sei W die Breite und L die Länge des Bereiches der Zeichnung, dann

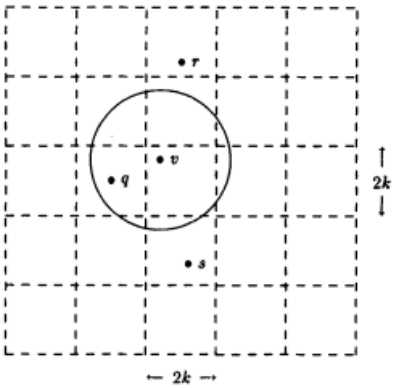


Abbildung 5: Fruchterman und Reingold: Berechnung der Abstoßungskräfte auf Grundlage eines quadratischen Rasters [12]

berechnet sich die Area A für jeden Knoten wie folgt:

$$\begin{aligned} A &= \frac{WL}{|V|} \\ k &= \sqrt{A} \end{aligned} \tag{4}$$

Die Anzahl der Rasterfelder ergibt sich aus:

$$\text{Anzahl} = \frac{W}{2k}\frac{L}{2k} = WL\frac{|V|}{4WL} = \frac{|V|}{4} \tag{5}$$

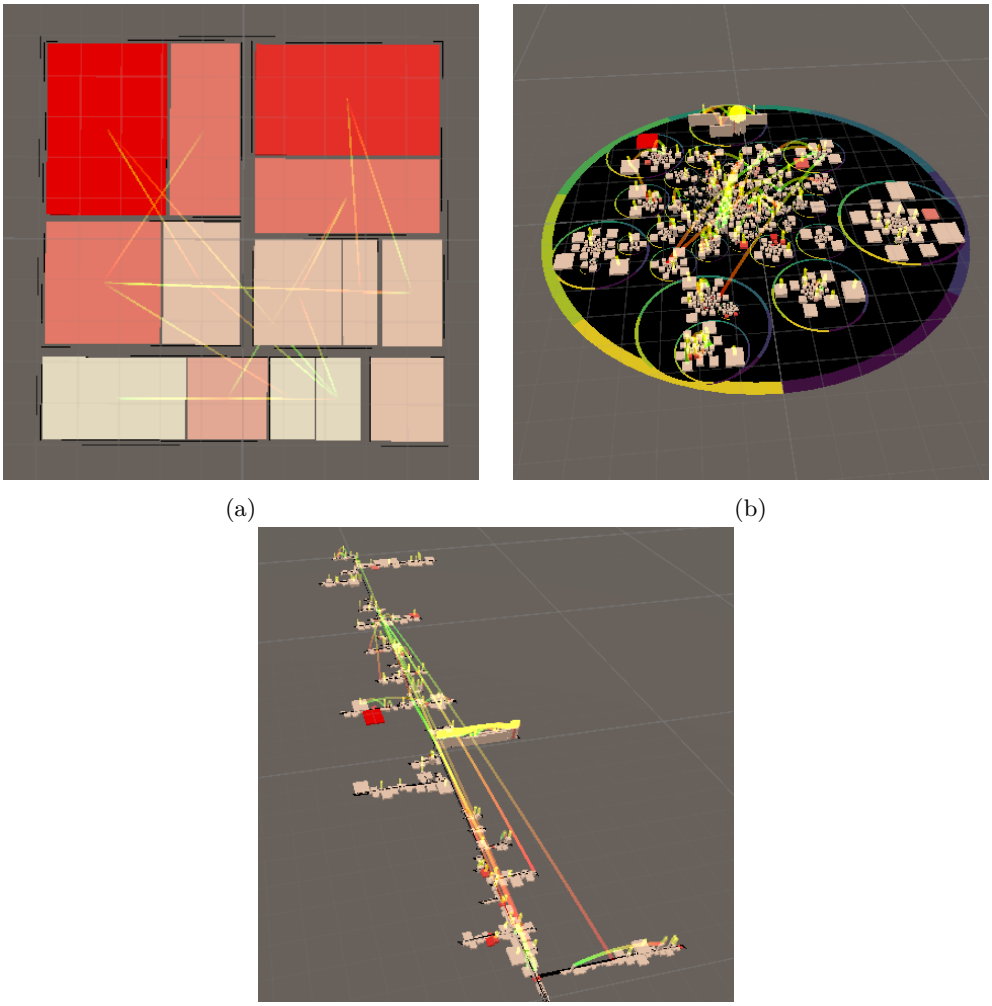
Um die Ergebnisse des Algorithmus noch weiter zu verbessern, kann die Abkühlungsstrategie (eng. cooling schedule) für die maximale Verschiebung eines jeden Knotens innerhalb einer Iteration verbessert werden. Fruchterman und Reingold kamen zu den besten Ergebnissen, wenn der maximale Wert in der ersten Phase bei einer hohen Temperatur startet, dann stetig, schnell abkühlt und in der zweiten Phase auf einer konstant niedrigen Temperatur bleibt [12].

2.6 Codecity

Eine Codecity (dt. Software Stadt) soll den Benutzer dabei unterstützen, die sonst immaterielle Software durch grafische Darstellung besser zu überschauen. In einer Codecity werden die einzelnen Komponenten (z.B. Klassen) meist als Häuser und Verzeichnisse als Bereiche dargestellt. Der Aufbau der Stadt hängt von der gewählten Visualisierung ab. Das Layout einer Codecity ist essentiell für das Verständnis des Benutzers. Eine gute grafische Darstellung hilft dem Benutzer das System zu verstehen, eine schlechte hingegen wirkt verwirrend und irreführend.

2.7 SEE- Software Engineering Experience

SEE - SOFTWARE ENGINEERING EXPERIENCE ist ein Programm, welches darauf abzielt, ein Software System als Codecity in der virtuellen Realität darzustellen. SEE soll dabei dem Benutzer nicht nur die Struktur einer Software näher bringen, sondern auch den Zustand und die Evolution des Softwaresystems. Durch Analyse des Quellcodes werden bestehende Codesmells gefunden und durch zusätzliche grafische Indikatoren, wie Icons und Kanten, in die Codecity integriert. Beispiele sind in Abbildung 6 aufgezeigt.



(a) (b)

(c)

Abbildung 6: Verschiedene Codecities aus SEE: (a) Treemap-Layout; (b) Circle-Packing-Layout; (c) Evo-Street-Layout

3 Entwurf

Im folgendem Abschnitt werden die getroffenen Entwurfsentscheidungen erläutert. Zu Beginn wird in Kapitel 3.1 auf die Anforderungen an das Layout eingegangen. Der Entscheidungsprozess wird in Kapitel 3.2 dargestellt. Im letzten Kapitel des Abschnitts wird detailliert auf den gewählten Layoutalgorithmus eingegangen.

3.1 Anforderungen an das Layout

Das Ziel dieser Ausarbeitung ist es, eine neue Visualisierung in das Programm SEE zu integrieren. Das Layout muss folgende Kriterien erfüllen:

- K1: Die Visualisierung soll durch ein *Force-Directed* Algorithmus erzeugt werden.
- K2: Der hierarchische Aufbau von Softwaresystemen soll in der entstehenden Softwarestadt sichtbar sein.
- K3: Kanten zwischen Knoten, welche nicht im gleichen Zweig der Hierarchie liegen, sollen möglich sein.
- K4: Kanten sollen auch zwischen inneren Knoten der Hierarchie möglich sein.
- K5: Die Kanten sollen gerade sein.
- K6: Sowohl gerichtet als auch ungerichtet verlaufende Kanten sollen möglich sein.

3.2 Wahl der Visualisierungsmethode

Bei *Force-Directed* Algorithmen werden Kräfte auf die Menge der Kanten und auf die Menge der Knoten eines Graphen angewandt. Eine Darstellung des Graphen durch ein Knoten-Kanten-Diagramm ist naheliegend. Die Abbildung der hierarchischen Struktur (K2) setzt eine gegebene Hierarchie, zum Beispiel in Form eines Inklusionsbaum (engl. inclusion tree), voraus. Einen Graphen mit verschachtelten Ebenen und zulässigen Kanten zwischen Knoten in verschiedenen Zweigen des Inklusionsbaumes (K3, engl. intergraph edges) nennt man Compound Graphen. Hierarchisch verschachtelte Graphen, welche nur Kanten zwischen Blättern des

Inklusionsbaumes zulassen, heißen hierarchisch geclusterte Graphen (engl. hierarchical clustered graphs) (vgl. [17, S. 196]).
Um Compound Graphen darzustellen und eine ansprechende Zeichnung als Endergebnis zu erhalten, sollten die Konventionen/ ästhetischen Kriterien und Einschränkungen, welche Compound Graphen mit sich bringen, eingehalten werden. Diese wurden in Kapitel 2.4.1.1 erläutert.
Folgende Kriterien werden den Anforderungen an das Layout hinzugefügt:

- K7: Die Visualisierung des Graphen soll ein Knoten-Kanten-Diagramm sein.
- K8: Die Graph-Klasse des Algorithmus soll ein Compound Graph sein.
- K9: Die Konventionen/ ästhetischen Kriterien und Einschränkungen von Compound Graphen sollen eingehalten werden.

Viele mögliche Visualisierungen von Compound Graphen sind eingeschränkt durch den Typ der Graphen, auf welche das Layout ausgerichtet ist (beispielsweise geclusterte Graphen [18–21]). Die Auswahl an Algorithmen, welche auf geclusterten Graphen arbeiten, ist deutlich größer als jene, die auf Compound Graphen ausgerichtet sind. Aus dieser Beobachtung sind zwei Ansätze zur Visualisierung von Compound Graphen entstanden. Diese werden im weiteren Verlauf vorgestellt.

3.2.1 Mehrdimensionaler Ansatz

Da viele Algorithmen nur auf bestimmten Typen von Graphen arbeiten, kann versucht werden, den Compound Graphen auf einen geclusterten Graphen zu reduzieren und die verloren gegangenen Informationen auf andere Art und Weise darzustellen. Ein mögliche Methode dafür ist der Multilevel Visualisierungsansatz. Veröffentlichungen zu diesem Themengebiet wurden beispielsweise von Eades und Feng erarbeitet [22]. Die Grundidee hinter dem mehrdimensionalen Ansatz ist es, Graphen im dreidimensionalen Raum über verschiedene Schichten darzustellen. Die Schichten stellen unterschiedliche Abstraktionsgrade des Graphen dar. Jede Schicht in dem dreidimensionalen Raum entspricht einem Level der Hierarchie aus dem Inklusionsbaum des Compound Graphen und wird entsprechend des Levels auf dessen Z-Koordinate gezeichnet. Eingeschlossene Knoten werden über eine Kante, welche zwischen zwei Ebenen verläuft, mit ihren Eltern-Knoten verbunden.

Um K1 zu entsprechen und einen Force-Directed Algorithmus zu verwenden, könnten die Positionen der Knoten auf den einzelnen Schichten durch einen Force-Directed Algorithmus für geclusterte Graphen berechnet werden. Ein mögliches

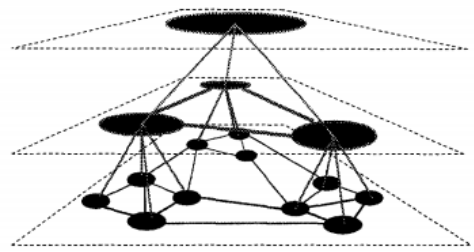


Abbildung 7: Multilevel Visualisierung von geclusterten Graphen [22]

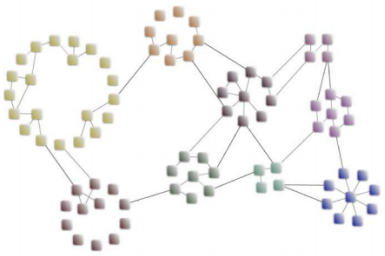


Abbildung 8: CiSE: Circular Spring Embedder [21]

Vorgehen ist zum Beispiel die Verwendung des CiSE: Circular Spring Embedder Algorithmus [21]. Eine Beispielvisualisierung ist in Abbildung 8 zu sehen. K2, K3, K5, K6 und K7 werden entweder durch den Multilevel Visualisierungsansatz oder durch das CiSE-Layout erfüllt. Um K4 und K8 umzusetzen, könnten zusätzliche Kanten zwischen den Knoten der einzelnen Schichten zugelassen werden. K9 wird teilweise erfüllt. A8 wird mit wenig Priorität betrachtet, da Kanten zwischen Compound Knoten und/oder Blatt-Knoten im Nachhinein eingefügt und nicht direkt über den Algorithmus betrachtet werden. C3 ist nicht erfüllt, da die Inklusion nur durch eine Kante und nicht durch Einschluss dargestellt wird. Ein weiterer Nachteil dieses Ansatzes ist die mangelhafte Übersichtlichkeit in der virtuellen Realität, welche in SEE genutzt werden kann. Ein Vorteil ist, dass die verschiedenen Schichten getrennt betrachtet werden können und somit eine höhere visuelle Abstraktion des Graphen möglich ist.

3.2.2 Force-Directed für ungerichtete Compound Graphen

Auch wenn die Auswahlmöglichkeit durch wenige zur Verfügung stehende Algorithmen eingeschränkt ist, ist ein weiterer möglicher Ansatz einen existierenden Force-Directed Algorithmus zu wählen, welcher auf Compound Graphen ausgelegt ist. Für die Umsetzung dieser Methode wurde das Layout CoSE : Compound Spring Embedder [9] gewählt. Dieses basiert auf traditionellen Force-Directed Algorithmen mit Erweiterungen für Verschachtelung auf beliebig vielen Ebenen, variierender Knotengröße und mit der Möglichkeit andere anwendungsspezifische Einschränkungen zu integrieren. Darüber hinaus sind Kanten zwischen beliebigen Verschachtelungsebenen und zwischen inneren Knoten in der Verschachtelungshierarchie möglich (vgl. [9]). Das CoSE-Layout unterstützt somit K1, K2, K3, K5, K6, K7 und K8. Die Erfüllung der genannten Kriterien wird auch anhand der Abbildung 9 sichtbar.

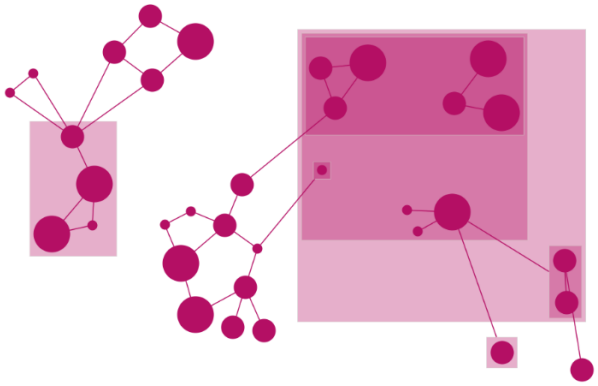


Abbildung 9: Erstellt mit Cytoscape.js Demo für CoSE : Compound Spring Embedder [23]

Das Kriterium K4 wird ebenfalls größtenteils durch den CoSE Algorithmus abgedeckt. Eine Einschränkung existiert jedoch: das CoSE-Layout lässt keine Kanten zu, welche einen Knoten mit einem seiner Vorgänger oder Nachfolger aus dem Inklusionsbaum verbindet, weil dies zu Problemen mit dem Kräftemodell führen kann (vgl. [9]). Ein möglicher Lösungsansatz ist es, diese Kanten aus dem Layoutprozess auszuschließen und sie erst nach dem erfolgreichen Berechnen des Layouts wieder einzufügen. C3, C4 und A8 aus K9 werden durch das Kräftesystem realisiert. Wenn die Kräfte, abhängig von dem Graphen, geeignet spezifiziert sind, dann werden diese erfüllt. R4 wird durch CoSE umgesetzt.

3.2.3 Fazit

Für die Force-Directed Visualisierung von Compound Graphen wird das CoSE -Layout gewählt. Dieses erfüllt nahezu alle aufgestellten Kriterien. Im Gegensatz zu dem multidimensionalen Ansatz ist die Darstellung übersichtlicher und für Personen in der virtuellen Realität besser erfassbar. Eine mögliche Erweiterung ist es, die beiden Ansätze zu kombinieren. Hierbei kann der Vorteil des Multilevel Ansatzes, die höhere Abstraktionsmöglichkeit, mit eingebracht werden. Diese Variante wird in dieser Ausarbeitung jedoch nicht betrachtet, da es den zeitlichen Aufwand der Arbeit überschreiten würde.

3.3 CoSE-Layout

Die Grundlage für das CoSE-Layout (vgl. [9]) ist eine Veröffentlichung von Fruchterman und Reingold: *Graph drawing by force-directed placement* [12] und eine Vorarbeit dieser von Eades [13]. Diese Verfahren wurden bereits in Kapitel 2.5.1

erläutert. Die Definition für Compound Graphen erschließt sich aus dem Kapitel 2.4.1. Hinzu kommt die Einschränkung, dass das CoSE-Layout keine Kante zulässt, welche einen Knoten mit einem seiner Vorgänger oder Nachfolger verbindet (vgl. [9])).

Im Gegensatz zu dem FR-Layout und dem Layout von Eades, abstrahiert das CoSE-Layout die Berechnung des idealen Abstandes und der Abstoßungskraft nicht auf die Anzahl der Knoten und der zur Verfügung stehenden Fläche, sondern setzt auf Eingaben des Benutzers. Gründe dafür könnten sein, dass eine unbegrenzte Fläche zur Verfügung steht, die Größe der Knoten variieren darf und Compound-Knoten visualisiert werden. Durch diese Erweiterungen ist die von Fruchterman und Reingold vorgestellte Formel zur Berechnung des idealen Abstandes zwischen allen Knoten nicht mehr übertragbar.

3.3.1 Graphmanager

Ein Graph wird für das CoSE-Layout durch einen Graphmanager [24] abgebildet (vgl. [9]). Die Topologie eines Compound Graphen wird in einzelne Graphen aufgespalten, welche ineinander verschachtelt sind. Ein Graphmanager ist ein Tripel $M = (S, I, F)$. $S = \{G_1, G_2 ... G_i\}$ bildet eine Menge von Graphen ab, I beschreibt die Menge der Intergraph-Kanten und $F = \{V^F, E^F\}$ ist ein verwurzelter (verschachtelter) Baum, welcher auch der Navigationsbaum genannt wird (vgl. [24]).

Graphen werden in ihre übergeordneten Knoten verschachtelt. Dadurch wird das Zeichnen mehrerer Graphen und das ihrer Wechselbeziehungen erleichtert. Diese Beziehung wird über eine Kante zwischen dem Knoten n und dem Graph G_i im Navigationsbaum ausgedrückt. G_i darf dabei nicht der Besitzer von n sein. G_i kann als der Kind-Graph (engl. child graph) des Eltern Knoten (engl. parent node) n bezeichnet werden. Außerdem kann n als *erweitert* betitelt werden. Ein erweiterter Knoten ist so groß wie sein Kind-Graph.

Beziehungen zwischen Knoten unterschiedlicher Graphen werden Intergraph-Kanten genannt. Sei $u \in V^{G_i}$, $v \in V^{G_j}$, $i \neq j$ und $G_i, G_j \in S$. Dann existiert eine Intergraph-Kante $(u, v) \in I$.

Um den Aufbau eines Graphmanagers zu verdeutlichen, ist die Topologie des Compound Graphs aus Kapitel 2.4.1 in Abbildung 10 als Graphmanager beschrieben.

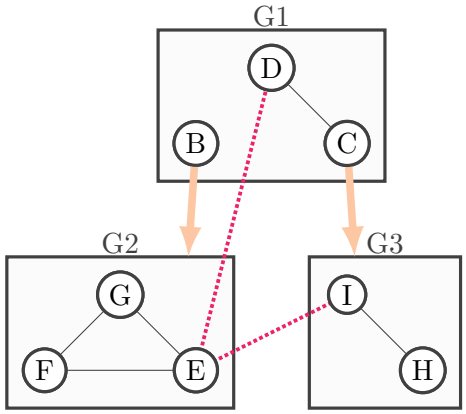


Abbildung 10: Der Graphmanager für den Compound Graphen aus Kapitel 2.4.1. Die dicken Pfeile zeigen die Inklusionsbeziehungen und die gepunkteten Kanten sind die Intergraph-Kanten.

3.3.2 Physikalisches Modell

Die folgenden Abschnitte basieren auf [1, 9, 24].

3.3.2.1 Anziehungs- und Abstoßungskräfte Die Knoten werden als physikalische Objekte mit einer elektrischen Ladung betrachtet, welche über Sprungfedern mit einer spezifizierten idealen Länge verbunden sind. Abhängig von der aktuellen Länge der verbundenen Kanten ziehen sich die Knoten an oder stoßen sich ab. Außerdem wirken Abstoßungskräfte auf jedes Paar von zu nah beieinander stehenden Knoten, um Überlappungen zu vermeiden. Dabei gilt, dass Knoten sich nur abstoßen, wenn sie Mitglied des gleichen Graphen sind.

3.3.2.2 Variierende Knotengröße Um eine variierende Knotengröße zu unterstützen und das Überlappen von benachbarten Knoten zu verhindern, basiert die Berechnung der Kantenlänge auf dem Teil der Kante, welcher zwischen den Grenzen der Endknoten liegt.

3.3.2.3 Verschachtelte Graphen Ein erweiterter Knoten wird mit seinem verschachtelten Graphen (im folgenden Verlauf auch Compound Knoten genannt) als eine Einheit beziehungsweise bildlich, als eine Karte (engl. cart) betrachtet, welche sich frei in orthogonale Richtungen bewegen kann. Knoten und Kanten des verschachtelten Graphen dürfen sich nur innerhalb der Grenzen des Graphen bewegen. Es wird angenommen, dass eine Karte aus einem speziellen Material besteht, welches elastisch genug ist, sich an die Grenzen des verschachtelten

Graphen anzupassen. Sollte ein Knoten aus dem verschachtelten Graphen seine Position nach außen oder innen verschieben, passen sich die Grenzen des Compound Knotens entsprechend an.

3.3.2.4 Gravitationszentren Zusätzlich wirken Gravitationskräfte, um die einzelnen Graphen zusammenzuhalten. Jeder verschachtelte Graph besitzt ein Gravitationszentrum, welches die Kindknoten in Richtung der Mitte des Graphen zieht. Die Stärke der Gravitationskräfte ist unabhängig von der Größe des Knotens und von der Distanz zwischen dem Knotenzentrum und der Mitte des verschachtelten Graphen. Gravitationskräfte sind im Vergleich zu den Kräften der Sprungfedern und der Abstoßungskräfte schwach.

3.3.2.5 Intergraph-Kanten Kanten zwischen verschiedenen Graphen werden gesondert behandelt. Der Teil einer Intergraph-Kante e, falls vorhanden, von dem Endknoten u, welcher Mitglied des Graphen G_u ist, zu der Grenze des Graphen G_u, wird durch eine konstante Kraft, ähnlich der Gravitationskraft, modelliert. Der Knoten u wird dadurch so nah wie möglich an der Grenze von G_u gehalten. Der verbleibende Teil einer Intergraph-Kante wird durch die Kräfte der Sprungfedern repräsentiert. Um die Rechenleistung so niedrig wie möglich zu halten, kann die spezifizierte ideale Länge einer Intergraph-Kante variiert werden. Dabei erhöht sich die Länge der Kante proportional zur Summe der Tiefe der Endknoten, die gemeinsame Vorgänger im Verschachtelungsbaum besitzen.

Um das Kräftemodell zu verdeutlichen, ist dieses in Abbildung 11 dargestellt.

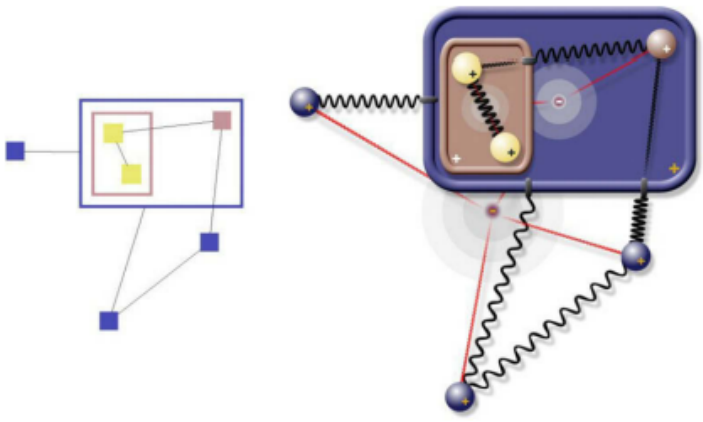


Abbildung 11: Kräftemodell des CoSE-Layout [9]

4 Implementierung

In dem folgenden Kapitel wird eine Beschreibung der Implementierung des CoSE-Layouts aufgeführt. Zu Beginn wird die Umsetzung des Basisalgorithmus erklärt. Anschließend wird in Abschnitt 4.2 auf die Ausarbeitung der anwendungsspezifischen Einschränkungen eingegangen. Im vorletzten Abschnitt dieses Kapitels wird die Implementierung der Features beschrieben. Am Ende werden Beispiele des entstandenen Layouts vorgestellt.

Die Implementierung erfolgte in der Programmiersprache C-Sharp (C#). Große Teile der Implementierung des Basisalgorithmus und der Features wurden dem CHILAY-Projekt [6] entnommen und von der Programmiersprache Java in C# übertragen. Davon ausgenommen sind die Features: Kennzahlen, Parameterauswahl und iterative Berechnung der Parameter. Im Folgenden wird auf eine individuelle Referenzierung auf das CHILAY-Projekt verzichtet. Der globale Zusammenhang mit dem CHILAY-Projekt wird hiermit betont.

4.1 Basisalgorithmus

Um das CoSE-Layout auszuführen wird ein `CoseLayout` Objekt erstellt und der Layoutprozess über die Funktion `Layout()` gestartet. Zu Beginn wird ein `GraphManager` Objekt erstellt und alle Graphen, Knoten und Kanten entsprechend der in das Layout hinein gegebenen Graphstruktur erzeugt (s. Kapitel 3.3.1). Die erstellten Objekte sind Instanzen der Klassen `CoseGraph`, `CoseNode` und `CoseEdge`. Diese Klassen halten alle Werte der Objekte sowie beispielsweise die Größe, Position und alle Kräfte, wie die Abstoßungs-/ und Anziehungskräfte. Nachdem die Topologie erstellt wurde, wird die Initialisierung gestartet. Während des Initialisierungsprozesses werden die niedrigsten gemeinsamen Vorgänger der Start-/Endknoten der Kanten, die erwartete Größe der Knoten und die ideale Kantenlänge aller Kanten, sowie die Knoten, auf welche Gravitationskräfte wirken, berechnet. Des Weiteren werden Werte wie die maximale Verschiebung von Knoten und die maximale Anzahl von Iterationen bestimmt.

Daraufhin startet das Berechnen des Layouts. Der Layoutprozess wird solange fortgeführt, bis die Anzahl der maximalen Iterationen erreicht ist. Ein weiteres Abbruchkriterium besteht, wenn die Gesamtverschiebung der Knoten während einer Iteration einen vorher festgelegten Grenzwert unterschreitet oder die Gesamtverschiebung sich im Vergleich zu der vorherigen Iteration nicht mehr entscheidend verändert.

Als nächstes wird die Temperatur des Systems berechnet. Dieser Wert wirkt sich auf die maximale Verschiebung der Knoten innerhalb einer Iteration aus. Die gewählte Abkühlungsstrategie wird in Kapitel 4.3.4 erklärt. Daraufhin werden die Methoden `CalculateSpringForce()`, `CalculateRepulsionForce(nodeA, nodeB)` und `CalcGravitationalForce()` ausgeführt. Die wichtigsten Aspekte der Methoden sind im folgenden Pseudocode beschrieben.

Die Methode `CalculateSpringForce()` wird für jede Kante ausgeführt. Die Zeitkomplexität für die Berechnung der Anziehungskräfte ist $\mathcal{O}(|E|)$, da jeder Schritt innerhalb der Methode `CalculateSpringForce` in $\mathcal{O}(1)$ Schritten ausführbar ist.

```
function CalculateSpringForce() begin
    springforce = (length - idealEdgeLength) * Spring_Strength;
    springforceX = springforce * (lengthX / length);
    Calculate springforceY equivalent to springforceX;
    Add the springforces to the source node springforces;
    Substract springforces from the target node springforces;
end
```

Abbildung 12: CalculateSpringForce()

```
function CalculateRepulsionForce(nodeA, nodeB) begin
    Calculate distanceX and distanceY from the clipping points of nodeA
     and nodeB;
    if distanceX < minDistance then
        distanceX = Math.Sign(distanceX) * minDistance;
    end
    if distanceY < minDistance then
        distanceY = Math.Sign(distanceY) * minDistance;
    end
    distanceSquared = distanceX * distanceX + distanceY * distanceY;
    distance = Mathf.Sqrt(distanceSquared) # Length of distance vector;
    repulsionforce = Repulsion_Strength * nodeA.NoOfChildren *
     nodeB.NoOfChildren / distanceSquared;
    repulsionforceX = repulsionforce * distanceX / distance;
    repulsionforceY = repulsionforce * distanceY / distance;
    Substract repulsionforces from nodeA repulsionforces;
    Add repulsionforces to nodeB repulsionforces;
end
```

Abbildung 13: CalculateRepulsionForce(nodeA, nodeB)

`CalculateRepulsionForce(nodeA, nodeB)` wird für jedes Paar von Knoten auf-

gerufen, welche Kindknoten vom selben Graphen sind. Der Zeitaufwand für die Berechnung der Abstoßungskräfte ist $\mathcal{O}(|V|^2)$. Wurde vor dem Layoutprozess das Feature *Smart Repulsion Range* (die Rastervariante) ausgewählt, weicht die Berechnung von der hier darstellten Methode ab. Die Zeitkomplexität verringert sich auf $\mathcal{O}(|V|)$.

```
function CalculateGravitationalForce() begin
    distanceX = centerposition.X - ownerCenterX;
    distanceY = centerposition.Y - ownerCenterY;
    estimatedSize = ownerGraph.estimatedSize *
     Compound_Gravity_Range_Factor;
    if distanceX > estimatedSize || distanceY > estimatedSize then
        node.gravitationForceX = -Gravity_Strength * distanceX *
         Compound_Gravity_Strength;
        node.gravitationForceY = -Gravity_Strength * distanceY *
         Compound_Gravity_Strength;
    end
end
```

Abbildung 14: CalculateGravitationalForce()

Die Gravitationskräfte werden nur für die Knoten von Graphen berechnet, welche nicht zusammenhängend sind. Für diese Knoten wird die Methode `CalculateGravitationalForce()` jeweils einmal ausgeführt. Der Pseudocode beschränkt sich auf die Darstellung der Berechnung der Gravitationskräfte, die auf alle inneren Knoten wirken. Des Weiteren werden spezielle Gravitationskräfte auf die Knoten ausgeübt, welche Kinder des Wurzelgraphen sind. Der Zeitaufwand für die Berechnung der Gravitationskräfte ist $\mathcal{O}(|V|)$.

Nach der Berechnung der Kräfte werden die Knoten bewegt und die Methode `updateBounds()` aufgerufen, welche die Größe und Position der Graphen entsprechend der neuen Position der Kindknoten aktualisiert.

Am Ende des Layoutprozesses werden die neu berechneten Positionen und Größen auf die übergebenen Objekte übertragen.

4.2 Anwendungsspezifische Einschränkungen

Eine Einschränkung, welche durch die Anwendungsumgebung SEE gegeben ist, ist die Integration der Möglichkeit der Auswahl von Sublayouts für Teilbäume

der hierarchischen Struktur eines Graphen. Außerdem wurde versucht die inneren Knoten ellipsenförmig darzustellen, um die Wahrnehmung und Ästhetik der Visualisierung weiter zu verbessern (vgl. [25]).

4.2.1 Sublayout

Sei $M = (S, I, F)$ ein Graphmanager, dann kann jeder Graph $G_i \in S$ durch ein von CoSE abweichendes Layout dargestellt werden, ein sogenanntes Sublayout. In der SEE GUI kann der Benutzer für jeden dieser Graphen ein Sublayout, sowie die Darstellungsform der inneren Knoten auswählen. Als Sublayout stehen alle Layouts zur Verfügung, welche bereits in SEE integriert sind, wie beispielsweise das Treemap-Layout oder das CirclePacking-Layout. Manche Layouts sind nur auf Graphen anzuwenden, die ausschließlich aus Blattknoten bestehen, da diese keine inneren Knoten visualisieren. Für jedes Sublayout kann festgelegt werden, welche Art von inneren Knoten zulässig ist und welche anderen Layouts in dem Sublayout als Sublayouts eingefügt werden dürfen.
Während der Implementierung wurde versucht, die Struktur der Sublayout-Klassen so generisch wie möglich zu halten, damit andere Layouts auch auf diese Struktur zurückgreifen können.

Im Folgenden Pseudocode werden die benötigten Schritte zur Berechnung der Knoten eines Sublayouts, sowie die Schritte, welche nach einer Sublayoutberechnung ausgeführt werden, beschrieben. Der Pseudocode beschreibt die signifikantesten Codeanweisungen.

```
function CalculateNodesForSublayouts() begin
    Result: A list of sublayouts and their nodes
    rootList = List with all sublayout root nodes;
    Sort the list in descending order depending on the level of the nodes in
     the hierarchy tree;
    foreach root in rootList do
        Add all child nodes of the root node to the sublayout;
        if A child node is already part of another sub-layout then
            removedChildren.Add(child node)
            Remove this node from the list of sub-layout nodes;
        end
    end
end
```

Abbildung 15: CalculateNodesForSublayouts()

Vor der Berechnung eines Sublayouts werden diesem alle Kindknoten wieder hinzugefügt, welche einen Wurzelknoten eines unterliegenden Sublayouts bilden. Da die Sublayouts entsprechend der Reihenfolge ihrer Tiefe im Hierarchiebaum gelayoutet werden, liegen die Wurzelknoten bereits in der Größe des unterliegenden Sublayouts vor und werden dem überliegenden Sublayout als einfache Blattknoten hinzugefügt.

```
function CalculateSublayouts() begin
    Result: A calculated sublayout
    sublayout = calculate sublayout;
    foreach layoutNode in sublayout do
        Set associated node to layoutNodes position, scale and rotation;
        if layoutnode is root of a another sublayout then
            foreach sublayoutNode in layoutNode.Sublayout.sublayoutNodes
             do
                Apply rotation, set the node relative to the root node of the
                 current sublayout and add this node to the current
                 sublayouts node list;
            end
        end
    end
    if !sublayout.NodeLayout.InnerNodesEncloseLeafNodes then
        Set the scale and position of the root node so that it encloses all
         sublayout nodes. Save the old scale / position for later;
    end
    Set all sublayout nodes relative to the root node;
end
```

Abbildung 16: CalculateSublayouts()

4.2.2 Kreisförmige Compound Knoten

In dem CoSE-Layout werden innere Knoten visuell durch Rechtecke dargestellt. Um die Visualisierung des Graphen weiter zu verbessern, wurde überlegt, die inneren Knoten als Ellipsen darzustellen, da diese Form generell als ästhetischer angesehen und angenehmer wahrgenommen wird (vgl. [25]).

Der Algorithmus des CoSE-Layouts geht grundsätzlich von rechteckig geformten inneren Knoten aus. Eine Möglichkeit ellipsenförmige Compound Knoten zu unterstützen, ohne die Grundstruktur des Layoutalgorithmus zu verändern, ist das Einfügen von inneren Rändern zu den Compound Knoten. Eine Darstellung dieser Idee ist in Abbildung 17 zu sehen. Das gräulich gefärbte Rechteck ist die Fläche von einem inneren Knoten, in der Kindknoten platziert werden können.

Der orange gefärbte Rand ist der zusätzliche Platz, der benötigt werden würde, um die Fläche des gräulich gefärbte Rechtecks von einer Ellipse zu umschließen. Der innere Knoten könnte nun als Ellipse gezeichnet werden, ohne zu riskieren, dass Kindknoten nicht mehr durch die Ellipse umschloßen werden.

Da dieses Feature jedoch nicht effizient ist, wurde es nicht eingebaut. Der Flächeninhalt eines inneren Knoten würde sich zu sehr vergrößern und statt einer verbesserten Darstellung das Gegenteil bewirken. Im Beispiel in Abbildung 17 vergrößert sich der Flächeninhalt von ≈ 6 Einheiten^2 auf 12 Einheiten^2.

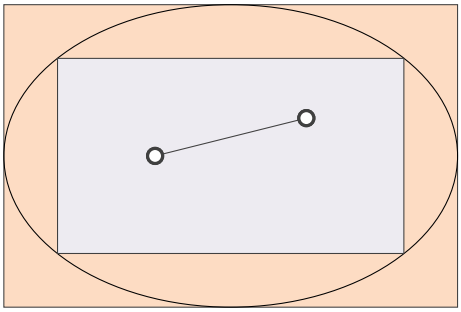


Abbildung 17: Darstellung einer möglichen Umsetzung von ellipsenförmigen Compound Knoten

4.3 Features

Für den CoSE-Algorithmus gibt es Erweiterungsvorschläge, welche darauf abzielen die Laufzeit und die Ergebnisse des Layouts zu verbessern.

4.3.1 Multilevel-Scaling

Für das CoSE-Layout wird ein Multilevel-Scaling Algorithmus von Walshaw vorgeschlagen [26, 27]. In diesem Prozess werden Knoten zu Gruppen zusammengefügt, welche ein Cluster bilden. Die Cluster werden genutzt, um einen neuen Graphen zu definieren. Das Vorgehen wird iterativ wiederholt, bis die Größe des Graphen unter einen festgelegten Schwellwert fällt. Der gröbste (engl. coarest) Graph wird als initiales Layout ausgelegt und verfeinert, bis der originale Graph dargestellt wird. Damit wird die Geschwindigkeit das Layout zu erstellen und die Qualität des Layouts erhöht (vgl. [27]).

Walshaw kombiniert in seinem Ansatz Force-Directed Visualisierung mit der Multilevel-Scaling Technik. Hierbei setzte er auf das Prinzip der Vergröberung

(engl. coarsening approach) und auf eine Variante der von Hendrickson / Leland vorgeschlagenen Heuristik der Kantenkontraktion (vgl. [27, S. 3]).

Dieser Ansatz beruht auf der Methode immer zwei Knoten zu einem Cluster zusammenzufassen. Dabei wird eine zufällig angeordnete Liste aller Knoten erstellt und jeder Knoten nacheinander besucht, wobei jeder nicht zusammengefasste (engl. matched) Knoten mit einem nicht zusammengefassten benachbarten Knoten (oder mit sich selbst) gepaart wird. Die Entscheidung, welche benachbarten Knoten zusammengefasst werden, kann zufällig sein. Um jedoch den gröberen Graphen so einheitlich wie möglich zu halten, wird ein Knoten immer mit dem Nachbarn mit dem geringsten Gewicht gematcht.

Sobald der gröbste Graph gefunden ist, startet der Layoutprozess. Der Graph wird durch das Layout ausgelegt. Die verfeinerten Graphen werden daraufhin so positioniert, dass die Knoten eines Clusters immer auf der Position des Clusters positioniert werden, welcher sie repräsentiert. Der Layoutprozess endet, wenn der originale Graph ausgelegt wurde. Um den idealen Abstand k an die Größe des vergröberten Graphen anzupassen, setzt Walshaw k relativ zu dem derzeitigen Graph G_l. Dabei seien $l, L \in \mathrm{N}$, $l < L$ und G_L der originale Graph:

$$k_l = \sqrt{4/7} * k_{l+1} \tag{6}$$

Außerdem schlägt Walshaw vor, falls der Multilevel-Scaling Ansatz mit der Rastervariante eines Force-Directed Algorithmus kombiniert wird, dass die Größe des Kreises mit dem Radius R, welcher den Raum bestimmt in dem die Abstoßungskräfte eines Knoten zu anderen Knoten berechnet wird, entsprechend der folgenden Formel (vgl. [27]) verändert wird:

$$R_l = 2(l + 1) * k_l \tag{7}$$

Das Multilevel-Scaling wurde entsprechend des von Walshaw aufgezeigten Vorgehens umgesetzt. Ebenso wurde die Abänderung der Berechnung innerhalb der Rastervariante übernommen. Die Berechnung der relativen Kantenlänge wurde in das Feature *Smart Multilevel-Scaling* ausgelagert. Diese Entscheidung wurde getroffen, da die Topologie eines Graphen nicht immer viele Vergröberungen zulässt und das Anpassen der Kantenlänge damit nicht immer notwendig ist.

In dem folgenden Pseudocode ist das Grundgerüst für den Multilevel-Scaling

Algorithmus beschrieben.

```
function MultiLevelScaling() begin
    Result: A layout calculated with multilevel-scaling
    List<GraphManager> gmList = graphmanager.coarsenGraph();
    settings.level = gmList.count -1 ;
    if LayoutSettings.smartMultilevelScaling then
        Calculate smart ideal edge length for multilevel scaling;
    end
    while settings.level >= 0 do
        graphManager = gmList[settings.level];
        classicLayout();
        if settings.level >= 1 then
            uncoasen();
        end
        settings.level–;
    end
end
```

Abbildung 18: MultiLevelScaling()

Ob sich die Laufzeit oder die Ergebnisse durch das Feature Multilevel-Scaling verbessert haben, wurde im Rahmen dieser Ausarbeitung nicht untersucht. Jedoch wurde in [26] gezeigt, dass sich das Multilevel-Scaling positiv auf die Qualität des Layouts und die Laufzeit des Algorithmus auswirkt. Da große Teile der Implementierung von dem Projekt Chilay [6] übernommen wurden, welches ebenfalls in [26] genutzt wurde, ist anzunehmen, dass die Ergebnisse übertragbar sind.

4.3.2 Rastervariante

Die Rastervariante des Fruchterman/ Reingold Spring Embedders wurde bereits im ersten Abschnitt in Kapitel 2.5.2 erläutert. Die Berechnung des Radius des Kreises, innerhalb dessen die Abstoßungskräfte zu anderen Knoten berechnet werden, richtet sich in der Umsetzung nach der Formel, welche in Kapitel 4.3.1 beschrieben wurde. Die Berechnung von k, wie in Kapitel 2.5.2 dargestellt, wurde nicht umgesetzt. Wie bereits für das CoSE-Layout erläutert wurde, wird die ideale Kantenlänge, sowie die Stärke der Abstoßungskräften durch den Benutzer über die Benutzeroberfläche eingegeben.

4.3.3 Smarte Berechnung der Kantenlänge

In dem Feature *Smarte Berechnung der Kantenlänge* (engl. smart edge length calculation) wird die ideale Kantenlänge einer Intergraph-Kante um die geschätzte Größe der Graphen, in welchem die Start-/ Endknoten (SourceInLca, TargetInLca) der Kante liegen, erweitert. Die Idee zielt darauf ab, die Überlappung der Graphen durch eine zu große Anziehungskraft (engl. springforce) zu verhindern. Die Formel beschreibt sich wie folgt:
$idealEdgeLength+ = sizeOfSourceInLca+sizeOfTargetInLca-2*NodeSize$

Um die Begriffe *SourceInLca* und *TargetInLca* zu verdeutlichen, sind diese in Abbildung 19 dargestellt.

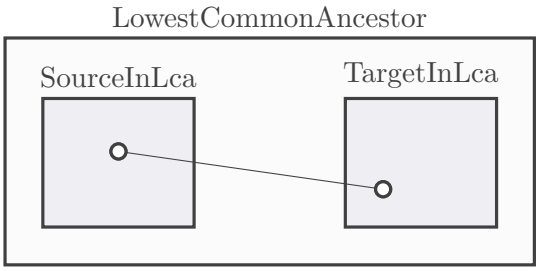


Abbildung 19: Niedrigster gemeinsamer Vorgänger (engl. Lowest Common Anchestor)

4.3.4 Abkühlungsstrategie

Die Idee eine Abkühlungsstrategie in einen Force-Directed Algorithmus zu integrieren, welche die maximale Verschiebung eines jeden Knoten innerhalb einer Iteration limitiert, wurde von Fruchterman und Reingold in [12] eingeführt.

In [28] wurde eine Abkühlungsstrategie (engl. Cooling Strategy) für den CoSE Algorithmus vorgeschlagen, welche an die folgende Formel in Abbildung 20 angelehnt ist. Diese Berechnung der Abkühlungsstrategie wurde aus [28] übernommen. Mit verschiedenen Strategien wurde nicht experimentiert.

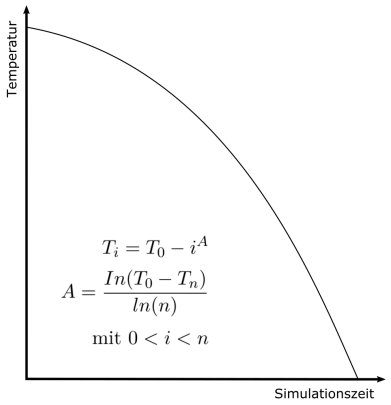


Abbildung 20: Abkühlungsstrategie [29]

4.3.5 Kennzahlen

Die Berechnung von Kennzahlen zu einem entstandenen Layout kann über die SEE GUI angewählt werden.
Folgende Kennzahlen können während des Layoutprozesses erhoben werden:

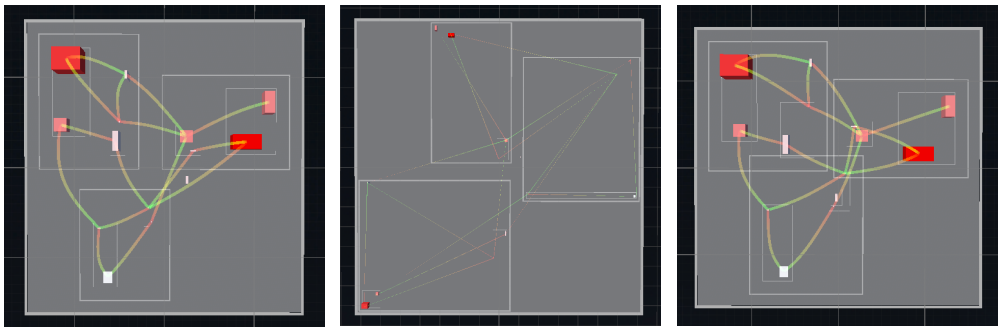
- Fläche des Layouts
- Anzahl der sich überlappenden Blatt-/ inneren Knoten
- Anzahl der Kantenüberkreuzungen
- Durchschnitt der Kantenlänge
- Längste/ kürzeste/ totale Kantenlänge
- Durchschnitt der Kantenlänge relativ zu der Fläche des Layouts
- Längste/ kürzeste/ totale Kantenlänge relativ zu der Fläche des Layouts
- Standardabweichung und Varianz der Kantenlänge
- Standardabweichung und Varianz der Kantenlänge relativ zu der Fläche des Layouts
- Benötigte Zeit, um den Layoutalgorithmus auszuführen

Diese sollen den Benutzer dabei unterstützen, die Qualität der entstandenen Visualisierung besser einschätzen zu können. Darüber hinaus kommen die erhobenen Kennzahlen in der Evaluation des CoSE-Layouts zum Einsatz.

Die Berechnung der Kennzahlen der Kantenlängen beruht auf der Annahme, dass die Kanten gerade sind. Die Art der Visualisierung der Kanten ist davon unabhängig.
Die Anzahl der Kantenüberkreuzungen wird auf die Berechnung, ob sich zwei Geraden in einem 2D Raum schneiden, reduziert. Der gewählte Algorithmus stammt ursprünglich von F. Antonio [30]. Die Implementierung wurde aus [31] übernommen.

4.3.6 Parameterauswahl

Das entstehende Layout ist abhängig von einer Vielzahl zuvor festgelegter Konstanten. Dazu zählen zum Beispiel die Kantenlänge, die Abstoßungskraft, die Gravitationskonstante und die Stärke der Anziehungskräfte. Über die SEE GUI kann der Benutzer die Werte für eine Teilmenge dieser Konstanten festlegen. Die entstehende Visualisierung, sowie die Qualität des Layouts, sind demzufolge abhängig von den Eingaben des Benutzers. Wählt der Benutzer passende Werte, wird die Visualisierung den Ansprüchen des Benutzers genügen. Werden die Konstanten mit unvorteilhaften Werten belegt, wird die Visualisierung kein zufriedenstellendes Ergebnis liefern. Abhängig von der Größe eines Graphen kann der Benutzer viel Zeit damit verbringen, die Werte der Parameter iterativ zu wählen und zu verändern.



(a) (b) (c)

Abbildung 21: Verschiedene Visualisierungen durch das CoSE-Layout. Die Layouts unterscheiden sich durch die Werte Kantenlänge und Abstoßungskraft.

Die Layouts in Abbildung 21 stellen denselben Graphen dar. Die Visualisierungen unterschieden sich nur in der Wahl der Parameter Kantenlänge und Abstoßungskraft. Abbildung 21a zeigt eine gute Visualisierung des Graphen. Die Knoten überlappen sich nicht und die Fläche des Layouts ist nicht zu groß. In Abbildung 21b überlappen sich ebenfalls keine Knoten, jedoch steht die Fläche der Visualisierung in keinem guten Verhältnis zur Knotengröße. Die dritte Abbildung

zeigt zwar einen Graphen mit einer guten Größe, dafür überlappen allerdings die Knoten.

Um den Benutzer bei der Belegung der Konstanten zu unterstützen, sollen die Werte zuvor so bestimmt werden, dass die Auswahl ein gutes Layout erzeugt.

Die Parameter so zu belegen, dass das best mögliche Layout erzeugt wird, ist nicht möglich, da es sich um ein NP-hartes Optimierungsproblem handelt. Daher muss ein Approximationsalgorithmus herangezogen werden, welcher die Parameter für ein möglichst gutes Layout liefert.

Kriterien, um die Qualität einer Visualisierung zu beurteilen, wurden in Kapitel 2.4.1.1 vorgestellt. Die zwei Kriterien, welche die Qualität des Layouts am offensichtlichsten negativ beeinflussen, sind die Größe der eingenommenen Fläche und die Überlappung von Knoten. Da es dennoch sehr schwer festzustellen ist, wann ein Layout gefunden wurde, welches den Graphen möglichst gut darstellt, wurde von Approximationsverfahren wie *hill climbing* (dt. Bergsteigeralgorithmus) abgesehen.

Vielmehr wurde sich auf die Abhängigkeiten der Parameter von Eigenschaften eines Graphen konzentriert. Um die Vorhersage der Konstanten möglichst aussagekräftig zu halten, wurden diese minimiert und sich nur auf die Kantenlänge und den Wert der Abstoßunsgkraft fokussiert.

Dazu wurden 505 zufällige Graphen erzeugt. Die Anzahl der Blattknoten der Graphen liegt in dem Bereich 1-50, die Anzahl der inneren Knoten zwischen 1-10 und die Dichte der Kanten zwischen Blattknoten liegt zwischen dem Wert 0.001 - 0.05. Diese Beschränkungen wurden gewählt, da die Zeit für die Berechnung des Layouts bei steigender Knoten- /Kantenanzahl zunimmt. Ebenso wurde auf die Anwendung von implementierten Features, wie Multilevel-Scaling verzichtet. Bei Testdurchläufen stellte sich heraus, dass die Aktivierung von Features den Zusammenhang zwischen den Eigenschaften eines Graphen und den Konstanten Kantenlänge und Abstoßungskraft verringert.
Das Layout der Graphen wurde iterativ berechnet. Bei jeder Iteration wurden die Werte der Kantenlänge und der Abstoßungskräfte verändert. Ebenso wurden Messwerte zu den Eigenschaften des Graphen festgehalten. Dazu zählt die Gesamtzahl der Knoten, die Anzahl der inneren/ Blattknoten, die Anzahl der Kanten, die maximale Tiefe des Hierarchiebaums, die durchschnittliche Tiefe eines Graphen, sowie die durchschnittliche Anzahl von ausgegeben Kanten eines Knoten. Der Prozess wurde beendet, sobald jede gewünschte Kombination von

Kantenlänge und Abstoßungskraft als Layout ausgelegt wurde. Das Layout, welches die beste Visualisierung erzeugte, das heißt eine möglichst kleine Fläche in Anspruch nahm und keine Überlappungen zwischen Knoten und Graphen erzeugte, wurde separat festgehalten. Der entstandene Datensatz umfasst 494 Datenreihen, für 11 zuvor generierte Graphen wurde kein passendes Layout gefunden.

Um den Zusammenhang zwischen den Eigenschaften eines Graphen und der Kantenlänge bzw. der Abstoßungskraft zu ermitteln, wurde das Verfahren der multiplen Regressionsanalyse (kurz MLR) gewählt. Der zuvor entstandene Datensatz wurde als Grundlage dafür verwendet. Die Analyse wurde mit den Programm SPSS [32] und mit Hilfe der Programmiersprache `R` umgesetzt. Vor dem Beginn der MLR wurden Ausreißer aus dem Datensatz entfernt. Der Datensatz wurde auf 470 Datenreihen verkleinert.

Die MLR wurde für zwei Modelle durchgeführt. Für den ersten Durchlauf wurde die Kantenlänge als abhängige Variable gewählt und zuvor ausgewählte Eigenschaften der Graphen bilden die unabhängigen Variablen. Im zweiten Durchlauf ist die Abstoßunskraft die abhängige Variable. Die Analysen wurden durchgeführt und überprüft, ob die Voraussetzungen für die Aussagekraft der MLR erfüllt sind.

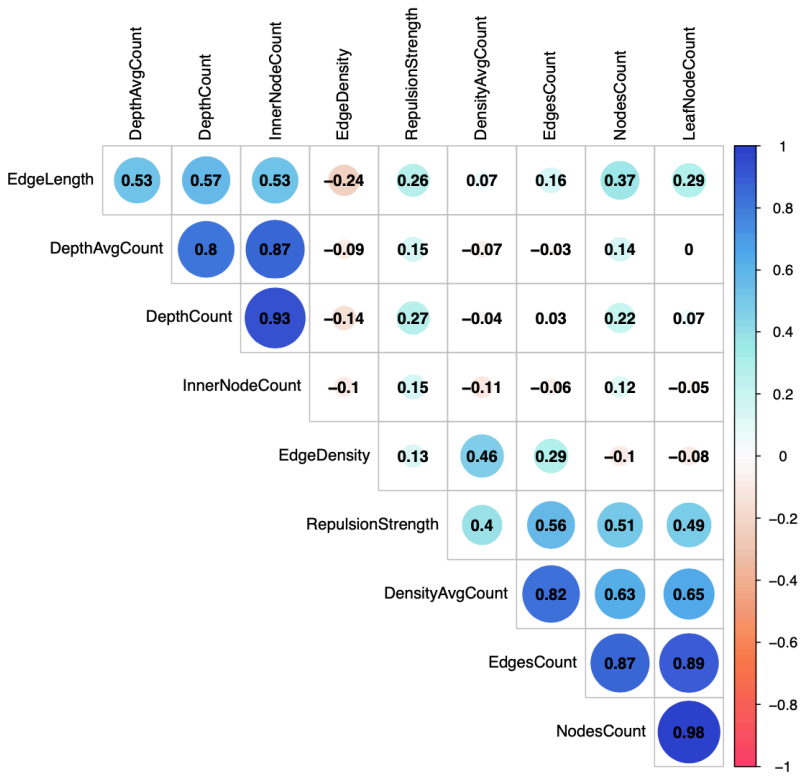


Abbildung 22: Korrelationsmatrix (erzeugt mit `R` und der Bibliothek `corrplot`)

Infolgedessen wurden die unabhängigen Variablen für das erste Model auf die Anzahl der Knoten, die Kantendichte sowie auf die maximale Tiefe des Hierarchiebaums begrenzt, da sonst Multikollinearität (s. Abbildung 22) zwischen den Variablen besteht. Die Auswahl der Variablen erfolgte manuell.

Bei der Untersuchung der Voraussetzungen für die MLR fiel ebenfalls auf, dass die Residuen keine Varianzgleichheit aufwiesen, also eine Heteroskedastizität der Residuen vorlag. Um die Werte der Standardfehler, die daraus berechnete t-Statistik und die p-Werte nicht zu verzerren, wurde Bootstrapping während der MLR angewandt.

Das Vorgehen für das Modell *Abstoßungskraft* verlief äquivalent. Die unabhängigen Variablen für das zweite Model sind die Anzahl der Blattknoten, die maximale Tiefe des Hierarchiebaumes und die Kantendichte.

Die folgenden Ergebnisse konnten der MLR entnommen werden:

Modellzusammenfassung

Modell	R	R-Quadrat
Kantenlänge	,737	,543
Abstoßungskraft	,561	,314

Tabelle 1: Modellzusammenfassung

Varianzanalyse

Modell		Quadratsumme	df	Mittel der Quadrate	F	Sig.
Kantenlänge	Regression	10765,344	3	3588,448	184,852	,000
	Nicht standardisierte Residuen	9046,222	466	19,412		
	Gesamt	19811,566	469			
Abstoßungskraft	Regression	5011,179	3	1670,393	71,241	,000
	Nicht standardisierte Residuen	10926,270	466	23,447		
	Gesamt	15937,449	469			

Tabelle 2: Varianzanalyse

Koeffizienten

Modell		Regressions-koeffizient	Std.-Fehler	Sig.
Kantenlänge	(Konstante)	-5,055	,943	,001
	Tiefe	2,279	,167	,001
	Knotenanzahl	,206	,016	,001
	Kantendichte	-57,829	15,385	,002
Abstoßungskraft	(Konstante)	-1,995	,924	,030
	Blattknotenanzahl	,209	,016	,001
	Tiefe	1,105	,172	,001
	Kantendichte	91,799	16,504	,001

Tabelle 3: Koeffizienten

Tabelle 1 zeigt Werte der Modellzusammenfassung auf. R (Korrelationskoeffizient) gibt die Korrelation zwischen den vorhergesagten Werten des Modells und den tatsächlichen Werten an. Dieser kann Werte zwischen -1 und +1 annehmen. Der Wert R^2 (Determinationskoeffizient) gibt Aufschluss über die aufgeklärte Varianz des Models.

Die Werte von R und R^2 für das Modell *Kantenlänge* können aus Tabelle 1 entnommen werden. Aus diesen wird ersichtlich, dass es einen relativ hohen Zusammenhalt zwischen der Variable Kantenlänge und der Anzahl der Knoten/Tiefe/Kantendichte des Hierarchiebaumes gibt. Die Varianzaufklärung ist für das Modell akzeptabel. Der Wert sagt aus, dass die unabhängigen Variablen 54,3 % der Variable *Kantenlänge* aufklären. Das Modell *Abstoßungskraft* liefert nicht so gute Ergebnisse.

Die Ergebnisse der Varianzanalysen sind in Tabelle 2 zusammengefasst. Diese gibt Auskunft über die Signifikanz eines Regressionsmodells. Dazu wird ein F-Test durchgeführt. Bei einem Signifikanzwert von $< 0,005$ enthält das Modell signifikante erklärende Variablen.

Für das Modell *Kantenlänge* kann aus der Tabelle ein Signifikanzwert von $\approx 0,000$ entnommen werden. Das Modell ist somit signifikant, ebenso wie das Modell *Abstoßungskraft.*

Die Tabelle 3 gibt Auskunft über die Regressionskoeffizienten, aus denen die Regressionsgleichung aufgestellt werden kann. Die Signifikanz der Koeffizienten wird mittels eines t-Tests berechnet. Ein Ergebnis von $< 0,005$ gilt als signifikant.

Für die beiden Modelle wird in Tabelle 3 ersichtlich, dass die meisten Koeffizienten signifikant sind. Allein die Konstante des Modells *Abstoßungskraft* ist größer $0,005$. Dies stellt jedoch für die MLR kein Problem dar.

Aus den Regressionskoeffizienten sind für die Modelle folgende Formeln entstanden:

$$\begin{aligned}\text{Kantenlänge} = -5,055 + 0,206 * \text{KA} + 2,279 * \text{Tiefe} + (-57,829 * \text{KD}) \\ \text{mit KA} = \text{Kantenanzahl, KD} = \text{Kantendichte}\end{aligned} \tag{8}$$

$$\begin{aligned}\text{Abstoßungskraft} = -1,995 + 0,209 * \text{BA} + 1,105 * \text{Tiefe} + 91,799 * \text{KD} \\ \text{mit BA} = \text{Blattknotenanzahl, KD} = \text{Kantendichte}\end{aligned} \tag{9}$$

Im Folgenden werden Vermutungen über die Kausalitäten der Modelle aufgestellt. Zuerst werden die Zusammenhänge für das Modell *Kantenlänge* untersucht.

- *Tiefe*: Ein Grund für die Signifikanz ist möglicherweise die Einbeziehung der Tiefen der Endknoten einer Kante in die Berechnung der idealen Länge.
- *Anzahl der Knoten*: Die Relevanz der Anzahl der Knoten ist eventuell darauf zurückzuführen, dass die Anziehungskraft der Abstoßungskraft entgegenwirkt. Da in der Berechnung der Abstoßungskräfte die Anzahl der Blattknoten eine deutliche Rolle spielt, überträgt sich diese auch auf die Anziehungskraft.
- *Kantendichte*: Gegebenenfalls besteht der Zusammenhang aus der Tatsache, dass die Anziehungskräfte nur zwischen Knoten berechnet werden, welche über eine Kante verbunden sind. Diesen Zusammenhang spiegelt vielleicht diese Variable wieder.

Nun werden die Ursachen für das Modell *Abstoßungskraft* betrachtet.

- *Tiefe*: Die Bedeutsamkeit der Tiefe entsteht eventuell aus der Tatsache, dass die Distanz zwischen den Endknoten der Intergraph-Kanten bei steigender Tiefe des Hierarchiebaumes zunimmt.
- *Anzahl der Blattknoten*: Die Anzahl aller Kindknoten eines Knoten wird in die Berechnung der Abstoßungskräfte mit einbezogen. Dadurch kann der kausale Zusammenhang dieser Variable mit der Abstoßungskraft entstehen.

- *Kantendichte*: Die Einbeziehung der Kantendichte ist möglicherweise auf den Grund zurückzuführen, dass die Abstoßungskraft der Anziehungskraft entgegenwirkt und bei der Kantendichte in dem Modell *Kantenlänge* eine signifikante Rolle spielt.

Das durch die MLR entstandene Modell ist in seiner Gültigkeit eingeschränkt. Zum einen wurden für den zugrundeliegenden Datensatz nur Graphen innerhalb festgelegter Grenzwerte betrachtet. Dadurch kann es sein, dass das Modell auf Graphen, die außerhalb dieser Limitationen liegen, nicht das gewünschte Verhalten zeigt und Werte zurückliefert, welche nicht ideal sind. Durch eine Vergrößerung des betrachteten Bereiches könnte diese Restriktion gelockert werden.
Ebenso verändern sich die Modelle für andere unabhängige Variablen. Die Auswahl dieser Variablen ist auf die erhobenen Messwerte beschränkt. Bei anderen Messwerten, wie beispielsweise der zusätzlichen Erhebung des Verhältnisses zwischen der Anzahl der Intergraph-Kanten zu der Gesamtanzahl von Kanten oder die durchschnittliche Anzahl von Kindknoten eines Graphen, könnte sich die Aussagekraft des Modells verändern.
Überdies besteht der Datensatz aus einer Menge von zufällig generierten Graphen. Diese sollen ein repräsentatives Abbild aller möglichen Graphen innerhalb der Grenzwerte schaffen. Falls der Datensatz durch Fehler in der Generierung die Allgemeinheit aller Graphen nicht richtig widerspiegelt, ist davon auszugehen, dass das Modell Fehler aufweist. Daraus geht zum Teil auch die dritte Beschränkung hervor. Das Modell ist auf Knoten einer bestimmten Größe angepasst. Variieren die Knotengrößen zu stark, zum Beispiel durch die Deaktivierung des Features *Z-Score scaling* in `SEE`, wird das Modell ungenügende Werte liefern. Ebenso beruht das Modell auf dem derzeitigen Implementierungsstand. Werden Schritte innerhalb des CoSE-Layouts verändert, muss auch das Modell erneuert werden.

Die entstandenen Formeln wurden in `SEE` als Feature *Automatische Berechnung der Parameter* eingebaut.

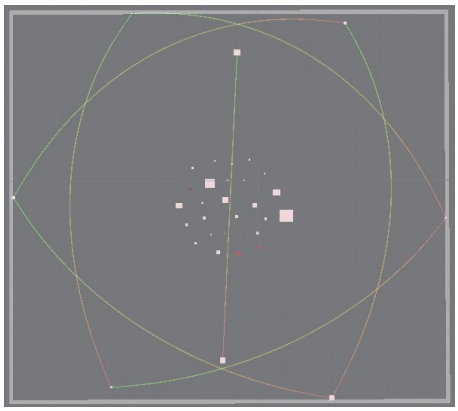
4.3.7 Iterative Berechnung der Parameter

Da die Vorhersage der Parameter nicht immer Werte liefert, welche eine gute Darstellung erzeugen, wurde `SEE` um ein weiteres Feature erweitert. Der Benutzer kann vor dem Layoutprozess die Option *Iterative Berechnung der Parameter* auswählen. Die Werte für die Kantenlänge und die Abstoßungskräfte werden, wie

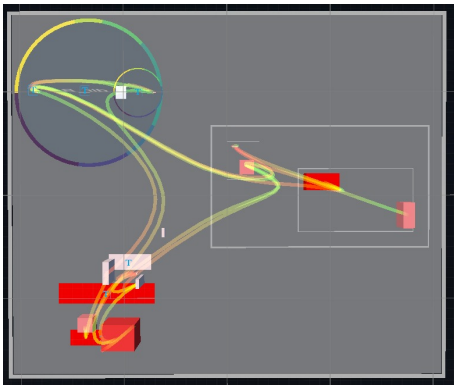
oben aufgezeigt, berechnet. Falls sich in der Zeichnung weiterhin Knoten überlappen, werden die Parameter iterativ vergrößert und das Layout neu berechnet. Dies geschieht solange bis eine maximale Anzahl an Iterationen erreicht ist oder eine Darstellung erzeugt wurde, in der sich keine Knoten überlagern.

4.4 Fertiges Layout

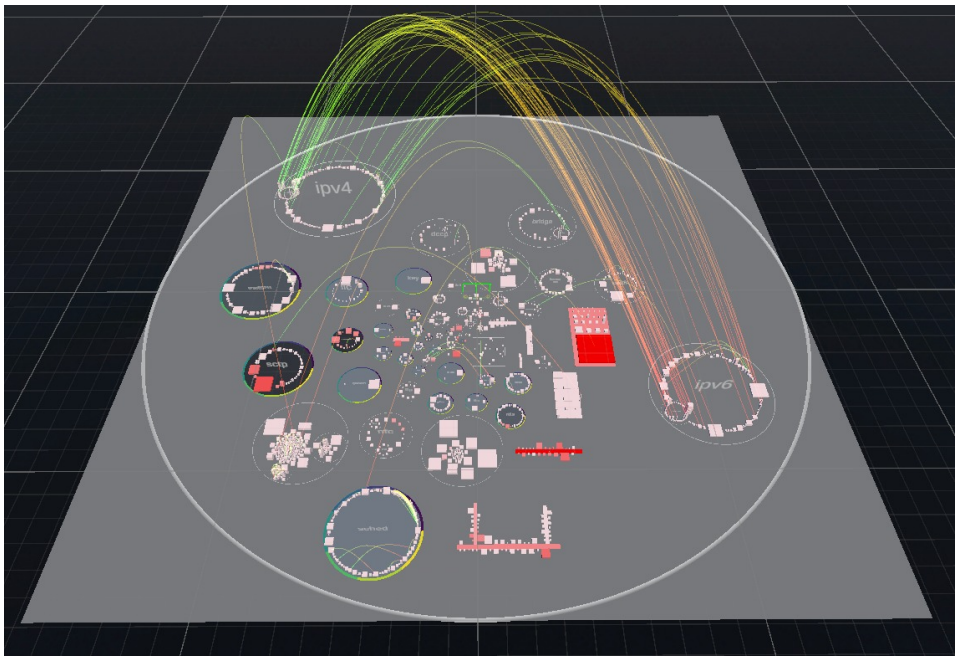
Die folgenden Abbildungen zeigen Beispiele von Visualisierungen, die durch das CoSE-Layout und durch den Einsatz von Sublayouts entstanden sind.



(a)



(b)



(c)

Abbildung 23: (a) Visualisierung durch das CoSE-Layout; (b) Visualisierung durch das CoSE-Layout mit Sublayouts; (c) Ein Circle-Packing-Layout, welches sich aus vielen Sublayouts zusammensetzt.

5 Evaluation

In diesem Kapitel wird die entstandene Implementierung evaluiert. Am Anfang wird erörtert, ob die angestrebten Eigenschaften der Visualisierung erreicht werden konnten. Folgend wird das CoSE-Layout mit den bereits in SEE existierenden Layouts verglichen. Am Ende erfolgt eine Zusammenfassung der Ergebnisse.

5.1 Angestrebte Eigenschaften des CoSE Layouts

In Kapitel 3 wurden Kriterien aufgestellt, anhand derer das CoSE-Layout für die Umsetzung der Visualisierung von Compound Graphen gewählt wurde. Die Kriterien werden in diesem Abschnitt wieder aufgegriffen und es wird evaluiert, ob diese durch die erzeugten Visualisierungen erfüllt werden.

Die Kriterien K1 bis K8 sind wie in Kapitel 3 erläutert, durch das CoSE-Layout gegeben. Auch K9 wird theoretisch durch das CoSE-Layout erfüllt. Die Frage inwiefern die theoretischen Ergebnisse von den tatsächlichen abweichen ist hier aber näher zu betrachten. K9 beinhaltet die Einhaltung von Konventionen/ ästhetischer Kriterien und Einschränkungen von Compound Graphen. Hinzu kommen die ästhetischen Kriterien aus Kapitel 2.3.1.2.

Die Erfüllung der ästhetischen Kriterien und Einschränkungen wird nicht anhand von vorgenommenen Messungen oder einer Benutzerstudie belegt, sondern durch das Hinzuziehen von Studien zu diesem Themengebiet und durch Erfahrungen, die während der Entwicklung gesammelt wurden.

- Der CoSE Algorithmus ist so aufgebaut, dass C3, C4 und R4 erfüllt werden. Ausnahmen treten auf, wenn die Werte der Parameter, zum Beispiel der Kantenlänge, nicht gut gewählt sind. Um dem entgegenzuwirken, wurden die Features *Automatische Berechnung der Parameter* und *Iterative Berechnung der Parameter* entwickelt.
- Grundlage für den CoSE Algorithmus ist der Spring Embedder von Fruchterman und Reingold (FR). Angestrebte ästhetische Kriterien von FR sind das Erreichen einer einheitliche Kantenlänge (Minimierung der Varianz, A5) und eine gleichmäßige Verteilung der Knoten. Die Umsetzung dieser wurde von [33] gezeigt.
- A6, die Darstellung der symmetrischen Eigenschaften eines Graphen, wird von Fruchterman und Reingold in ihrer Veröffentlichung als das erfreulichste Ergebnis des Algorithmus bezeichnet [12]. In [33] wurde bestätigt, dass der

FR Algorithmus das Ziel, Symmetrien gut darzustellen, erreicht. Auch bei entstandenen Visualisierungen des CoSE Algorithmus ist zu erkennen, dass Symmetrien gut ausgelegt werden (vgl. Fertiges Layout).

- Knoten werden so nah wie die gewählte Kantenlänge es zulässt, beieinander platziert (A8). Dies geht auch aus entstandenen Visualisierungen hervor.

Demzufolge werden alle explizit genannten Konventionen, ästhetische Kriterien und Einschränkungen für Compound Graphen erfüllt. Zusätzlich wird die Auslegung der Symmetrie und eine Verringerung der Varianz durch das Layout erzielt.

5.2 Vergleich zu anderen Visualisierungen

Das Ziel des folgenden Abschnitts der Evaluation ist es herauszufinden, wie sich die Ergebnisse des CoSE-Layouts von anderen Graphzeichnungen unterscheiden. Der Vergleich erfolgt mit allen Layout Algorithmen aus SEE, welche ebenfalls Compound Graphen darstellen können. Dies sind die Folgenden:

- Circle-Packing-Layout
- Balloon-Layout
- Evostreet-Layout
- Treemap-Layout
- Rectangle-Packing-Layout

Für die Auswertung wurden 3000 Graphen zufällig generiert. Die erstellten Graphen haben 1-50 Blattknoten, 1-10 innere Knoten und eine Kantendichte von 0.001 - 0.05. Jeder dieser Graphen wird durch ein Layout visualisiert. Die Anzahl der ausgelegten Graphen ist für jeden Layoutalgorithmus derselbe. Nach dem Layoutprozess wurden zu jeder Iteration Messwerte erhoben. Diese setzen sich aus den in Kapitel 4.3.5 beschriebenen Kennzahlen und den Eigenschaften der Graphen zusammen.

Im Fokus stehen die Messwerte der Varianz sowie der Standardabweichung der Kantenlängen, da diese ein ästhetisches Ziel des CoSE Algorithmus widerspiegeln. Die Umsetzungen der Darstellung der Symmetrie werden nicht gemessen. Die Symmetrie der Visualisierung kann besser durch eine Benutzerstudie und nicht durch eine automatisierte Evaluation erhoben werden.

Um die Unterschiede der Messwerte in Bezug auf die Kanten auszuwerten, wurden für deren Auswertung alle Graphen aus dem Datensatz entfernt, welche keine Kanten besitzen.
Da die Graphen zufällig generiert werden, kann es dazu kommen, dass sich der „durchschnittliche“ Graph eines Layouts von dem „durchschnittlichen“ Graphen eines anderen Layouts unterscheidet. Um eine Verfälschung der Ergebnisse durch eine ungleichmäßige Verteilung der Graphen auf die Layouts zu vermeiden, wurde die Erhebung der Daten und die darauf folgende Auswertung dreimal wiederholt. Das Endergebnis dieser Evaluation setzt sich aus den Ergebnissen aller Durchführungen zusammen.

In der Auswertung werden jeweils die Unterschiede der Mittelwerte der Kennzahlen eines Layouts mit denen des CoSE-Layouts untersucht. Die Auswertung erfolgt mithilfe der Programmiersprache `R`.

Für die Umsetzung wurde der Kruskal-Wallis-Test sowie ein paarweise durchgeführter Mann-Whitney-U-Test gewählt.

Zu jeder Kennzahl wurde zu Beginn ein Kruskal-Wallis-Test durchgeführt. Dieser testet, ob sich unabhängige Stichproben in ihrer zentralen Tendenz (Mittelwert) unterscheiden. Bei einem p-Wert von < 0.05 kann davon ausgegangen werden, dass es signifikanten Unterschiede in den Mittelwerte der Gruppen gibt [34]. Der Test gibt jedoch keine Auskunft darüber, welche Gruppen sich voneinander unterscheiden.

Um herauszufinden, welche Gruppen sich signifikant im Mittelwert unterscheiden, wurde der Mann-Whitney-U-Test paarweise durchgeführt. Wie beim Kruskal-Wallis-Test ist der Unterschied signifikant, wenn der p-Wert < 0.05 ist.
Um die Ergebnisse visuell darzustellen, wurden bei einem signifikanten Unterschied Boxplot-Diagramme erzeugt. Die weitere Auswertung erfolgte über diese Boxplot-Diagramme und eine zusammenfassende Statistik über die Messwerte der Layouts.

Im folgenden sind die Ergebnisse aufgeführt.

Fläche: Für alle Durchläufe ergab sich das gleiche Ergebnis.

- *Kruskal-Wallis p-Wert*: $\approx 0,0$
- *Mann-Whitney-U-Test*: Es gibt signifikante Unterschiede zwischen dem CoSE-Layout und allen anderen Layouts.

- *Auswertung*: Eine durchschnittlich kleinere Fläche als das CoSE-Layout benötigen das Circular-/ Evostreet- und Rectangle-Packing-Layout.

Laufzeit: Für alle Durchläufe ergab sich das gleiche Ergebnis.

- *Kruskal-Wallis p-Wert*: $\approx 0,0$
- *Mann-Whitney-U-Test*: Es gibt signifikante Unterschiede zwischen dem CoSE-Layout und allen anderen Layouts.
- *Auswertung*: Das CoSE-Layout hat bei weitem die schlechteste Laufzeit.

Überkreuzungen von Kanten:

- *Kruskal-Wallis p-Wert für alle Iterationen*: $\approx 0,000$
- *Mann-Whitney-U-Test*: In einem Durchlauf ergab sich kein signifikanter Unterschied zwischen dem CoSE-Layout und den anderen Layouts. In zwei Durchläufen unterschied sich das CoSE-Layout signifikant von dem Balloon- und Treemap-Layout.
- *Auswertung*: Es gibt nur geringe Unterschiede in der Anzahl der Kantenüberkreuzungen zwischen den unterschiedlichen Layouts.

Für die Auswertung der Kantenlängen wurden jeweils die angepassten Datensätze verwendet. Diese enthalten keine Graphen ohne Kanten. Ebenso wurden nur die Messwerte zu den Kanten ausgewertet, welche in Relation zu der Fläche gesetzt wurden.

Durchschnittliche Kantenlänge in Relation zu der Fläche: Für alle Durchläufe ergab sich das gleiche Ergebnis.

- *Kruskal-Wallis p-Wert*: $\approx 0,000$
- *Mann-Whitney-U-Test*: Es gibt signifikante Unterschiede zwischen dem CoSE-Layout und allen anderen Layouts.
- *Auswertung*: Das CoSE-Layout befindet sich im Mittelfeld.

Maximale Kantenlänge in Relation zu der Fläche:

- *Kruskal-Wallis p-Wert für alle Iterationen*: $\approx 0,000$

- *Mann-Whitney-U-Test*: Es gibt signifikante Unterschiede in allen Durchläufen zwischen dem CoSE-Layout und dem Evostreet-/ Rectangle-Packing-/ Treemap- und Balloon-Layout. In zwei Iterationen ergab sich kein signifikanter Unterschied zwischen dem CoSE-Layout und dem Circle-Packing-Layout.
- *Auswertung*: Bei einem Durchlauf schnitt das Circle-Packing-Layout besser ab, d.h. es hatte die kleinsten Werte für die maximale Kantenlänge. Bei den anderen Durchläufen hatte das CoSE-Layout die besten Ergebnisse. Im Allgemeinen ist anzunehmen, dass das Circle-Packing-Layout und das CoSE-Layout ähnliche Ergebnisse in Bezug auf die maximale Kantenlänge liefern.

Minimale Kantenlänge in Relation zu der Fläche:

- *Kruskal-Wallis p-Wert für alle Iterationen*: $\approx 0,000$
- *Mann-Whitney-U-Test*: Für die Durchläufe treten widersprüchliche Ergebnisse auf.
- *Auswertung*: Die Ergebnisse für die minimale Kantenlänge sind für alle Layouts relativ ausgeglichen und es gibt keine signifikanten Unterschiede.

Standardabweichung der Kantenlängen in Relation zu der Fläche:

- *Kruskal-Wallis p-Wert für alle Iterationen*: $\approx 0,000$
- *Mann-Whitney-U-Test*: In einem Durchlauf gab es keinen signifikanten Unterschied zwischen dem CoSE-Layout und dem Circle-Packing-Layout. Für alle anderen Durchläufe und Layouts ergaben sich signifikante Unterschiede.
- *Auswertung*: Das CoSE-Layout liefert die besten Ergebnisse für die durchschnittliche Standardabweichung von Kantenlängen. Das entstandene Box-Plot-Diagramm aus der ersten Durchführung wird in Abbildung 24 dargestellt.

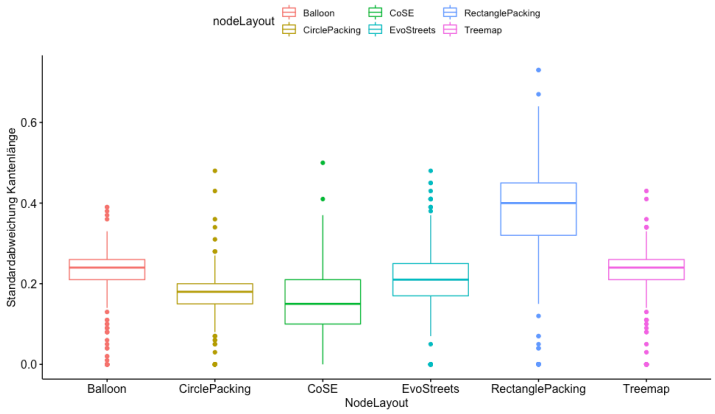


Abbildung 24: Durchlauf 1: Standardabweichung der Kantenlänge

Varianz der Kantenlängen in Relation zu der Fläche:

- *Kruskal-Wallis p-Wert für alle Iterationen*:$\approx 0,000$
- *Mann-Whitney-U-Test*: Ähnlich wie bei der Standardabweichung gibt es zwischen dem CoSE-Layout und allen anderen Layouts signifikante Unterschiede. Für einen Durchlauf ergab sich kein signifikanter Unterschied zu dem Circle-Packing-Layout.
- *Auswertung*: Das Circle-Packing-Layout schneidet ähnlich wie das CoSE-Layout ab. Die anderen Layouts haben eine höhere Varianz innerhalb der Kantenlängen.

Zusammenfassung

Messwert	Bewertung
Fläche	
Laufzeit	
Überkreuzungen von Kanten	
Durchschnittliche Kantenlänge in Relation zu der Fläche	
Maximale Kantenlänge in Relation zu der Fläche	
Minimale Kantenlänge in Relation zu der Fläche	
Standardabweichung der Kantenlängen in Relation zu der Fläche	
Varianz der Kantenlängen in Relation zu der Fläche	

gut, mittelmäßig, neutral, schlecht

Tabelle 4: Zusammenfassung: Vergleich CoSE-Layout mit anderen Visualisierungen

Das CoSE-Layout schneidet in den Tests überwiegend positiv ab. Besonders auffällig ist, dass die Werte für die Standardabweichung, sowie für die Varianz für das CoSE-Layout im Vergleich zu den anderen Layouts besser sind und die ästhetischen Ziele des CoSE-Layouts widerspiegeln. Die Ergebnisse der durchgeführten Tests sind in Tabelle 4 zusammenfassend dargestellt.

5.2.1 Threads to Validity

Gründe für eine eingeschränkte Gültigkeit/ nicht signifikante Ergebnisse könnten sein:

- Die Kennzahlen, welche für den Vergleich der Layouts gewählt wurden, beschreiben die Vorteile der CoSE Visualisierungen nicht ausreichend.
- Die Ergebnisse des CoSE-Layouts sind von Grund auf nicht so gut wie erwartet/ nicht sehr signifikant.
- Die Implementierung beinhaltet Fehler, was zu einer Verschlechterung der Visualisierung führt.
- Die angestrebten Resultate des CoSE-Layouts unterscheiden sich nicht signifikant von den anderen Layouts.
- Das eingesetzte Feature *iterative Berechnung der Parameter* liefert keine guten Ergebnisse. Außerdem werden in dem Feature die anderen Erweiterungen wie *Multilevel-Scaling* und *SmartRepulsionRange* nicht angewendet. Bei einer Einbeziehung könnte es sein, dass das Layout bessere Visualisierungen liefert.
- Die erzeugten Graphen sind, trotz mehrfacher Wiederholung, nicht gleichmäßig auf die Layouts verteilt, sodass das Endergebnis verfälscht wird.

5.3 Auswertung

Die angestrebten Eigenschaften für die Visualisierung von Compound Graphen konnten durch das CoSE-Layout weitestgehend umgesetzt werden. Dies geht aus der durchgeführten Evaluation hervor.

Zum einen konnte durch das Hinzuziehen von Studien und durch Erfahrungen, die während der Implementierung gesammelt wurden, gezeigt werden, dass die ästhetisch angestrebten Kriterien für Compound Graphen umgesetzt wurden. Diese

Ergebnisse konnten durch die darauf folgenden Tests für den Vergleich von dem CoSE-Layout mit anderen Visualisierungen aus SEE untermauert werden.

Die Eigenschaften, welche in der theoretischen Evaluation nicht bestätigt werden konnten, wie die Minimierung von Überkreuzungen, Minimierung des Bereichs, Minimierung der Summe der Längen und die Minimierung der maximale Länge, wurden in dem zweiten Teil der Evaluation weitestgehend aufgegriffen und bewertet.

6 Fazit

Das Ziel dieser Ausarbeitung war es, eine ästhetisch ansprechende Visualisierung von einem hierarchisch aufgebauten Graphen durch einen Force-Directed Ansatz zu erzeugen. Die Implementierung erfolgte für die Software SEE. Dabei sollten festgelegte Eigenschaften, wie die nahe Platzierung von Graphkomponenten, welche durch eine Kante verbunden sind, umgesetzt werden. Diese zielen darauf ab, die Lesbarkeit und Benutzbarkeit der Graphzeichnung zu erhöhen. Ebenso sollten anwendungsspezifische Einschränkungen, wie die Umsetzung von Sublayouts, integriert werden.

Zu Beginn wurden die Grundkonzepte der Softwarevisualisierung und Graphzeichnungen vorgestellt, sowie eine Einführung zu Force-Directed Methoden gegeben. Im Folgenden wurden verschiedene Strategien zur Erstellung von Zeichnungen für Compound Graphen vorgestellt und Entwurfsentscheidungen begründet dargelegt. Daraufhin wurde das CoSE-Layout als Grundlage für die entstandene Implementierung gewählt. Dieses, sowie weitere anwendungsspezifische Einschränkungen und Features, wurden in SEE integriert. Schlussendlich wurde eine theoretische Evaluation durchgeführt, in der geprüft wurde, ob das CoSE-Layout alle festgelegten Eigenschaften der Visualisierung erfüllt. Diese wurde durch Tests erweitert, welche das CoSE-Layout mit anderen Layouts aus SEE verglichen.

Die Evaluation bestätigt das erfolgreiche Umsetzen zuvor festgelegter Eigenschaften, zeigte aber auch die bestehenden Schwachstellen, wie die lange Laufzeit des CoSE-Layouts auf.

6.1 Ausblick

Die Implementierung muss an dieser Stelle nicht beendet sein. Eine Erweiterung der Features ist durchaus denkbar. Auch ist das CoSE-Layout anpassbar und kann im Rahmen von SEE erweitert werden. Im Folgenden werden mögliche Weiterentwicklungen aufgezählt.

Die Sublayouts können um die Möglichkeit erweitert werden, diese automatisch, auf Grundlage der Grapheigenschaften und der Eigenschaften des Layouts, auszuwählen. Jedes Layout besitzt Vorteile, welche durch einen gezielten Einsatz besser zur Geltung kommen könnten. Ein Circle-Packing-Layout ist beispielsweise gut für Graphen anwendbar, die eine überschaubare Anzahl an Kindknoten haben und dessen Kinder alle Blattknoten sind.
Die benötigte Grundstruktur, um diese Erweiterung zu implementieren, wurde

im Zuge dieser Ausarbeitung umgesetzt. Ein berechnetes Sublayout kann durch ermittelte Kennzahlen ausgewertet werden. Auf Grundlage dessen und mit Einbezug der Grapheigenschaften kann die Eignung eines Sublayouts bewertet werden.

Das CoSE-Layout könnte des Weiteren auch für interaktive Graphvisualisierungen eingesetzt werden. Es profitiert von dem eingesetzten Graphmanager [24]. Dieser kann effizient auf interaktive Operationen reagieren, da die Abstraktionslevel des Compound Graphen klar voneinander abgespalten sind. Funktionen wie Erweitern oder Zusammenfassen von Knoten, ebenso wie einzelne Bestandteile des Graphmanagers ausblenden, können effizient auf der Struktur des Graphmanagers implementiert werden (vgl. [24]).

Zudem gibt es noch weitere Features, welche das CoSE-Layout weiter verbessern könnten. Manche, wie die Integration eines Spektral Layouts, können die Ergebnisse und die Laufzeit weiter verbessern [35].

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] E. Giral. *A layout algorithm for undirected compound graphs.* PhD thesis, Bilkent University, August 2005.

[2] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing - Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.

[3] H. C. Purchase, R. F. Cohen, and M. I. James. An experimental study of the basis for graph drawing algorithms. *ACM Journal of Experimental Algorithmics*, 2:4, 1997.

[4] S. Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software.* January 2007.

[5] D. Graanin, K. Matkovi, and M. Eltoweissy. Software visualization. *Innovations in Systems and Software Engineering*, 1:221–230, September 2005.

[6] https://github.com/iVis-at-Bilkent/chilay. (abgerufen: 10.02.2019).

[7] E. R. Gansner and Y. Koren. Improved circular layouts. In Michael Kaufmann and Dorothea Wagner, editors, *Graph Drawing*, pages 386–398, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[8] A. Bondy and M. Ram Murty. *Graph Theory.* Graduate Texts in Mathematics. Springer, 2008.

[9] U. Dogrusoz, E. Giral, A. Cetintas, A. Çivril, and E. Demir. A layout algorithm for undirected compound graphs. *Inf. Sci.*, 179:980–994, March 2009.

[10] K. Sugiyama and K. Misue. Visualization of structural information: automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):876–892, July 1991.

[11] I. Fuhrmann. Layout of compound graphs. Master's thesis, 2012.

[12] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software-Practice & Experience*, 21(11):1129–1164, November 1991.

[13] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[14] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[15] M. Schönfeld and J. Pfeffer. Fruchterman/reingold (1991): Graph drawing by force-directed placement. In *Schlüsselwerke der Netzwerkforschung*, chapter Fruchterman/Reingold (1991): Graph Drawing by Force-Directed Placement, pages 217–220. Springer VS, Wiesbaden, 2019.

[16] P. A. Tippler and G. Mosca. *Physik für Wissenschaftler und Ingenieure.* Springer Spektrum, 2015.

[17] M. Kaufmann and D. Wagner. *Drawing graphs. Methods and models*, volume 2025. January 2001.

[18] P. Eades and M. L. Huang. Navigating clustered graphs using force-directed methods. *Journal of Graph Algorithms and Applications*, 4:157–181, 2000.

[19] Y. Frishman and Ayellet Tal. Dynamic drawing of clustered graphs. In *IEEE Symposium on Information Visualization*, pages 191–198, October 2004.

[20] G. Wallner. Force directed embedding of hierarchical cluster graphs. January 2008.

[21] U. Dogrusoz, M. E. Belviranli, and A. Dilek. Cise: A circular spring embedder layout algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 19(6):953–966, June 2013.

[22] P. Eades and Q. Feng. Multilevel visualization of clustered graphs. In Stephen North, editor, *Graph Drawing*, pages 101–112, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[23] https://cytoscape.org/cytoscape.js-cose-bilkent/demo-compound.html. (abgerufen: 10.02.2019).

[24] U. Dogrusoz and B. Genc. A multi-graph approach to complexity management in interactive graph visualization. *Computers & Graphics*, 30(1):86–97, 2006.

[25] M. Bar and M. Neta. Humans prefer curved visual objects. *Psychological Science*, 17(8):645–648, 2006.

[26] A. Karacelik. *An improved spring embedder layout algorithm for compound graphs.* PhD thesis, Bilkent University, 2012.

[27] C. Walshaw. A multilevel algorithm for force-directed graph drawing. pages 171–182, January 2000.

[28] https://github.com/iVis-at-Bilkent/cose-base. (abgerufen: 14.05.2020).

[29] www.btluke.com/simanf1.html. (abgerufen: 8.12.2019).

[30] F. Antonio. *Faster Line Segment Intersection*, pages 199–202. Academic Press Professional, Inc., 1992.

[31] https://www.habrador.com/tutorials/math/5-line-line-intersection/. (abgerufen: 14.05.2020).

[32] https://www.ibm.com/de-de/products/spss-statistics. (abgerufen: 5.06.2020).

[33] F. Brandenburg, M. Himsolt, and C. Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. In *Graph Drawing*, 1995.

[34] https://www.methodenberatung.uzh.ch/de/datenanalyse_spss/unterschiede/zentral/kruskal.html. (abgerufen: 24.05.2020).

[35] https://github.com/iVis-at-Bilkent/cytoscape.js-fcose. (abgerufen: 26.05.2020).