Visualisierung von Folgen von Graphen mit der TimeArcTrees-Methode

Diplomarbeit

Martin Greilich

Universität Trier
Fachbereich IV - Informatik
Lehrstuhl für Softwaretechnik
Prof. Dr. Stephan Diehl

Trier im März 2009

Erklärung zur Diplomarbeit

Hiermit erkläre ich, dass ich die Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe. Die Diplomarbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher auch nicht veröffentlicht.

Martin Greilich Trier, den 13.03.2009

Danksagung

Ich danke Herrn Prof. Dr. Stephan Diehl für die Unterstützung bei der Anfertigung meiner Diplomarbeit. Seine zahlreichen Ideen und Verbesserungsvorschläge haben mir sehr geholfen.

Weiterer Dank gilt Michael Burch und Fabian Beck, die immer ein offenes Ohr für meine Fragen hatten und durch ihre Vorkenntnisse wichtige Hinweise für die Umsetzung der Programmierung gegeben haben.

An dieser Stelle möchte ich ebenfalls Andreas Marx danken, der mir bei der Fehlersuche und der Wahl von verständlichen Formulierungen sehr geholfen hat.

Ganz besonderer Dank gilt meinen Eltern, die mich während meiner gesamten Studienzeit unterstützten, und meiner Verlobten Kasia Urbanek, die mich stets motiviert und immer an mich geglaubt hat.

Inhaltsverzeichnis

Einleitung

Die Beschreibung von Beziehungen und Abhängigkeiten innerhalb einer Menge von Objekten mittels Graphen ist ein weit verbreitetes Mittel in der Informatik und allen anderen Wissenschaften. So lassen sich zum Beispiel Netzwerke jeglicher Art beschreiben. Ein konkretes Beispiel ist das Computernetzwerk World Wide Web (WWW) in denen Server miteinander über das Internet verbunden sind. In der Psychologie werden soziale Netzwerke benutzt, um Beziehungen zwischen Personen oder Gruppen von Personen zu beschreiben. Diese Liste von Beispielen aus der Wissenschaft könnte man beliebig fortsetzen.
Das Daten-Beziehungs-Modell der Graphen beschränkt sich nicht nur auf die Wissenschaften, sondern kann auch im alltäglichen Leben gefunden werden. Ein typisches Beispiel ist der Graph des Autobahnnetzes in Kapitel 4.2, welcher Straßenverbindungen zwischen Autobahnknoten beschreibt.

Oft ändern sich die Beziehungen über die Zeit. Die Graphen sind dynamisch und können durch eine Folge von einzelnen Graphen dargestellt werden. Jede Änderung der Struktur des Graphen kann durch einen separaten Graphen in der Folge modelliert werden. Zusätzlich unterliegen die Objekte oft einer hierarchischen Ordnung. Diese Ordnung wird Informationshierarchie genannt und kann gemeinsam mit den Relationen zwischen den Objekten mit einem Compound Graphen dargestellt werden. Informationshierarchien sind in vielen Anwendungsbereichen zu finden. Beispiele sind hierarchische Strukturen eines Betriebes, geographische Einteilung der Welt und Dateisysteme mit Ordnern und Dateien. Die Evolution der Relationen zwischen Objekten einer Informationshierarchie kann mithilfe einer Folge von Compound (Di-)Graphen modelliert werden. Die Stärke der Relation kann mit gewichteten Kanten ausgedrückt werden.

Durch die bildliche Darstellung kann ein Mensch den Graphen oftmals besser begreifen. Dabei spielt die Art und Weise, wie der Graph gezeichnet wird, eine entscheidende Rolle. Die Zeichnung soll die Struktur möglichst so darstellen, dass besondere Eigenschaften hervorgehoben werden.
Um Graphen bildlich darzustellen, wurden bereits viele Algorithmen entwickelt. Viele dieser Algorithmen können aber nur statische Graphen zeichnen und nicht Folgen von Graphen. Nur wenige sind in der Lage mit wechselnden Relationen umzugehen.

Das Ziel dieser Diplomarbeit ist es, ein Werkzeug zu entwickeln, dass eine gesamte Folge von Graphen als traditionelles Knoten-Kanten Diagramm in einer einzigen Ansicht zeigt. Die Knoten der einzelnen Graphen der Folge sollen auf einer Vertikalen von oben nach unten gezeichnet werden. Die Reihenfolge wird von der Informationshierarchie bestimmt, die auf der linken Seite der Ansicht als Knoten-Kanten Diagramm abgebildet werden soll. Die Kanten in den Graphen der Folge sollen links bzw. rechts der Vertikalen verlaufen. Aufwärts laufende Kanten befinden sich links und abwärts laufende rechts der Geraden. Diese Anordnung soll beim visuellen Verfolgen von Kanten helfen und die Graphen möglichst strukturiert darstellen.

Das TimeArcTrees-Werkzeug soll mithilfe mehrerer interaktiver Funktionen den Benutzer bei der visuellen Analyse der Daten helfen. Eine der wichtigsten Funktionen ist das Aufklappen und Zusammenfassen von Teilen der Hierarchie. Die aktuell angezeigten Blätter der Hierarchie bestimmen die Knoten in der Folge von Graphen. Eine weitere wichtige Funktion soll die Zusammenfassung von aufeinanderfolgenden Graphen der Folge sein.
Zusätzlich soll das Werkzeug in der Lage sein einige graphtheoretische Probleme wie kürzeste Pfade oder maximale Flüsse zwischen Knoten zu berechnen und somit deren Veränderungen über die Zeit zu visualisieren.
Ein wichtiges Feature ist, dass die sogenannte Mental Map bei den flüssigen Animationen aufrecht erhalten bleibt. Darunter versteht man, dass die Topologie des Graphen in der mentalen Repräsentation des Betrachters durch die Verschiebung der Knoten nicht zerstört wird.

Die Diplomarbeit gliedert sich in die Teile Grundlagen, Beschreibung des Werkzeuges, Beschreibung der Implementierung sowie Anwendungsfällen.
In Kapitel 1 werden alle zum Verständnis der Arbeit benötigten theoretischen Grundlagen behandelt. Außerdem werden einige grundlegende Konzepte der Informationsvisualisierung genannt und bestehende verwandte Werkzeuge aufgeführt.
In Kapitel 2 folgt die Beschreibung der Funktionsweise des TimeArcTrees-Werkzeuges. Zuerst wird kurz das Datenmodell präsentiert. Im Anschluß folgt eine detaillierte Beschreibung der Ansicht, die aus den beiden Teilen Hierarchie und Zeittafel besteht. Das Kapitel schließt mit der Erklärung der zusätzlichen interaktiven Funktionen ab.
In welcher Art und Weise das Werkzeug implementiert wurde, wird in Kapitel 3 beschrieben. Einige ausgewählte Implementierungsdetails werden darin genauer behandelt.
Zuletzt werden in Kapitel 4 Anwendungsfälle des Werkzeuges mithilfe mehrerer Datensätze präsentiert. Dabei werden die wichtigsten Funktionen benutzt, um aussagekräftige Visualisierungen zu erzeugen.
Abgeschlossen wird diese Diplomarbeit in Kapitel 5 mit einer Diskussion und Zusammenfassung der Arbeit. Zusätzlich wird ein Ausblick über mögliche Weiterentwicklungen des Werkzeuges gegeben.

Kapitel 1

Grundlagen

1.1 Graphentheoretische Grundlagen

Die folgenden Abschnitte führen die nötigen Grundlagen der Graphentheorie ein. Sie sollen keine vollständige Einleitung dazu darstellen, sondern nur die in der Arbeit benutzen Begriffe und Eigenschaften näher bringen. Für eine detailliertere Beschreibung der Graphentheorie sei deshalb auf [Diestel, 2006] verwiesen.

1.1.1 Graphen

Relationale Daten, die aus einer Menge von Elementen und Beziehungen zwischen den Elementen bestehen, sind allgegenwärtig und werden in der Informatik meist als *Graphen* modelliert. Die Elemente werden als *Knoten* und die Relationen zwischen den Elementen als *Kanten* bezeichnet. Ein Graph $G = (V, E)$ besteht also aus der Menge der Knoten V und der Menge der Kanten E, wobei $E \subseteq V^2$.
Eine Kante (u, v) mit $u = v$ ist eine *Schleife* oder *Schlinge* und Kanten, die mehr als einmal im Graphen vorkommen (E muss dabei als Multimenge definiert sein), nennt man *Mehrfachkanten*.

In Zeichnungen von Graphen werden Knoten oft als Kreise oder Rechtecke und Kanten als Linien zwischen den Knoten dargestellt.
Von besonderem Interesse sind *gerichtete* Graphen, da diese einseitige Beziehungen darstellen können. Die Kanten aus $E \subseteq (V \times V)$ bestehen hierbei aus *geordneten Paaren* von Knoten. Eine andere Bezeichnung hierfür ist Digraph (directed

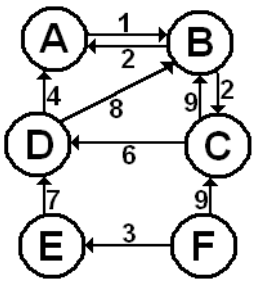

Abbildung 1.1: Beispiel für eine klassische Darstellung eines Graphen.

graph). Das bedeutet, dass alle Kanten Richtungen haben, die durch eindeutige *Start-* und *Zielknoten* beschrieben werden.

Um zusätzliche Informationen zu den Relationen angeben zu können, benutzt man *gewichtete* Digraphen. Kanten wird ein Wert $W \in \mathbb{R}^{m \times m}$ mit $m = |V|$ zugeordnet. Damit lassen sich z.B. Entfernungen zwischen den Knoten angeben.

1.1.2 Compound Digraphen

Oft unterliegen die Knoten V eines Digraphen einer zusätzlichen Hierarchie. Dies kann durch einen *Compound Digraph* modelliert werden. Sei V eine Menge von Knoten und E_1 sowie E_2 Mengen von Kanten. Sei weiter $H = (V, E_1)$ ein Baum, also ein spezieller Graph, mit dem sich Hierarchien modellieren lassen und ein Graph $G = (I, E_2)$, wobei $I \subset V$. Der Baum definiert somit eine Hierarchie auf der Menge I der Knoten des Graphen G dar.
Ein einfaches Beispiel für einen Compound Digraph ist der Graph in Abbildung 1.2. Hierbei dient die geographische Unterteilung der Welt als Hierarchie H und der Digraph G wird mittels Kanten dargestellt.

1.1.3 Pfade

Ein (gerichteter) *Pfad* in einem (gerichtetem) Graph $G = (V, E)$ ist eine Folge $(v_1, v_2, \ldots, v_k)$ von voneinander verschiedenen Knoten $v_i \in V$ aus G, sodass gilt $(v_i, v_{i+1}) \in E$ für $1 \leq i \leq k-1$. Wenn ein (gerichteter) Pfad ebenfalls die Kante $(v_k, v_1) \in E$ enthält, so wird er (gerichteter) *Zyklus* genannt.
Ein besonderer Pfad ist der *kürzeste Pfad*. Ein Pfad p von u nach v ist ein

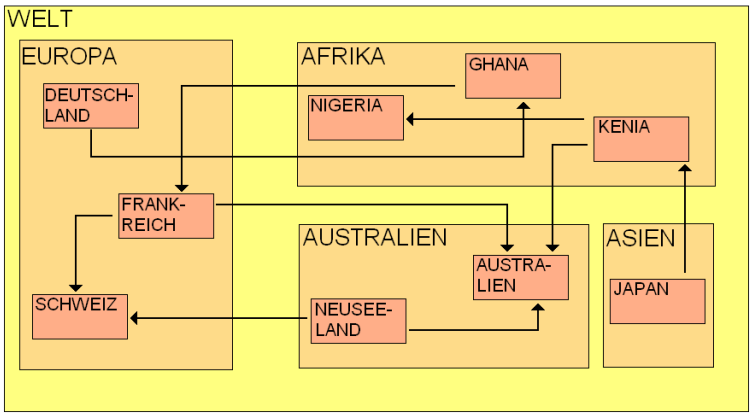

Abbildung 1.2: Beispiel für einen Compound Digraph.

kürzester Pfad, wenn das Gesamtgewicht $w(p)$ minimal ist. Sei $w : E \to \mathbb{R}$ die Gewichtsfunktion des Graphen G. So ist das Gesamtgewicht eines Pfades $p = (v_1, v_2, \ldots, v_k)$ durch $w(p) = \sum_{i=2}^{k} w(v_{i-1}, v_i)$ definiert. Falls nun mindestens ein Pfad von u nach v existiert, so ist das Gewicht des kürzesten Pfades $\delta(u, v) = min\{w(p) : u \to^p v\}$.
Ein Knoten v ist von einem Knoten u *erreichbar*, wenn mindestens ein Pfad $p = (u, u_1, u_2, \ldots, v)$ existiert.

1.1.4 Maximaler Fluß

Sei $G = (V, E)$ ein gerichteter Graph mit Kapazitätsfunktion $c : E \to \mathbb{R}^+$. Sei ferner der Knoten $s \in V$ eine *Quelle* und der Knoten $t \in V$ eine *Senke*. Den *Fluss* zwischen Quelle und Senke beschreibt man mit der Funktion $f : V \times V \to \mathbb{R}^+$, für die folgende Bedingungen gelten:

- Kapazitätsbeschränkung: Für alle $u, v \in V$ gilt $f(u, v) \leq c(u, v)$
- Flusserhaltung: Für alle $u \in V \setminus \{s, t\} : \sum_{v \in u^+} f(u, v) = \sum_{v \in u^-} f(v, u)$, wobei u^+ die Menge der Knoten, die mit ausgehenden Kanten von u und u^- die Menge der Knoten, die mit eingehenden Kanten von u verbunden sind.

Der Wert des Flusses ist $|f| = \sum_{v \in V} f(s, v)$.

Ein Fluß f von s nach t ist *maximal*, wenn kein anderer Fluß f' zwischen s und t existiert, sodass $|f| < |f'|$.

1.2 Informationsvisualisierung und Graphzeichnen

1.2.1 Interaktive Visualisierung von Graphen

Zunächst liegen die Graphen in textueller Form in einer Datei vor. Die Aufgabe des Visualisierungswerkzeuges besteht nun darin, eine für den Benutzer überschaubare Visualisierung daraus zu erzeugen. Wie schon im ersten Abschnitt dieses Kapitels erwähnt, werden Graphen meist mithilfe von Kreisen und Linien dargestellt. Die Anordnung der Elemente spielt dabei eine große Rolle bei der visuellen Verarbeitung durch den Benutzer. Das Graphzeichnen ist eine eigene Disziplin der Informatik und beschäftigt sich mit u.a. dem Problem, einen Graphen dem Benutzer möglichst effizient zu präsentieren. Dabei ist zwischen *statischen* und *dynamischen* Graphen zu unterscheiden. Im statischen Fall wird nur ein Graph dargestellt, wogegen im dynamischen Fall eine Folge von Graphen gezeichnet wird. Für kleine einzelne Graphen genügt es, sich mit dem statischen Graphzeichnen zu beschäftigen. In diesem Kapitel werden jedoch dynamische Graphen vorausgesetzt.

Der Begriff Interaktion beschreibt die Möglichkeit des Benutzers mittels einer Mensch-Maschine-Schnittstelle, wie zum Beispiel einer Maus oder Tastatur, Einfluß auf die derzeitige Ausgabe der Informationen zu nehmen. Die interaktive Präsentation eines Graphen ermöglicht dem Benutzer die Daten so aufzuarbeiten, dass er neue Erkenntnisse gewinnen kann, die durch eine statische Darstellung nicht oder nur mit viel Mühe vermittelbar wären. Dies wirkt sich besonders bei großen und dichten Graphen aus, die eventuell nicht komplett auf dem Bildschirm dargestellt werden können. Hierbei ist es nützlich dem Leitfaden **Visual Information Seeking Mantra** [Shneiderman, 1996] von Ben Shneiderman zu folgen:

- **Overview First:** Zuerst wird dem Benutzer ein Überblick der gesamten Datenmenge (kompletter Graph) gezeigt.
- **Zoom & Filter:** Der Benutzer hat nun die Möglichkeit durch Programmfunktionen die Daten zu filtern und in interessante Teilmengen zu zoomen. Bei Graphen sollte man Kanten und Knoten mit bestimmter Eigenschaft (z.B. minimales Gewicht) herausfiltern und in Teilgraphen hineinzoomen können.
- **Details on Demand:** Bei Bedarf kann sich der Benutzer zusätzliche Informationen z.B. in textueller Form anzeigen lassen. Für Graphen können Zusatzinformationen zu Knoten und Kanten eingeblendet werden.

Außerdem bietet es sich bei dynamischen Graphen an, Funktionen zur Verfügung zu stellen, mit denen man den Graphen „verschönern“ kann. Zwei der Kriterien eines „schönen“ Graphen sind, dass sie möglichst wenige Kantenkreuzungen beinhalten und dass Kanten gleich lang gezeichnet werden. Dies erreicht man durch eine Umpositionierung der Knoten und Änderungen im Verlauf der Kanten.

Eine weitere Technik, die besonders gut bei großen Graphen anwendbar ist, heißt Fokus & Kontext. Dabei wird die Übersichtsdarstellung mit der Detailansicht kombiniert. Der Teil des Graphen, der für den Benutzer von besonderer Interesse ist, wird vergrößert dargestellt und trotzdem im gesamten Kontext gezeigt.

1.2.2 Vor- und Nachteile

Bei der interaktiven Visualisierung von Graphen sind außer des *Visual Information Seeking Mantra* noch andere Gegebenheiten zu beachten, um den Benutzern eine möglichst optimale Datenanalyse zu ermöglichen. Hierbei spielt die Mental Map [Misue et al., 1995] [Bridgeman and Tamassia, 1998] [Diehl and Görg, 2002] [Purchase et al., 2006], also die im Gedächtnis vorhandene Repräsentation, eine wichtige Rolle. Diese wird besonders dann benötigt, wenn sich der Benutzer für Strukturen und nicht nur für einzelne Elemente im Graphen interessiert. Große Änderung der Anordnung der Knoten oder Kanten zerstören diese Mental Map. Deshalb ist es von Vorteil die Anordungung nur dann zu verändern, wenn es auch wirklich notwendig ist und dann dem Betrachter einen möglichst fließenden

Übergang zu präsentieren. So kann die mentale Repräsentation beibehalten und angepasst werden.
Im Folgenden werden die Vor- und Nachteile einzelner Funktionen einer interaktiven Visualisierung von Graphen kurz geschildert.

- **Zooming:** Der größte Vorteil des Zooms ist, dass kleine Details besser ins Auge fallen. Mittels des Vergrößerns eines Ausschnittes oder eines gesamten Teilgraphens können kleine Unterschiede der graphischen Objekte, wie z.B. Kanten ähnlicher Farbe, deutlicher hervorgehoben werden. Es hilft ebenfalls, die kleinen Teilstrukturen des Graphen zu erkennen und mental zu erfassen. Wenige kurze Kanten zwischen eng aneinanderliegenden Knoten gehen oft unter vielen langen Kanten verloren und können durch den Zoom deutlicher dargestellt werden.
Es gibt aber auch Nachteile: Wenn einzelne Teile des Graphen vergrößert dargestellt werden, ist es unvermeidlich, dass der Rest des Graphen undeutlicher dargestellt wird, da für die Vergrößerung Platz benötigt wird. Es gibt nun zwei Möglichkeiten Platz für einen Zoom zu schaffen: Entweder behält man die Größen des restlichen Graphen bei und zeichnet die Vergrößerung teils darüber oder man verkleinert ihn. Bei der ersten Möglichkeit geht der Gesamtüberblick verloren, jedoch werden die Positionen im restlichen Teil des Graphen beibehalten. Bei der zweiten Möglichkeit wird der Überblick behalten, die Elemente um den Zoom herum werden jedoch kleiner dargestellt. Diese Technik wird Fisheye View [Sarkar and Brown, 1992] [Furnas, 1986] [Tominski et al., 2006] genannt.

- **Filter:** Eine wichtige Eigenschaft von Graphen ist das Vorhandensein von wiederkehrenden Mustern und „Ausreißern“, also solchen Kanten, die sich besonders vom restlichen Graphen abheben. Dieser Begriff stammt aus der Statistik und beschreibt einen Messwert, der von der erwarteten Messreihe signifikant abweicht. Muster können mittels Filterfunktionen hervorgehoben werden, indem man die stark abweichenden Teile des Graphen ausblendet. Umgekehrt geht man beim Hervorheben der Ausreißer vor. Man filtert immer wieder kehrende Muster aus, indem man diese Teilgraphen oder Kanten mit durchschnittlichen Werten ausblendet.
Ein weiterer Vorteil ist, dass man Datenmengen, die zu groß für eine visuelle Darstellung wären, nur in Teilmengen darstellen kann. Nach dem *divide-and-*

conquer-Prinzip können nun Teilmengen der Daten analysiert werden, um daraus ein Gesamtergebnis abzuleiten.
Filterfunktionen haben aber auch einen Nachteil. Dieser kann durch eine ungeschickte Wahl der Parameter entstehen. Zum Beispiel könnten Knoten oder Kanten ausgefiltert werden, die für die Anwendung wichtig sind.

- **Einblendung zusätzlicher Informationen:** Oft bieten die Datensätze nicht nur Knoten und Relationen zwischen diesen, sondern auch zusätzliche Informationen an. Diese können bei Bedarf in Textform interaktiv eingeblendet werden und stören somit nicht die Gesamtdarstellung des Graphen. Sie können entweder für einzelne Komponenten oder für alle eingeblendet werden und unterstützen den Benutzer in den Situationen, in denen die reine Zeichnung des Graphen nicht ausreicht, um ein gutes Verständnis der Datenmenge zu erhalten.

- **Änderung der Farbe der Knoten / Kanten:** Durch die Änderung der Einfärbung können zusätzliche Informationen vermittelt werden. Signalfarben können benutzt werden, um bestimmte Knoten oder Kanten gegenüber den restlichen hervorzuheben.
Meist werden Farben in Visualisierungen von Graphen eingesetzt, um zusätzliche Informationen wie Gewichte von Knoten oder Kanten darzustellen. Sie können aber auch benutzt werden, um besondere Eigenschaften hervorzuheben, wie Gruppen von zusammengehörenden Knoten oder Kanten.

- **Änderung des Layouts:** Durch die Änderung der Positionen kann der Graph übersichtlicher dargestellt werden. Eine Repräsentation des Graphen mit vielen Kantenkreuzungen oder vielen langen Kanten ist nicht vorteilhaft, weil man Kanten nur schwer verfolgen kann und das Gehirn so nur schwer die Zusammenhänge des Graphen nachvollziehen kann. In diesen Fällen ist eine Optimierung des Layouts von Vorteil.

1.3 Verwandte Arbeiten

Auf dem Gebiet der Visualisierung von Graphen existieren bereits sehr viele Arbeiten. Es ist verständlich, dass an dieser Stelle nur der Teil vorgestellt wird, der eng mit dem TimeArcTrees-Werkzeug verwandt ist. Wer sich für das gesamte Teilgebiet Graphzeichnen interessiert, bekommt im Buch [Battista et al., 1999] einen sehr guten Einblick in das Thema. Es behandelt alle fundamentale Algorithmen zur Visualisierung von Graphen als traditionelles Node-Link Diagramm. Allerdings stoßen diese Darstellungen bei großen Datensätzen an ihre Grenzen. In [Ghoniem et al., 2004] vergleichen die Autoren die Knoten-Kanten Repräsentation mit der Darstellung als graphische Adjazenzmatrix. Bei der Studie stellte sich heraus, dass Matrixdarstellungen bei Graphen mit mehr als 20 Knoten für das Lösen von Aufgaben gegenüber dem Node-Link Ansatz bevorzugt wurden. Lediglich bei der Pfadsuche war die traditionelle Darstellung bei jeder Größe im Vorteil.

1.3.1 Visualisierung von Hierarchien

Für die Visualisierung von Hierarchien können neben den bisher genannten Varianten auch andere kartesische und radiale Darstellungen benutzt werden. Die bekannteste kartesische Technik ist die TreeMap [Johnson and Shneiderman, 1991]. Die hierarchische Struktur wird hierbei durch ineinander verschachtelte Rechtecke gezeigt. Die Fläche ist dabei proportional zur Größe der Elemente, wodurch Größenverhältnisse besonders gut dargestellt werden können. TreeMaps sind sehr effizient, da der ganze Bildschirm flächenfüllend genutzt wird und sie deshalb auch bei großen Hierarchien informative Visualisierungen liefern.
Nach der Veröffentlichung von TreeMaps durch Johnson und Shneiderman folgten mehrere Variationen wie Cushion TreeMaps [van Wijk and van de Wetering, 1999] in denen Schatteneffekte genutzt werden oder Voronoi TreeMaps [Balzer and Deussen, 2005], in denen statt rechteckiger Darstellungen Voronoi Diagramme eingesetzt werden.
In [Zhao et al., 2005] wird der klar strukturierte Node-Link Ansatz mit dem platzeffizienten TreeMap Ansatz zu sogenannten Elastic Hierarchies kombiniert.

In [Andrews and Heidegger, 1998] werden Information Slices vorgestellt. Sie sind eine radiale Visualisierungstechnik für Hierarchien, in der Elemente als halbkreisförmige kaskadierte Scheiben gezeichnet werden. Der Bildschirm ist dabei in zwei Hälften geteilt. Auf der linken Seite wird die gesamte Hierarchie gezeigt und rechts kann ein Teil der Hierarchie vergrößert dargestellt werden.
Einen Überblick weiterer Visualisierungstechniken findet man in [Herman et al., 2000]. In den nächsten Abschnitten werden Arbeiten vorgestellt, die direkt mit dem TimeArcTrees-Werkzeug verwandt sind. Zuerst werden Techniken zur Visualisierung von Compound Digraphen vorgestellt, dann die Arbeiten zur Darstellung von Folgen von Graphen.

1.3.2 Visualisierung von Compound Digraphen

Zwei grundlegende Arbeiten zum Thema Visualisierung von Compound Digraphen sind [Sugiyama and Misue, 1991] und [Bertault and Miller, 1999]. Die Autoren stellen Algorithmen vor, welche Compound Digraphen zeichnen und dabei die Lesbarkeit berücksichtigen. Auf der Grundlage dieser Algorithmen entstanden im Laufe der Zeit einige Variationen. Diese nutzen die verfügbare Fläche allerdings nur suboptimal aus. Ein Ansatz, der für Hierarchien flächenfüllende Visualisierungen benutzt und für die Relationen zwischen den Hierarchieelementen Kanten darüber gezeichnet, ist der *ZTree* [Bartram et al., 2000]. Hierbei werden die Elemente der Hierarchie als Fenster dargestellt. Falls nun Relationen zwischen zwei Fenstern oder deren untergeordneten Elementen bestehen, wird dies durch eine Kante angezeigt. Der Benutzer hat durch Anklicken der Fenster die Möglichkeit, tiefer in die Hierarchie hineinzuschauen. So hat er stets alle Relationen zwischen den aufgeklappten Fenstern im Blick. Die Visualisierung erfolgt nach einem speziellen Layoutalgorithmus, der dem Benutzter beim Navigieren durch große Hierarchien helfen soll.
Einen ähnlichen Ansatz findet man in *Overlaying Graph Links on TreeMaps* [Fekete et al., 2003]. Hier wird die Hierarchie als traditionelle TreeMap dargestellt und die Kanten verlaufen jeweils zwischen den Mittelpunkten der Flächen. Die Kanten werden als Bezier-Kurven gezeichnet und deuten durch ihren Schwung die Laufrichtung an. So können auch gerichtete Graphen dargestellt werden. Durch die Anordnung der Elemente als TreeMap kann man auch bei großen Datensätzen

die Teilhierarchien erkennen, zwischen denen viele Relationen bestehen. Zwischen diesen Bereichen der TreeMap verlaufen besonders viele Kurven. Zusätzlich hat der Benutzer die Wahl, sich entweder alle Linien anzeigen zu lassen oder nur diejenigen von bzw. zu einem Element der TreeMap. Diese Funktion hilft dem Benutzter, wenn er sich nur für bestimmte Knoten und deren Beziehungen interessiert.

Auch in *Trees in a TreeMap* [Burch and Diehl, 2006] wird die komplette Fläche genutzt. Hier beschränkt sich die Visualisierung der Daten darauf, dass Bäume über die TreeMap gezeichnet werden, um Taxonomien der Elemente darstellen zu können. Die Kanten werden als Linien zwischen den Flächen gezeichnet und es kann zwischen geradlinigem und orthogonalem Layout gewählt werden.

In [Holten, 2006] wird gezeigt, wie man Visualisierungen mit vielen Kanten geordneter darstellen kann, indem man Kanten mit „ähnlichen“ Start- und Zielknoten bündelt. Wie „ähnlich“ die jeweiligen Start- und Zielknoten sind bestimmt die Hierarchie. Durch einen Bündelungsfaktor kann der Grad der Bündelung frei angegeben werden. Je kleiner der Faktor, desto geradliniger sind die Kanten und je größer, desto enger werden sie gebündelt. Die Autoren zeigen, dass sich diese Technik mit bestehenden Visualisierungstechniken für Bäume kombinieren lässt. Die Hierarchie kann als TreeMap oder traditionelles Node-Link Diagramm dargestellt werden, wobei alle Kanten zwischen den Elementen gebündelt gezeichnet werden. Besonders bei großen Datensätzen hilft dieser Ansatz visuell überladene Zeichnungen zu vermeiden.

Ähnlich zum TimeArcTrees-Ansatz (siehe Kapitel 2) werden in *ArcTrees* [Neumann et al., 2005] alle Elemente auf einer Geraden angeordnet und Beziehungen als runde Bögen um die Gerade herum gezeichnet. Allerdings sind nur ungerichtete Graphen darstellbar und alle Kanten befinden sich auf einer Seite. Die Hierarchie wird wie bei der traditionellen TreeMap mithilfe ineinander verschachtelter Rechtecke dargestellt. Eine Aggregation über die Hierarchie ist möglich, indem man entweder Kanten oder Rechtecke anklickt. Der Grad der Aggregation einer Kante wird durch Transparenz angedeutet und die Breite der Kante repräsentiert das Gesamtgewicht.

1.3.3 TimeRadarTrees und Timeline Trees

Wenn verwandte Werkzeuge vorgestellt werden, können TimeRadarTrees [Burch and Diehl, 2008] und Timeline Trees [Burch et al., 2008] nicht ausgelassen werden, da sie die Idee für die Entwicklung von TimeArcTrees geliefert haben.
Alle drei Werkzeuge beschäftigen sich mit Folgen von Graphen, oder Transaktionen deren Knoten einer Informationshierarchie unterliegen. Allerdings werden die Daten unterschiedlich visualisiert.

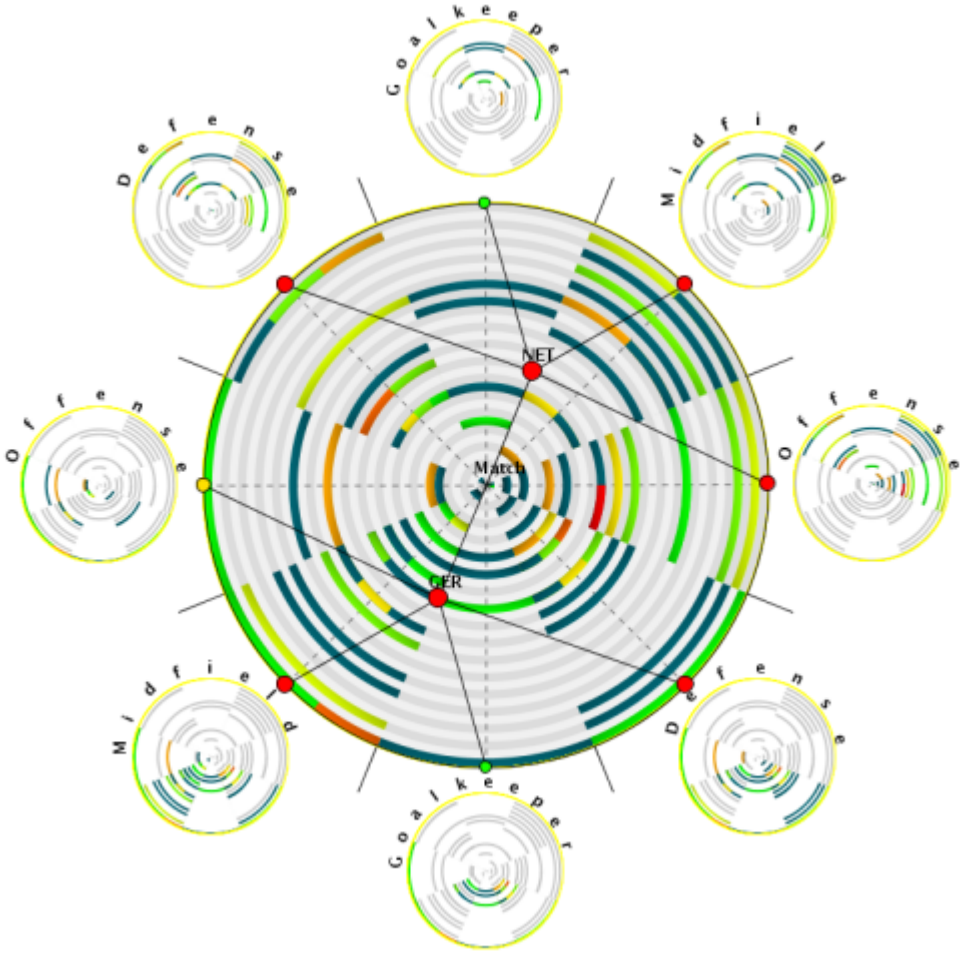

Abbildung 1.3: TimeRadarTrees-Visualisierung eines Fußballspiels.

Das TimeRadarTrees-Werkzeug benutzt ein radiales Baumlayout, um die Hierarchie darzustellen und Kreissektoren repräsentieren die zeitlichen Änderungen des Graphen. Die Beziehungen zwischen den Elementen der Hierarchie werden nicht

explizit als Kanten dargestellt. Alle eingehenden Kanten eines Knotens werden aufsummiert und als gefärbter Kreissektor dargestellt. Um herauszufinden, ob zwei Elemente in Beziehung stehen, werden rund um den gesamten Kreis an jedem Sektor kleine äußere Kreise gezeichnet, die nur die ausgehenden Kanten des Knotens als Kreissektoren zeigen. Der Zeitverlauf wird im Kreis von innen nach außen dargestellt.
Zusätzlich beinhaltet das Werkzeug viele interaktive Funktionen zur Erforschung der Datensätze. Es ist möglich, Teilbäume der Informationshierarchie zu einem Knoten zusammenzufassen oder einzelne Graphen der Folge herauszufiltern.
Der große Vorteil von TimeRadarTrees ist, dass die Visualisierung nicht durch viele Kanten und damit auch durch viele Kantenkreuzungen überladen wird. Die einzigen Kanten die gezeichnet werden, sind die der Hierarchie, welche kreuzungsfrei angeordnet werden können.

Das Werkzeug Timeline Trees visualisiert Folgen von Transaktionen zwischen Elementen einer Informationshierarchie. Die Hierarchie wird wie beim TimeArcTrees-Werkzeug auf der linken Seite des Bildschirms als Baum von links nach rechts gezeichnet. Alle aktuellen Blätter besitzen rechts der Hierarchie eine Zeitleiste, welche die Abfolge der Transaktionen visualisiert. Einzelne Teilbäume der Hierarchie lassen sich beliebig auf und zu klappen, sodass sich Relationen auf verschiedenen Abstraktionsebenen untersuchen lassen.
In der Zeitleiste werden die Transaktionen als farbige Rechtecke dargestellt. Die Farben und Größen der Rechtecke zeigen, wie stark ein Element an der Transaktion beteiligt ist.
Zusätzlich werden zu jedem Element Miniaturbilder angezeigt, welche die Zeitleiste so darstellen, dass nur Transaktionen, an denen das Element beteiligt ist, farbig gezeichnet werden.

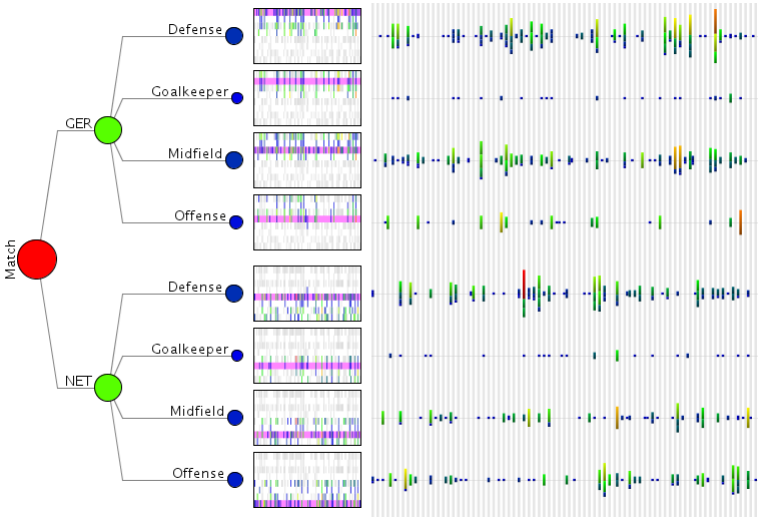

Abbildung 1.4: Timeline Trees-Visualisierung eines Fußballspiels.

Kapitel 2

Das TimeArcTrees-Werkzeug

Im folgenden Kapitel werden die Eigenschaften des vom Autor entwickelten Werkzeugs vorgestellt. Zunächst wird eine Einführung in das Datenmodell, mit dem das Tool arbeitet und aus dem das Eingabeformat abgeleitet wird, gegeben. Es folgt dann die Beschreibung der Visualisierung mit Hierarchieansicht für die Informationshierarchie der Knoten und der Zeittafelansicht für den Verlauf der Relationen über die Zeit. Schließlich werden die interaktiven Funktionen inklusive der Filter und der Pfadalgorithmen erklärt, die der Benutzer zur Analyse der Daten nutzen kann.

2.1 Datenmodell

Für die Graphdaten benötigt man ein Datenmodell, das alle darzustellenden Eigenschaften beinhaltet. Die Menge der Knoten V des Graphen G repräsentiert die darzustellenden Elemente der Visualisierung und die Menge der Kanten E die Relationen zwischen den Elementen. Da es sich im Digraphen um gerichtete Kanten handelt, also $E \subseteq V \times V$, spielt die Reihenfolge der Elemente beim Modellieren der Relation eine wichtige Rolle. Häufig besitzen die Beziehungen zwischen den Elementen ein Gewicht, welches zugeordnet zur Kante durch einen numerischen Wert angegeben werden kann.
Zusätzlich soll die Möglichkeit bestehen, Knoten der Menge V hierarchisch zu ordnen. Diese Hierarchie wird Informationshierarchie genannt. Jeder Knoten kann höchstens einen übergeordneten Knoten haben. Dies führt dazu, dass eine Mono-

hierarchie in Form eines Baumes T zur Modellierung benutzt werden kann. Eine Knotenposition im Baum läßt sich durch einen Pfad von der Wurzel zum Knoten beschreiben.
Aus allen bisher erwähnten Eigenschaften kann bereits ein einzelner Graph modelliert werden. Die wichtigste Eigenschaft des Datenmodells besteht nun darin, Folgen von Graphen zu beinhalten. Es wird hierfür jedem Graphen ein Zeitstempel zugeordnet. So kann der Verlauf der Beziehungen zwischen den Elementen innerhalb einer Zeitspanne modelliert werden, um daraus Informationen herzuleiten.
Basierend auf diesem Datenmodell kann die Form der Eingabedateien erzeugt werden. Eine Datei hat immer folgende Form: Sie beginnt mit der Schlüsselzeile `arbitrary graph;`. Nun folgen die einzelnen Graphen beginnend mit dem optionalen Graphkommentar, der in runden Klammern notiert wird und dem Zeitstempel, der aus Datum und Uhrzeit besteht. Hinter der Zeitangabe werden nun alle Kanten aufgezählt. Diese werden mittels Startknoten, Zielknoten und Gewicht angegeben. Jeder Knoten ist Teil einer Informationshierarchie und die Position darin wird als Pfad beschrieben. Das Gewicht besteht aus einem Vorzeichen und dem numerischen Wert. Auch bei Kanten ist es möglich, einen Kommentar mit anzugeben. Dieser wird ebenfalls umschlossen von runden Klammern vor die Kante geschrieben. Ein Graph wird durch ein Semikolon abgeschlossen.
So kann Graph für Graph der gesamte Verlauf kodiert werden. Die Anzahl der Knoten, Kanten und Graphen ist im Datenmodell unbeschränkt. So können theoretisch auch sehr große Datenmengen beschrieben werden.
Ein Beispiel für eine Eingabedatei sieht folgendermaßen aus:

```
1 arbitrary graph;
2 (1.Graph) 2008-11-20 06:00:00 alle/A/1 alle/B/2 +23 alle/B/1 alle/A/1 -45;
3 2008-12-20 12:00:00 (erste Kante) alle/B/1 alle/B/2 -43;
4 2005-11-05 23:59:59 alle/C/1 alle/A/2 +5.78 alle/C/2 alle/B/1 +0;
```

Listing 2.1: Beispiel für eine Eingabedatei

2.2 Aufbau der Visualisierung

Nachdem die Eingabedatei vom Programm eingelesen wurde, erhält man eine visuelle Darstellung auf dem Bildschirm. Der genaue Aufbau der Ansicht wird nun mithilfe des Standardbeispiels (siehe Abb. 2.1) erklärt.
In dieser Folge von Graphen stellen die Knoten Computer im Netzwerk mit ihren IP-Adressen und die Kanten Verbindungen zwischen den Rechnern dar. Um mit diesem Beispiel alle Funktionen sinnvoll zu erklären, wird das Gewicht z.B. als Nachrichtenlaufzeit oder Nachrichtenaufkommen interpretiert.
In der Abbildung sieht man, dass im zweiten Graphen der Folge die Verbindung zwischen den Knoten 134.96.7.179 und 136.199.55.175 wegfällt und im dritten Graph fällt ein Rechner aus und kann keine Nachrichten mehr senden und empfangen.

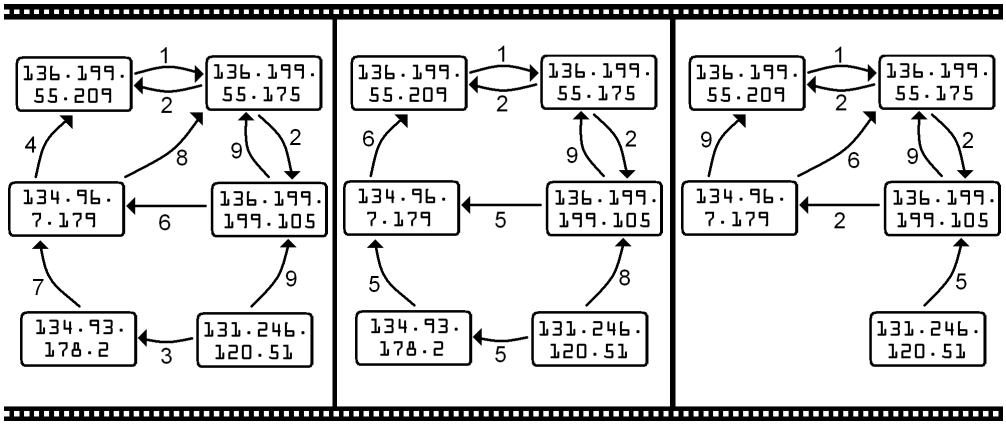

Abbildung 2.1: Folge von Graphen in traditioneller Node-Link Repräsentation.

Die gleiche Folge von Graphen wird in Abbildung 2.2 visualisiert. Das TimeArcTrees-Werkzeug stellt die Graphen ebenfalls als traditionelles Knoten-Kanten Diagramm dar. Die Kantengewichte werden allerdings farblich kodiert dargestellt und bedienen sich verschiedener Farbskalen. In dieser Abbildung werden niedrige Werte grün, mittlere Werte gelb und hohe Werte rot dargestellt.
Wie man in Abbildung 2.2 sieht, besteht die Visualisierung aus zwei Teilen. Auf der linken Seite befindet sich die Informationshierarchie und auf der rechten Seite die sogenannte Zeittafel, die den Verlauf der Graphen zeitlich geordnet darstellt.

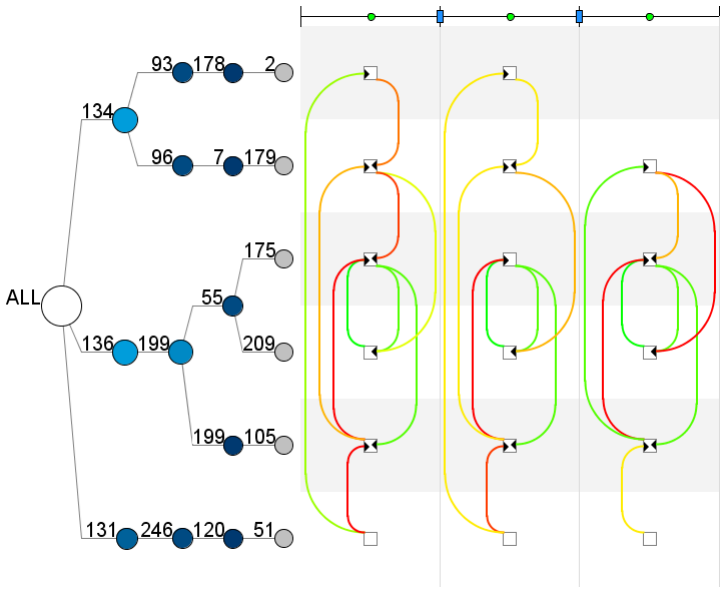

Abbildung 2.2: Beispielgraph in TimeArcTrees Repräsentation.

2.2.1 Hierarchie

Die Hierarchieansicht zeigt die hierarchische Ordnung der Elemente als Baum-Diagramm. Sie stellt die Verbindungen der Knoten mittels einer klassischen Knoten-Kanten Repräsentation dar, wobei alle Knoten als Kreise gezeichnet werden. Die Wurzel der Hierarchie, also der Knoten, dem alle anderen untergeordnet sind, befindet sich ganz links wogegen die Blätter der Hierarchie rechts auf einer Vertikalen untereinander dargestellt sind. Der Baum wird also nicht wie üblich von oben nach unten gezeichnet, da alle aktuellen Blattknoten in der Zeittafel Repräsentanten haben, die auf der Horizontalen rechts des Baumes in der Zeittafel dargestellt werden. So kann jedem Knoten der Hierarchie auf einfache Art und Weise jeder Repräsentant zugeordnet werden und umgekehrt.
Indem man den Kanten entlang in Richtung Wurzel folgt, sieht man alle übergeordneten Knoten.
Die Größe und Farbe der Knoten in der Hierarchie spiegeln die Anzahl der Knoten im Unterbaum wieder, also die Anzahl untergeordneter Knoten plus eins. Als zweite Alternative kann jedoch auch die Anzahl der Blätter im Unterbaum als Maß genommen werden. Im Werkzeug stehen mehrere Farbskalen zur Einfärbung der Knoten zur Verfügung. Im Beispiel wird die „Blue Tone“-Skala benutzt, die

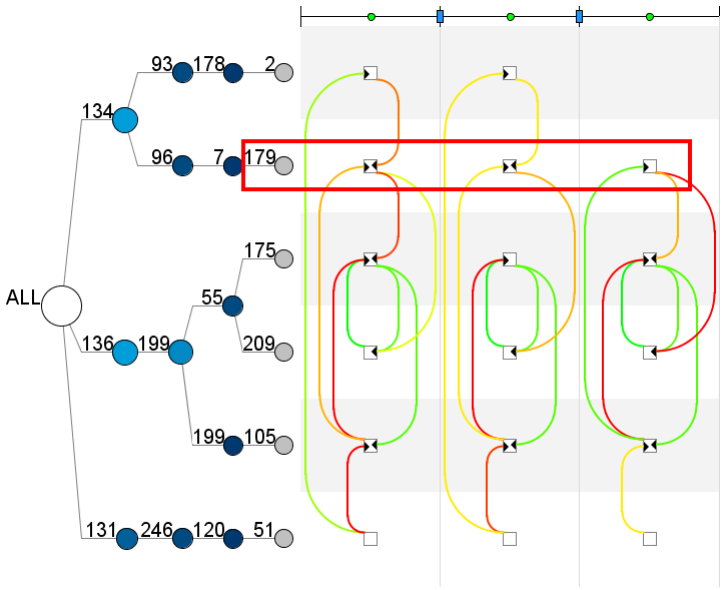

Abbildung 2.3: Knoten in der Hierarchie und dessen Repräsentanten in Zeittafel.

von Schwarz über Blau zu Weiß verläuft. Alle Blattknoten werden durch eine Graufärbung kenntlich gemacht.
Der Baum wird nicht nur zum Zweck der Visualisierung der Informationshierarchie angezeigt, er dient auch dazu, Interaktionen mittels Anklicken auszuführen. Es stehen einige Funktionen bereit, die eine Manipulation der Darstellung erlauben.
Die wohl wichtigste Funktion ist die Aggregation über die Hierarchie. Sie erlaubt das Zusammenfassen eines Teilbaumes zu einem Knoten, indem man auf den Knoten, der die Wurzel des Teilbaumes darstellt, klickt. So wird dieser nun als Blatt in der Visualisierung angezeigt und dementsprechend wird auch die Visualisierung in der Zeittafel angepasst. Alle Repräsentanten der untergeordneten Knoten verschmelzen zu einem. Die Kanten zwischen den Repräsentanten, die von dieser Manipulation betroffen sind, werden dementsprechend angepasst und verlaufen nun entweder aus dem neuen Repräsentanten hinaus, hinein oder bilden eine Schleife. Der erste Fall entsteht, wenn die Kante zuvor aus einem anderen Unterbaum in den zugeklappten Teilbaum verlief. Im zweiten Fall führte sie von dem Teilbaum in einen anderen Teilbaum. Der dritte Fall tritt auf, wenn sie Knoten innerhalb des Teilbaums verbunden hat.
In der Abbildung 2.4 sieht man, wie der Teilbaum unter dem Knoten 136.* zu-

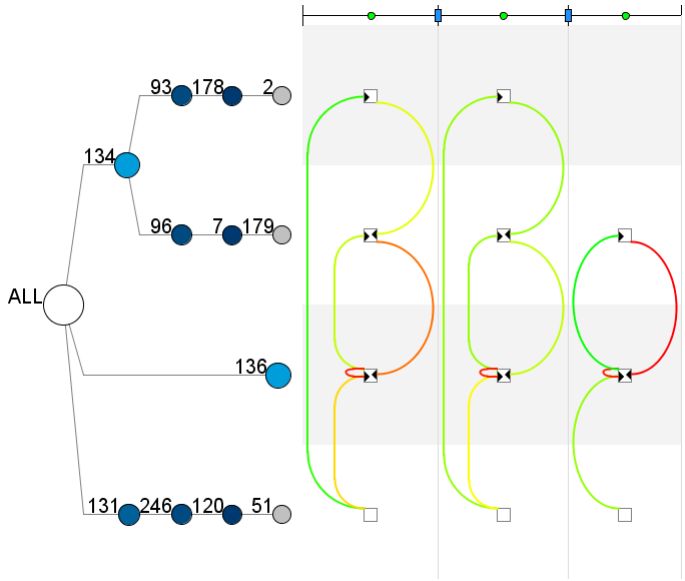

Abbildung 2.4: Zusammengefalteter Knoten 136.* in Hierarchie.

sammengeklappt und gleichzeitig die Visualisierung der Zeittafel angepasst wurde. Zum Beispiel wurden im ersten Graphen der Folge die Kanten (134.96.7.179, 136.199.55.175) und (134.96.7.179, 136.199.55.209) zu einer Kante (134.96.7.179, 136.*) verschmolzen und die vier Kanten innerhalb des Unterbaumes 136.* sind nun zu einer Schleife geworden. Man sieht auch deutlich, dass sich die Gewichtsverhältnisse stark verändert haben, da nun beinahe alle Kanten anders gefärbt werden. Wie die Gewichtung und Färbung der Kanten genau funktioniert, wird in den Kapiteln 2.2.2 und 2.3.1 genauer erläutert.
Ein erneuter Klick auf den zusammengefalteten Knoten erlaubt es, diesen Teilbaum wieder in die ursprüngliche aufgefaltete Form zu bringen.
Der Übergang beim Zusammenfalten und Aufklappen eines Teilbaumes wird animiert dargestellt, sodass der Benutzer die Änderungen der Visualisierung besser nachvollziehen kann (Stichwort Mental Map - siehe Kapitel 1.2.2).
Um die Bildschirmfläche optimal auszunutzen wird der Baum so gezeichnet, dass der vertikale Abstand der aktuellen Blätter maximal ist und somit für jedes Blatt und deren Repräsentanten ein gleich hoher Streifen zur Verfügung steht. Die Streifen der Blätter und Repräsentanten sind durch helle Farben im Hintergrund kenntlich gemacht. Dies hilft bei Bäumen mit vielen Blättern die Zuordnung zu finden. Außerdem ist die Breite der Hierarchie beeinflussbar. Um Platz für die

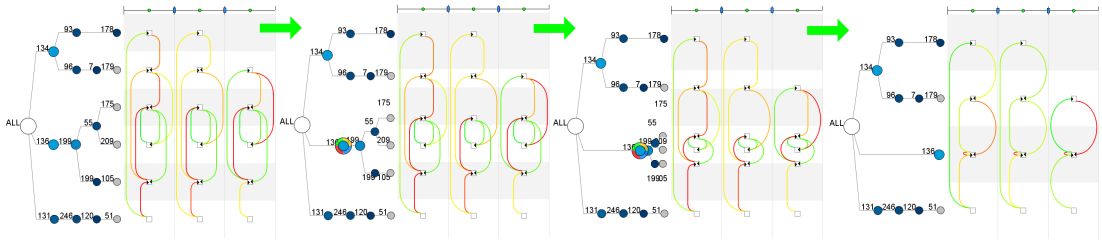

Abbildung 2.5: Bildabfolge vom animierten Zusammenklappen eines Teilbaumes.

Zeittafel zu schaffen, kann die Breite der Hierarchie verringert werden. Durch eine andere Anordnung der Beschriftungen oder das Verkleinern der Schrift und der Knoten kann eine kompaktere Darstellung erreicht werden. Mehr zu diesem Thema wird im Kapitel 3.3.2 erörtert.

Wenn der Mauszeiger auf Hierarchieknoten kommt, erkennt man, dass ein Ring mit drei oder vier Farben um den Knoten herum erscheint. Er dient zum einen dazu, den Knoten zu fokussieren, zum anderen kann man mit diesem Funktionen auf dem Knoten ausführen.

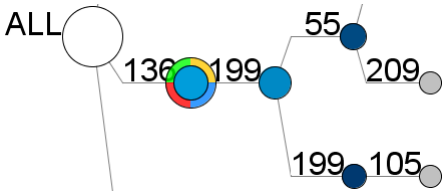

Abbildung 2.6: Funktionen ringförmig um Hierarchieknoten.

Jedem der vier farbigen Bereiche wird ein anderer Zweck zugeschrieben. Der gelbe Bereich hat die gleiche Funktion wie die innere Fläche. So kann man mit einem Klick darauf den Knoten auf- bzw. zuklappen. Weil dies bei Blättern nicht zur Verfügung steht, haben diese nur die drei anderen Bereiche.
Die blaue Fläche dient der Deaktivierung des Knotens und damit auch all seiner Repräsentanten. Alle ein- und ausgehenden Kanten der korrespondierenden Knoten in der Zeittafel werden nun nicht mehr angezeigt. Es handelt sich also um eine Filterfunktion, die benutzt werden kann, um uninteressante Knoten aus der Darstellung zu entfernen.
Die Deaktivierung funktioniert auch, wenn sie auf einen Knoten der nicht ein

aktuelles Blatt ist, ausgeführt wird. Dann werden gleichzeitig auch alle untergeordneten Knoten deaktiviert. Um Knoten wieder zu aktivieren, reicht ein zweiter Klick auf die blaue Fläche.

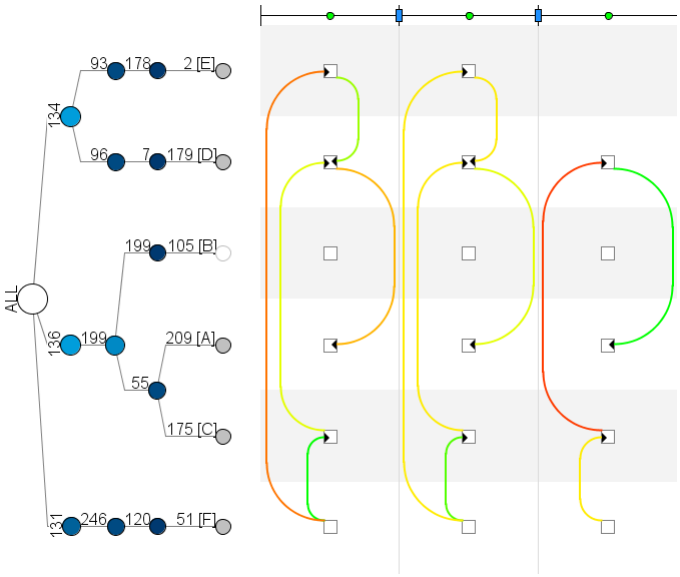

Abbildung 2.7: Deaktivierter Knoten 136.199.199.105.

In Abbildung 2.7 wurde der Knoten, der einen Server mit der IP-Adresse 136.199.199.105 darstellt, deaktiviert. Da dieser viele Verbindungen zu anderen hatte, entfällt ein großer Teil der Kanten. Da im dritten Graph auch der Server 134.93.178.3 nicht vorhanden war, ist der Knoten 131.246.120.51 isoliert und hat keine Verbindung zum Restgraphen.
Mit der grünen Fläche im Ring kann ein Knoten als Startknoten bzw. Quelle der Pfade bzw. der Flüsse bei Anwendung der Pfadalgorithmen markiert werden. Wird diese Funktion auf einen Knoten angewendet, der gerade nicht als Blatt dargestellt wird, so wird dieser Knoten auch zusammengefaltet, da nur ein Knoten als Start bzw. Quelle markiert werden kann und nicht mehrere gleichzeitig.
Die rote Fläche hat eine ähnliche Funktion. Sie markiert Zielknoten bzw. Senken der Pfade bzw. der Flüsse. Es gelten die gleichen Bedingungen wie für die Startknoten, bis auf den Fall, dass bei der Berechnung mehrere Knoten als Zielknoten bzw. Senken markiert werden können.
Sowohl Startknoten/Quellen als auch Zielknoten/Senken erhalten eine grüne bzw. rote Markierung und können mit einem erneuten Klick auf den Ringabschnitt

oder die Markierung wieder in den unmarkierten Zustand versetzt werden.
Alle vier Funktionen, die hier aufgezählt wurden, sind auch über das Kontextmenü der Hierarchieknoten verfügbar.

2.2.2 Zeittafel

Die Zeittafel stellt die chronologisch sortierte Folge von Graphen dar. Zunächst wird die Darstellung eines einzelnen Graphen erklärt.

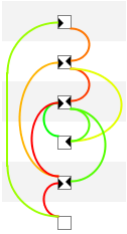

Abbildung 2.8: Darstellung der Graphen.

Die Knoten werden als Rechtecke dargestellt und liegen alle auf einer Vertikalen durch die Mitte der Zeichenfläche eines Graphen. Sie sind genau auf der Höhe ihrer korrespondierenden Knoten der Informationshierarchie untergebracht.

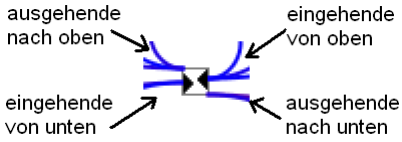

Abbildung 2.9: Knoten mit Ankerpunkten der Kanten.

Alle Kanten werden als farbige Bögen vom Startknoten zum Zielknoten gezeichnet und verlaufen entweder links oder rechts der Vertikalen. Auf der linken Seite befinden sich alle Kanten, deren Startknoten niedriger liegen als die Zielknoten oder die Schleifen beschreiben. Auf der rechten Seite sind entsprechend alle Kanten eingezeichnet, deren Startknoten höher liegen als die Zielknoten. Oder kürzer: Alle nach oben laufenden Kanten sind links angebracht und alle nach unten laufenden rechts. Dies soll das Lesen des Graphen erleichtern. Je größer der Abstand zwischen Start- und Zielknoten ist, desto größer ist auch der Abstand der Kante zur Vertikalen. Der Betrachter gewöhnt es sich meist schnell an, die Kanten in

korrekter Richtung zu verfolgen und wird dabei auch durch die in den Knoten eingezeichneten Pfeilspitzen unterstützt. Außerdem kann durch diese Anordnung die Anzahl der Kantenkreuzungen minimal gehalten werden, ohne dass die hierarchische Gruppierung der Knoten gestört wird. Mehr zur Minimierung der Anzahl der Kantenkreuzungen folgt in Kapitel 3.3.3.
Das Gewicht der Kante bestimmt deren Farbe. Dabei spielt die eingestellte Farbskala und die Einstellung, ob der Graph einzeln(lokal) oder alle Graphen bei der Einfärbung berücksichtigt werden sollen(global), eine Rolle. Die Kanten mit Minimalgewicht werden mit der Anfangsfarbe, die mit Maximalgewicht mit der Endfarbe der Farbskala versehen. Für dazwischenliegende Werte wird die entsprechende Zwischenfarbe der Skala benutzt (siehe dazu auch Seite 35). Als Minimal-/Maximalgewicht kann entweder das Minimum/Maximum der Gewichte im einzelnen Graph (lokal) oder das Minimum/Maximum der Gewichte aller Graphen (global) verwendet werden. Der erste Fall ist von Interesse, wenn man die Kanten mit besonders niedrigem oder hohem Gewicht in den einzelnen Graphen herausfinden will. Im zweiten Fall ist man an den Extremwerten der gesamten Folge interessiert.
In Abbildung 2.10 ist das Beispiel (mit zusammengefassten Knoten 136.199.55.*) in beiden Einstellungen zu sehen. Links wird das Minimum/Maximum aller Graphen als Maß genommen. Man sieht, dass im mittleren Graphen keine der Kanten rot dargestellt ist, d.h. keine der Kanten das Maximalgewicht besitzt. Rechts wird das Minimum/Maximum der einzelnen Graphen als Maß genommen, worauf man nun auch im mittleren Graphen die Extremwerte erkennt. Der Nachteil zur ersten Einstellung ist jedoch, dass die Farbe der Kanten keinen Bezug zur Farbe der Kanten anderer Graphen mehr hat.
Die Einstellung der Minimum-/Maximumwahl erfolgt in der Werkzeugleiste über die Schaltfläche 'Min/Max'. Eine detaillierte Beschreibung der verfügbaren Einstellungen der Farbskala folgt im Kapitel 2.3.2.
Als Alternative zur Standarddarstellung der Kanten als Bögen kann man auf ein rein orthogonales Layout umschalten. Alle Kanten verlaufen dann auf einem rechtwinkligen Gitter. Der Vorteil des orthogonalen Layouts ist, dass alle sich kreuzenden Kanten im rechten Winkel aufeinander treffen. Dies ist optimal für die menschliche Wahrnehmung, wenn es darum geht, dem Verlauf einer Kante zu folgen.

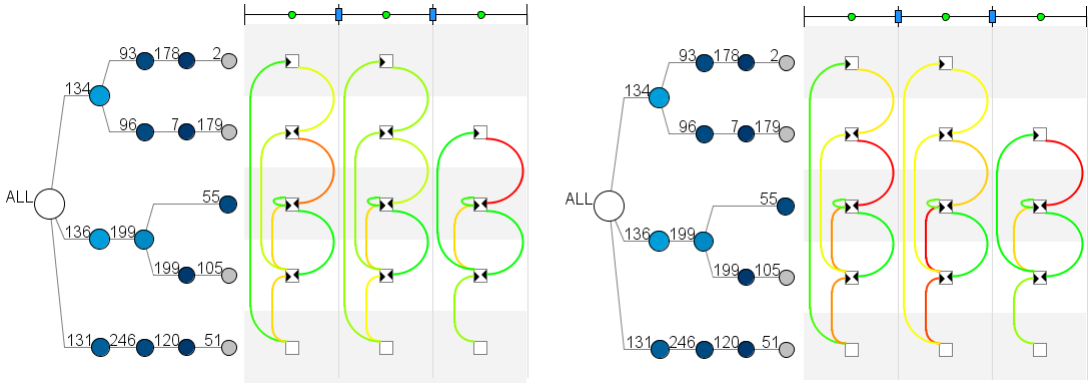

Abbildung 2.10: Kantenfärbung links mit 'globalen' und rechts mit 'lokalen' Min/-Max.

Ein orthogonales Layout bringt aber auch Probleme mit sich. Bei mehreren ein- oder ausgehenden Kanten, die auf der selben Seite eines Knotens gezeichnet werden müssen, kann eine Überlappung der Kanten nicht verhindert werden, denn alle diese Kanten laufen vom selben Punkt aus auf der Horizontalen entweder links oder rechts vom Knoten hinaus. Dies wird bisher so gelöst, dass eine der Kanten den Vorrang hat und diese auf dem gemeinsamen Teilstück alle anderen überdeckt. In vertikalen Teilstücken existiert dieses Problem nicht, da ein spezielles Verfahren angewendet wird (siehe Kapitel 3.3.1).

Die Zeitleiste befindet sich am oberen Rand der Zeittafel und teilt die Zeitspanne aller Graphen in Zeitintervalle. Jeder angezeigte Graph ist also so gesehen nicht der Graph eines einzelnen Zeitpunktes, sondern der Graph eines Zeitintervalls, der natürlich die Zeitpunkte der Graphen beinhaltet. Beim Programmstart wird für jeden Zeitpunkt der Daten ein eigenes Intervall angelegt. Alle Graphen der Folge werden also in separaten Intervallen gezeichnet. Die Intervalle teilen die Zeittafel in vertikale Streifen. In der Mitte jedes Streifens befindet sich die Vertikale, auf der alle Knoten angeordnet werden. Standardmäßig wird die Fläche so unterteilt, dass alle Streifen gleich breit sind, so dass jeder Graph gleich viel Platz zur Verfügung hat. Dies ist allerdings nicht mehr der Fall, wenn ein Graph mehr Breite benötigt, weil er zu viele Kanten besitzt, die ordnungsgemäß ohne Überlappung gezeichnet werden müssen. Ein Algorithmus versucht den Platz optimal aufzuteilen. Sollte aber der Fall eintreffen, dass alle Graphen gemeinsam

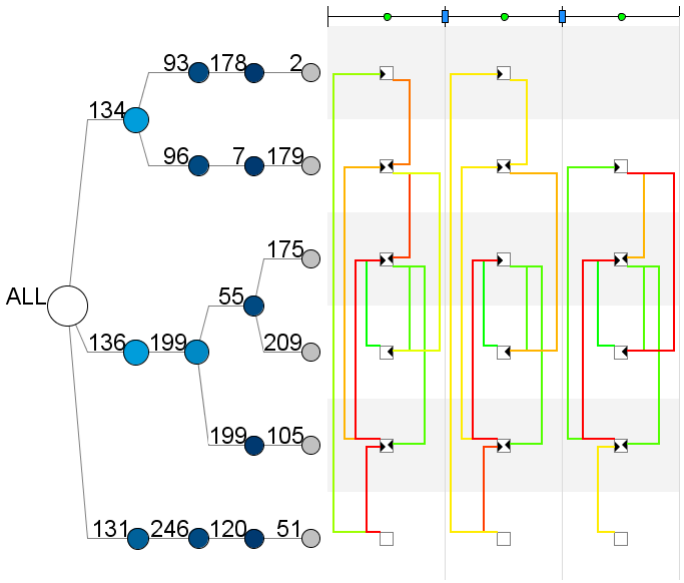

Abbildung 2.11: Orthogonales Layout.

eine größere Breite benötigen als im Fenster vorhanden ist, wird die Zeichenfläche über das Fenster hinaus vergrößert und nur zum Teil dargestellt. Mittels Scrollschaltflächen ist es dann möglich nach links und rechts zu navigieren um den Rest der Zeittafel zu sehen.

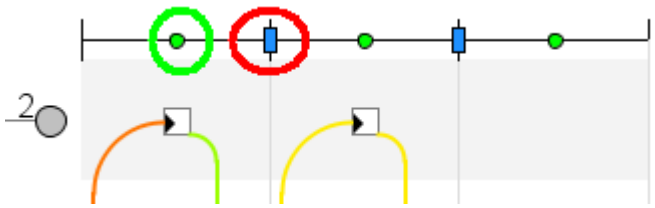

Abbildung 2.12: Zeitleiste - Roter Kreis: Separator - Grüner Kreis: Zeitpunkt.

In der Zeitleiste wird die aktuelle Einteilung dargestellt, mit der Möglichkeit darauf Einfluß zu nehmen. Jede Intervallgrenze wird mit einem sogenannten Separator (Abb. 2.12 roter Kreis) kenntlich gemacht. Per Drag&Drop kann dieser verschoben werden (siehe Abb. 2.13). So wird der Streifen des Graphen (und damit auch der Graph selbst) vergrößert oder verkleinert, die Zeitspanne des Intervalls ändert sich jedoch nicht. Da die Größe der gesamten Zeichenfläche auch beim Verbreitern eines Graphen beibehalten werden soll, müssen die Größen der anderen Intervalle verkleinert bzw. vergrößert werden. Dies erkennt man in Ab-

bildung 2.13, inder die rechten Graphen schmaler werden, falls der linke breiter gezogen wird.

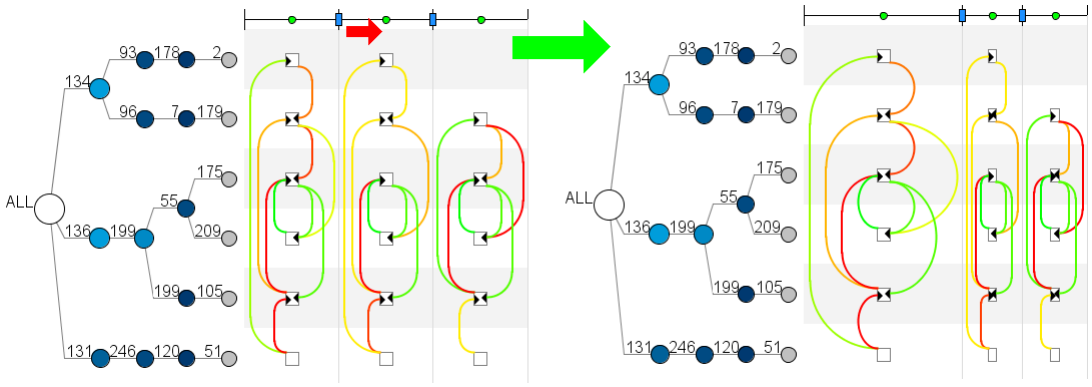

Abbildung 2.13: Vergrößern eines Graphen mittels Verschiebung der Separatoren.

Um die Intervalle zu verändern, gibt es drei Funktionen, die in der Zeitleiste per Kontextmenü angeboten werden. Mit der ersten Funktion lässt sich ein Separator entfernen. Die benachbarten Intervalle verschmelzen so zu einem einzigen und die Graphen werden entsprechend zusammengefasst. Dabei werden Knoten, die ein Element der Informationshierarchie repräsentieren, zusammengefasst. Kanten, die entweder in einem oder beiden Intervallen vorhanden waren, sind ebenfalls Teile des neuen Graphen. Wenn eine Kante zwischen zwei gleichen Elementen jeweils in beiden Zeitabschnitten bestand, so wird sie zu einer Kante aggregiert. Das neue Gewicht der Kante wird vom Modus für Kantengewichte bestimmt (siehe Kapitel 2.3.1).

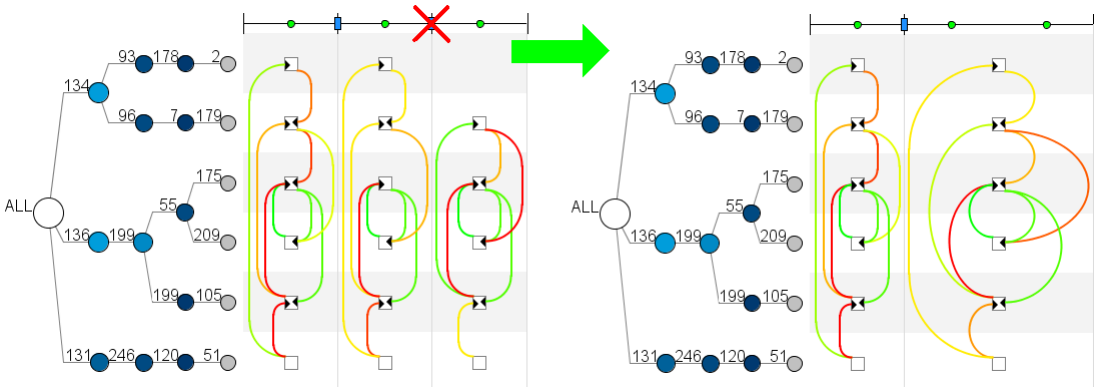

Abbildung 2.14: Entfernen eines Separatoren in der Zeitleiste.

Die zweite Funktion implementiert das Löschen aller Separatoren. Sie funktioniert nach dem gleichen Prinzip wie das Entfernen eines einzelnen Separators und führt diese Aktion auf alle vorhandenen Trennmarkierungen aus. Dadurch entsteht ein Graph, der alle Kanten der Folge enthält.
Die letzte Funktion ist möglich, wenn Separatoren entfernt wurden. Sie erlaubt neue Separatoren wieder einzufügen. Jeder Zeitpunkt, für den ein Graph vorhanden ist, wird in der Zeitleiste durch einen grünen Punkt dargestellt (Abb. 2.12 grüner Kreis). Wenn nun mehrere Zeitpunkte innerhalb eines Intervalls liegen, gibt es die Möglichkeit, zwischen zwei Zeitpunkten eine neue Grenze zu setzen. Bei der Ausführung wird der Streifen zweigeteilt und der Graph für beide Zeitabschnitte rekonstruiert. Das Einfügen eines Separators ist die Umkehrfunktion zum Entfernen eines Separators. Es können jedoch keine leeren Abschnitte gebildet werden.
In Abbildung 2.14 wird das zweite mit dem dritten Intervall zusammengelegt. Man erkennt, dass der zusammengefasste Graph nun alle Knoten und Kanten beider Vorgänger besitzt. Bei aggregierten Kanten wird hier als Gewicht das Durchschnittsgewicht gewertet. Wenn die Gewichte der Kanten im Datensatz z.B. die Nachrichtenlaufzeit zwischen den Rechnern der IP-Adressen darstellten, könnte mit dieser Funktion die durchschnittliche Laufzeit mehrerer Zeitspannen visualisiert werden. Dies ist nützlich, wenn der Datensatz viele Zeitpunkte enthält, aus denen man durch die Visualisierung jedes einzelnen weniger Erkenntnisse gewinnen könnte, als bei der Zusammenfassung zu größeren Intervallen. So ließen sich beispielsweise Tendenzen oder langfristige Trends erkennen. Wenn man alle Graphen zu einem einzigen aggregieren würde, bekäme man so die durchschnittliche Nachrichtenlaufzeit im gesamten Zeitrahmen der Messung. Diese Funktion kann auch als Fokus+Kontext Technik benutzt werden, indem man die interessanten Intervalle kurz wählt und die weniger interessanten Graphen in längere Intervalle packt. Ein Beispiel dafür findet man im Kapitel 4.2, in dem die Visualisierung von Staudaten behandelt wird. Dort werden benachbarte Zeitpunkte, in denen nur wenige Veränderungen bestehen, zu einem Intervall zusammengefasst. Wie die Funktion weiter sinnvoll genutzt werden kann, wird in den Anwendungsbeispielen im Kapitel 4 beschrieben.

Der genaue Zeitpunkt wird per Tooltip unter dem Mauszeiger eingeblendet und

kann benutzt werden, um Intervallgrenzen zeitgenau einzufügen. Falls die Bildpunktanzahl im Intervall es zulässt, werden Zeitmarkierungen mit Strichen und Beschriftungen eingezeichnet.

Eine zusätzliche Hilfe für den Benutzer ist das Hervorheben der ausgehenden Kanten, wenn der Mauszeiger auf einem Knoten gezogen wird. Die Zielknoten der hervorgehobenen Kanten werden ebenfalls mit der Kantenfarbe markiert. So sieht der Benutzer sofort, welche Knoten mit dem Ausgewählten verbunden und wie stark die Verbindungen zu diesen Knoten sind.

2.3 Zusätzliche interaktive Funktionen

Das Programm wird durch die Werkzeug- und Menüleiste vervollständigt, welche die Funktionalität um einige Optionen erweitern.

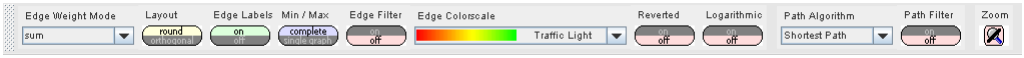

Abbildung 2.15: Werkzeugleiste mit interaktiven Funktionen.

Die Werkzeugleiste ist mit Schaltflächen und Auswahlboxen versehen, mit denen man interaktiv Einfluss auf die Visualisierung nehmen kann. Es sind unter anderem Pfadalgorithmen zur Bestimmung von kürzesten Pfaden und maximalem Fluß, sowie Filter und Optionen zur Veränderung der Farbskala anwendbar.
Schließlich sind noch einige weitere Funktionen in der Menüleiste untergebracht. So gibt es dort die Möglichkeit, die Visualisierung als Bild zu exportieren oder zu drucken und einige Einstellungen am Werkzeug vorzunehmen.

In den folgenden Abschnitten folgen die Beschreibungen der wichtigsten interaktiven Funktionen und Tooltips.

2.3.1 Modi für Kantengewichte

Mit der Auswahlbox „Edge Weight Mode“ kann das Gewicht der Kanten manipuliert werden. Je nach Datensatz und Anwendung sind unterschiedliche Einstel-

lungen sinnvoll.

- Summe: Die erste Einstellung ist die Summe. Bei Kanten, die aus mehreren zusammengefassten Kanten entstanden sind, bildet die Summe der ursprünglichen Kantengewichte das neue Gewicht der aggregierten Kante. Dies entspricht der Standardeinstellung und sollte bevorzugt werden, wenn die Gewichte der Kanten absolute Werte darstellen und deshalb ein Aufsummieren über die Zeit sinnvoll ist.

- Durchschnitt: Bei Gewichten mit relativem Charakter ist das aufsummieren nicht sinnvoll, da der Vergleich zu anderen Graphen nicht mehr vorhanden ist. In diesen Fällen sollte die Option „Durchschnitt“ gewählt werden. Hierbei werden aggregierten Kanten der Durchschnittswert aller Gewichte der beinhalteten Kanten als neues Kantengewicht zugeordnet.

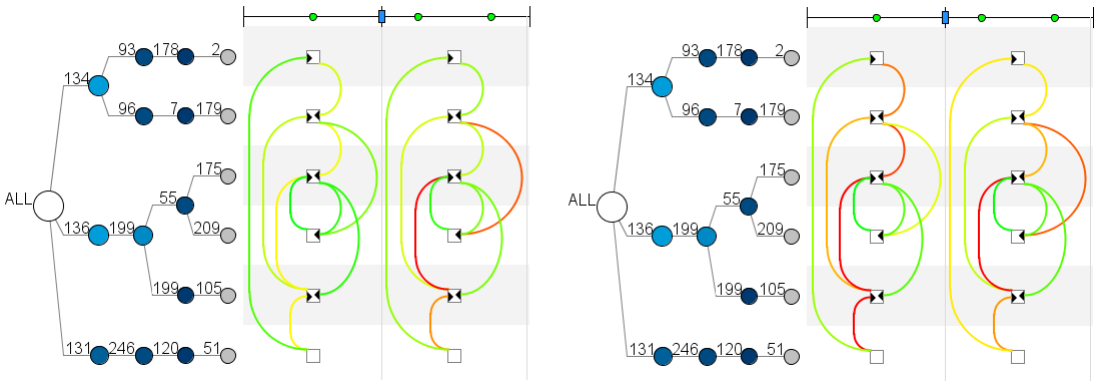

Abbildung 2.16: Zwei Gewichtsmodi aggregierter Kanten (Summe / Durchschnitt).

In Abbildung 2.16 sieht man den Vergleich zwischen den Optionen „Summe“ und „Durchschnitt“. Angenommen wird, dass die Graphen die Anzahl der Nachrichten pro Stunde darstellen (die Daten werden stündlich als Graph aufgezeichnet). Die Abbildung zeigt also die Messung im Intervall der ersten Stunde im Vergleich zum Intervall über die Stunde zwei und drei. Da es sich um relative Daten handelt (Nachrichten pro Stunde) zeigt die linke Visualisierung (Summe) nicht mehr die Anzahl der Nachrichten pro Stunde an. Dies ist allerdings oft wünschenswert und wird mit der Bildung des Durchschnitts erreicht. So kann sinnvoll zwischen unterschiedlich langen

Intervallen verglichen werden, wie es in der rechten Abbildung dargestellt wird. Zum Beispiel erkennt man hier, dass sich das Nachrichtenaufkommen von 131.246.120.51 zu 136.199.199.105 nicht etwa erhöht, wie es die linke Darstellung durch die Farben gelb zu orange andeutet, sondern verringert, wie es rechts durch die Farben rot zu orange signalisiert wird.

- Differenz Hin-/Rückkante: Falls eine Kante in einem Graphen auch eine Rückkante besitzt, also eine Kante vom Ziel- zum Startknoten zurück, so wird das Kantengewicht durch die Differenz zwischen den Kanten als eine Relation visualisiert. Wenn also eine Kante von A nach B mit Gewicht 3 existiert und eine Rückkante von B nach A mit Gewicht 8, so ändert sich das Gewicht der Kante von B nach A auf 5 und das Gewicht der Kante A nach B auf -5. Das bedeutet, dass die Kante B nach A im Vergleich zur Kante A nach B bzgl. dem Gewicht um 5 größer ist. Bei Kanten ohne Rückkanten verändert sich hingegen nichts.
  Eine Anwendung für diese Option sind Graphen, bei denen man an der Differenz zwischen den Beziehungen der Knoten interessiert ist. Ein Beispiel ist der Exportgraph in Abbildung 2.17.
  Hier ist es interessant zu visualisieren, welches Land einen Überschuss an Gütern produziert, also mehr exportiert als es importiert. Dies ist durch die Darstellung der absoluten Gewichte der Kanten nur schwer möglich, da alle Kanten ein hohes Gewicht haben und dadurch alle eine ähnliche Färbung besitzen. Dies wird erst durch die Visualisierung der Differenz deutlich dargestellt. Bei ausgeglichenem Verhältnis sind die Kanten gelb. Wenn die Differenz zwischen Import und Export zweier Länder groß ist, dann werden die Kanten rötlicher (Export des Ausgangsknoten kleiner als Import) oder grüner (Export des Ausgangsknoten größer als Import) dargestellt.

- Anzahl der Kanten: Es wird die Anzahl der Kanten als Maß genommen. Bei nicht-aggregierten Kanten ist dies also gleich eins. Bei aggregierten Kanten spiegelt das Gewicht wieder, wie viele Kanten diese eine zusammenfasst.
  Wenn man die Graphen im Standardbeispiel über die Zeit zu einem zusammenfasst und diese Option wählt, wird man an der Farbe der Kanten sofort erkennen, dass nicht alle Verbindungen im Netzwerk durchgängig vorhanden sind, wie zum Beispiel zwischen 134.96.7.179 und 136.199.55.175. Bei

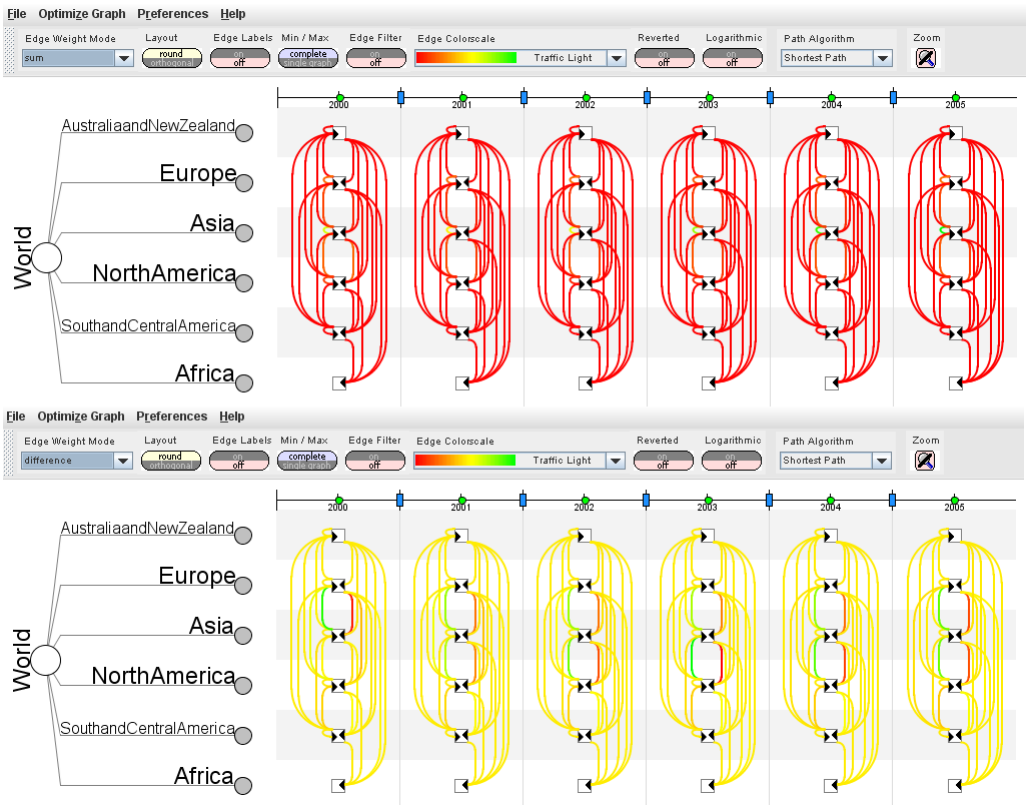

Abbildung 2.17: Exportgraph - oben: absolutes Gewicht - unten: Gewicht ist Differenz zwischen Hin- und Rückkante.

größeren Netzwerkverbindungsgraphen kann auf diese Weise eine Visualisierung der Verfügbarkeit erstellt werden.

- Kantengewicht = 1: Alle Kanten haben das Gewicht eins. Wenn man nur daran interessiert ist, welche Knoten mit welchen anderen Knoten in einem Intervall in Beziehung stehen und nicht, welches Gewicht sie besitzen, ist dies die optimale Einstellung. Alle Kanten sind gleich gefärbt und werden somit gleichberechtigt dargestellt.
  Die zweite Verwendungsmöglichkeit dieser Einstellung ist in Kombination mit den Pfadalgorithmen zu sehen. Bei kürzesten Pfaden wird so der Pfad errechnet, der über die kleinste Anzahl von Kanten zum Ziel führt (kürzester Pfad bei nicht gewichteten Graphen). Beim Algorithmus zur Berechnung des maximalen Flusses wird so jede Kante nur für eine Einheit benutzt und man kann herausfinden, wie viele Einheiten von Quelle zum

Ziel transportiert werden können, ohne eine Kante zweimal zu benutzen.

2.3.2 Farbskalen

Die Kantengewichte in der Zeittafel werden mit der Hilfe von Farben dargestellt. Dazu verwendet man Farbskalen. Eine Farbskala ist eine Funktion, die numerische Werte zwischen 0 und 1 auf eine Farbe abbildet. Um also Kantengewichte in Farben zu kodieren, benötigt man zunächst eine Funktion, die Gewichte auf das Intervall zwischen 0 und 1 abbildet. Dazu verwendet man den Wertebereich aller Gewichte (mehr dazu auf Seite 26). Die Kante mit dem niedrigsten Gewicht bekommt den Wert 0, diejenige mit dem höchsten Gewicht erhält den Wert 1 und dazwischen wird linear oder logarithmisch interpoliert.
So können relative Gewichte berechnet werden und auf eine Farbe mittels Farbskala abgebildet werden. Dabei ist allerdings zu beachten, dass die Farbskala die Werte geeignet darstellt. Besonders wichtig ist, dass sich die Farbe der niedrigen Werte stark von der Farbe der hohen Werte unterscheidet. Problematisch sind auch Farbskalen, die Farben enthalten, die sich kaum vom Hintergrund abheben. Viele Farbskalen sind so aufgebaut, dass sie beim Benutzer Assoziationen auslösen. Dies unterstützt die Wahrnehmung beim Verarbeiten der Visualisierung. Beispiele für solche Farbskalen sind die kalt-heiß-Skala (blau-rot), die Ampel-Skala (rot-gelb-grün), Ozean-Skala (schwarz-blau-weiß) und die Vegetations-Skala (blau-grün-gelb-rot). Nützlich sind aber auch Farbskalen, die einen hell zu dunkel (oder umgekehrten) Verlauf haben. Je mehr Kontrast eine Farbe zum Hintergrund hat, desto deutlicher wird sie wahrgenommen.

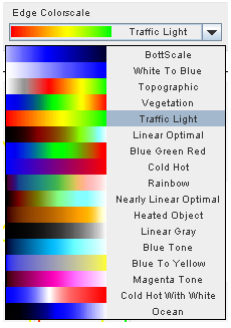

Abbildung 2.18: Combobox zur Auswahl der Farbskala.

In der Werkzeugleiste stehen viele verschiedene Farbskalen zur Auswahl bereit.

Wenn der Farbverlauf umgekehrt werden soll, kann dies mit der Schaltfläche „Reverted“ vorgenommen werden. Außerdem steht mit der Schaltfläche „Logarithmic“ zur Auswahl, ob man die Werte linear oder logarithmisch interpolieren will. Der Vorteil der logarithmischen Interpolation ist, dass sich niedrige Werte farblich deutlicher voneinander unterscheiden. Der Nachteil ist allerdings, dass sich hohe Werte weniger voneinander unterscheiden. Wenn sich also viele Variablen im unteren Wertebereich befinden und differenzierter dargestellt werden sollen, kann dies mit dieser Option geschehen. Wenn die Variablen allerdings gleichmäßig im Wertebereich verteilt sind oder sich sogar größtenteils im oberen Bereich befinden, sollte eine lineare Interpolation verwendet werden.

2.3.3 Pfadalgorithmen

Das Werkzeug ist in der Lage, zwei graphtheoretische Probleme zu berechnen und anschließend zu visualisieren. Der erste Pfadalgorithmus bestimmt die kürzesten Pfade zwischen einem Start- und einem Zielknoten in jedem Graph der Folge. Hierzu benutzt das Programm den Algorithmus von Bellman-Ford, da dieser im Gegensatz zum bekannten Kürzeste Wege-Algorithmus von Dijkstra auch mit negativen Katengewichten zurecht kommt und negative Kreise erkennt. Wenn solche Kreise im Graph vom Startknoten aus erreichbar sind, so ist das Problem nicht lösbar.
Nachdem nun die Pfade errechnet wurden (nicht immer existiert ein Pfad), werden sie auf dem Bildschirm dargestellt.
Hierzu werden alle Kanten, die nicht im Pfad vorhanden sind, hellgrau gezeichnet, wogegen die anderen ihre Farbe behalten, um hervorgehoben zu werden. Als zusätzliche Information wird um jeden Knoten im Pfad ein ringförmiger Balken eingeblendet, der die akkumulierte Pfadlänge relativ zur Länge des längsten berechneten kürzesten Pfades der Folge darstellt.
Der Längste der kürzesten Pfade der Folge hat also am Endknoten einen komplett ausgefüllten Ring. In Abbildung 2.20 ist ein ringförmiger Balken dargestellt, der 59% der längsten Pfadlänge repräsentiert.
Als Beispiel werden die kürzesten Pfade des Netzwerkgraphen zwischen Knoten 131.246.120.51 und 136.199.55.175 berechnet (Abb. 2.19). Dazu wird angenommen, dass die Gewichte die Nachrichtenlaufzeiten darstellen, um das Beispiel an

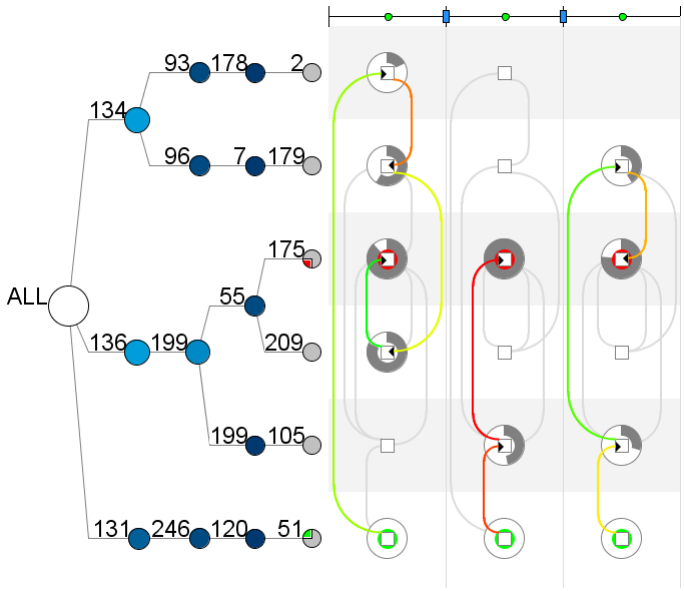

Abbildung 2.19: Kürzeste Pfade von 131.246.120.51 zu 136.199.55.175.

Abbildung 2.20: Ringförmiger Balken um den Knoten zeigt die akkumulierte Pfadlänge.

das Problem anzupassen. Man sieht deutlich, dass sich der schnellste Pfad zu jedem Zeitpunkt ändert und die kürzesten Pfade aus unterschiedlichen Kanten gebildet werden. In dem mittleren Graph befindet sich der längste der kürzesten Pfade (voller Ring), obwohl nur zwei Verbindungen benutzt werden. In den anderen Graphen ist ein Umweg über mehrere Kanten günstiger.

Der zweite Algorithmus, der zur Verfügung steht, berechnet den maximalen Fluss zwischen zwei Knoten. Dazu wird angenommen, dass die Gewichte der Kanten die maximale Kapazität darstellen. Auf dieser Basis lässt sich nun herausfinden, wie viele Einheiten maximal vom Startknoten (Quelle) durch den Graphen zum Zielknoten (Senke) transportiert werden können. Hierfür wird die Methode von Ford und Fulkerson benutzt. Diese sucht Pfade von Quelle zu Senke, an denen eine

Flussvergrößerung vorgenommen werden kann, bestimmt die maximale Erhöhung und aktualisiert den Fluss der Kanten entlang des Pfades. Der Algorithmus läuft solange, bis kein solcher Pfad mehr gefunden wird.
Wie auch bei kürzesten Pfaden werden alle Kanten, über die kein Fluss verläuft, hellgrau angezeigt. Die Kanten, die zum Transport verwendet werden, erhalten nun eine erweiterte Form. Sie werden nun dicker gezeichnet und mit zwei Farben versehen. Die äußere Farbe stellt die Kapazität der Kante dar und die innere Linie den Wert des Flusses, der über diese Kante gesendet wird. So hat der Benutzer die Information, wie stark eine Verbindung ausgelastet wird.
Mit dem ringförmigen Balken am Quellknoten wird der maximale Fluss dargestellt. Auch hier ist er relativ zum größten Gesamtfluss der Folge angegeben.

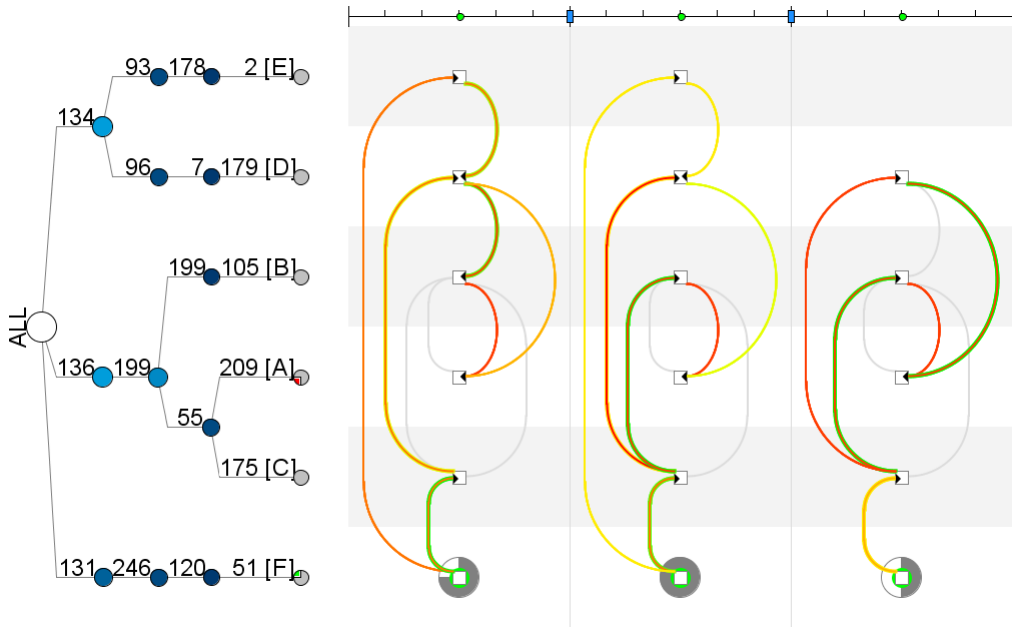

Abbildung 2.21: Maximaler Fluß von 131.246.120.51 zu 136.199.55.209.

Im Beispiel (Abbildung 2.21) wird der maximale Nachrichtenfluss von 131.246.120.51 zu 136.199.55.209 visualisiert. Die Gewichte der einzelnen Kanten stellen den maximalen Fluss der Verbindung dar. Der Algorithmus berechnet den maximalen Fluss durch das gesamte Netzwerk und beachtet dabei die maximale Last der einzelnen Kanten. Man sieht, dass nicht alle Kanten voll ausgelastet und deshalb zweifarbig dargestellt werden. Einige Kanten könnten deutlich mehr Nachrichten weiterleiten. Dies erkennt man an den grünen Außenfarben und den

roten oder orangenen inneren Farben. An dem komplett ausgefüllten Balken des Quellknotens im mittleren Graphen erkennt man den maximalen Nachrichtenfluss der Folge. Im letzten Graph ist nur halb soviel Fluss möglich.

2.3.4 Kantenfilter

Um die Anzahl der gezeichneten Kanten zu verkleinern, existiert im Werkzeug ein Filter, mit dem man den Wertebereich der Gewichte einschränken kann. Diese Kanten werden auch bei der Berechnung von Pfaden ignoriert. Oft trifft der Fall ein, dass man Kanten mit sehr kleinen Werten hat, die nicht viel zum Verständnis des Graphen beitragen, aber trotzdem mit visualisiert werden und zu sehr dichten Zeichnungen führen. Es wäre also sinnvoller, diese Kanten nicht mit anzuzeigen, weil sie vernachlässigbar sind. Dies kann mit dem Kantenfilter erreicht werden. Der Filter hat nicht nur die untere Schranke als Parameter sondern auch eine obere. Diese kann zum Beispiel benutzt werden, um unerwünschte Extremwerte herauszufiltern. Wenn man nur das gewöhnliche Verhalten des Graphen darstellen will, sind solche Ausreißer vernachlässigbar und können mit dem Filter entfernt werden.

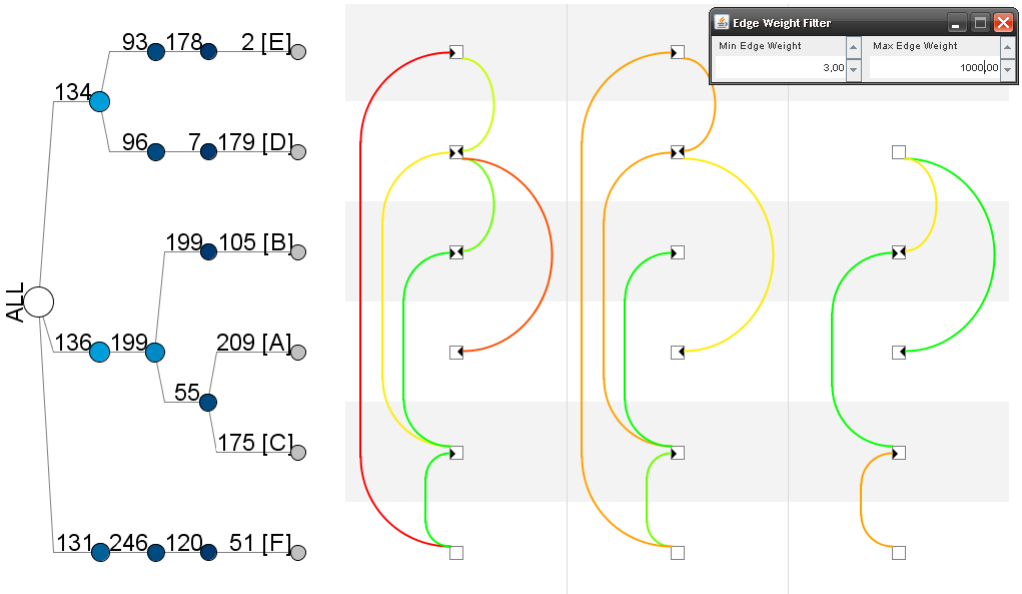

Abbildung 2.22: Kantenfilter.

In der Abb. 2.22 werden die Kanten mit Gewicht < 3 herausgefiltert. Verbindun-

gen mit niedrigem Nachrichtenaufkommen sollen also nicht angezeigt werden.

Die Schaltfläche „Edge Filter“ in der Werkzeugleiste aktiviert den Filter. Es öffnet sich daraufhin ein kleines Fenster in dem die Parameter eingestellt werden können. Beim Schließen des Fensters wird der Filter wieder deaktiviert.

2.3.5 Tooltips und Kantenbeschriftung

Oft will man detailliertere Angaben zur reinen Zeichnung der Daten bekommen und den Weg der reinen textuellen Daten zur Visualisierung wieder zurück verfolgen. Hierzu können bei Bedarf Informationen in Textform eingeblendet werden. Eine erste Variante sind Tooltips, die erscheinen, wenn der Mauszeiger längere Zeit auf einer Stelle verbleibt. Es wird dann neben dem Mauszeiger ein Textfeld mit Informationen zum Element unter dem Zeiger eingeblendet. Als Beispiel wird der Tooltip über einem Knoten in der Zeittafel erklärt. Er beinhaltet den Namen und den aktuellen Zustand (aktiviert oder deaktiviert) aber auch eine Beschreibung jeder ausgehenden Kante mit Anzahl (bei mehreren zusammengefassten Kanten) und das Gewicht.

Die zweite Variante ist eine dauerhaft angezeigte Beschriftung. Standardmäßig ist sie nur an den Knoten der Hierarchie sichtbar. Dort ist sie unerlässlich, um den Graphen analysieren zu können. Ohne Beschriftung der Hierarchie ist die Zuordnung der Knoten nicht nachvollziehbar. Bei Bedarf können auch Beschriftungen an den Kanten der Zeittafel eingeschaltet werden. Sie beinhalten das Gewicht und werden in der Höhe der halben Kante rechts positioniert. Die Schaltfläche namens „Edge Labels“ befindet sich in der Werkzeugleiste.

2.3.6 Zoom

Die Darstellung großer und dichter Graphen mit vielen Kanten bringt Probleme dergestalt mit sich, dass Kanten stellenweise sehr dicht aneinander gezeichnet werden müssen und dass viele Kreuzungen existieren. Der Benutzer hat Schwierigkeiten, diese Linien auseinander zu halten. Die eingebaute Zoomfunktion, die sich in der Werkzeugleiste mittels Schaltfläche ein- und ausschalten lässt, hilft solche Ballungsstellen der Zeichnung genauer zu betrachten, indem sie an der

Stelle des Mauszeigers eine Lupe simuliert.

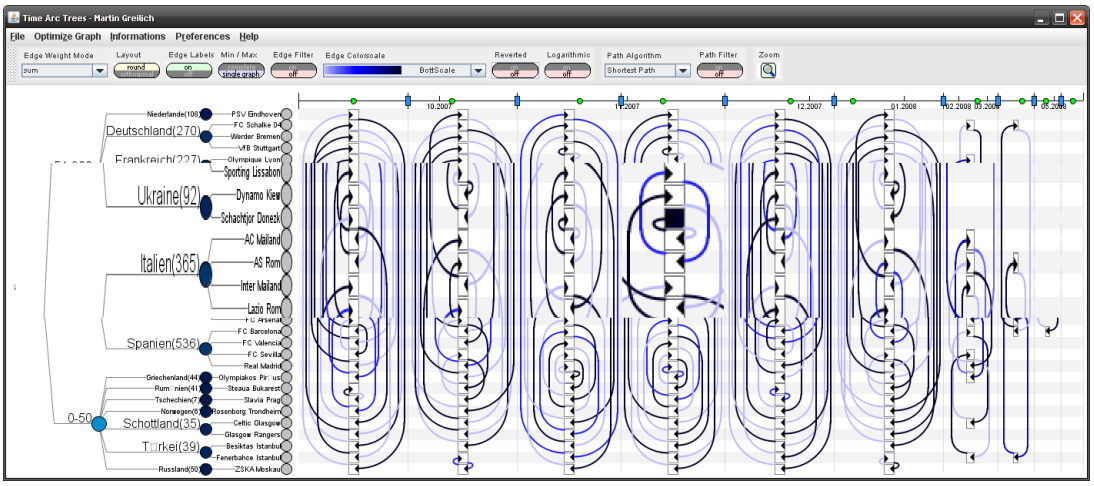

Abbildung 2.23: Zoomfunktion.

Der Zoomausschnitt wird in doppelter Größe dargestellt. Ein horizontaler Streifen auf der Höhe des Mauszeigers wird über die gesamte Breite des Fensters gestreckt gezeichnet, damit auch während der Benutzung die Zuordnung der Repräsentanten zu den korrespondierenden Hierarchieknoten zu erkennen ist.
Der Ausschnitt in dem Intervall unter dam Mauszeiger wird besonders genau dargestellt, indem auch in die Breite gestreckt wird.

2.3.7 sonstige Funktionen

In der Menüleiste unter dem Punkt „File“ befinden sich die Funktionen „Print“ und „Export“ mit denen man die aktuelle Darstellung der Graphen ausgeben kann. Beim Starten der Druckfunktion erzeugt das Werkzeug eine einseitige Ausgabe der Visualisierung an den Drucker im Querformat. Mit der Exportfunktion wird die Darstellung als PNG-Bilddatei an einem frei wählbaren Speicherort abgelegt.

Kapitel 3

Implementierung

Im vorherigen Kapitel wurde die Funktionsweise des TimeArcTrees-Werkzeugs beschrieben. Im nun folgenden Kapitel wird die eigentliche Implementierung erläutert. Als Programmiersprache wurde Java von Sun Microsystems gewählt. Java ist objektorientiert und plattformunabhängig. Darüber hinaus bietet Java noch diverse weitere Vorteile, die es als Implementierungssprache empfehlen, auf die hier jedoch nicht näher eingegangen wird.
Bei der Entwicklung wurde zuerst die Grundarchitektur ermittelt, welche die grobe Aufteilung des Systems beinhaltet. Aus dieser Grundaufteilung wurde dann ein Klassenentwurf hergeleitet, der alle wichtigen Komponenten enthält.
Bei der Implementierung der Klassen wurde schrittweise vorgegangen. Zuerst entstand ein Prototyp, der zwar schon die Visualisierung der Hierarchie und der Folge beherrschte, jedoch noch keine Möglichkeit der Manipulation bot. Aus diesem Prototyp konnten schon einige kritische Stellen der Visualisierung hergeleitet werden, wie z.B. die Überlappung der Kanten und der Platzverbrauch. Der Entwicklungsprozess bestand aus diversen Phasen in denen Funktionalität hinzugefügt oder die Visualisierungstechnik verfeinert wurde. In der letzten Phase wurden alle Funktionen getestet und kleine Fehler und Ungenauigkeiten in der Visualisierung entfernt.

In diesem Kapitel wird zunächst die Architektur vorgestellt. Es folgt die Klassenbeschreibung, aufgeteilt in die logischen Blöcke GUI, Informationshierarchie, Zeittafel und Hilfsklassen. Abschließend werden in „Ausgewählte Implementie-

rungsdetails“ einige Algorithmen zur Verbesserung der Visualisierung vorgestellt.

3.1 Architektur

Der Grobentwurf indentifiziert die wichtigsten Teile des Werkzeuges. Bei Visualisierungswerkzeugen bietet sich eine Aufteilung in Komponenten nach dem ModelViewController-Entwurfsmuster an, indem Daten-, Ansichts- und Kontrollklassen existieren. Das TimeArcTrees-Werkzeug ist in Benutzeroberfläche, Datenklassen der Informationshierarchie und Datenklassen der Zeittafel zerlegt. Zusätzlich existieren Kontrollklassen in Form von MouseListenern, die alle graphischen Komponenten der Benutzeroberfläche überwachen und Änderungen an die Datenklassen weiterleiten. Die Datenklassen passen dann ihre Eigenschaften an und benachrichtigen die Ansichtsklassen, dass eine Änderung eingetreten ist und eine Teilmenge oder die komplette Visualisierung neu gezeichnet werden muss.
Die Hauptklasse TimeArcTrees verbindet alle Komponenten und ist somit der Kern der gesamten Anwendung. Nach dem Programmstart wird nach einer Eingabedatei gefragt, die anschließend geladen wird. Diese Aufgabe übernimmt die Hilfsklasse *TATDataReader*, die den Text der Datei in einzelne Elemente zerlegt und diese zu den Hauptdatenklassen *HierarchyTree* und *Timetable* sendet. Dort geschieht der Aufbau der gesamten Datenstruktur. Es werden Instanzen der einzelnen Datenklassen erzeugt und miteinander verbunden. Im Hierarchieteil werden Objekte vom Typ *HierarchyNode*, sowie ihre Beschriftungen von Typ *HierarchyLabel* erzeugt und eine initiale Anordnung der Elemente auf der Oberfläche berechnet. In der Zeittafel wird zunächst aus allen Zeitpunkten eine Zeitleiste vom Typ *Timeline* und für jeden Zeitpunkt ein Intervall vom Typ *TimeInterval* erzeugt. Anschließend wird für jede Kante der Folge eine Instanz vom Typ *TimetableEdge* erzeugt. Dazu werden die Start- und Zielknoten der Klasse *TimetableNode* im Intervall gesucht und in der Kante referenziert. Falls sich noch keine Instanzen der Knoten im Intervall befinden, werden sie nun erzeugt. Jeder Knoten enthält Referenzen auf alle ausgehenden Kanten.

Nachdem der Datensatz vollständig geladen wurde, wird die Visualisierung auf dem Bildschirm ausgegeben. Die graphische Benutzeroberfläche wurde mit dem

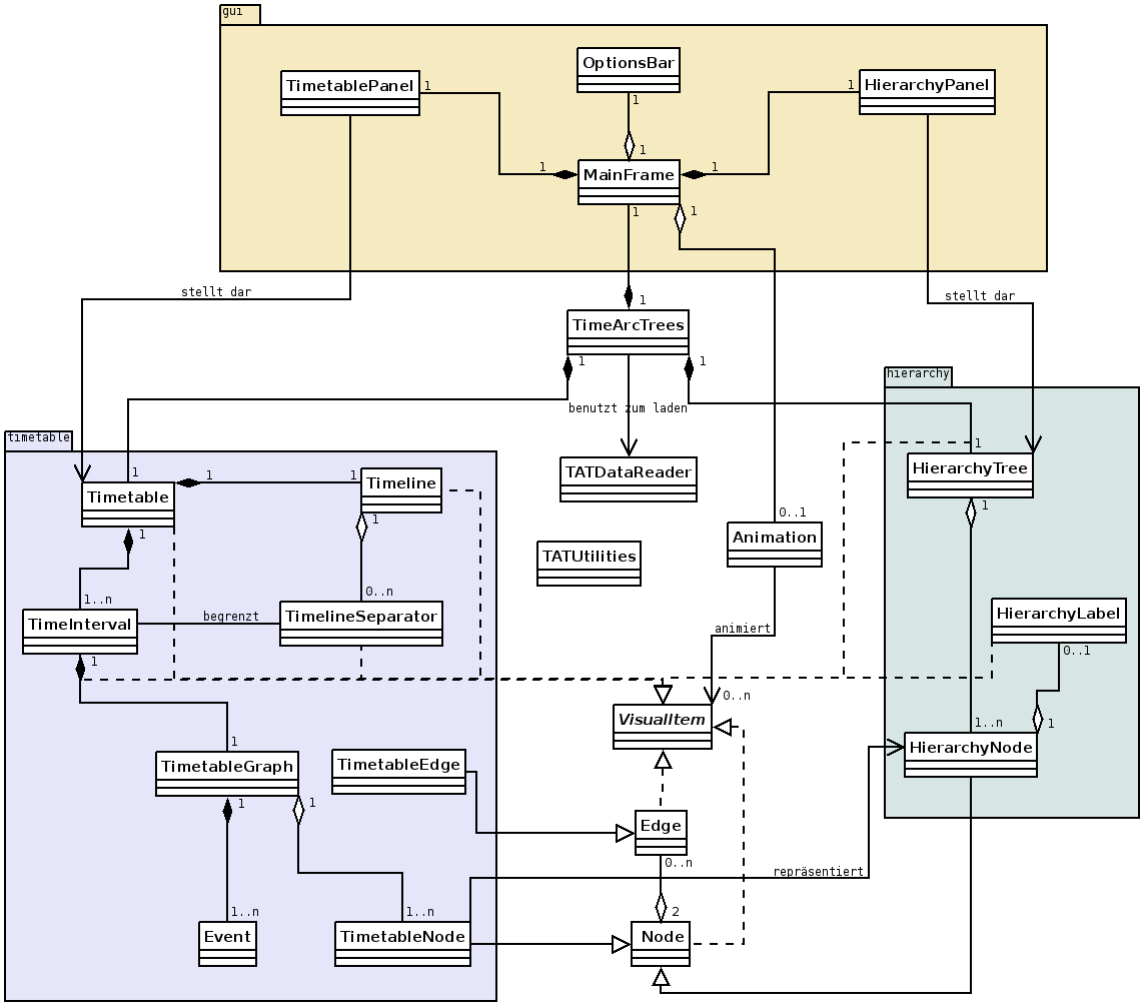

Abbildung 3.1: UML-Klassendiagramm mit den wichtigsten Klassen.

in Java mitgelieferten Paketen AWT und Swing erzeugt. Mit dem Abstract Window Toolkit (AWT) kann auf einfache Art und Weise eine Ereignisbehandlung für die Benutzerinteraktion implementiert werden, wogegen das Swing-Toolkit die Möglichkeit, plattformunabhängige graphische Benutzeroberflächen zu erstellen, bietet. Für die TimeArcTrees-Oberfläche werden u.a. Schaltflächen, Auswahlboxen und Menüs benötigt, die mit dem Toolkit schnell realisiert werden können. Das Hauptfenster der Oberfläche ist vom Typ *MainFrame*, welches von JFrame abgeleitet ist. Es enthält die Werkzeugleiste *OptionsBar* und die Zeichenflächen *HierarchyPanel* und *TimetablePanel*, die beide von JPanel abgeleitet sind und Zeichenmethoden der Informationshierarchie bzw. Zeittafel implementieren.

Alle Datenobjekte sind Bestandteile der Visualisierung und haben damit alle Eigenschaften, die für die Ausgabe auf dem Bildschirm benötigt werden, gemeinsam. Dies wurde realisiert, indem alle Klassen die abstrakte Klasse *VisualItem* implementieren.

In der Abbildung 3.1 ist das UML-Klassendiagramm[1] des Programms dargestellt. Es enthält die wichtigsten Klassen inklusive Einteilung in die drei Pakete *gui*, *timetable* und *hierarchy*. Man erkennt die hierarchische Architektur innerhalb der Pakete mit den Hauptklassen *MainFrame*, *Timetable* und *HierarchyTree*, die wiederum aus mehreren einzelnen Komponenten bestehen. Alle Klassen werden im nächsten Abschnitt kurz beschrieben.

3.2 Klassenbeschreibung

- **TimeArcTrees:** Diese Klasse verbindet die wichtigsten Programmteile miteinander. Sie lädt bei Programmstart alle Einstellungen und sorgt für die Initialisierung der Hierarchie und der Zeittafel.
- **VisualItem:** Diese Klasse ist eine vereinfachte Version von *Component* und dient als Oberklasse aller auf der Zeichenfläche befindlichen Elemente, wie Knoten und Kanten. Sie speichert deren Flächen innerhalb des Zeichenpanels und sorgt damit, dass alle Elemente an der korrekten Position und in korrekten Größen dargestellt werden. Außerdem ist sie für die Kommunikation zwischen Maus und Objekten auf der Zeichenfläche verantwortlich.

3.2.1 Die GUI-Klassen

- **MainFrame:** Die Klasse des Hauptfensters erweitert die Klasse *JFrame*. Dieses enthält die Zeichenfläche der Hierarchie und der Zeittafel sowie die Werkzeugleiste. Zusätzlich hat das Fenster eine Menüleiste. Sie registriert alle Veränderungen der visualisierten Elemente und leitet die Neuzeichnung der betroffenen Teile ein.

[1] Damit es nicht unübersichtlich ist, werden Felder und Methoden nicht dargestellt.

- **HierarchyPanel:** Die Visualisierung der Informationshierarchie geschieht durch diese Klasse. Sie ist abgeleitet von *JPanel*. Die Zeichenmethode zeichnet die Knoten, Kanten und Labels der Hierarchie.

- **TimetablePanel:** Sie ist ebenfalls abgeleitet von der Klasse *JPanel* und stellt die komplette Zeittafel inklusive Zeitleiste dar.

- **OptionsBar:** Die Klasse der Werkzeugleiste wird abgeleitet von *JToolBar*. Sie beinhaltet Schaltflächen und Auswahlboxen zur Manipulation der Visualisierung.

3.2.2 Die Klassen der Informationshierarchie

- **HierarchyNode:** Die Knoten in der Hierarchie werden mit einem *HierarchyNode*-Objekt beschrieben. Diese Klasse ist eine Spezialisierung der Klasse *Node*, die Referenzen zu allen ausgehenden Kanten hält und auch für Knoten der Teiltafel benutzt wird. Sie enthält alle Eigenschaften, die für alle korrespondierenden Knoten in der Zeittafel gleichzeitig gelten. Außerdem ist jeder Knoten dafür verantwortlich, Veränderungen im Layout an die untergeordneten Knoten weiter zu leiten.

- **HierarchyTree:** Ein *HierarchyTree*-Objekt repräsentiert die gesamte Informationshierarchie und enthält alle Eigenschaften, die für den ganzen Baum gelten. Von hier aus wird die Berechnung des Layouts gestartet und versucht, die Kantenkreuzungen in der Zeittafel durch umsortieren der Knoten zu verringern.

- **HierarchyLabel:** Die Beschriftungen der Hierarchieknoten werden mit dieser Klasse realisiert. Die Eigenschaften sind z.B. Orientierung, Schriftgröße und der angezeigte Text.

3.2.3 Die Klassen der Zeittafel

- **Timetable:** Die gesamte Zeittafel wird durch eine Instanz der Klasse *Timetable* beschrieben. Sie besteht aus der Zeitleiste und den Zeitintervallen

inklusive derer Graphen. Alle Eigenschaften, die für jedes Zeitintervall gelten, sind hier untergebracht. Beim Auslösen einer Änderung der Visualisierung kümmert sich diese Klasse um eine Neuberechnung der Attribute oder den Neustart der Pfadalgorithmen.

- **Timeline:** Die Zeitleiste bestimmt mithilfe von Separatoren die Intervalle, die in der Zeittafel angezeigt werden. Hier werden auch alle Zeitpunkte der Graphen registriert, die für die Bildung von Intervallen benötigt werden. Es stehen Methoden bereit, die Intervalle vereinen, trennen und die Breite der entsprechenden Zeichenflächen ändern.

- **TimelineSeparator:** Die Separatoren sind Trennmarkierungen zwischen Intervallen. Jedem Separator ist das Intervall zugeteilt, welches sich an der linken Seite befindet. Als Eigenschaften hat ein Separator die aktuelle Position, sowie die Grenzen, in denen er verschoben werden kann.

- **TimeInterval:** Die Zeitleiste wird in Intervalle geteilt. Die Klasse speichert die zeitlichen Grenzen, die enthaltenen Zeitpunkte der Graphen, einen Zeiger auf das nächste Intervall sowie den Graphen, der im Intervallbereich abgebildet werden soll. Jeder Graph benötigt eine Minimalbreite, die somit auch Bedingung für die Breite des Intervalls ist.

- **TimetableNode:** Die Knoten der Zeittafel werden durch Instanzen der Klasse *TimetableNode* erzeugt. Sie erweitert die Klasse *Node*, die u.a. alle ausgehenden Kanten enthält, um einige Zeichenmethoden.

- **TimetableEdge:** Die Klasse *TimetableEdge* erbt die Eigeschaften der Klasse *Edge* und repräsentiert die Kanten in der Zeittafel.

- **Graph:** Diese Klasse enthält komplette Graphen eines Zeitpunktes oder einer Zeitspanne. Alle im Graphen vorhandenen Knoten werden hier gespeichert. Außerdem beinhaltet sie die Algorithmen zur Berechnung der kürzesten Pfade und der maximalen Flüsse.

- **Event:** Ein *Event*-Objekt stellt einen Zeitpunkt dar, zu dem Graphen im Datensatz existieren. Die Klasse besteht aus einem Zeitstempel, dem Graphen des Zeitpunktes und eventuell einem Kommentar.

- **EventMap:** Die Klasse *EventMap* speichert chronologisch geordnete Referenzen zu allen Events.

3.2.4 Die Hilfsklassen

- **TATUtilities:** Diese Klasse enthält statische Hilfsmethoden zur Berechnung der Farben aus Farbskalen, Datumsausgaben und den Algorithmus zur Berechnung der überlappungsfreien Positionen der Kanten (Eine genaue Erklärung des Alg. befindet sich im Kapitel 3.3.1).

- **TATDataReader:** Hier befindet sich der Parser für die Eingabedatei. Er liefert alle Relationen inklusive Zeitangaben und Kommentaren.

- **MenuFactory:** Die Menüleiste wird mithilfe dieser Klasse und einer *Property*-Datei erzeugt. In der Datei *menu.properties* befinden sich alle Eigenschaften wie Menüstruktur, Beschriftungen und andere. Aus den Einträgen der Datei formt die *MenuFactory* die Menüleiste des Werkzeuges.

3.2.5 Sonstige Klassen

- **HierarchyMouseListener / TimetableMouseListener:**
  Die MouseListener-Klassen kümmern sich um die Eingaben der Maus auf der Hierarchie- und Zeittafelfläche. Außerdem sind sie für die Fokussierung der Elemente unter dem Mauszeiger zuständig.

- **ColorScaleIcon:** Für die Auswahl von Farbskalen in Auswahlboxen werden deren Piktogramme benötigt. Diese Klasse implementiert die Schnittstelle *Icon*.

- **Colors:** In der Klasse *Colors* befinden sich alle verfügbaren Farbskalen der Anwendung. Sie werden in einem zweidimensionalen Feld definiert. Zusätzlich existiert ein Feld mit den Namen der Skalen.

- **Animation:** Alle Instanzen der Klasse *VisualItem* können animiert werden. Diese Aufgabe übernimmt die *Animation*-Klasse, indem sie Eigenschaften der sichtbaren Elemente manipuliert.

- **AnimationObject und AnimationJob:** Sobald ein Element der Visualisierung, also eine Instanz die von der Klasse *VisualItem* erbt animiert werden soll, wird eine Instanz der Klasse *AnimationObject* erzeugt. Diese enthält außer der Referenz auf das Objekt eine Referenz auf einen *AnimationJob*. Dieser beschreibt den Typ der Animation mit deren Parametern.

3.3 Ausgewählte Implementierungsdetails

3.3.1 Berechnung der überlappungsfreien Anordnung der Kanten

Je nach Art der Zeichnung des Graphen wird die enthaltene Information besser oder schlechter zum Ausdruck gebracht. Deshalb wurden ästhetische Kriterien [Biedl et al., 1998] [Purchase, 2002] [Ware et al., 2002] [Taylor and Rodgers, 2005] [Bennett et al., 2007] festgelegt, welche die Lesbarkeit von Graphen erhöhen sollen. Eines dieser Kriterien ist die überlappungsfreie Anordnung der Verbindungen. Es ist klar, dass bei Kreuzungen eine Kante über die andere gezeichnet werden muss. Mit überlappungsfreier Anordnung ist aber gemeint, dass Kanten im vertikalen Verlauf nicht über anderen dargestellt werden.
Die Kanten sollen den größtmöglichsten Abstand zueinander haben, um sie besser auseinander halten zu können. Normalerweise wird dieses Problem gelöst, indem man die Knoten und Kanten so anordnet, dass diese Bedingung möglichst optimal erfüllt wird. In dem TimeArcTrees-Ansatz ist man allerdings darauf beschränkt, nur die Anordnung der Kanten variieren zu können, da Knoten eine feste Position besitzen. Bei der Anordnung der Kanten ist man ebenfalls eingeschränkt, da sich Abwärtskanten auf der rechten Seite und Aufwärtskanten auf der linken Seite befinden müssen.
Ein erster naiver Ansatz des Problems ist, die Kanten je nach Distanz zwischen Start- und Zielknoten zu zeichnen und zwar so, dass die Weite des Bogens mit weiterem Abstand der Knoten größer wird. So haben alle Kanten durch das konstante Verhältnis von Länge zu Weite eine homogene Form und es wird gewährleistet, dass jede Kante eine andere höchstens einmal kreuzt. Dem Benutzter wird durch die Distanz der Kante zur Vertikalen der Knoten der ungefähre Abstand der Knoten vermittelt. Durch das Wissen der Aufteilung nach Aufwärts- und

Abwärtskanten ist auch klar, in welche Richtung die Kante verläuft.

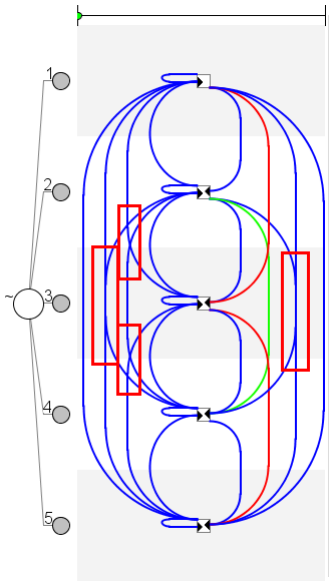

Abbildung 3.2: Naiver Ansatz für vollständigen Graph mit fünf Knoten.

Dieser naive Ansatz hat allerdings einen entscheidenden Nachteil. Wie in Abbildung 3.2 in den roten Rechtecken dargestellt, entstehen Stellen, an denen sich Kanten in vertikaler Richtung überlappen. Deren Distanzen zwischen Start- und Zielknoten sind gleich, sodass der Startknoten der einen zwischen den beiden Knoten der anderen liegt. Zum Beispiel kann nicht unterschieden werden, ob rechts die Kanten von 1 zu 5 und von 2 zu 4 laufen, oder (wie es richtig wäre) von 1 zu 4 und 2 zu 5.
Um dieses Problem zu umgehen, muss die Bedingung, dass eine gleiche Distanz der Knoten auch zur gleichen Weite der Kante führt, etwas aufgeweicht werden. Die Verbindungen, die gleiche Distanz der Knoten haben und sich kreuzen (wie im Beispiel beschrieben), müssen unterschiedliche Bogenweiten zugewiesen bekommen. Die Bedingung, dass Verbindungen mit höherer Distanz der Knoten eine höhere Distanz zur Vertikalen haben, soll bestehen bleiben.
In der Implementierung bekommt jede Kante mit der Methode `setRelWidth()` eine relative Breite in ihrer Intervallhälfte zugewiesen. Um Überlappung zu vermeiden, den Platz nicht zu verschwenden und dabei noch die Bedingung, dass alle Kanten maximalen Abstand zueinander haben sollen, zu berücksichtigen, muss ein Mechanismus implementiert werden, der die nötigen Werte berechnet. Dies

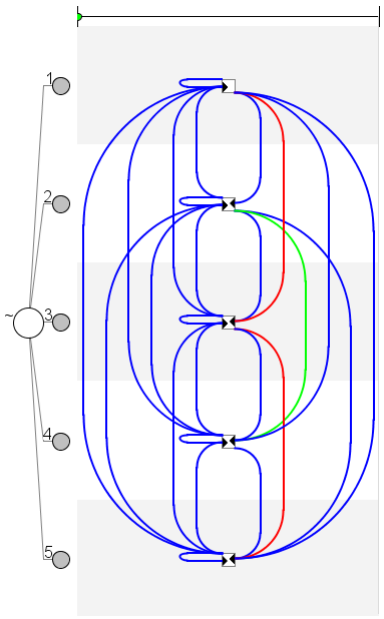

Abbildung 3.3: Vollständiger Graph mit fünf Knoten ohne vertikalen Überlappungen.

wird mit der statischen Methode `initEdgePositions(int n)` umgesetzt. Sie berechnet das dreidimensionale Feld `edge_widths`, in dem man mit der Anzahl der Knoten und der Zeilennummer der Start- und Zielknoten die Bogenweite erhalten kann.
In der Methode wird zunächst für jede mögliche Distanz zwischen Start- und Zielknoten (1 bis $AnzahlZeilen - 1$) die Anzahl der unterschiedlichen Bogenweiten berechnet. Bei Distanz 1, also bei Verbindungen zu direkten Nachbarn in der Vertikalen, ist nur eine unterschiedliche Weite nötig, da keine Kreuzungen vorkommen können. Bei Distanz 2 sind es schon zwei. Allerdings ist die Anzahl nicht linear zur Distanz, da die Anzahl der möglichen Kombinationen bei langen Kanten wieder abnimmt. Für die maximale Distanz $AnzahlZeilen - 1$ gibt es nur eine mögliche Kante, also wird nur eine Bogenweite benötigt. Die Funktion f_1 zur Berechnung ergibt sich wie folgt:

- z = Zeilenanzahl
- l = Länge der Kante in Differenz der Zeilen
- $a(z, l)$ = Anzahl der Kanten der Länge l bei z Zeilen

- $f_1(z,l) = \left\{ \begin{array}{c} \frac{a(z,l)}{\lceil a(z,l)/l \rceil} \text{ wenn } a(z,l) < l \\ l \text{ sonst} \end{array} \right\}$

Die Werte für unser Beispiel mit 5 Knoten sehen folgendermaßen aus:

l	a(5,l)	$f_1(5,l)$
1	4	1
2	3	2
3	2	2
4	1	1

Nun muss man nur noch für jede Kante der gleichen Distanz eine dieser unterschiedlichen Bogenweiten zuordnen. Dies geschieht mithilfe der Modulo-Funktion in der Funktion f_2.

- $s =$ oberste Zeile der Kante
- $f_2(z,l) = s \mod f_1(z,l)$

So entsteht eine vertikal überlappungsfreie Anordnung der Kanten, wie es in Abbildung 3.3 für einen vollständigen Graphen mit fünf Knoten dargestellt wird. Die grüne und die roten Kanten haben die gleiche Distanz der Knoten, werden aber unterschiedlich weit gezeichnet.

3.3.2 Berechnung eines komprimierten Layouts der Hierarchie

Einer der kritischen Punkte bei der Visualisierung von Folgen von Graphen als Node-Link Diagramm ist der Platzverbrauch. Je mehr Platz zur Verfügung steht, desto mehr Graphen können gleichzeitig auf dem Bildschirm dargestellt werden. Wie groß die Zeichenfläche ist, hängt vor allem von der Auflösung des Bildschirms ab. Um die Fläche der Visualisierung der Folge zu vergrößern, muss in einem anderen Bereich an Größe gespart werden. Die Maße der Menü- und Werkzeugleisten sind fix. Der einzige Bereich mit variabler Größe ist die Informationshierarchie. Zwar wird die Höhe durch die Fensterhöhe bestimmt, die Breite bietet aber Einsparpotenzial.

Der Hierarchiebaum wird so gezeichnet, dass alle Knoten der gleichen Schicht im Baum (gleiche Kantenanzahl im Pfad zur Wurzel) auf einer Vertikalen angeordnet sind. Die Beschriftung der Knoten befindet sich vor jedem Knoten. Da Überlappung vermieden wird, ist jede Schicht im Baum mindestens so breit wie der breiteste Knoten inklusive Beschriftung. Die Fläche der Hierarchie hängt von der Fläche der einzelnen Schichten und somit von den Beschriftungen und den Knotendurchmessern ab.
In der Implementierung werden die Durchmesser der Knoten durch die Variablen `minDiameter` und `maxDiameter` der Klasse *HierarchyNode* bestimmt. Sie geben an, wie groß die Blätter und die Wurzel gezeichnet werden. Für die Knoten dazwischen wird der Durchmesser je nach Anzahl der Knoten im Unterbaum linear interpoliert. Diese Werte können in den Einstellungen geändert werden.
Die Beschriftungen werden mit der Klasse *HierarchyLabel* realisiert. Sie sind standardmäßig waagerecht vor dem Knoten angeordnet und haben die Schriftgröße `maxFontSize`.
Wenn die Hierarchie mit diesen Standardwerten nun im Verhältnis zur Visualisierung der Folge zu groß dargestellt wird, so kann das Programm die Anordnung, Schriftgröße und angezeigten Text solange manipulieren, bis entweder die berechnete Optimalbreite der Hierarchie erreicht wird oder die Hierarchie gänzlich ohne Beschriftungen angezeigt wird.
Um dies zu realisieren wurden Platzoptimierungsstufen eingeführt. Die Enumeration hierfür heißt `SpaceOptimization` und befindet sich in der Klasse *HierarchyNode*. Die Optimierungsstufen sind `NO`, `VERTICAL_ABOVE`, `VERTICAL_MIDDLE`, `SMALLER_FONTS`, `ABBREVIATED_TEXT` und `NO_TEXT`. Es wird zuerst versucht, den Text über den Knoten anzuordnen. Wenn dies nicht ausreicht, wird versucht den Text um 90° zu drehen. Danach folgt die Verkleinerung der Schrift und falls dies immer noch nicht ausreicht, wird nur eine Abkürzung des Textes gezeichnet oder schließlich die Beschriftung ganz weggelassen.
Der Algorithmus durchläuft jede der Stufen zunächst schichtweise im Baum. Es wird also z.B. zuerst versucht für alle Schichten im Baum die Beschriftungen abzukürzen, bevor begonnen wird die Texte ganz wegzulassen.
Die entsprechende Methode heißt `optimizeSpace()` und ist in der Klasse *HierarchyTree* untergebracht. Sie wird beim Berechnen des Layouts so oft aufgerufen, bis eine der oben genannten Bedingungen zutrifft.

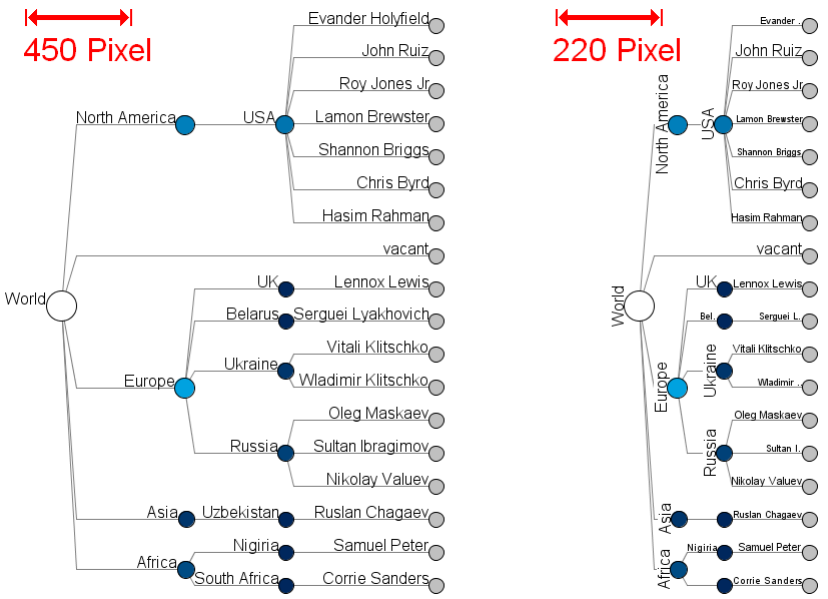

Abbildung 3.4: Maximale und komprimierte Hierarchieausgabe.

In der Abbildung 3.4 sieht man die standardmäßige Version einer Hierarchie im Vergleich zur komprimierten. Man spart hier 230 Pixel auf dem Bildschirm. Bei einer Bilschirmauflösung von 1024×768 ist dies etwa ein Viertel der Bildschirmbreite. Man erkennt, dass der Algorithmus die letzte Stufe erreicht und die langen Beschriftung der Knoten *World/Africa/South Africa* und *World/Africa/Uzbekistan* entfernt hat. Andere werden mit einer kleineren Schriftart oder abgekürzt dargestellt.

3.3.3 Minimierung der Anzahl der Kantenkreuzungen und der Kantenlängen

Im Kapitel „Berechnung der überlappungsfreien Anordnung der Kanten“ wurden die Kriterien für die Lesbarkeit eines Graphen erwähnt. Ein weiteres Kriterium ist, den Graphen so anzuordnen, dass möglichst wenige Kantenkreuzungen [Biedl et al., 1998] [Harel, 1998] gezeichnet werden. Aber um dies zu erreichen, müsste wiederum die Anordnung der Knoten und Kanten manipuliert werden. Durch die Konvention, dass alle Kanten nach ihrer Richtung auf rechte und linke Seite eingeteilt werden, ist ein Wechsel der Seiten zur Vermeidung von Kreuzungen nicht

möglich. Innerhalb der Konvention ist durch die in Kapitel 3.3.1 beschriebene Vorgehensweise garantiert, dass eine minimale Anzahl von Kreuzungen vorliegt. Die einzige Möglichkeit weitere Kreuzungen zu entfernen, ist eine neue Reihenfolge der Knoten auf der Vertikalen. Diese wird aber durch die Anordnung der Knoten in der Hierarchie bestimmt.
Die Minimierung der Überschneidungen ist also nur durch eine andere Reihenfolge der Blätter der Hierarchie möglich und wirkt sich auf alle Graphen der Folge aus. Die Suche nach einer neuen Reihenfolge ist durch die Struktur der Hierarchie eingeschränkt, da keine Kantenkreuzungen im Hierarchiebaum erlaubt sind. Es kann also jeweils nur die Reihenfolge der Kinder jedes Knotens geändert werden. Für einen Knoten mit n Kindern existieren $n!$ viele Möglichkeiten, die Kinder anzuordnen. Bei einer Hierarchie mit 50 Knoten, die alle an der Wurzel hängen, gibt es 50! Kombination, die überprüft werden müssten. Dies ist auch bei einer effizienten Implementierung und schnellen Rechnern nicht in einer kurzen Zeitspanne zu bewerkstelligen. Das Problem kann auf das NP-vollständige Optimal Linear Arrangement Problem(OLAP) reduziert werden und ist deshalb nur näherungsweise innerhalb einer maximalen Zeitspanne zu lösen.
Im TimeArcTrees-Werkzeug wird die Minimierung durch ein Umsortieren der Kinder jedes Knotens ausgeführt. Um Überschneidungen effizient aufzufinden, wird aus der aktuellen Reihenfolge der Knoten und der Graphen eine Adjazenzmatrix der Kanten abgeleitet. Als Indizes wird die aktuelle Zeile des Startknotens, des Zielknotens und die Folgenummer des Graphen benutzt. Die Einträge der Matrix sind vom Typ `Integer` und geben die Anzahl der Kanten von Startknoten zum Zielknoten an (0 oder 1).
Für den Graphen in der Abbildung 3.5 sieht die Matrix[2] also folgendermaßen aus:

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

[2]Man beachte, dass alle 1er Einträge sich oberhalb der Diagonalen der Matrix befinden, da im Beispiel alle Kanten von oben nach unten laufen.

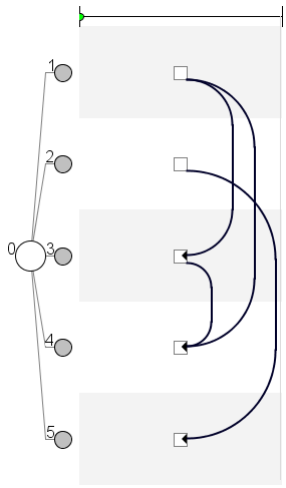

Abbildung 3.5: Bedingung für Kantenkreuzungen.

Um nun alle Kreuzungen aus dieser Matrix zu bestimmen, muss zunächst eine Bedingung für Kantenkreuzungen hergeleitet werden:

Eine Kante $e1$ kreuzt eine Kante $e2$ gleicher Richtung, wenn der Startknoten von $e1$ zwischen und der Zielknoten von $e1$ nicht zwischen Start- und Zielknoten der Kante $e2$ liegt. Wenn die Richtungen von $e1$ und $e2$ verschieden sind, kann keine Kreuzung vorliegen.

Zwar führt der umgekehrte Fall ebenfalls zu einer Überschneidung, diese wird aber nicht berücksichtigt um alle Kreuzungen nur einmal zu zählen.
Die Bedingung kann nun in der Matrix angewendet werden, um alle Kreuzungen zu finden. Für jeden Eintrag mit dem Wert 1 wird in einem Bereich der Matrix nach Überschneidungen gesucht. Der Bereich lässt sich mithilfe der Indizes der Einträge auf eine Teilmenge der Matrix beschränken.
Als Beispiel wird die erste 1 in der Beispielmatrix betrachtet. Sie befindet sich in der ersten Zeile in der dritten Spalte. Somit läuft die Kante vom ersten zum dritten Knoten. Eine Kante, die diese Kante schneidet, müsste also nach der Bedingung im zweiten Knoten starten und unter dem dritten Knoten enden. In der Matrix sind dies also alle Einträge in der zweiten Zeile deren Spaltennummer

höher als drei ist.
Auf diese Weise können alle Kreuzungen des Graphen zusammengezählt werden. Wenn nun bei einer Vertauschung festgestellt wird, dass eine kleinere Anzahl von Kreuzungen besteht, wird diese Reihenfolge beibehalten. Sonst geht das Vertauschen mit der ursprünglichen Reihenfolge der Kinder weiter.
So wird, angefangen bei der Wurzel, mittels Tiefendurchlauf bis zu den Blättern vorgegangen. Die Methode `minimizeCrossings()` der Klasse HierarchyTree wird so oft durchgeführt, bis keine Verbesserung mehr eintritt. Die Vertauschungen finden innerhalb der Methode `minimizeCrossings(int[][][] adj_matrix)`, die sich in der Klasse HierarchyNode befindet, statt. Dazu müssen alle Blätter im Unterbaum der Kinder bestimmt werden (`nodeMinChild` und `nodeMaxChild`). Für den Tausch zweier Knoten müssen alle Zeilen / Spalten der Blätter im Unterbaum der Knoten in der Matrix getauscht werden.
Der Quelltext der verwendeten Methoden befidet sich im Anhang.

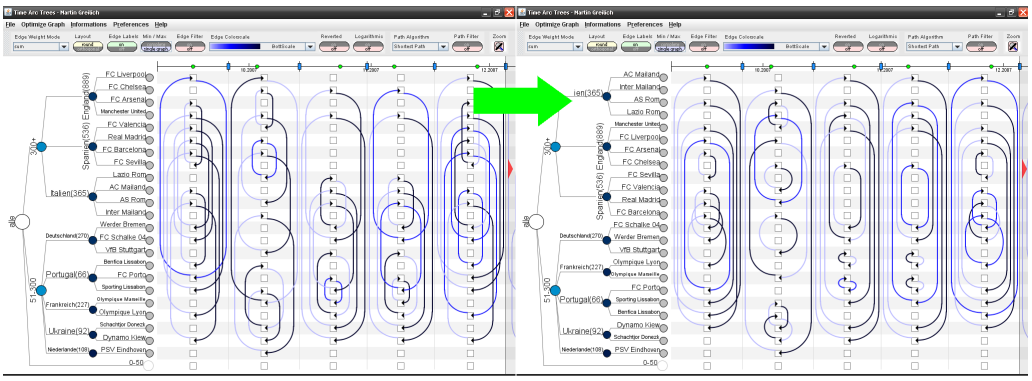

Abbildung 3.6: Ein Graph vor und nach der Minimierung der Anzahl von Kantenkreuzungen.

In Abbildung 3.6 sieht man einen Graphen vor und nach der Minimierung[3] der Anzahl der Kantenkreuzungen. Sie ist im gesamten Graphen[4] um 65% von 136 auf 48 gesunken.

Die Minimierung der Kantenlänge erfolgt mit dem selben Verfahren wie die Mi-

[3]Bei der Minimierung im Beispiel wird eine Kombination mit der Kantenlängenminimierung benutzt - diese erzielt ein besseres Ergebnis.

[4]Weil die Folge viele Graphen enthält, ist nur ein Teil der Visualisierung sichtbar.

nimierung der Anzahl der Kantenkreuzungen. Der Algorithmus liefert ebenfalls keine optimale Lösung sondern nur eine Näherung. Als Maß wird die Differenz zwischen Zeilennummer des Start- und Zielknotens genommen. Diese lässt sich ebenfalls mithilfe der Adjazenzmatrix herausfinden indem die Differenz zwischen Spalten- und Zeilenindex gebildet wird. Ein Tausch der Reihenfolge wird nur durchgeführt, wenn die Gesamtlänge der Kanten dadurch kleiner wird.

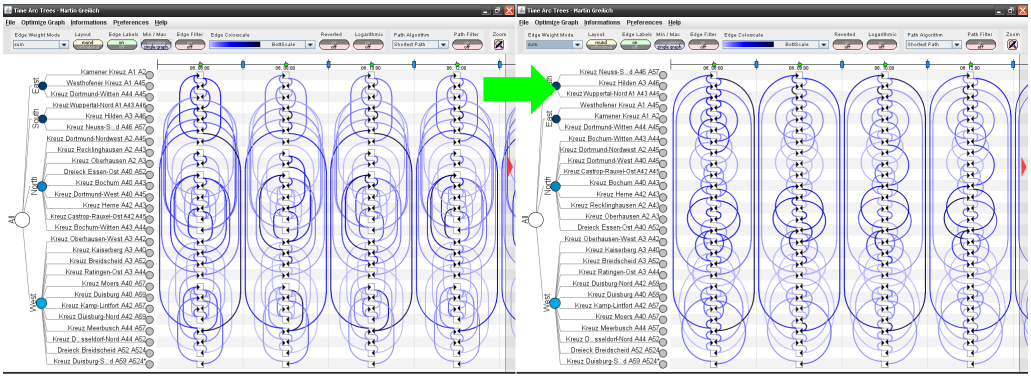

Abbildung 3.7: Ein Graph vor und nach der Verkürzung der Kanten.

In Abbildung 3.7 ist die Anwendung der Längenminimierung auf einen größeren Graphen zu sehen. Der Wert der Gesamtlänge sinkt um 26% von 2734 auf 2012.

Kapitel 4

Anwendungen

In diesem Kapitel werden einige konkrete Anwendungen der TimeArcTrees-Visualisierung vorgestellt. Dazu wurden Datensätze aus dem Bereich Fußball und Verkehr erstellt und visualisiert. Unter der Benutzung der im Kapitel „Das TimeArcTrees-Werkzeug“ vorgestellten interaktiven Funktionen, wie Aggregation über die Zeit, Zusammenfalten von Teilbäumen oder Pfadberechnung, werden einzelne Abbildungen erstellt und analysiert.

4.1 Aggregation: Fußballergebnisse Champions League

Der erste Datensatz beschreibt Fußballergebnisse der UEFA Champions League aus der Saison 2007/08. Die Champions League ist ein Wettbewerb für die besten europäischen Vereinsmannschaften. Alle 125 Spiele des Wettbewerbs sind im Datensatz als Folge von Graphen kodiert. Knoten stellen die 32 teilnehmenden Vereine dar und Kanten repräsentieren Spiele. Das Gewicht der Kanten ist die Anzahl der erzielten Punkte. Wenn Mannschaft A gegen Mannschaft B gewonnen hat, so existiert in der Datei eine Kante von A nach B mit dem Gewicht +3 und eine Kante von B nach A mit dem Gewicht 0. Bei Unentschieden haben beide Kanten das Gewicht +1.
Die Mannschaften unterliegen einer Hierarchie. Jede Mannschaft hat ihr Land als Oberknoten, denn einige Länder haben mehrere Mannschaften im Wettbewerb. Die Länder wurden wiederum drei verschiedenen Oberknoten zugeordnet,

welche die finanzielle Stärke der Länder repräsentieren. Die finanzielle Stärke der Länder wird gemessen in „Ausgaben für neue Spieler aller Erstligavereine vor der Saison(in Millionen Euro)“. Es wurden drei Gruppen gebildet: 0-50, 50-300 und 300+. Die finanzstärksten Vereine kommen aus England(889 Mio. Euro), Spanien(536) und Italien(365). Deutschland(270) befindet sich in der mittleren Gruppe. Die Einteilung wurde so gewählt, dass in jeder Gruppe ungefähr gleich viele Vereine vertreten sind.
Die einzelnen Spieltage bilden die Folge. Am Anfang befindet sich die Gruppenphase mit 6 Spieltagen, gefolgt von den K.O.-Runden bis zum Finale. Die komplette Folge der Graphen[1] wird in Abbildung 4.1 visualisiert.
Diese Darstellung sagt noch nicht viel aus. Man kann lediglich einzelne Partien heraussuchen und nachschauen, wer Punkte erzielt hat. Deswegen werden nun einige Funktionen des Werkzeuges angewendet, um die Daten etwas mehr unter die Lupe zu nehmen.
Zunächst werden einige Intervalle zusammengefasst, da man aus einzelnen Spieltagen keine relevanten Schlüsse ziehen kann. Aus den 6 Spieltagen der Gruppenphase wird ein Intervall gebildet, das alle Spiele darstellt. Das Gleiche wird mit der K.O.-Phase gemacht.
Die Gruppierung nach Finanzstärke soll nun ausgenutzt werden, um herauszufinden, wie sich die Spielereinkäufe auf die Spielergebnisse auswirken. Dazu werden die Ergebnisse der drei finanzstärksten Länder untereinander und den unteren zwei Finanzgruppen verglichen. Hierfür werden nun Teilbäume der Hierarchie zusammengeklappt, so dass der Baum nur noch fünf Blätter besitzt: England, Spanien, Italien, Gruppe 0-50 und Gruppe 50-300.
Die entstandenen Graphen zeigen nun, wie viele Punkte die Vereine der Länder oder Gruppen gegeneinander erzielt haben. Man muss allerdings beachten, dass jeder der Knoten eine unterschiedliche Anzahl von Vereinen repräsentiert und die Anzahl der Spiele gegeneinander verschieden ist. Deshalb sind die Farben in der Form nicht aussagekräftig. Es fanden in der Gruppenphase 20 Spiele zwischen Vereinen der Gruppen 0-50 und 50-300, aber nur zwei Spiele zwischen italienischen und spanischen Vereinen statt. Um die Kanten nun sinnvoll miteinander vergleichen zu können, muss das Kantengewicht von Summe auf Durchschnitt

[1] Wegen der Übersicht wurden alle Kanten mit Gewicht 0 herausgefiltert. Es werden also nur Kanten zu Mannschaften gezogen, gegen die gepunktet wurde.

Abbildung 4.1: Champions League-Graph - vollständig.

der Punkte gesetzt werden(Auswahlbox „Edge Weight Mode“). Jetzt stellen die Farben die durchschnittliche Anzahl der erzielten Punkte dar und geben Auskunft, wie gut oder schlecht die Gruppen (Vereine der Gruppen) gegeneinander abgeschnitten haben. Die entstandene Ausgabe ist in Abbildung 4.2 zu sehen[2].
Die Visualisierung bestätigt, dass finanzstarke Vereine erfolgreicher waren. Dies zeigt sich besonders in dem Graphen der Gruppenphase. Hier sind alle Kanten von Reicheren zu Ärmeren gelb bis grün. England, mit den größten Ausgaben für neue Spieler, hat nur grüne ausgehende Kanten. Man erkennt auch, dass der Unterschied zwischen den unteren Gruppen 0-50 Mio. und 50-300 Mio. nicht mehr so groß ist, wie der Unterschied zwischen den Top-Ländern und der Gruppen 50-300 und 0-50. Die Kante von 50-300 zu 0-50 ist nur gelb-grün.
Nachdem die Hälfte der Vereine ausgeschieden war, sind die Verhältnisse in den K.O.-Runden ausgeglichener. Die Tendenz hält jedoch; je reicher desto erfolgreicher. Einzig die Spanier durchbrechen diese Regel. Sie unterlagen den Italienern und haben zur Gruppe 0-50 Mio. lediglich ein ausgeglichenes Verhältnis.

4.2 Kürzeste Pfade: Staudatensatz vom Autobahnnetz Ruhrgebiet

Das Werkzeug bietet die Möglichkeit, kürzeste Pfade in den einzelnen Graphen der Folge zu berechnen. Diese Funktion soll nun anhand des Staudatensatzes vom Autobahnnetz Ruhrgebiet vorgeführt werden. Die Knoten des Datensatzes stellen Verbindungspunkte mehrerer Autobahnen dar (Autobahndreiecke und Autobahnkreuze) und die Kanten repräsentieren die Autobahnabschnitte zwischen den Verbindungspunkten. Die Kanten beinhalten als Gewicht die geschätzte Fahrzeit durch den Abschnitt. Den Fahrzeiten wurden Stauzeiten hinzugerechnet, die anhand einer Staumeldungsanalyse eines gesamten Tages ermittelt wurden. Pro Kilometer Stau jedes Abschnittes wurden pauschal eine konstante Anzahl von Minuten zur staufreien Fahrzeit addiert. Die Daten wurden von 6 bis 20 Uhr alle zwei Stunden aufgezeichnet und ergeben daraus eine Folge von 8 Graphen.
Die Knoten wurden geographisch in vier Himmelsrichtungen aufgeteilt und in die

[2]Es wird die Ampel-Farbskala benutzt - rot für niedrige, gelb für mittlere und grün für hohe Werte

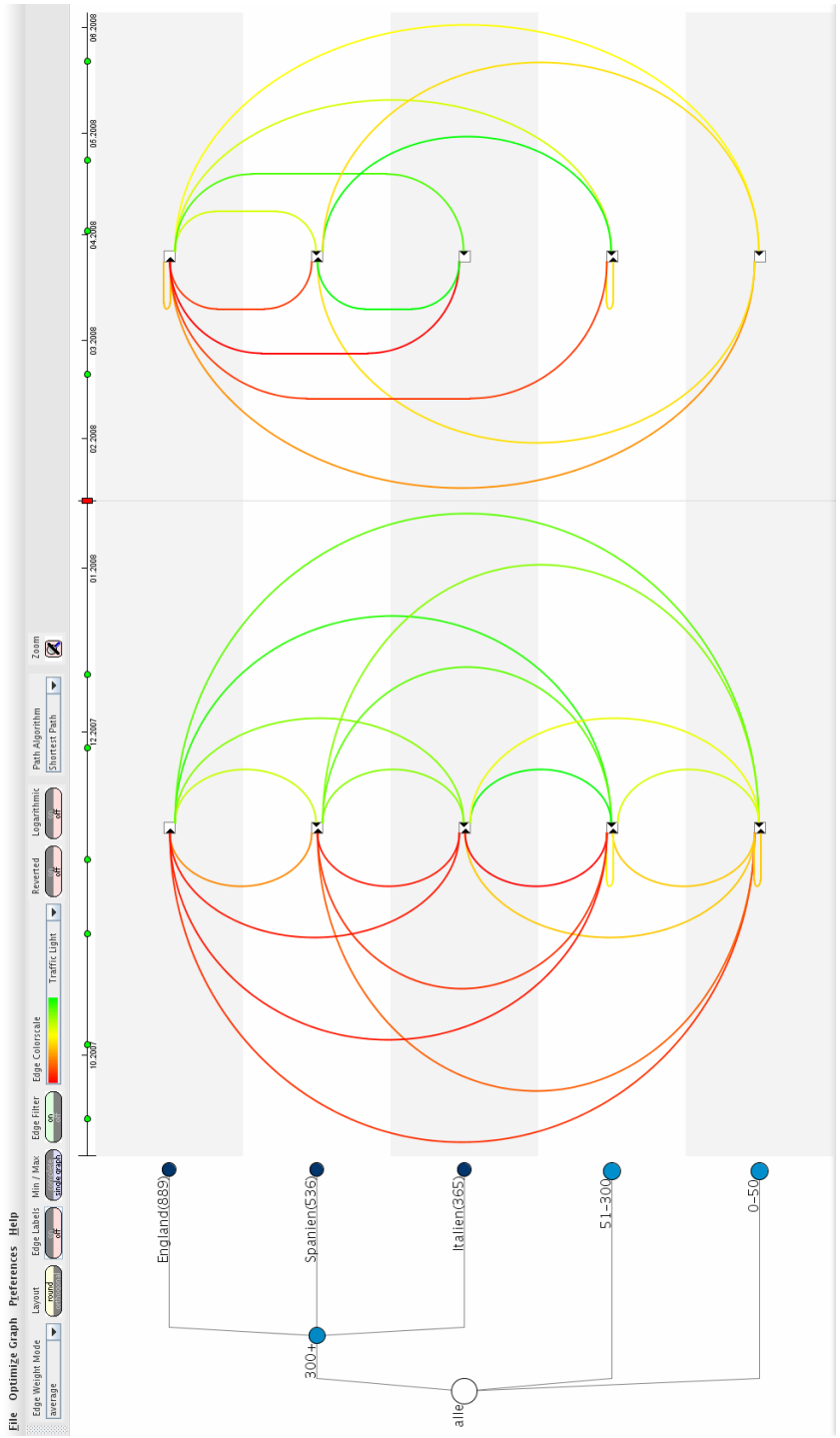

Abbildung 4.2: Champions League-Graph - Vergleich finanzieller Gruppen.

Hierarchie unter den Knoten North, East, South und West eingebunden. Die Abbildung 4.3 zeigt die Karte des aufgezeichneten Ausschnittes des Autobahnnetzes mit der Einteilung in Himmelsrichtungen.

Abbildung 4.3: Autobahnnetz Ruhrgebiet - Unterteilung in Himmelsrichtungen.

In der Abbildung 4.4 sieht man die TimeArcTrees-Visualisierung der Verbindungen. Es wird eine Vegetationsfarbskala benutzt. Die Farbe blau deutet eine staufreie Verbindung an, grün bedeutet wenig Stau, gelb langer Stau und rot sehr langer Stau. Jede Kante hat natürlich auch eine Rückkante, da Autobahnen in beide Richtungen befahrbar sind.
Nun soll dieser Datensatz genutzt werden, um den kürzesten Pfad (schnellste Verbindung) vom Knoten „Kreuz Meerbusch“ zum Knoten „Kamener Kreuz“ zu berechnen und zu visualisieren. Dazu wird in der Auswahlbox „Path Algorithm“ die Option „Shortest Path“ gewählt und der Start- und Zielknoten in der Hierarchie markiert. Die entstandene Ausgabe ist in Abbildung 4.5 zu sehen[3].
Die berechneten Pfade unterscheiden sich im Laufe des Tages enorm. Um 6 Uhr, wenn alle Autobahnen noch staufrei sind, sollte die nördliche Route benutzt wer-

[3]Graphen von 12 und 14 Uhr wurden aus Platzgründen zu einem aggregiert, da zwischen diesen beiden nur ein minimaler Unterschied existiert.

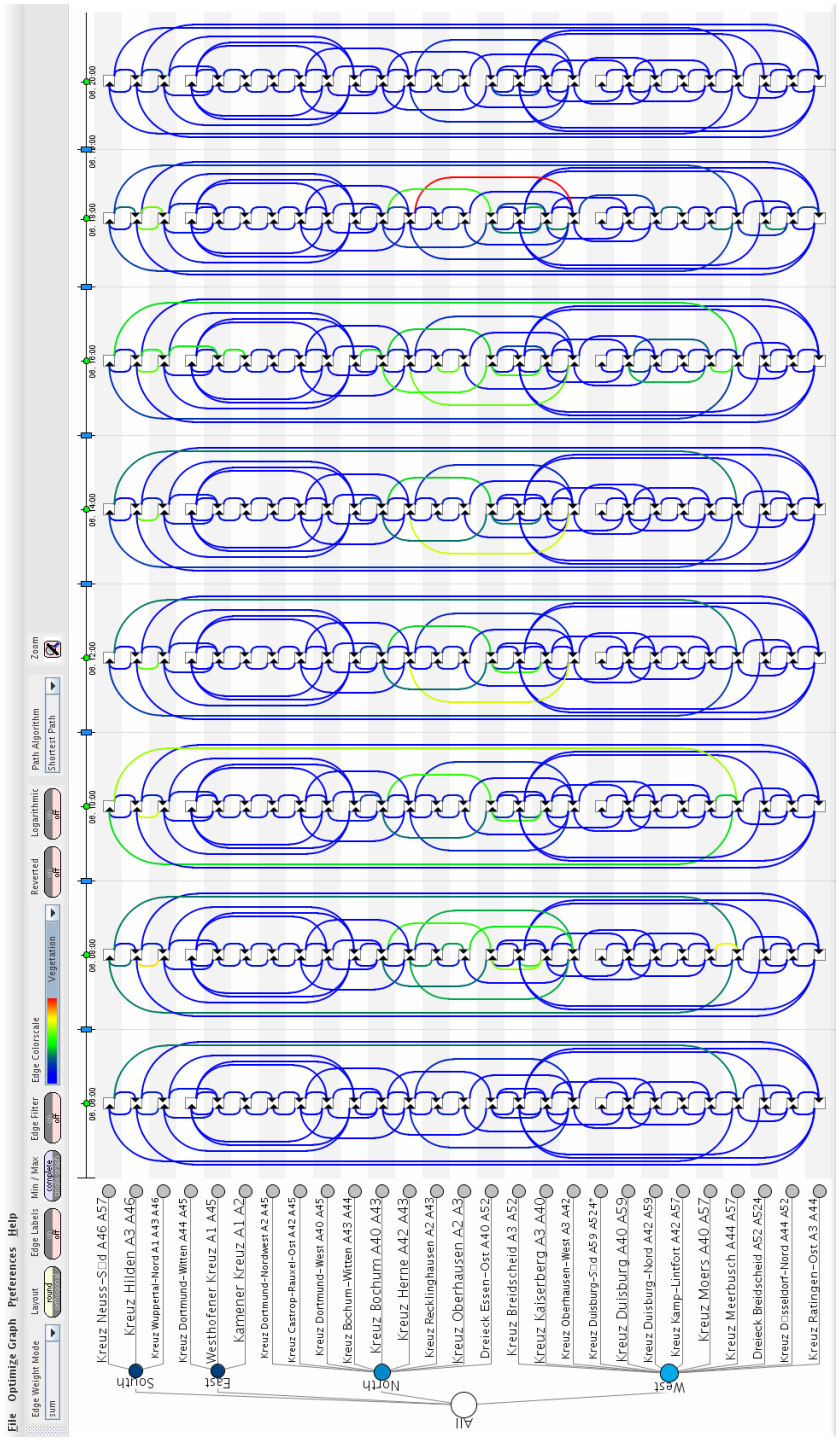

Abbildung 4.4: Autobahnnetz Stau-Graph - vollständig.

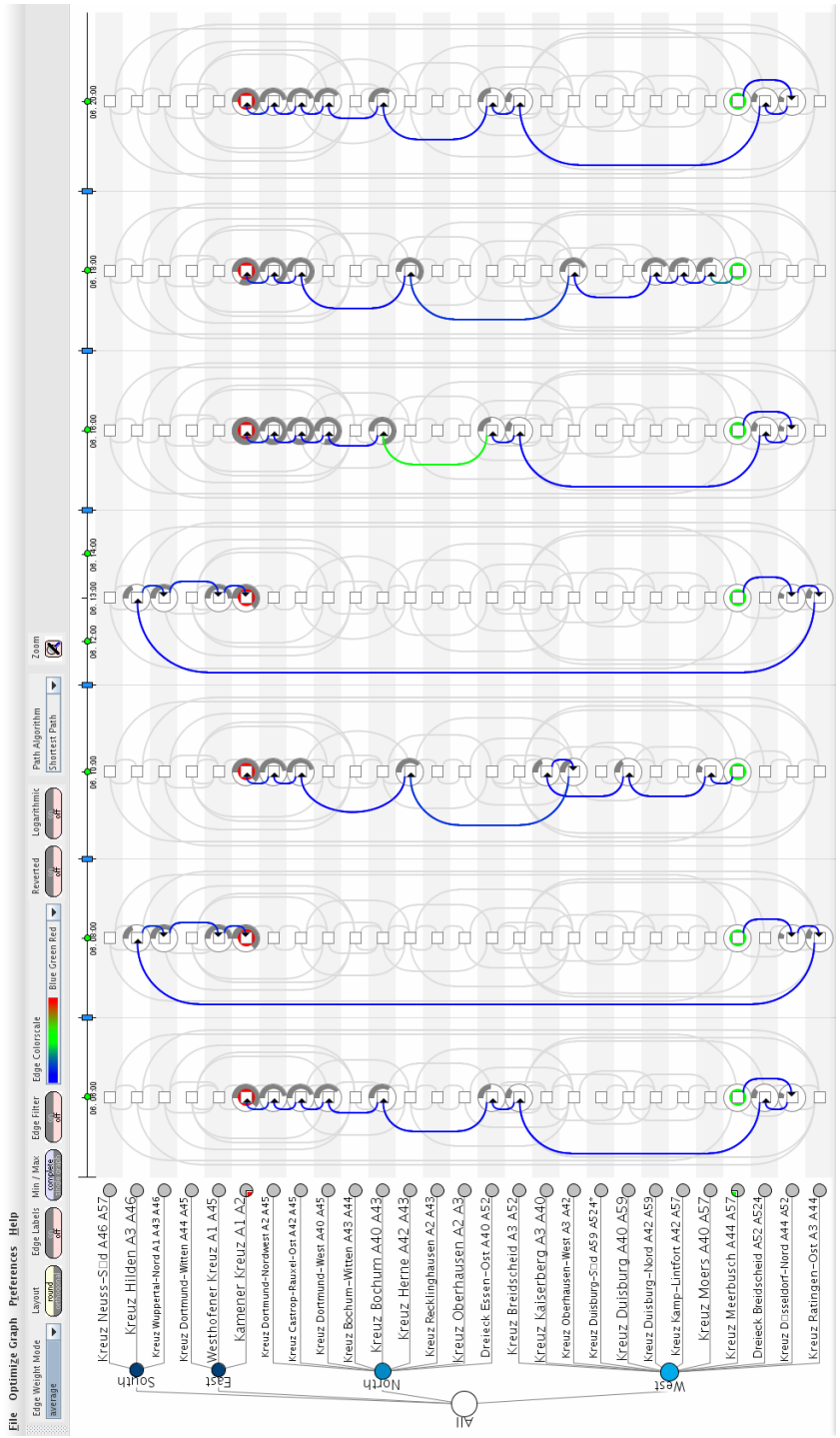

Abbildung 4.5: Schnellste Route - Visualisierung als Folge von Graphen.

den. Um 8 Uhr und 12-14 Uhr ist die nördliche Route allerdings nicht die schnellste und deshalb sollte über den südlichen Pfad gefahren werden. Um 16 Uhr sind alle Routen mit Verkehr überlastet und man sollte die kürzeste Route trotz des Staus zwischen „Dreieck Essen-Ost“ und „Kreuz Bochum“ benutzen. Dies erkennt man an dem vollen Ring um den Zielknoten. Die Fahrzeit ist mit 102 min fast doppelt so lang als um 6 Uhr (62 min). Nach 16 Uhr nimmt der Verkehr wieder ab.

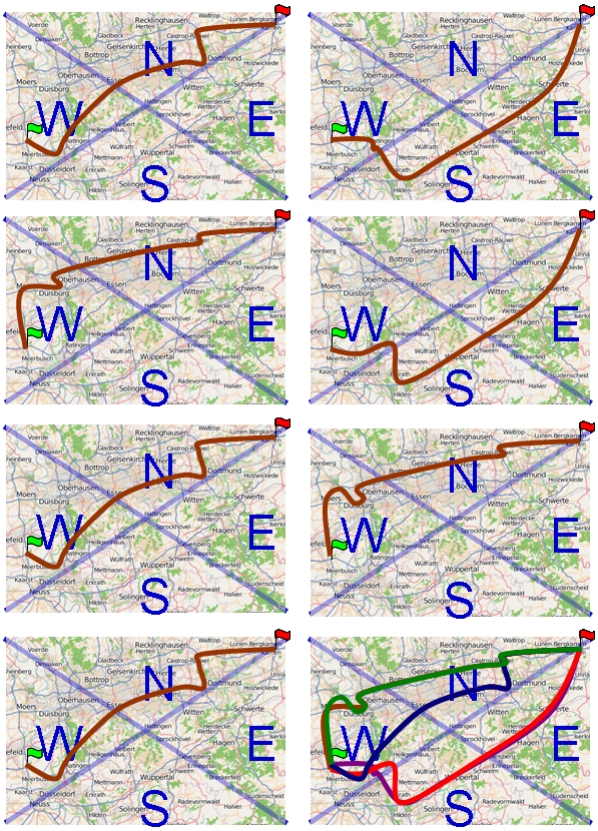

Abbildung 4.6: Schnellste Route - Visualisierung in Karten eingezeichnet.

Die Abbildung 4.6 zeigt die gleichen Routen eingezeichnet in Karten. Unten rechts sind alle Routen in einer einzigen Karte eingezeichnet. Der Vorteil der Routendarstellung in einer Karte ist, dass der geographische Bezug erhalten bleibt. Allerdings benötigt die Visualisierung der Pfade in einzelnen Karten große Bildschirmflächen. Die Visualisierung in einer einzelnen Karte führt zu unübersichtlichen Darstellungen. Eine dritte Möglichkeit wäre die Darstellung der einzelnen

Karten als Animation. Dies führt aber zu einer hohen kognitiven Belastung.
Die TimeArcTrees-Darstellung verliert gegenüber der Kartendarstellung den geographischen Bezug. Der Vorteil ist aber ein effizienter Platzverbrauch und ein hoher Grad an Interaktivität.

4.3 Große Datensätze: Fußballspiel Deutschland-Portugal

Ein weiterer Datensatz beschreibt ein Fußballspiel und stellt ein Beispiel für eine Anwendung mit großen Datenmengen dar. Es handelt sich hierbei um das Viertelfinalspiel Deutschland gegen Portugal bei der Fußball Europameisterschaft 2008. Es umfasst die ersten 30 min und enthält alle Pässe, Schüsse und sonstige Ereignisse, die während des Fußballspieles auf dem Platz geschehen. Alle Spieler und alle möglichen Ereignisse, wie Eckball, Abseits oder Tor, sind Knoten der Hierarchie. Sie werden in logische Gruppen in der Hierarchie geordnet. In der obersten Stufe befinden sich die Knoten der beiden Mannschaften und der Knoten „Ereignisse“. In den beiden Knoten der Mannschaften wird weiter nach Mannschaftsteilen Torwart, Abwehr, Mittelfeld und Angriff gruppiert. Die Ereignisse werden ebenfalls in sinnvollen Gruppen in der Hierarchie eingebunden.
Jeder Spielzug beginnt an einem Ereignisknoten und führt zu einem Spieler. Jeder Pass wird ebenfalls als Kante modeliert und enthält die Passlänge als Gewicht. Die Kanten, die zu Ereignissen führen, denen keine Länge zugeordnet werden kann (z.B. Foulspiel), enthalten das Gewicht +1. Das Ende eines Spielzuges wird mit einer Kante vom letzten ballführenden Spieler zu einem Ereignis gebildet.
Jeder Graph der Folge stellt einen Spielzug dar. Insgesamt sind 99 Graphen in der Folge. Dies führt zum ersten Problem der Visualisierung dieses Datensatzes. Es ist nicht möglich, die gesamte Folge der Graphen gleichzeitig auf dem Bildschirm darzustellen. Zwar sind durchschnittlich nur 7 Kanten in jedem Graph, dennoch benötigt jeder Graph mindestens 30 Pixel damit er korrekt gezeichnet werden kann. Wer sich trotzdem für die genauen Graphen interessiert, hat die Möglichkeit, mit zwei Scrollflächen den aktuellen Teilausschnitt nach links bzw. rechts zu verschieben.

Ein anderes Problem ist die Größe der Hierarchie. Der komplett aufgeklappte Baum enthält 51 Blätter. Dadurch liegen alle Knoten eng beieinander und die Zuordnung der Knoten in der Zeittafel zu den korrespondierenden Knoten der Hierarchie ist schwierig.
Ein Ausschnitt mit komplett aufgeklappter Hierarchie ist in Abbildung 4.7 zu sehen[4].
Mit diesem Datensatz soll nun eine Visualisierung der durchschnittlichen Passlänge erstellt werden. Dazu werden zunächst die Intervalle 0-15 min und 15-30 min gebildet. Da Ereignisse keine Rolle bei der Passanalyse spielen, wird der komplette Unterbaum am Knoten „Ereignisse“ deaktiviert. Jeder Mannschaftsteil beider Teams wird zusammengefasst. Der daraus entstandene Graph zeigt die aufsummierten Passlängen zwischen und innerhalb der Mannschaftsteile. Wenn der Gewichtsmodus nun auf Durchschnitt gestellt wird, so entsteht eine Ausgabe, wie sie auch in Abbildung 4.8 zu sehen ist. Rote Kanten repräsentieren kurze Pässe und grüne lange Pässe.
Welche Schlüsse können nun aus dieser Visualisierung hergeleitet werden? Besonders interessant sind Kanten, die im Spiel nach vorne entstehen. Es sind die Kanten von Abwehr zu Mittelfeld, von Abwehr zu Angriff und Mittelfeld zu Angriff. Am Graphen kann man erkennen, ob eine Mannschaft eher ein Kurzpassspiel bevorzugt oder ob sie mit langen Bällen zum Erfolg kommen will. Je nach Taktik der Mannschaften kann sich dieses Verhalten während eines Spieles mehrfach ändern.
In dieser Begegnung spielten die Deutschen in den ersten 15 Minuten eher kurze Pässe nach vorne und die Portugiesen versuchten, vom Mittelfeld mit langen Pässen in den Angriff zu spielen. Diese Taktik ändert sich nun in den zweiten 15 Minuten. Jetzt spielen die Portugiesen kurze Pässe und die Deutschen längere Pässe nach vorne.
Auch das Passverhalten innerhalb von Mannschaftsteilen kann interessante Erkenntnisse liefern. Werden kurze Pässe gespielt wie im Mittelfeld der Deutschen, deutet es an, dass ein enges Mittelfeld vorliegt. Bei langen Pässen, wie im Mittelfeld der Portugiesen oder der deutschen Abwehr, liegen die taktischen Positionen weiter auseinander und man versucht, mit vielen Querpässen Raum zu gewinnen.

[4]Auf der linken und rechten Seite der Zeittafel sieht man die Scrollflächen.

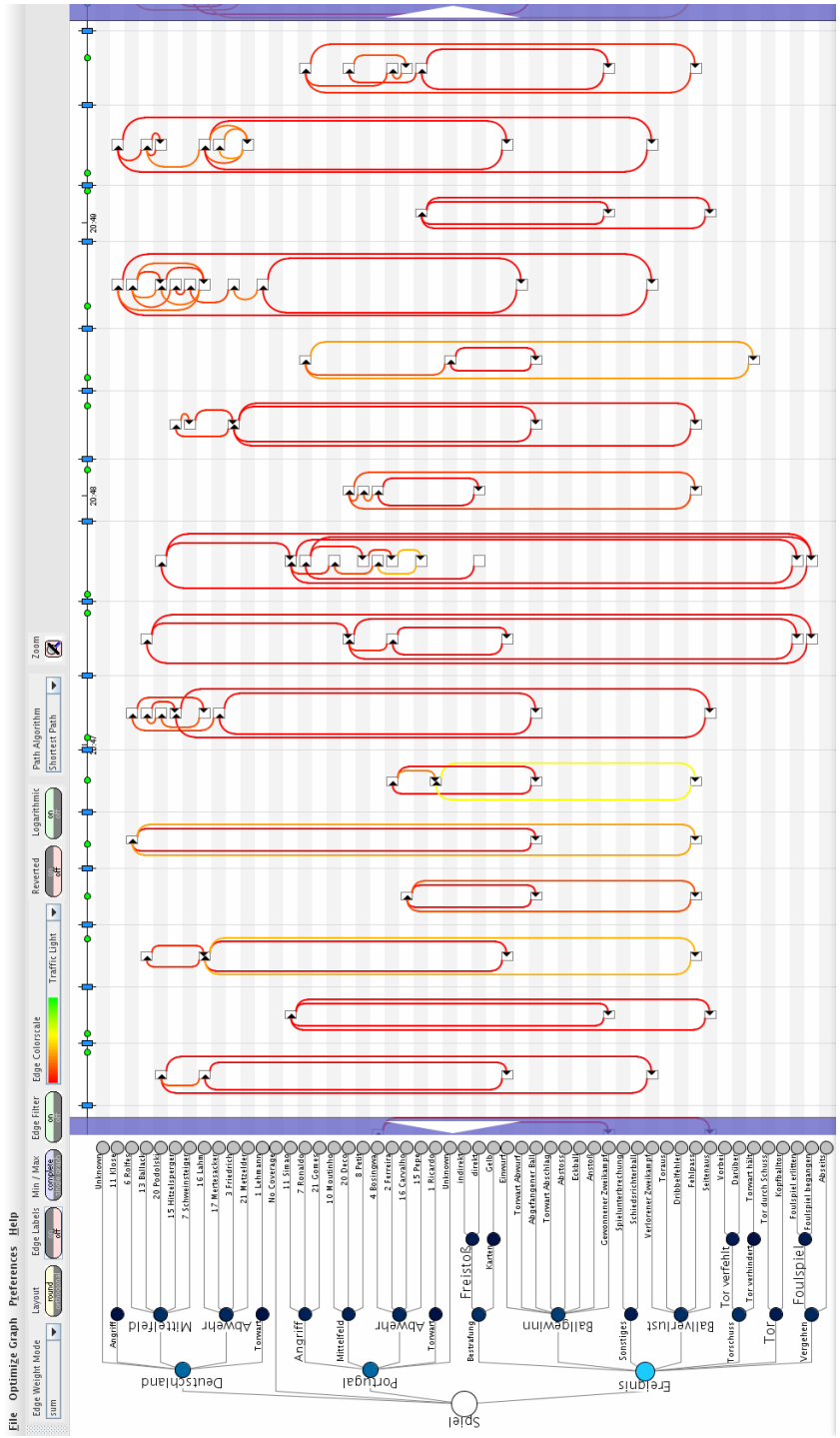

Abbildung 4.7: Fußballspiel-Graph - Teilausschnitt mit aufgeklappter Hierarchie.

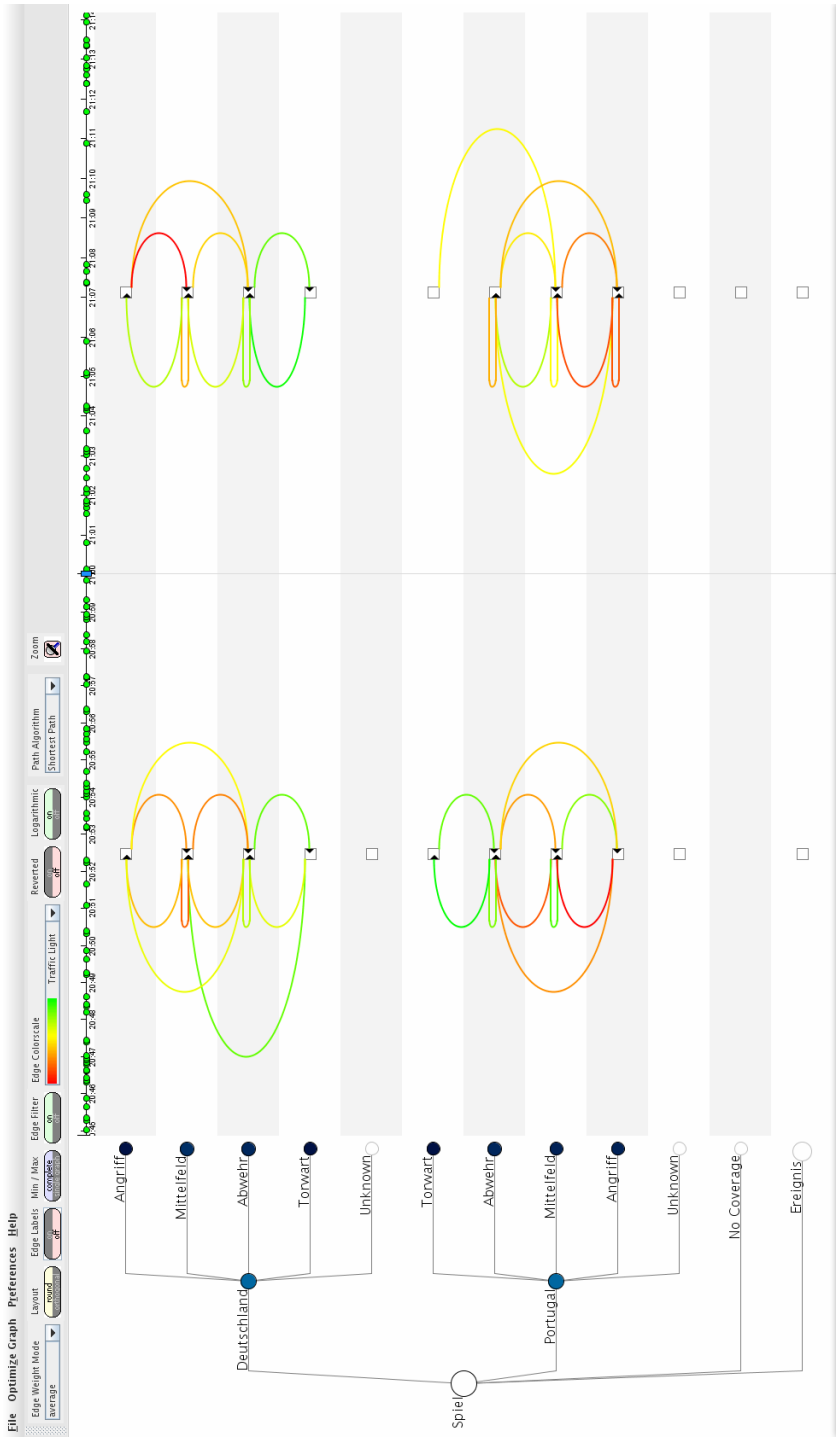

Abbildung 4.8: Fußballspiel-Graph - Passspielanalyse.

Diese Passlängen-Visualisierung ist nur ein Beispiel, wie dieser Datensatz genutzt werden kann. Es bieten sich aber noch viele andere Analysen an. An dieser Stelle werden einige genannt. Die genaue Betrachtung würde aber den Rahmen dieser Arbeit überschreiten.
- Anzahl der Pässe
- genauere Betrachtung des Zusammenspiels innerhalb der Mannschaftsteile
- Einbeziehen von Ereignissen wie gewonnene Zweikämpfe oder Fehlpässe
- ...

4.4 Algorithmenanimation: Ford Fulkerson Maxflow Algorithmus

Die letzte Anwendung benutzt einen Datensatz, der die Funktionsweise des Ford Fulkerson Maxflow Algorithmuses an einem Beispielgraph (siehe Abbildung 4.9) erklärt. Der Knoten s stellt die Quelle dar und der Knoten t die Senke. Knoten 1, 2 und 3 sind unter einem Knoten 1 zusammengefasst, da sie im Graphen die Entfernung 1 zur Quelle haben. Das gleiche gilt für die Knoten 4, 5 und 6 die Entfernung 2 haben.
Der erste Graph der Folge zeigt den kompletten Ausgangsgraphen. Die Gewichte der Kanten sind deren Kapazitäten. Die folgenden Graphen stellen nun Schritt für Schritt den anwachsenden Fluss dar.
Der Algorithmus sucht Pfade von der Quelle zur Senke, an denen eine Flusserhöhung vorgenommen werden kann. In dem Beispiel wird der Fluss vier mal erhöht bis schließlich kein Pfad mehr existiert, auf dem zusätzlicher Fluss gesendet werden könnte.
Mit diesem Beispiel soll gezeigt werden, dass man das TimeArcTrees-Werkzeug nutzen kann, um die Funktionsweise verschiedener inkrementeller Graphalgorithmen vorzuführen. Der Vorteil gegenüber Animationen mit bewegten Bildern ist, dass der gesamte Ablauf als Folge von Graphen in einer Abbildung visualisiert werden kann.

Abbildung 4.9: Visualisierung des Ford Fulkerson Maxflow Algorithmus.

Kapitel 5

Diskussion, Zusammenfassung und Ausblick

5.1 Diskussion

5.1.1 Inhaltliche Probleme bei der Realisierung

Bei der Realisierung von TimeArcTrees tauchten einige inhaltliche Probleme auf, die nun kurz beschrieben werden sollen.
Das größte Problem ist die Skalierbarkeit. Der Bildschirm des Benutzers ist in seinen Maßen beschränkt. Wenn der Datensatz nun viele Knoten oder eine lange Folge von Graphen enthält, kann er nicht mehr komplett auf dem Bildschirm visualisiert werden. Eine Verkleinerung der Knoten und Kanten ist ab einem gewissen Grad nicht mehr sinnvoll. Jedes Objekt besitzt eine minimale Größe, um korrekt identifiziert zu werden, und einen Minimalabstand zu anderen Objekten, um unterschieden werden zu können.
Die Graphen sollen so übersichtlich wie möglich dargestellt werden, ohne die Vorgaben (Hierarchie ohne Kantenkreuzungen, alle Knoten auf einer Vertikalen, Position der Kanten ja nach Richtung links oder rechts der Vertikalen) zu verletzen. Die Vorgaben schränken die Möglichkeiten, ein ästhetisches Layout zu finden, sehr ein.
Ein anderes Problem war die Kantenüberlappung. In der ersten Lösung war vorgesehen, dass alle Kanten, welche die gleiche Länge haben, mit einem gleich weiten

Bogen gezeichnet werden. Dies führte aber dazu, dass sich viele Kanten überdeckten. Sie konnten so nicht mehr eindeutig verfolgt werden. Die Bedingung musste also wegfallen.
Im Laufe der Entwicklung ist auch die Frage aufgetaucht, wie nun mit Gewichten vorgegangen werden sollte, wenn Kanten zusammengefasst werden. Die erste Idee, einfach alle Gewichte zusammen zu zählen, erfüllt nicht immer die Erwartung. Für relative Daten musste die Möglichkeit geschaffen werden, Durchschnittswerte als Gewichte zu benutzen. Anschließend wurden auch noch einige andere Gewichtsmodi für Kanten eingeführt (siehe Kapitel 2.3.1).

5.1.2 Vergleich zu verwandten Werkzeugen

Nachdem nun die TimeArcTrees-Visualisierungstechnik ausführlich vorgestellt wurde, soll ein Vergleich zu den verwandten Werkzeugen (siehe Kapitel 1.3) gezogen werden. Jede Technik hat Vor- und Nachteile und meist bringt die Verbesserung einer Eigenschaft die Benachteiligung einer anderen mit sich. Zum Beispiel haben kreuzungsfreie Techniken (*Matrix*, *Timeline Trees*, *TimeRadarTrees*) das Problem, dass es schwer ist, Pfaden zu folgen. Deshalb müssen beim Entwurf neuer Visualisierungen immer Kompromisse zwischen den Eigenschaften gefunden werden.
Wie schon in Kapitel 1.3 erwähnt, lieferten *Timeline Trees* und *TimeRadarTrees* die Idee für *TimeArcTrees*. Die Intention war, Graphen so darzustellen, dass ein einfaches Folgen von Pfaden möglich ist und somit den größten Nachteil der beiden Vorgänger wett zu machen. Das menschliche Auge benötigt dazu Verbindungslinien, die zwischen den Knoten gezeichnet werden. Dies führt aber zwangsläufig zu Kreuzungen und ist nicht mehr in sehr kompakter Form zu visualisieren.

In Tabelle 5.1 werden die Eigenschaften der Repräsentation und in Tabelle 5.2 die Eigenschaften der Eingabegraphen verglichen. Die Visualisierungstechniken *Hierarchical Edge Bundles* und *Graph Links on Treemaps* sind entworfen worden, um große Datenmengen kompakt darzustellen. Allerdings können sie weder dynamische noch gewichtete Daten darstellen. *ArcTrees* haben durch ihre Darstellung der Kanten und der ähnlichen Kompaktheit auf den ersten Blick eine große Ähn-

Visualisierungstechnik	**kreuzungsfrei**	**Folgen von Pfaden**
Node-Link	nein	einfach
Matrix	ja	schwierig
Hierarchical Edge Bundles	nein	schwierig
ArcTrees	nein	schwierig
Graph Links on Treemaps	nein	schwierig
Timeline Trees	ja	schwierig
TimeRadarTrees	ja	schwierig
TimeArcTrees	**nein**	**einfach**

Tabelle 5.1: Vergleich der Repräsentationseigenschaften.

Visualisierungstechnik	**dynamisch**	**(un-)gerichtet**	**gewichtet**
Node-Link	schwierig	beide	ja
Matrix	nein	beide	ja
Hierarchical Edge Bundles	nein	gerichtet	nein
ArcTrees	nein	ungerichtet	nein
Graph Links on Treemaps	nein	ungerichtet	nein
Timeline Trees	ja	beide	ja
TimeRadarTrees	ja	beide	ja
TimeArcTrees	**ja**	**gerichtet**	**ja**

Tabelle 5.2: Vergleich der Grapheigenschaften.

lichkeit zu einzelnen Graphen der TimeArcTrees. Jedoch sind sie weder in der Lage gerichtete noch gewichtete Graphen zu visualisieren. Das TimeArcTrees-Werkzeug ist das einzige, dass die gesamte Folge des dynamischen gewichteten Digraphen in einer statischen Node-Link-basierten Darstellung zeigt.

5.2 Zusammenfassung

Die Aufgabe der vorliegenden Arbeit bestand darin, ein neues Visualisierungswerkzeug für Folgen von gerichteten Graphen zu entwickeln. Die Visualisierung sollte mithilfe von Knoten-Kanten-Repräsentationen realisiert werden und in der Lage sein, gewichtete Graphen darzustellen. Zusätzlich sollte es möglich sein, Knoten in Informationshierarchien zu ordnen. Teilhierarchien sollten interaktiv auf- und zusammengeklappt werden können.

Der Benutzter sollte möglichst die komplette Folge der Graphen in einer Ansicht analysieren können. Um dies zu unterstützen, wurden interaktive Funktionen wie Filter und Aggregation implementiert. Alle Funktionen wurden so umgesetzt, dass sie benutzerfreundlich mit der Maus gesteuert werden können.

Um die Funktionalität des Werkzeuges zu erweitern, wurden Pfadalgorithmen zur Berechnung und Visualisierung von kürzesten Pfaden und maximalen Flüssen implementiert.

Die Lesbarkeit der einzelnen Graphen der Folge wurde verbessert, indem Algorithmen zur Minimierung der Anzahl der Kantenkreuzungen und Kantenlängen hinzugefügt wurden. Zusätzlich wurde eine Methode entworfen, die eine möglichst kompakte aber dennoch überlappungsarme Darstellung der Kanten errechnet.

Das Werkzeug wurde mit Java implementiert und anschließend getestet. Zum Schluß wurden mehrere Datensätze visualisiert. Die entstandenen Abbildungen dienen als Beispiele für Anwendungen von TimeArcTrees.

5.3 Ausblick

Es ist geplant eine Evaluation des Werkzeuges durchzuführen. Dabei soll der Vergleich zu *TimeRadarTrees* und *Timeline Trees* gezogen werden. Zur Zeit entsteht am Lehrstuhl Softwaretechnik der Universität Trier ein AJAX-Framework für webbasierte Evaluation von interaktiven Anwendungen mithilfe dessen die Studie per Internet durchgeführt werden könnte.

Die Funktionalität des Werkzeuges kann zusätzlich zur Berechnung der kürzesten Pfade und maximalen Flüsse durch andere graphtheoretische Algorithmen erweitert werden. Ein Beispiel dafür wäre die Berechnung der starken Zusammenhangskomponenten.
Um Unterschiede zwischen aufeinanderfolgenden Graphen deutlicher darzustellen, wäre ein zusätzlicher Gewichtsmodus der Kanten denkbar. Hierbei würden die Gewichte der Kanten den Unterschied zu Gewichten der gleichen Kanten im Vorgraph visualisieren. Zusätzlich könnten Kanten, die in der Folge neu erscheinen oder entfallen, durch eine besondere Markierung kenntlich gemacht werden.
Auf die Möglichkeit, einzelne Zeitabschnitte herauszufiltern, wurde bisher verzichtet. Jedoch könnten dieser und einige andere Filter helfen, umfangreiche Datensätze effektiver ohne vorherige Überarbeitung zu visualisieren.
Schließlich sollte erwähnt werden, dass die verwendeten Algorithmen Raum für Optimierung bieten. Vor allem der Algorithmus zur Minimierung der Anzahl der Kantenkreuzungen, der lediglich eine Näherung berechnet, könnte durch einen effizienteren ersetzt werden. Für die Umsetzung der graphtheoretischen Probleme wurden Standardalgorithmen verwendet. Hier besteht kein Handlungsbedarf, da alle visualisierbaren Datensätze aus einer begrenzten Menge von Knoten bestehen und daher auch schnell mit den verwendeten Algorithmen bearbeitet werden können.

Bisher ist das Erstellen eigener Datensätze nur durch zeitaufwendiges Eintippen möglich. Ein Editor zum softwaregestützten Erstellen einer Eingabedatei könnte Abhilfe schaffen. Da TimeRadarTrees mit dem gleichen Datenmodell arbeitet und ebenfalls noch keinen Editor besitzt, wäre es sinnvoll, diesen als eigenständiges Werkzeug zu erstellen.

Abbildungsverzeichnis

Literaturverzeichnis

[Andrews and Heidegger, 1998] Andrews, K. and Heidegger, H. (1998). Information slices: Visualising and exploring large hierarchies using cascading, semi-circular discs (late breaking hot topic paper). In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS'98)*, pages 9–12, Research Triangle Park, NC.

[Balzer and Deussen, 2005] Balzer, M. and Deussen, O. (2005). Voronoi treemaps. In *INFOVIS*, page 7. IEEE Computer Society.

[Bartram et al., 2000] Bartram, L., Uhl, A., and Calvert, T. (2000). Navigating complex information with the ztree. In *Graphics Interface*, pages 11–18. Canadian Human-Computer Communications Society.

[Battista et al., 1999] Battista, G. D., Eades, P., Tamassia, R., and Tollis, I. G. (1999). *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall.

[Bennett et al., 2007] Bennett, C., Ryall, J., Spalteholz, L., and Gooch, A. (2007). The aesthetics of graph visualization. In *Computational Aesthetics*, pages 57–64. Eurographics Association.

[Bertault and Miller, 1999] Bertault, F. and Miller, M. (1999). An algorithm for drawing compound graphs. In *Graph Drawing*, volume 1731 of *Lecture Notes in Computer Science*, pages 197–204. Springer.

[Biedl et al., 1998] Biedl, T. C., Marks, J., Ryall, K., and Whitesides, S. (1998). Graph multidrawing: Finding nice drawings without defining nice. In *Graph Drawing*, volume 1547 of *Lecture Notes in Computer Science*, pages 347–355. Springer.

[Bridgeman and Tamassia, 1998] Bridgeman, S. S. and Tamassia, R. (1998). Difference metrics for interactive orthogonal graph drawing algorithms. In *Graph Drawing*, volume 1547 of *Lecture Notes in Computer Science*, pages 57–71. Springer.

[Burch et al., 2008] Burch, M., Beck, F., and Diehl, S. (2008). Timeline trees: visualizing sequences of transactions in information hierarchies. In *AVI*, pages 75–82. ACM Press.

[Burch and Diehl, 2006] Burch, M. and Diehl, S. (2006). Trees in a treemap. In *Proceedings of 13th Conference on Visualization and Data Analysis (VDA 2006)*, San Jose, California.

[Burch and Diehl, 2008] Burch, M. and Diehl, S. (2008). Timeradartrees: Visualizing dynamic compound digraphs. *Comput. Graph. Forum*, 27(3):823–830.

[Diehl and Görg, 2002] Diehl, S. and Görg, C. (2002). Graphs, they are changing. In *Graph Drawing*, volume 2528 of *Lecture Notes in Computer Science*, pages 23–30. Springer.

[Diestel, 2006] Diestel, R. (2006). *Graphentheorie*. Springer Verlag.

[Fekete et al., 2003] Fekete, J.-D., Wand, D., Dang, N., Aris, A., and Plaisant, C. (2003). Overlaying graph links on treemaps. In *Poster Compendium of the IEEE Symposium on Information Visualization (INFOVIS'03), Los Alamitos, CA, USA*. IEEE.

[Furnas, 1986] Furnas, G. W. (1986). Generalized fisheye views. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 16–23, New York, NY, USA. ACM Press.

[Ghoniem et al., 2004] Ghoniem, M., Fekete, J.-D., and Castagliola, P. (2004). A comparison of the readability of graphs using node-link and matrix-based representations. In *INFOVIS*, pages 17–24. IEEE Computer Society.

[Harel, 1998] Harel, D. (1998). On the aesthetics of diagrams (summary of talk). In *MPC*, volume 1422 of *Lecture Notes in Computer Science*, pages 1–5. Springer.

[Herman et al., 2000] Herman, I., Melançon, G., and Marshall, M. S. (2000). Graph visualization and navigation in information visualization: A survey. *IEEE Trans. Vis. Comput. Graph.*, 6(1):24–43.

[Holten, 2006] Holten, D. (2006). Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Trans. Vis. Comput. Graph.*, 12(5):741–748.

[Johnson and Shneiderman, 1991] Johnson, B. and Shneiderman, B. (1991). Tree maps: A space-filling approach to the visualization of hierarchical information structures. In *IEEE Visualization*, pages 284–291.

[Misue et al., 1995] Misue, K., Eades, P., Lai, W., and Sugiyama, K. (1995). Layout adjustment and the mental map. *J. Vis. Lang. Comput.*, 6(2):183–210.

[Neumann et al., 2005] Neumann, P., Schlechtweg, S., and Carpendale, M. S. T. (2005). Arctrees: Visualizing relations in hierarchical data. In *EuroVis*, pages 53–60. Eurographics Association.

[Purchase, 2002] Purchase, H. C. (2002). Metrics for graph drawing aesthetics. *J. Vis. Lang. Comput.*, 13(5):501–516.

[Purchase et al., 2006] Purchase, H. C., Hoggan, E. E., and Görg, C. (2006). How important is the mental map? - an empirical investigation of a dynamic graph layout algorithm. In *Graph Drawing*, volume 4372 of *Lecture Notes in Computer Science*, pages 184–195. Springer.

[Sarkar and Brown, 1992] Sarkar, M. and Brown, M. H. (1992). Graphical fisheye views of graphs. In *CHI*, pages 83–91.

[Shneiderman, 1996] Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. In *VL*, pages 336–343.

[Sugiyama and Misue, 1991] Sugiyama, K. and Misue, K. (1991). Visualization of Structural Information: Automatic Drawing of Compound Digraphs. *IEEE Transactions on Systems, Man and Cybernetics*, 21(4):876–892.

[Taylor and Rodgers, 2005] Taylor, M. and Rodgers, P. (2005). Applying graphical design techniques to graph visualisation. In *IV*, pages 651–656. IEEE Computer Society.

[Tominski et al., 2006] Tominski, C., Abello, J., van Ham, F., and Schumann, H. (2006). Fisheye tree views and lenses for graph visualization. In *IV*, pages 17–24. IEEE Computer Society.

[van Wijk and van de Wetering, 1999] van Wijk, J. J. and van de Wetering, H. (1999). Cushion treemaps: Visualization of hierarchical information. In *INFOVIS*, pages 73–78.

[Ware et al., 2002] Ware, C., Purchase, H. C., Colpoys, L., and McGill, M. (2002). Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110.

[Zhao et al., 2005] Zhao, S., McGuffin, M. J., and Chignell, M. H. (2005). Elastic hierarchies: Combining treemaps and node-link diagrams. In *INFOVIS*, page 8. IEEE Computer Society.