Visualisierung und dynamische Aggregierung von semantischen Graphen

Techniken, Konzepte, Evaluierung
und Implementierung

Martin Kandlhofer

TU Graz

Visualisation and Dynamic Aggregation of Semantic Graphs

Techniques, Concepts and Implementation

Master's Thesis

at

Graz University of Technology

submitted by

Martin Kandlhofer

Know-Center, Knowledge Management Institute
Graz University of Technology
A-8010 Graz, Austria

August 2009

Supervisor: Univ.-Prof. Dr. Klaus Tochtermann
Advisor: DI Vedran Sabol

TU Graz

center . graz Know

Visualisierung und dynamische Aggregierung von semantischen Graphen

Techniken, Konzepte und Implementierung

Masterarbeit

an der

Technischen Universität Graz

vorgelegt von

Martin Kandlhofer

Know-Center, Institut für Wissensmanagement,
Technische Universität Graz
A-8010 Graz

August 2009

Diese Arbeit ist in deutscher Sprache verfasst.

Gutachter: Univ.-Prof. Dr. Klaus Tochtermann
Betreuer: DI Vedran Sabol

TU Graz

center . graz Know

Abstract

The enormous growth of data and content over the last few years resulted in large amounts of complex information hosted on heterogeneous systems. Semantic technologies, which have seen rapid adoption in recent years, employ a formalized, machine-understandable representation of knowledge to facilitate sharing and reuse of the stored information and enable interoperability between various systems. In consideration of these facts there is clearly a need for tools enabling efficient, user-centred exploration and understanding of large amounts of information and complex knowledge models. In this context computer supported visualisation, in particular visualisation of semantic graphs plays a decisive role.

This Master's Thesis first discusses relevant theoretical background including selected topics from the areas of information visualisation and graph visualisation. Subsequently an evaluation of recent graph visualisation tools, frameworks and packages is performed with the purpose of assessing and comparing their capabilities. Within the scope of the practical part of this Master's Thesis a tool for visualisation and dynamic aggregation of RDF-graphs is implemented atop of a selected graph visualisation framework. To reduce visual complexity and clutter and allow the user to focus on the relevant information aggregation of visualised graphs is implemented, either depending on semantic information, or by dynamic, level-of-detail dependent clustering of the layout.

Kurzfassung

Der enorme Zuwachs an Daten und Inhalten führte über die letzen Jahre hinweg zu einer riesigen Menge an komplexen, mitunter auf heterogenen Systemen bereitgestellten Informationen. Semantische Technologien, welche in den letzten Jahren eine rasche Verbreitung erfuhren, nutzen die formalisierte, maschineninterpretierbare Repräsentation von Wissen um auf diese Weise die Wiederverwendung von gespeicherten Informationen sowie die Interoperabilität zwischen verschiedenen Systemen zu ermöglichen bzw. zu erleichtern. In Anbetracht dieser Tatsachen besteht ein offenkundiger Bedarf an Tools zur effizienten, nutzerorientierten Handhabung riesiger Informationsmengen und komplexer Wissensmodelle. Computerunterstützte Visualisierung und speziell die Visualisierung von semantischen Graphstrukturen spielt dabei eine zentrale Rolle.

Diese Masterarbeit beleuchtet zunächst den relevanten theoretischen Hintergrund, einschließlich ausgewählter Themen aus den Bereichen der Graph- und Information-Visualisation. Die im Anschluss durchgeführte Evaluierung aktueller Graph-Visualisation Tools, Packages und Frameworks dient der Abschätzung und dem Vergleich der unterschiedlichen Fähigkeiten der einzelnen Softwarelösungen. Im Rahmen des praktischen Teils der Masterarbeit erfolgt, unter Berücksichtigung der Ergebnisse der Evaluierung und aufbauend auf einem ausgewählten Graphvisualisierungs-Framework, die Implementierung eines Tools zur Visualisierung und dynamischen Aggregierung von RDF-Graphen. Zur Reduktion der visuellen Komplexität von dargestellten Graphen wird ein Aggregierungsverfahren implementiert, welches auf semantischen Informationen bzw. auf dem dynamischen, vom Detaillierungsgrad abhängigen Clustering des Layouts basiert. Auf diese Weise soll es dem User ermöglicht werden, sich auf relevante Informationen und Bereiche des visualisierten Graphen zu fokussieren.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....

(date) (signature)

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am..... .....

(Unterschrift)

Danksagung

An dieser Stelle möchte ich zunächst meinen Eltern danken. Erst durch ihre Unterstützung, ihr Verständnis und ihre Rücksichtnahme wurde meine schulische und universitäre Ausbildung ermöglicht. Besonderer Dank gilt auch meinem Betreuer Vedran Sabol, der mir in der finalen Phase des Studiums stets mit Rat und Tat zur Seite stand und dessen Ideen, Anregungen und Feedbacks maßgeblich zur Erstellung dieser Arbeit beitrugen. Weiters bedanke ich mich bei Prof. Klaus Tochtermann und dem Team des Know-Center für die hervorragende Unterstützung während der Abfassung dieser Masterarbeit.

Martin Kandlhofer
Graz, Austria, August 2009

Credits

- Diese Arbeit wurde mit Hilfe der LaTex-Vorlage von Keith Andrews erstellt [Andrews, 2008].

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Diese Masterarbeit widmet sich grundsätzlich der Visualisierung von semantischen Graphen. Im Zuge dessen werden verschiedene Themen und Konzepte aus den Bereichen Information Visualisation, Graph Visualisation, Semantic Web, Clustering, Aggregation, Graph Theory und Graph Drawing aufgegriffen. Zunächst erfolgt eine Betrachtung relevanter Techniken und Mechanismen vom theoretischen Standpunkt aus. Der anschließenden Evaluierung ausgewählter, bereits vorhandener und frei verfügbarer Visualisierungswerkzeuge folgt letztlich die Implementierung einer neuen Software zur interaktiven Darstellung von semantischen Graphen.

Das Einführungskapitel bietet einen Überblick über die Thematiken, Problemstellungen und Ziele dieser Masterarbeit. Weiters werden zentrale Begriffe definiert und erläutert sowie die grundlegende Vorgehensweise zur Erreichung der Ziele skizziert. Den Abschluss bildet die überblicksmäßige Betrachtung vorhandener, ähnlicher Arbeiten zu diesem Thema sowie eine Abgrenzung der im Rahmen dieser Masterarbeit behandelten Themenbereiche.

1.1 Motivation

Enorme Datenwachstumsraten sowie immer größer werdende Informationsräume führten in den letzten Jahren zu einer wahren Datenflut. In Anbetracht dessen stellt sich die Frage nach der effizienten Handhabung sowie der geeigneten Aufbereitung dieser riesigen Mengen an Daten und Informationen verschiedensten Typs und unterschiedlichster Herkunft. Computerunterstützte Visualisierung spielt in diesem Zusammenhang eine entscheidende Rolle. Eine der Hauptaufgaben dieser Disziplin ist es, große Datenmengen bzw. abstrakte Informationen und Strukturen, nach einer entsprechenden Vorverarbeitung, visuell ansprechend darzustellen.

In den letzten Jahrzehnten gewann speziell der Bereich der Graphvisualisierung stetig an Bedeutung und ist mittlerweile in den verschiedensten Bereichen von zentraler Relevanz. Aufgrund dessen werden laufend neue Ansätze, Erkenntnisse sowie unterschiedliche Techniken und Konzepte entwickelt und publiziert. In Verbindung mit der Entstehung des *Semantic Web* stellt die Visualisierung und Navigation von semantischen Graphen ein aktuelles Thema mit vielen unterschiedlichen Herausforderungen und Aufgabengebieten dar.

1.2 Problemstellung und Ziele

Semantische Graphen bzw. Graph- und Netzwerkstrukturen im *RDF*-Format (Ressource Description Framework) weisen in der Regel eine hohe Anzahl an Knoten sowie eine hohe Verbindungsdichte auf. Im Zuge der Visualisierung solcher Strukturen kommt es aufgrund der Komplexität bzw. der hohen Elementanzahl häufig zu *Visual Clutter*, d.h. Knoten, Kanten und Beschriftungen (= Labels) können sich

überlappen und sind dadurch nicht mehr voneinander zu unterscheiden. Überlagerungen von Elementen oder Kantenüberschneidungen führen zu einer unübersichtlichen, für den menschlichen Betrachter unbrauchbaren Visualisierung. Eine Auswertung bzw. Analyse der Darstellung wird somit erschwert oder gänzlich verhindert. Weiters wird die Lesbarkeit von Labels sowie deren eindeutige Zuordnung zu Knoten oder Kanten erheblich beeinträchtigt. Auch im Hinblick auf die Interaktion mit der Visualisierung können Probleme auftreten. Eine zu große Anzahl an gleichzeitig darzustellenden Objekten führt ab einem gewissen Punkt unweigerlich zu Verzögerungen, d.h. eine Interaktion in Echtzeit ist nicht mehr möglich. In solchen Fällen ist es nicht zielführend, die gesamte Graphstruktur bzw. alle Knoten, Kanten und Labels auf einmal und in detaillierter Weise darzustellen.

Zur Lösung dieses Problems bedarf es spezieller Vorgehensweisen und Techniken. Eine Möglichkeit besteht darin, die Größe bzw. Komplexität des Graphen in einem Vorverarbeitungsschritt zu reduzieren. Dies kann zum Beispiel mithilfe von Cluster-, Aggregations- oder Filter-Techniken erreicht werden. Im Anschluss an diesen Schritt erfolgt die Darstellung bzw. das Layout des vereinfachten Graphen. Im Zuge der Komplexitätsreduktion dürfen Details nicht verloren gehen, sondern müssen bei Bedarf wieder verfügbar sein. Mittels Level-Of-Detail Techniken können diese Details dynamisch und in Abhängigkeit der aktuellen Zoomstufe ein- oder ausgeblendet werden.

Ein weiteres charakteristisches Merkmal semantischer Graphen ist, dass die Bedeutung der Graphelemente eindeutig definiert und Bestandteil des Graphen ist. Ein essentieller Punkt ist somit die Berücksichtigung dieser semantischen Informationen im Zuge der Visualisierung und der Analyse von RDF-Graphen. So können Aggregations- oder Filter-Vorgänge auf Basis semantischer Informationen erfolgen oder Graphelemente in Abhängigkeit ihrer Bedeutung dargestellt werden.

Vor dem Hintergrund dieser Problemstellung soll im Rahmen der Masterarbeit zunächst die aktuelle Situation bzw. der derzeitige Stand der Entwicklung auf diesem Gebiet ermittelt werden. Die beiden zentralen Fragestellungen dabei lauten: *Welche Visualisierungswerkzeuge sind bereits vorhanden?* und *Inwieweit eignen sich diese zur Adressierung der Problemstellung?*

Ziel ist es, ausgehend von den Ergebnissen dieser Untersuchung, ein neues Visualisierungssystem zu entwickeln bzw. ein bestehendes Tool entsprechend zu erweitern. Der Fokus liegt dabei auf der Implementierung eines Systems zur interaktiven Visualisierung von semantischen Graphen im RDF-Format, mit der Möglichkeit zur Reduktion der visuellen Komplexität von größeren Graphstrukturen. Weiters sollten Features zur dynamischen, zoomabhängigen Aggregation unterstützt werden.

1.3 Begriffsdefinitionen

In diesem Abschnitt werden einige im Kontext dieser Arbeit relevanten Begriffe und Konzepte kurz vorgestellt bzw. erläutert. Weiters soll die Definition häufig verwendeter Bezeichnungen zu einem besseren Verständnis der nachfolgend diskutierten Thematiken führen. Eine genauere Betrachtung einzelner Punkte sowie die Definition kontextspezifischer Begriffe erfolgt in den jeweiligen Unterkapiteln.

Daten, Information, Wissen

In der Literatur findet sich häufig eine Unterscheidung zwischen den Begriffen Daten, Information und Wissen. Je nachdem, in welchem Zusammenhang diese Begriffe verwendet werden, unterscheiden bzw. widersprechen sich die einzelnen Erklärungsversuche.

Eine weit verbreitete und bekannte Definition beruht auf der hierarchischen Beziehung zwischen Daten, Information und Wissen. *Daten* sind demnach einfache, isolierte Fakten. Sie stehen in keinem Kontext und bilden somit das Rohmaterial. In weiterer Folge entsteht *Information* wenn Daten strukturiert, manipuliert oder interpretiert werden. Zuvor isolierte Fakten stehen nun in einem Kontext. Wird Information interpretiert bzw. mit einer Bedeutung versehen, entsteht schließlich *Wissen*. Aufgrund von

Wissen können nun Entscheidungen und Vorhersagen getroffen sowie Schlussfolgerungen gezogen und Konsequenzen abgeleitet werden. [Tuomi, 1999] [Moscher et al., 2005]

Das folgende Beispiel soll die Relation zwischen Daten, Information und Wissen veranschaulichen:

- *Daten*: Einfache ganze Zahlen (z.B 30, 50,...) stehen in keinem Kontext. Eine Bedeutung dieser Zahlen ist nicht erkennbar.
- *Information*: Dieselben Zahlen auf einem Verkehrsschild stehen in diesem Fall für eine Geschwindigkeitsbeschränkung. Somit ist eine Interpretation möglich.
- *Wissen*: Eine Konsequenz aus der Überschreitung der Geschwindigkeitsbeschränkung wäre eine Geldstrafe. D.h. es kann einen Schlussfolgerung gezogen werden.

Wie zuvor bereits erwähnt, existieren zahlreiche Variationen dieses hierarchischen Konzepts sowie einige unkonventionellere Ansätze und Sichtweisen. Eine Auflistung verschiedener unterschiedlicher Definitionen findet sich auch in [Tuomi, 1999].

Visualisierung

Der Term *Visualisation* bzw. *Information Visualisation* wird in den verschiedenen Publikationen jeweils leicht unterschiedlich definiert bzw. interpretiert. [Card et al., 1999] unterscheidet zwischen Visualisation und Information Visualisation. Das charakteristische Merkmal der Information Visualisation besteht demnach in der Visualisierung von abstrakten Daten, während Visualisation auch nicht-abstrakte Daten (also Daten welche beispielsweise einen räumlichen Bezug aufweisen) behandelt. In [Card et al., 1999] findet sich folgende Definition des Begriffs Information Visualisation:

> "The use of computer-supported, interactive, visual representations of abstract data to amplify cognition."

Der Erwerb und die Verwendung von Wissen (Cognition) sowie das Erkennen von Mustern oder Relationen soll durch den Einsatz visueller Repräsentationen unterstützt werden. [Card et al., 1999] [Mueller, 2005]

Laut [Andrews, 2002] kann Information Visualisation bzw. deren Aufgaben und Ziele auf folgende Weise beschrieben werden:

> "The visual presentation of abstract information spaces and structures to facilitate their rapid assimilation and understanding."

Information Visualisation hat somit das Ziel, eine schnellere Aufnahme bzw. ein besseres Verständnis abstrakter Informationsräume mithilfe einer bildlichen Darstellung zu erreichen. Graph Visualisation kann als Teilbereich der Information Visualisation betrachtet werden.

Graphen

Als *Graph* wird generell eine abstrakte Struktur zur Beschreibung bzw. Abbildung von Relationen zwischen Objekten bezeichnet. Ein solches Konstrukt besteht aus einer Menge von Knoten (Objekte) und Kanten (Verbindungen zwischen Objekten). Synonyme Begriffe für Knoten und Kanten wären Nodes oder Vertices bzw. Edges oder Links. Nodes können mit beliebig vielen anderen Nodes verbunden sein. Faktoren wie die Größe eines Graphen oder die Komplexität der Beziehungen spielen eine entscheidende Rolle bei der Entwicklung bzw. der Wahl einer geeigneten Visualisierung. [Herman et al., 2000] [Andrews, 2002] [Kaufmann and Wagner, 2001] [Prinz, 2006] [Shneiderman, 1996]

Ein *semantischer Graph* ist ein gerichteter Graph, welcher spezielle Eigenschaften und Merkmale aufweist. Die Bedeutung von Nodes und Edges (Links) ist eindeutig definiert und ein Bestandteil des Graphen. Semantische Informationen existieren in maschinenlesbarer Form. Programme bzw. Anwendungen sind somit in der Lage, diese Informationen zu lesen, zu verarbeiten und in Folge die Bedeutung zu erfassen. Offene Standards wie *RDF (Ressource Description Framework)* oder *OWL (Web Ontology Language)* werden zur Definition bzw. Repräsentation der Semantik verwendet. [Wong et al., 2006] [Spivack, 2007]

Dynamische Aggregierung

Aggregierung stellt eine effektive Möglichkeit zur Behandlung großer Datenmengen dar. Vor allem in Verbindung mit größeren Graphen spielt das Konzept eine entscheidende Rolle. Im Kontext dieser Masterarbeit wird das Zusammenfassen bestimmter Teile des Graphen sowie die anschließende Visualisierung als *Aggregierung* bezeichnet. Die Identifikation einzelner Gruppen erfolgt unter Verwendung verschiedener Clustering-Techniken. Nodes, Edges bzw. Sub-Graphen werden aggregiert und mithilfe eines entsprechenden Symbols (Icons) dargestellt. Ziel ist die Reduktion der visuellen Komplexität bzw. eine Vereinfachung des ursprünglichen Graphen. Der dynamische Charakter ergibt sich aus der Kombination von Clustering-Techniken zur Auffindung von zusammengehörenden Elementen, und den verschiedenen Möglichkeiten zur interaktiven Anpassung bzw. Modifikation der Aggregation.

1.4 Vorhandene Arbeiten

Zu den Themen Information Visualisation im Allgemeinen bzw. Graph Visualisation im Speziellen findet sich eine Vielzahl von Arbeiten, Artikeln und Abhandlungen. Oft liegt der Fokus dabei auf graphtheoretischen Aspekten bzw. auf speziellen Fragestellungen des Graph Drawings. Eine Reihe von Publikationen widmet sich auch der Behandlung bzw. der Repräsentation semantischer Informationen mit dem Schwerpunkt der Visualisierung von RDF-Modellen in Form einer Graph- bzw. Netzwerkstruktur. Weiters existieren verschiedene Softwarepakete zur Graphvisualisierung sowie eine Reihe von Frameworks und Tools zur Visualisierung von Wissen bzw. Informationen. Ausgewählte Softwarelösungen zur Visualisierung von Graphen werden in Kapitel 3 näher betrachtet. Nachfolgend sollen einige vorhandene und ähnliche Arbeiten zum Thema kurz vorgestellt werden.

Das Paper von [Herman et al., 2000] bietet einen Überblick über verschiedene Graphvisualisierungs- und Navigations-Techniken. Der Schwerpunkt liegt auf der Betrachtung verschiedener Aspekte der Graphbehandlung aus Sicht der Information Visualisation. Im Zuge dessen werden grundlegende Layout-Algorithmen und Navigations-Konzepte besprochen. Besonderes Augenmerk gilt dabei den in Verbindung mit der Visualisierung großer Graphen auftretenden Problemen und Herausforderungen. Davon ausgehend werden unterschiedliche Methoden und Techniken zur Reduktion der visuellen Komplexität großer Graphen vorgestellt und analysiert.

Die Masterarbeit von [Prinz, 2006] widmet sich, neben allgemeinen Aspekten der Information Visualisation, in erster Linie den theoretischen Grundlagen des Graph Drawings bzw. den zugrundeliegenden Konzepten der Graphvisualisierung. Dabei werden u.a. Themen aus dem Bereich der Graphentheorie und Aspekte der Graph-Repräsentation näher behandelt. Einen weiteren Schwerpunkt bildet die Evaluierung allgemeiner Softwarepakete zur Visualisierung von Graphen. Im Rahmen dieser Arbeit wird auch die Implementierung eines Graphvisualisierung-Frameworks beschrieben. Die Software bietet eine geeignete Infrastruktur zur einfacheren Implementierung und Analyse von neuen Graph Drawing Algorithmen und dient in erster Linie Lehrzwecken.

[Wong et al., 2006] beschreibt ein Framework zur visuellen Analyse von großen semantischen Graphen. Im Rahmen dieses Papers werden das generelle Design des Frameworks sowie zugrundeliegende Konzepte und Techniken der Graph- bzw. Netzwerk-Visualisierung und Netzwerk-Analyse näher

erläutert. Das als *Have Green* bezeichnete Framework bietet eine interaktive Umgebung zur Visualisierung, Exploration, Navigation sowie diverse Möglichkeiten zur Abfrage (Querying) von großen semantischen Graphen.

[Abello et al., 2006] beschreibt ein Node-Link basiertes Visualisierungssystem für Graphen mit einer Größe von bis zu 16 Millionen Edges. Das Paper widmet sich in erster Linie den Problemen bei der Visualisierung großer Graphen und präsentiert in weiterer Folge mögliche Lösungsansätze. Das vorgestellte Visualisierungssystem verwendet zur Interaktion und Navigation ein Node-Link Layout eines geclusterten Graphen. Verschiedene Algorithmen erlauben dabei die Erstellung einer hierarchischen Cluster-Struktur eines beliebigen gewichteten Input-Graphen. Mittels Expand-/Collapse-Operationen können nach Abschluss dieser Vorberechnungsphase Cluster in willkürlicher Reihenfolge aufgefächert und somit die darin enthaltenen Sub-Graphen sichtbar gemacht werden.

Die Habilitationsschrift von [Andrews, 2002] bietet einen generellen Überblick über verschiedene Visualisierungstechniken für unterschiedliche Arten von Informationen.

1.5 Abgrenzung/Themenumfang

Die vorhandenen Publikationen beleuchten verschiedene Aspekte der Graph- bzw. der Information-Visualisation. Wie die Übersicht über einige dieser Arbeiten im vorangegangen Abschnitt zeigt, liegen die einzelnen Schwerpunkte auf unterschiedlichen Themenbereichen. Im Rahmen dieser Masterarbeit werden einige der darin vorgestellten Konzepte, Ideen und Techniken aufgegriffen und miteinander kombiniert. Der Fokus dieser Arbeit liegt somit auf folgenden Themen:

- **Information- bzw. Graph-Visualisation**: Visualisierung von Graphen, Netzwerken bzw. hierarchischen Strukturen.
- **Dynamische Aggregierung - Clustering**: Möglichkeiten zur Reduktion der visuellen Komplexität größerer Graphen.
- **Semantische Aspekte**: Visualisierung von semantisch annotierten Graphstrukturen bzw. von RDF-Graphen im Speziellen.

Hinsichtlich dieser Punkte erfolgt eine Evaluierung einiger aktuell verfügbarer Visualisierungs-Tools und -Frameworks. Aufbauend auf den daraus gewonnen Erkenntnissen wird im Anschluss eine neue Visualisierungskomponente implementiert. Kernthemen sind dabei die Visualisierung von RDF-Modellen sowie die Bereitstellung von Features zur Komplexitätsreduktion unter Berücksichtigung semantischer Informationen.

1.6 Kapitelübersicht

Kapitel 2 beleuchtet den theoretischen Hintergrund zu einzelnen ausgewählten Themenbereichen. Die im Kontext der Masterarbeit relevanten Konzepte, Techniken und Methoden werden dabei einer näheren Betrachtung unterzogen.

In Kapitel 3 erfolgt die Untersuchung aktuell vorhandener Softwarelösungen zur Visualisierung von Graphen sowie die anschließende Evaluierung ausgewählter Tools, Packages und Frameworks.

Ausgehend von den Ergebnissen bzw. Erkenntnissen der Evaluierung widmet sich Kapitel 4 der genaueren Betrachtung eines dieser Frameworks. Dabei werden Funktionalitäten, Eigenschaften, Mechanismen und technische Details der als *Cytoscape* bezeichneten Visualisierungsplattform erläutert.

Kapitel 5 befasst sich mit den verschiedenen Aspekten der Entwicklung einer neuen Visualisierungskomponente. Den Schwerpunkt bilden dabei die technischen Details der Implementierung. Darauf aufbauend widmet sich Kapitel 6 der Demonstration der Features und Merkmale des implementierten Visualisierungssystems.

In Kapitel 7 werden die im Rahmen der Masterarbeit erzielten Ergebnisse und gewonnenen Erkenntnisse nochmals zusammengefasst sowie mögliche Weiterentwicklungen und Verbesserungen der Implementierung diskutiert. Das Kapitel schließt mit einem generellen Ausblick auf zukünftige Entwicklungen.

Der User Guide in Anhang A erläutert essentielle Features sowie die grundlegende Bedienung des im Zuge des praktischen Teils der Masterarbeit entwickelten Visualisierungssystems. Weiters finden sich Informationen hinsichtlich Requirements, Installation und Start der Software.

Kapitel 2

Theoretische Grundlagen

Dieses Kapitel beleuchtet den theoretischen Hintergrund zu einzelnen ausgewählten Themenbereichen. Die im Kontext der Masterarbeit relevanten Konzepte, Techniken und Methoden werden dabei einer näheren Betrachtung unterzogen.

Zu Beginn werden grundlegende Prinzipien aus dem Bereich Information- bzw. Graph-Visualisation erläutert. Dieser Abschnitt stellt eine Einführung in die Informationsvisualisierung dar und bildet somit die Basis bzw. den Ausgangspunkt für nachfolgende Überlegungen und Betrachtung. Der anschließende Abschnitt widmet sich essentiellen Begriffen und Konzepten aus der Graphentheorie (Definition, Layout, etc.). Darauf aufbauend werden im nachfolgenden Abschnitt verschiedene Möglichkeiten und Ansätze zur dynamischen Aggregierung eines Graphen bzw. relevante Clustering-Techniken diskutiert. Die anschließende Betrachtung semantischer Aspekte im Zusammenhang mit der Visualisierung von Graphen repräsentiert einen weiteren Schwerpunkt dieses Kapitels. Den Abschluss bildet eine nähere Behandlung ausgewählter Punkte und Aspekte aus den zuvor behandelten Themengebieten.

2.1 Information Visualisation

Riesige Mengen an Informationen verschiedensten Typs und unterschiedlichster Herkunft, immer größer werdende Informationsräume sowie enorme Datenwachstumsraten führten in den letzten Jahren zu einer wahren Datenexplosion. Zahlreiche Prognosen und Vorhersagen deuten darauf hin, dass sich diese Entwicklung auch in nächster Zeit fortsetzen wird. Die Ursachen dafür sind mannigfaltig und eine Auflistung bzw. eine Analyse würde den Rahmen dieser Arbeit sprengen. Es stellt sich nun vielmehr die Frage nach der richtigen Handhabung dieser riesigen Datenmengen. Computerunterstützte Visualisierung spielt in diesem Zusammenhang eine wichtige Rolle und wird auch in Zukunft immer mehr an Bedeutung gewinnen.

Traditionelle Darstellungsmethoden und -techniken stoßen bei umfangreichen und komplexen Informationsmengen schnell an ihre Grenzen. Das Gebiet der Information Visualisation behandelt u.a. die Frage, wie große Datenmengen bzw. abstrakte Informationen und Strukturen in geeigneter und in, für die jeweilige Anwendung, brauchbarer Weise aufbereitet und dargestellt werden können. Durch die visuelle Repräsentation soll ein Überblick über abstrakte Informationsräume geschaffen und somit die schnellere Aufnahme bzw. ein besseres Verständnis ermöglicht werden. Das Ziel dabei: Ein Betrachter soll auf Basis der Visualisierung innerhalb kurzer Zeit in der Lage sein, komplexe Informationsmengen zu erfassen (sowohl was Inhalt als auch Struktur betrifft) und diese zu analysieren bzw. zu interpretieren. Anschließend sollten weitere Entscheidungen aufgrund des Verständnisses der visualisierten Inhalte getroffen werden können. Die bildhafte Darstellung abstrakter Daten, Informationen und Wissen (siehe dazu auch Abschnitt 1.3) erleichtert das Auffinden und Erkennen von Mustern, Clustern, Relationen oder anderen markanten Details (beispielsweise Ausreißer in statistischen Daten). Der Betrachter sollte

Einblick in zuvor verborgene Konzepte und Strukturen erhalten. Neben der zugrunde liegenden grafischen Repräsentation spielt dabei vor allem die Möglichkeit zur Interaktion mit der Darstellung eine entscheidende Rolle.

Schwerpunkte der Information Visualisation sind folglich die Wahl einer geeigneten grafischen Repräsentation, die Art der Berechnung der gewählten Repräsentation sowie abschließend die Frage nach der Unterstützung weiterer Funktionen (beispielsweise Interaktivität der Darstellung,...) [Andrews, 2002] [Shneiderman, 1996] [Lux, 2004] [Munzner, 2000] [Sabol, 2008] Es existiert eine Reihe unterschiedlicher Interpretationen des Begriffs *Information Visualisation*. Die für diese Arbeit relevanten Definitionen finden sich in Abschnitt 1.3.

Ein kurzer Blick auf die Entwicklung der letzten Jahrzehnte zeigt, dass der Interaktion zwischen Mensch und Maschine immer größere Aufmerksamkeit zuteil wird. Musste in den Anfangsjahren des Computers aufgrund kostbarer CPU-Zeit noch jeder In- bzw. Output genauest vorbereitet, überprüft und interpretiert werden, so entwickelte sich im Laufe der Zeit, begünstigt durch leistungsstärkere und kostengünstigere Hardware, die interaktive Ein- bzw. Ausgabe. Den Bereichen der Information Visualisation als Schnittstellen zwischen Benutzer und Rechner wurden infolgedessen stetig mehr Ressourcen (Prozessorleistung, Speicher, etc.) zur Verfügung gestellt bzw. eine größerer Aufmerksamkeit geschenkt. [Munzner, 2000]

2.1.1 Grundlegende Prinzipien und Tasks

Aufgabe eines Visualisierungssystems ist es, große Informationsmengen in einer übersichtlichen und intuitiven Weise aufzubereiten sowie Funktionalitäten zur Bewältigung typischer Tasks bereitzustellen. Es gibt eine Reihe unterschiedlicher Richtlinien, Leitsätze und Vorschläge hinsichtlich der Gestaltung einer Visualisierung. Einige der wichtigsten Prinzipien der Information Visualisation werden im Anschluss vorgestellt. [Andrews, 2009] [Andrews, 2002] [Mueller, 2005] [Munzner, 2000] [Sabol, 2001]

Nutzung der visuellen Fähigkeiten des Menschen

Das visuelle Wahrnehmungssystem des Menschen besitzt die Fähigkeit Muster und Änderungen (z.B. hinsichtlich Größe, Farbe, Form von Objekten) automatisch zu erkennen sowie Bewegungen wahrzunehmen. Bilder können in kurzer Zeit aufgenommen, verstanden und bei Bedarf wiedererkannt werden. So wird zum Beispiel ein roter Kreis in einer Menge von blauen Kreisen sofort und ohne einen bewussten Denkvorgang wahrgenommen (= präattentive Wahrnehmung). Diese Fähigkeiten sollen mithilfe verschiedenster Techniken und Methoden der Information Visualisation unterstützt und somit die kognitiven Ressourcen des Menschen für andere Aufgaben freigegeben werden.

Interaktivität

Zur Visualisierung großer Informationsmengen ist eine rein statische Repräsentation ungeeignet. Einerseits bieten selbst hochauflösende Monitore zu wenig Platz, um komplexe Strukturen und Datensätze in brauchbarer Weise darzustellen. Andererseits ist es nicht möglich, die Auswirkungen von veränderten Parametern mithilfe eines einfachen Bildes darzustellen. Eine Lösung dieses Problems liegt in der Nutzung der dynamischen Eigenschaften eines Computerdisplays. Mittels interaktiver Benutzerinterfaces kann die Darstellung je nach Bedarf verändert und angepasst werden. Neben der grafischen Repräsentation ist somit die Möglichkeit zur Interaktion von entscheidender Bedeutung und eine Grundvoraussetzung jeder computerunterstützten Visualisierung. Weitere Aspekte wie Animation, Navigation, Exploration, etc. sowie verschiedene Techniken zur Unterstützung von Interaktivität (beispielsweise Panning oder Zooming) werden im Rahmen dieser Arbeit u.a. in Abschnitt 2.5.1 behandelt.

Fokus+Kontext

Lokale Details, spezielle Bereiche (Areas of Interest) bzw. einzelne Items sollten im Kontext ihrer Umgebung dargestellt werden (Anmerkung: Als Items werden nachfolgend die mithilfe eines Visualisierungssystems dargestellten Objekte bzw. Elemente bezeichnet). Beispielsweise können fokussierte Objekte im Vordergrund, gleichzeitig jedoch Zusammenhänge im Hintergrund dargestellt werden. Diese Technik kommt u.a. bei *Perspective Walls* zum Einsatz. Eine Alternative wäre die Kombination einer Übersichtsdarstellung mit einer Detaildarstellung (*Overview+Detail*) bzw. der Einsatz von *Fisheye Views*(siehe auch [Card et al., 1999], [Herman et al., 2000]). Diese verschiedenen Ansätze helfen den Überblick zu bewahren und dienen zudem als Orientierungshilfe in den teils sehr komplexen Visualisierungen.

Tasks

Als das Grundprinzip der Information Visualisation wird häufig das von [Shneiderman, 1996] formulierte *Visual Information Seeking Mantra* bezeichnet:

> "Overview first, zoom and filter, details-on-demand."

Daraus können nun prinzipiell folgende Anforderungen (Tasks) an eine Visualisierung abgeleitet werden. Diese weisen einen hohen Abstraktionsgrad auf, wodurch eine Verfeinerung der einzelnen Punkte bzw. eine Erweiterung und Anpassung je nach Anwendungsbereich der Visualisierung vorgenommen werden muss. [Andrews, 2002] [Shneiderman, 1996]

- **Overview**
  Bereitstellung einer Gesamtübersicht über die Menge der visualisierten Elemente (Items). Eine bekannte Strategie hierfür wäre eine Überblicksdarstellung mit einer Detailansicht zu kombinieren. Eine sog. Field-Of-View Box der Überblicksdarstellung bestimmt dabei den Inhalt der Detailansicht. Diese Technik wird auch als *Overview+Detail* bezeichnet.

- **Zoom**
  Zoom auf bestimmte Items oder auf spezielle Regionen der Darstellung. Wichtige Punkte dabei: Faktor und Fokus sollten vom User geändert werden können. Mittels stufenlosem Zooming (Smooth Zoom) sollte das Gefühl für Position und Kontext bewahrt werden. Weiters sollte Zooming jeweils nur in eine Dimension zu einem Zeitpunkt erfolgen.

- **Filter**
  Das Aus- (bzw. auch das Wiedereinblenden) bestimmter Items oder Details der Visualisierung. Dynamische Queries oder Range-Sliders wären Beispiele für Filtermechanismen.

- **Details-on-Demand**
  Bereitstellung von Details ausgewählter Items oder Item-Gruppen. Beispielsweise könnten in einem separaten Fenster nähere Informationen über das ausgewählte Item angezeigt werden.

- **Relate**
  Darstellung von Beziehungen zwischen Items. Das Hervorheben von Items mit ähnlichen oder gleichen Eigenschaften wie das aktuell selektierte Item wäre ein beispielhafter Anwendungsfall.

- **History**
  Protokollierung der einzelnen User-Aktionen um somit Funktionen wie etwa Undo oder Redo zu unterstützen.

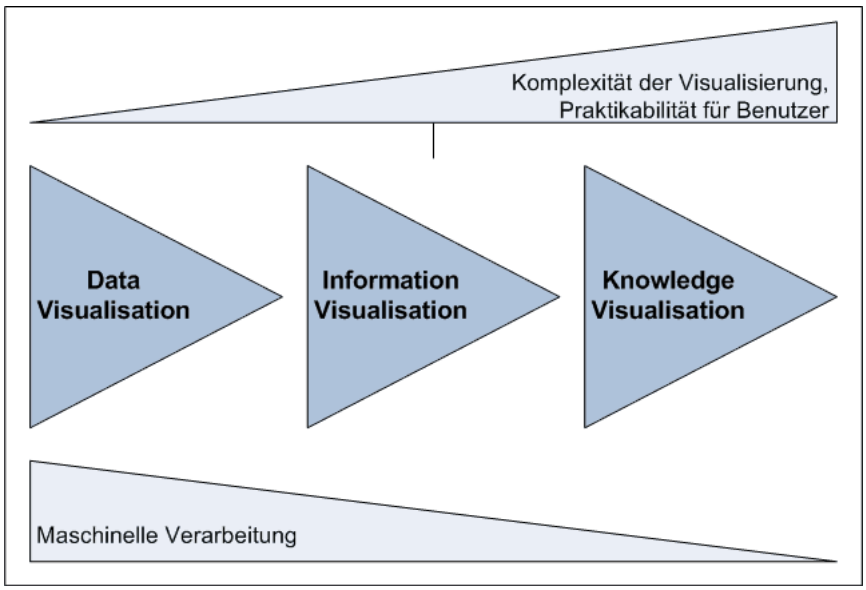


Abbildung 2.1: Teilgebiete der Information Visualisation sowie Relation zwischen Komplexität, Anwendbarkeit und Verarbeitungsaufwand. (basierend auf [Sabol, 2008])

- **Extract**
  Suche nach bzw. Extraktion einer Teilmenge der visualisierten Items. Die extrahierte Menge könnte beispielsweise gespeichert,ausgedruckt oder in einem neuen View visualisiert werden.

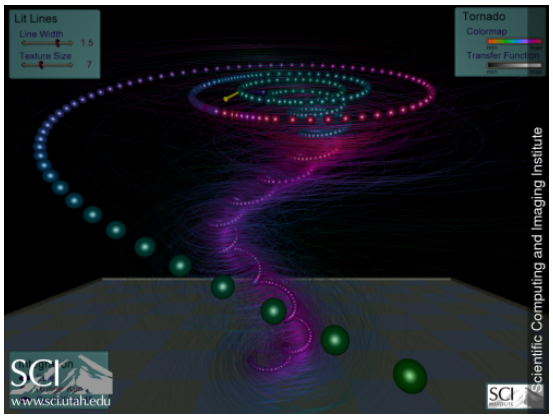
- **Organise**
  Spezifische manuelle Anordnung der visualisierten Items. Ziel ist es, diese zu einem späteren Zeitpunkt leichter wiederfinden zu können.

Wie bereits erwähnt, sollen die hier behandelten Punkte einen ersten Überblick über grundlegende Funktionalitäten, Prinzipien bzw. typische Tasks eines Visualisierungssystems verschaffen. Einzelne Aspekte werden an den entsprechenden Stellen dieser Arbeit nochmals aufgegriffen und ausführlicher behandelt. Im Zuge dessen werden in den jeweiligen Unterkapiteln noch weitere Konzepte und Begriffe vorgestellt und näher erläutert.

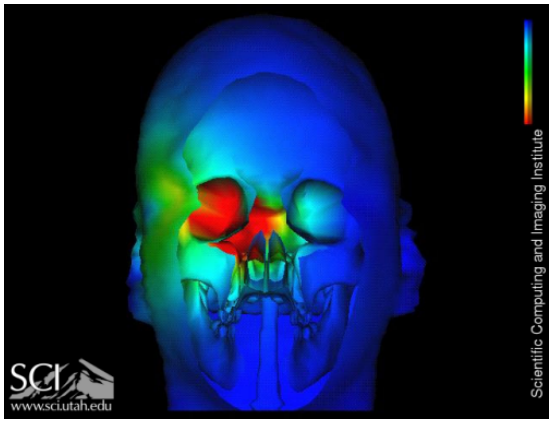
2.1.2 Teilbereiche

Information Visualisation stellt ein umfangreiches Fachgebiet mit zahlreichen Teildisziplinen, Nebenbereichen sowie verwandten Forschungsgebieten dar. Häufig überlappen und überschneiden sich die einzelnen Bereiche, wodurch eine klare Trennlinie nur schwer gezogen werden kann. Wie zuvor bereits erwähnt, werden im Bereich der Information Visualisation Daten, Informationen und Wissen (Definition siehe Abschnitt 1.3) mittels computerunterstützter Methoden in eine grafische Repräsentation konvertiert. Daraus ergibt sich eine mögliche Aufsplittung in die Teilgebiete Data Visualisation, Information Visualisation und Knowledge Visualisation. Abbildung 2.1 zeigt die Relation zwischen Komplexität der Darstellung, Praktikabilität für den Benutzer sowie maschinellem Verarbeitungsaufwand in Abhängigkeit von den einzelnen Teilbereichen. [Andrews, 2002] [Sabol, 2008]

Das verwandte Forschungsgebiet der *Scientific Visualisation* (oft auch als *Data Visualisation* bezeichnet) beschäftigt sich mit der Visualisierung von Daten welche häufig einen physikalischen, medizinischen oder geometrischen Hintergrund besitzen (z.B. Sensor- oder Simulationsdaten). Sie weisen eine inhärente 2D/3D Geometrie auf, wodurch sich eine Darstellung im zwei- bzw. dreidimensionalen Raum anbietet. So beinhalten beispielsweise die Daten einer Luftstromsimulation bereits räumliche Informa-



(a) Tornado-Simulation



(b) Bioelektrisches Feld

Abbildung 2.2: Beispiele für Scientific Visualisation (Quelle: [Utah, 2003]). (a) visualisierter Datensatz einer Tornado-Simulation. (b) Visualisierung des bioelektrischen Feldes eines menschlichen Kopfes.

tionen (z.B. 3D-Vektoren für die Strömungsrichtung). Eine naheliegende grafische Repräsentation wäre die Darstellung der Strömungsrichtung als Pfeile im dreidimensionalen Raum.

Im Gegensatz dazu sind Daten und Informationsstrukturen im Bereich der *Information Visualisation* abstrakt und besitzen demnach keine natürliche grafische Repräsentation (d.h. ein klare räumliche Abbildung (Mapping) der Informationen ist nicht möglich). Dadurch ergeben sich mehrere unterschiedliche Visualisierungsmöglichkeiten. Die Herausforderung der Information Visualisation besteht nun darin, eine geeignete grafische Repräsentation der abstrakten Informationen zu finden bzw. zu entwickeln. Typische Anwendungsbeispiele aus dem Bereich der Information Visualisation wären die Darstellung von Ähnlichkeiten mehrerer Textdokumente untereinander oder die Visualisierung von Metadaten, Datenbankeinträgen, Finanzdaten, o.ä.

Knowledge Visualisation widmet sich hingegen dem Transfer von Wissen. Dabei werden visuelle Repräsentationen eingesetzt um vorhandenes Wissen (zentrale Wissensobjekte, Beziehungen, Attribute, etc.) zwischen Personen oder Gruppen von Personen zu übertragen. [Andrews, 2002] [Munzner, 2000] [Sabol, 2008] [Prinz, 2006] [Burkhard, 2004]

Graph Visualisation, als einem Teilbereich der Information Visualisation, liegen häufig strukturierte Daten/Informationen zugrunde. D.h. es besteht eine inhärente Beziehung zwischen einzelnen Objekten des darzustellenden Datensatzes. In diesem Fall bietet sich die Visualisierung in Form von Graphen an, wobei die einzelnen Objekte als Nodes und deren Beziehung zueinander als Edges dargestellt werden können [Herman et al., 2000]. In weiterer Folge liegt das Hauptaugenmerk dieser Arbeit auf der Visualisierung von Graph- bzw. Netzwerkstrukturen.

Viele Konzepte aus anderen Disziplinen finden auch in der Information Visualisation Verwendung. So werden beispielsweise Ideen aus den Gebieten Wissensmanagement, Semiotik, Kartographie, Kunst, Psychologie, etc. aufgegriffen bzw. adaptiert. Die Haupteinflüsse stammen vor allem aus den Bereichen Computer Graphics und Human-Computer Interaction. [Munzner, 2000]

Die Abbildungen 2.2, 2.3, 2.4, 2.5, 2.6 sowie 2.7 zeigen einige Beispiele für Visualisierungen aus einzelnen Teilbereichen.

2.1.3 Klassifizierung

Informationen im Bereich der Information Visualisation stammen aus den verschiedensten Quellen bzw. besitzen einen unterschiedlichen Hintergrund. Sie weisen differenzierende Merkmale und Eigenschaften

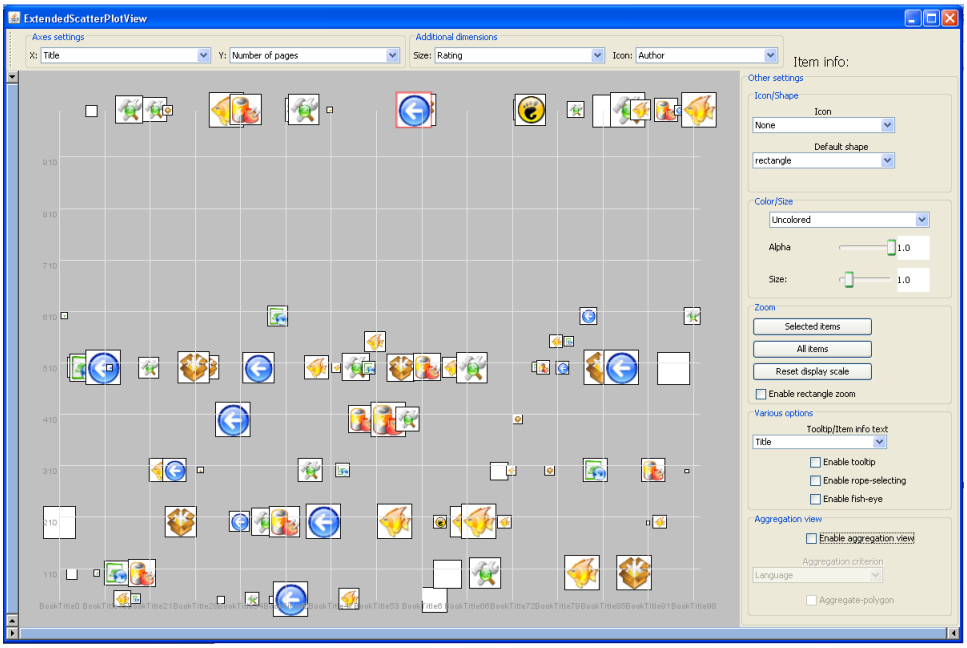


Abbildung 2.3: Scatterplot (Information Visualisation): Visualisierung von Buch-Metadaten entlang der X- bzw. Y-Achse. (Quelle: Screenshot aus [Kandlhofer, 2008])

auf, wodurch sich die Anforderungen an eine Visualisierung je nach Art der darzustellenden Information stark voneinander unterscheiden. Es bedarf einer generellen Einteilung der verschiedenen Informationsarten und der jeweils dafür geeigneten grafischen Repräsentationen. Die nachfolgende Auflistung stellt eine Möglichkeit zur Klassifizierung der Information bzw. der unterschiedlichen Visualisierungen dar. Häufig finden sich auch Variationen der einzelnen Untergruppen sowie Kombination aus mehreren Typen. Als Basis für die folgende Unterteilung dienen dabei die Klassifizierungsvorschläge aus [Shneiderman, 1996] und [Andrews, 2002].

Der Fokus dieser Arbeit liegt auf der Visualisierung von Graphstrukturen bzw. speziell auf der Visualisierung von semantischen Graphen (siehe Abschnitt 2.4). Hinsichtlich einer ausführlicheren Behandlung weiterer Informationstypen und Strukturen sowie einer Auflistung verschiedener Visualisierungsmöglichkeiten sei hier auf [Andrews, 2002] verwiesen.

Linear - Spatial

Lineare Datensätze weisen eine eindimensionale (lineare) Struktur bzw. eine sequentielle Anordnung auf. Beispiele dafür wären Textdokumente, Source-Codes, Namenslisten oder auch Tabellen. Jede Zeile repräsentiert dabei ein darzustellendes Objekt. Fragen der Visualisierung betreffen u.a. Farbe, Form und Größe der Items.

Spatiale (raumbezogene) Datensätze weisen eine inhärente zwei- bzw. dreidimensionale Geometrie auf. Die räumliche Information impliziert bereits eine geeignete Darstellung. Beispiele im zweidimensionalen Raum wären geographische Karten oder Grundrisspläne. Jedem Item wird dabei ein Teil der gesamten 2D-Visualisierungsfläche zugewiesen. CAD-Modelle oder Modelle von Molekülen sind Beispiele im dreidimensionalen Raum. Occlusion (Verdeckung) der dargestellten Objekte sowie Position und Ausrichtung des Betrachters in der 3D-Umgebung stellen dabei nur einige der Herausforderungen und Probleme bei der Visualisierung von Datensätzen diese Typs dar.

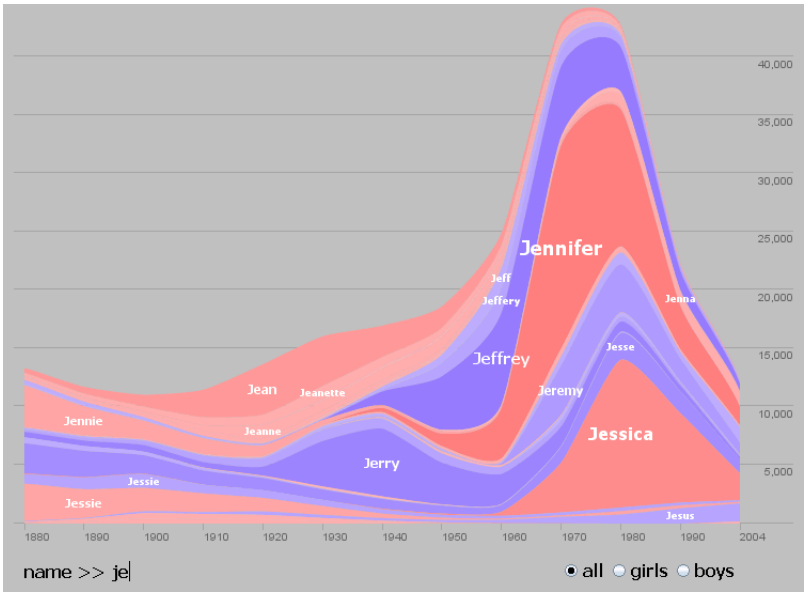


Abbildung 2.4: NameVoyager (Information Visualisation): Darstellung der Häufigkeit verschiedener Vornamen in Abhängigkeit des Geburtsjahres. (Quelle: [Heer, 2008a])

Temporal

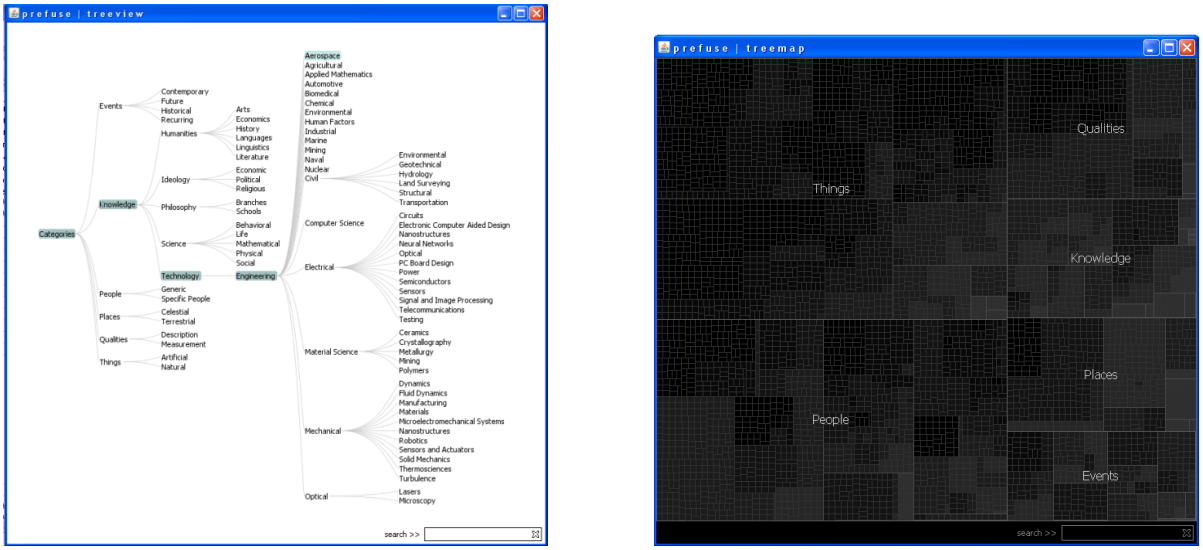
Ein charakteristisches Merkmal von Datensätzen mit zeitlichem Bezug ist, dass Items u.a. einen Start- bzw. Endzeitpunkt aufweisen und sich somit überlappen können. Die dargestellte Zeitachse kann dabei einem linearen, zyklischen oder verzweigten Verlauf folgen. Typische Anwendungsfälle wären das Auffinden von Ereignissen vor, nach, oder während eines bestimmten Zeitpunktes bzw. einer bestimmten Periode. Eine gute Einführung in das Thema Time-Oriented Data findet sich auch unter [Aigner et al., 2007] bzw. [Silva and Catarci, 2000].

Multidimensional

In mehrdimensionalen Datensätzen besitzen die einzelnen Items eine Vielzahl an darzustellenden Attributen. Jedes Attribut steht in diesem Fall für eine Dimension. So können beispielsweise einer Datei oder einem Dokument unterschiedliche Metadaten (Größe, Autor, Zugriffszeiten, Typ, etc.) zugeordnet sein. Ein darzustellendes Objekt wird somit entsprechend seiner n Attribute im n-dimensionalen Raum angeordnet. In weiterer Folge wird meist eine Abbildung (Mapping) der n Dimensionen in den zwei- bzw. dreidimensionalen Raum vorgenommen. Die Visualisierung von multidimensionalen Datensätzen kann z.B. mittels *Parallel Coordinates* Technik oder in Form eines *Scatterplots* erfolgen (siehe dazu auch Abbildung 2.3 sowie Abbildung 2.4). Eine ausführlichere Behandlung des Themas sowie weitere Visualisierungsmöglichkeiten finden sich unter [Weitlaner, 1999] sowie [Mueller, 2005].

Vector Spaces

Vektorräume bestehen aus aus einer Reihe hoch-dimensionaler Vektoren. Die einzelnen Vektoren werden durch den Inhalt der darzustellenden Objekte bestimmt (=content-based). Zur Darstellung ist wiederum ein Mapping in einen niedrig-dimensionalen Raum notwendig. Ein Anwendungsbeispiel wäre die Visualisierung eines Textdokuments mithilfe eines Vektors, wobei die Häufigkeit gewisser Schlüsselwörter innerhalb des Dokuments dabei den Vektor bestimmt. Die Visualisierung kann beispielsweise mittels *Landscape*, *Self Organising Maps* oder diverser Graph Drawing Techniken erfolgen. Die Herausforderung, die sich bei der Visualisierung von content-based Vector Spaces ergibt, ist einerseits die Generierung einer generellen Übersicht über alle Items. Anderseits sollten aber auch Details zu jedem einzelnen



(a) Treeview **(b)** Treemap

Abbildung 2.5: Visualisierung hierarchischer Strukturen bzw. Bäume (Information Visualisation). (Quelle: Screenshots einer Prefuse-Visualisierung [Heer, 2008b]). (a) Darstellung eines hierarchischen Datensatzes mittels Treeview. (b) Darstellung desselben Datensatzes mittels Treemap.

Item zur Verfügung stehen. Bei größeren Datensätzen führen diese widersprüchlichen Forderungen jedoch schnell zu Problemen (siehe dazu auch [Prinz, 2006]).

Tree

Hierarchische Strukturen oder Bäume bestehen aus einer Menge von Knoten (*Nodes*) sowie Verbindungen (*Links, Edges*) zwischen diesen Knoten. Charakteristisch ist dabei die Unterscheidung zwischen *Internal Nodes*, *Leave Nodes*, sowie der *Root Node*. Internal Nodes besitzen, im Gegensatz zu Leave Nodes, Verbindungen zu weiteren Knoten (*Children*). Jeder Knoten, mit Ausnahme des Roots, weist eine Verbindung zu einem übergeordneten Knoten (*Parent Node*) auf. Als Folge existiert ein eindeutiger Pfad vom Root Node zu jedem einzelnen Leave Node. Sowohl Nodes als auch Links können über mehrere unterschiedliche Attribute verfügen.

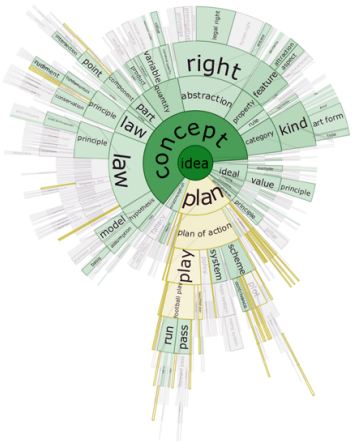


Abbildung 2.6: DocuBurst (Information Visualisation): Visualisierung von Dokumentinhalten mithilfe einer Radialdarstellung (Quelle: [Collins, 2008])

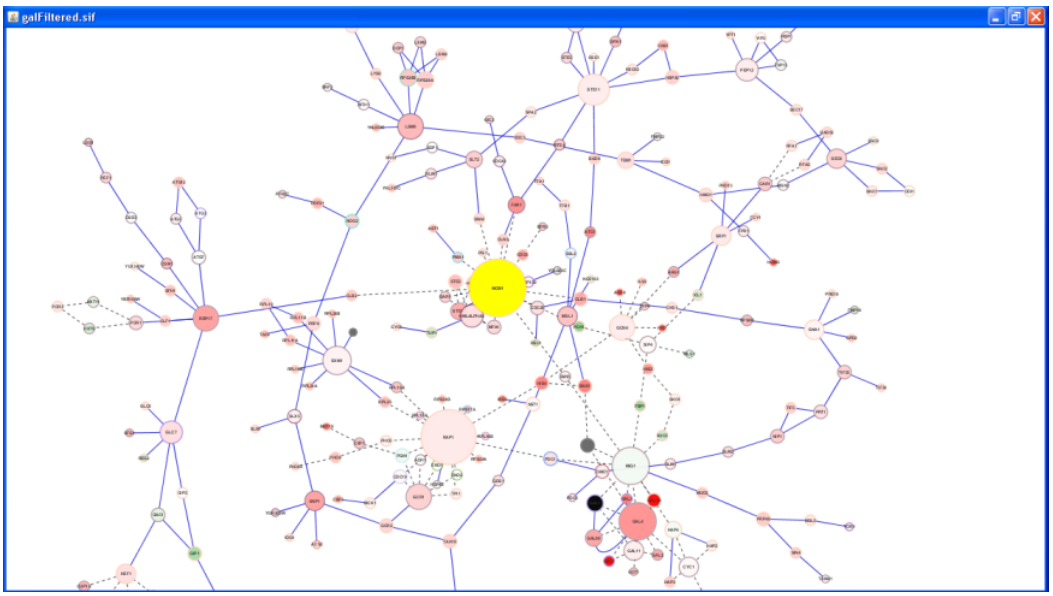


Abbildung 2.7: Beispiel Graph Visualisation: Darstellung eines Datensatzes aus dem Bereich der Bioinformatik (Quelle: Screenshot einer Cytoscape-Visualisierung [Cytoscape, 2009d])

Spezifische Problemstellungen betreffen meist strukturelle Aspekte (beispielsweise die Anzahl der Kinder eines Knoten oder die Anzahl der Ebenen des Baums). Ein Beispiel für eine Baumstruktur wäre ein Filesystem, welches Dateien in Ordner und Unterordner organisiert. Größere Hierarchien, wie beispielsweise solche Dateiverwaltungssysteme, benötigen eine adäquate grafische Repräsentation um Übersichtlichkeit und Verständlichkeit zu gewährleisten. Dazu gibt es eine Reihe unterschiedlicher Visualisierungstechniken und Konzepte. Die Bandbreite reicht dabei von, den aus diversen Dateiverwaltungssystemen bekannten Expand- bzw. Collapse-Funktionalitäten, über traditionelle Tree Drawing Methoden und Algorithmen (siehe z.B. [Reingold and Tilford, 1981] oder [Buchheim et al., 2002]) bis hin zu Tree-Maps, Radial-Maps oder 3D-Ansätzen wie Cone Trees oder 3D Hyperbolic Trees. Neben [Andrews, 2002] finden sich u.a. auch in [Herman et al., 2000], [Munzner, 2000] oder [Andrews et al., 2002] weitere Möglichkeiten, und Ansätze sowie Details bezüglich der Visualisierung von Baum- bzw. hierarchischer Strukturen. Abbildung 2.5 zeigt beispielhaft zwei Möglichkeiten zur Darstellung von Baumstrukturen.

Graphs, Networks

Als Netzwerke werden generelle Graphstrukturen bezeichnet. Solche Strukturen bestehen aus einer Menge von Knoten (*Nodes*) und Kanten (*Edges*), welche die Knoten miteinander verbinden. Nodes können mit beliebig vielen anderen Nodes verbunden sein. Im Gegensatz zu Baumstrukturen, welche eine sog. Top-To-Bottom Ordnung aufweisen (ausgehend vom Root Node gibt es immer einen eindeutigen Pfad zu den Leave Nodes), können die Verbindungen in einem Network/einem Graph auch zyklischen Charakters sein. Ein Baum stellt somit den Spezialfall eines azyklischen zusammenhängenden Graphen dar. Mithilfe von Graphstrukturen ist es möglich, komplexe Relationen zwischen Objekten zu beschreiben bzw. abzubilden. Die Größe eines Netzwerks oder auch die Komplexität der Beziehungen spielt eine entscheidende Rolle bei der Entwicklung bzw. der Wahl einer geeigneten Visualisierung. Eine Technik, welche für kleinere Graphen gut funktioniert, kann bei der Darstellung größerer Graphen bzw. Netzwerke versagen. Die Performance und Anwendbarkeit verschiedener Algorithmen hängt stark von der Anzahl der Nodes und Edges eines Graphen ab. In weiterer Folge widmen sich anschließende Abschnitte den unterschiedlichen Aspekten, Frage- und Problemstellungen welche sich im Kontext der Visualisierung von Graphen bzw. Netzwerken ergeben.

Alternative Klassifizierung

Wie bereits erwähnt, stellt die hier vorgenommene Einteilung nur eine Möglichkeit zur Strukturierung bzw. Klassifizierung von Informationen und Visualisierungen dar. Eine Alternative wäre beispielsweise die Unterscheidung zwischen *einfachen visuellen Strukturen* (Scatterplots, Pie Charts, Trees,...), *zusammengesetzten visuellen Strukturen* (Scatterplot-Matrizen, Parallel Coordinates, Graphen,...), *interaktiven visuellen Strukturen* (dynamische Abfragen, Overview+Detail Techniken,...) sowie *visuellen Fokus+Kontext Abstraktionen* (Distortion, Filtering, Aggregation,...) [Mueller, 2005].

2.1.4 Kodierung, Metaphern

Als visuelle Kodierung (*Visual Encoding*) wird die Abbildung von Informationen (sowohl numerisch als auch nicht-numerisch) auf graphische Elemente bezeichnet. Die Kombination mehrerer visueller Kodierungen führt zu einer visuellen Metapher (*Visual Metaphor*). Kodierungen und Metaphern sollen den Aufbau eines gedanklichen Modells (*Mental Map*) erleichtern. In der Literatur finden sich mehrere synonyme Bezeichnungen für die verschiedenen visuellen Kodierungen, darunter *Display Dimensions, Display Channels, Perceptual Dimensions oder Retinal Variables*.

Die räumliche Anordnung der einzelnen Objekte bildet die Basis der meisten Visualisierungen. Punkte, Linien, Symbole, oder Icons als grafische Repräsentationen der darzustellenden Objekte, werden im Raum platziert. Aufgrund dieser Anordnung können bereits Rückschlüsse, beispielsweise auf die Struktur der visualisierten Daten/Informationen, gezogen sowie weitere Erkenntnisse bzw. ein generellerer Überblick gewonnen werden. Zusätzlich zur räumlichen Anordnung können weitere Informationen beispielsweise mittels unterschiedlicher Farbe, Größe oder Form der Items dargestellt werden. Einige Beispiele für visuelle Kodierungen wären die Darstellung der Windrichtungen einer Wettersimulation als Pfeile oder die Kodierung der Seitenanzahl verschiedener Bücher mithilfe der Item-Größe. Die folgende Auflistung zeigt eine Übersicht über häufig verwendete Kodierungsformen. ([Munzner, 2000] [Sabol, 2001])

- **Position Encoding**
  Items werden entlang einer Achse in Abhängigkeit eines numerischen oder nicht-numerischen Wertes positioniert.
- **Proximity Encoding**
  Items welche eine größere Gemeinsamkeiten miteinander aufweisen, werden näher nebeneinander platziert. Gruppen von ähnlichen Objekten können so schneller erkannt werden
- **Size/Brightness/Color Encoding**
  Größe oder Helligkeit eignet sich besonders zur Kodierung von quantitativen Informationen. Aufgrund der unterschiedlichen Größe können beispielsweise Rückschlüsse auf die Anzahl der in den Items enthaltenen Elemente gezogen werden. Verschiedene Helligkeitswerte geben zum Beispiel Aufschluss über Relevanz oder Priorität der Items. Mithilfe von Farben lassen sich neben quantitativen Informationen auch Informationen über den Typ bzw. die Art des dargestellten Objektes kodieren.
- **Shape Encoding**
  Die Form eines Items wird häufig zur Kodierung von nicht-numerischen Daten verwendet. Geometrische Formen (Kreise, Rechtecke, Kuben, Kugeln, etc.), Icons oder Symbole repräsentieren einzelne Objekte. Relationen zwischen Objekten werden, wie bereits erwähnt, meist mittels Baumstrukturen dargestellt.

Im Hinblick auf die Visualisierung von Graphen/Netzwerken spielen vor allem die Punkte Position Encoding und Proximity Encoding eine zentrale Rolle. Auch die geeignete Darstellung von Beziehungen zwischen einzelnen Objekten stellt dabei eine der Schlüsselfragen dar.

2.2 Visualisierung von Graphen

Einer der Schwerpunkte im Kontext dieser Arbeit ist die Visualisierung von Graphen, Netzwerken bzw. hierarchischen Strukturen. Wie bereits in Abschnitt 2.1.2 erwähnt, kann *Graph Visualisation* als ein Teilbereich der Information Visualisation betrachtet werden. Zentrale Punkte im Zuge der Visualisierung abstrakter Informationen sind die Abbildung von Beziehungen zwischen Objekten sowie das Erkennen von Mustern und zuvor verborgenen Relationen. In manchen Fällen besteht bereits eine inhärente Beziehung zwischen einzelnen Objekten des darzustellenden Datensatzes. Liegen solche strukturierten Informationen vor, ist eine Darstellung in Form eines Graphen oder eines Baumes ebenfalls naheliegend.

In einigen Publikationen findet sich eine Differenzierung zwischen Graph Visualisation und dem Begriff des *Graph Drawings*. Grundlegendes Graph Drawing, also das Zeichnen von Graphen, beschäftigt sich demnach in erster Linie mit der Entwicklung von Algorithmen zur geometrischen Repräsentation von Graphen und Netzwerken. Das Hauptaugenmerk liegt dabei vor allem auf der geeigneten Platzierung von Knoten und Beschriftungen (*Labels*) sowie dem Zeichnen von Verbindungen (Links, Edges) zwischen den Knoten. Im Gegensatz dazu umfasst Graph Visualisation ein weitläufigeres Themengebiet. Mensch-Maschine Kommunikation, Graphalgorithmen, Graphentheorie oder Grafikdesign stellen nur einige der involvierten Bereiche dar ([Wong et al., 2006] [Herman et al., 2000] [Prinz, 2006] [Katifori et al., 2007] [Cui, 2007]).

Graph Visualisation gewann in den letzten Jahrzehnten stetig an Bedeutung. Im Zuge dessen wurden bzw. werden permanent neue Ansätze, Erkenntnisse sowie unterschiedliche Techniken und Konzepte entwickelt und veröffentlicht. Graph Visualisation ist Gegenstand aktueller Forschung. So behandelt beispielsweise das jährlich stattfindende Graph Drawing Symposium neben 'klassischen' Graph Drawing Problemen auch Themen aus dem Bereich der Graph Visualisation (siehe dazu z.B. [Symposium, 2009]).

Die Visualisierung von Graphen spielt auf den unterschiedlichsten Gebieten und in den verschiedensten Bereichen eine wichtige Rolle. Neben dem Gebrauch von Graphen im Alltag (z.B. die Darstellung eines Busfahrplans) finden sich vor allem auf dem Sektor der Computerwissenschaften eine Fülle von Anwendungsmöglichkeiten. Die Visualisierung von Netzwerken (Rechnernetze, soziale Netze, semantische Netzwerke, Hyperlink-Netzwerke, Netzwerke aus dem Bereich der Bioinformatik, etc.), die grafische Repräsentation von File- bzw. Dokumenthierarchien oder die Darstellung unterschiedlichster Datenstrukturen in Form eines Graphen wären nur einige Beispiele.

In den nachfolgenden Abschnitten werden einige grundlegende Ansätze und Erkenntnisse aus der Graphentheorie erläutert. Dies soll einerseits zu einem besseren Verständnis der in dieser Masterarbeit behandelten Thematik führen. Andererseits soll dadurch ein genereller Überblick über einige wichtige Grundaspekte des Graph Drawings verschafft werden. Eine detailliertere Sichtweise bieten die Standardwerke von [Battista et al., 1999], [Kaufmann and Wagner, 2001] und [Sugiyama, 2002]. Diese stellen einen guten Ausgangspunkt für eine intensivere Auseinandersetzung mit den Themen Graphentheorie, Graphalgorithmen bzw. Graph Drawing dar.

2.2.1 Graphen, Netzwerke, Bäume

Dieser Abschnitt widmet sich in erster Linie der Definition eines Graphen und seiner Bestandteile. Des weiteren soll an dieser Stelle eine allgemeine Unterscheidung zwischen den Begriffen Graph, Netzwerk und Baum vorgenommen werden.

Ein *Graph* kann als abstrakte Struktur, mit deren Hilfe sich Relationen zwischen Objekten beschreiben bzw. abbilden lassen, bezeichnet werden. Mathematisch formuliert besteht ein Graph G aus einer Menge V von Knoten (*Nodes, Vertices*) und einer Menge E von Kanten (*Edges, Links*), oder kurz $G = (V, E)$. Die endliche Menge V enthält keine doppelten Einträge (d.h. jeder Knoten ist nur einmal in V enthalten). Jeder Knoten hat zumindest einen Namen bzw. eine ID und ist somit eindeutig identifizierbar. E besteht hingegen aus Teilmengen der Menge V. Jedes Element der Menge E beschreibt dabei

die Relation zwischen zwei Nodes, d.h. eine Kante repräsentiert die Verbindung zwischen zwei Knoten. Dabei kann ein und dasselbe Knotenpaar auch mehrerer unterschiedliche Kanten aufweisen. Zusätzlich zu Namen bzw. Nummern zur eindeutigen Identifizierung können sowohl Nodes als auch Edges je nach Art des Graphen auch über weitere Attribute verfügen.

Ein *Baum (Tree)* stellt einen Sonderfall eines Graphen dar. Eine solche hierarchische Struktur weist spezielle Eigenschaften auf, wodurch sie sich von einem allgemeinen Graphen unterscheidet. In Abschnitt 2.1.3 wurde bereits auf die Charakteristika eines Baumes sowie auf die Unterschiede und Gemeinsamkeiten zwischen Bäumen und Graphen hingewiesen. Ein Baum ist demnach ein azyklischer zusammenhängender Graph (siehe dazu auch Abschnitt 2.2.2). Techniken zur Darstellung hierarchischer Strukturen sind auch im Hinblick auf die Visualisierung von Graphen von Bedeutung. Es gibt eine Reihe von Algorithmen zur Umwandlung von generellen Graphen in eine Baumstruktur. Das Ergebnis dieser Transformation wird als *Spanning Tree* bezeichnet. Layoutalgorithmen für Bäume sind in der Regel einfacher zu implementieren und weisen zudem eine niedrige Komplexität auf (siehe dazu auch [Herman et al., 2000] [Andrews, 2002]).

Die Bedeutung der Begriffe *Netzwerk (Network)* und Graph weist eine hohe Ähnlichkeit bzw. einige Gemeinsamkeiten auf. In der Literatur finden sich teils unterschiedliche Definitionen eines Netzwerks, teils wird gänzliche auf eine Unterscheidung verzichtet. So führt etwa [Prinz, 2006] als Unterscheidungsmerkmal an, dass ein Netzwerk eine eindeutige Semantik aufweist. Graphen können in Folge zur Beschreibung solcher Netzwerke verwendet werden. Eine weitere Möglichkeit wäre die Betrachtung eines Netzwerks als einen gewichteten, gerichteten Graphen ([Linz, 2002]). In [Shneiderman and Aris, 2006] werden hingegen die Begriffe Network und Graph als äquivalent betrachtet. Wie in Abschnitt 2.1.3 erwähnt, kann eine generelle Graphstruktur als Netzwerk bezeichnet werden. In Anlehnung daran werden im Rahmen dieser Arbeit, auch mangels einer exakten und sinnvollen Definition bzw. Unterscheidungsmöglichkeit, die Begriffe Netzwerk und Graph in synonymer Weise verwendet. [Kaufmann and Wagner, 2001] [Sugiyama, 2002] [Linz, 2002] [Prinz, 2006]

2.2.2 Eigenschaften von Graphen

Nodes und Edges als Bausteine eines Graphen weisen je nach Anwendungsfall und Anwendungsgebiet unterschiedliche Merkmale auf. In Abhängigkeit der Charakteristika von Edges (bzw. auch Nodes) können unterschiedliche Graphtypen identifiziert werden. Im Anschluss erfolgt nun die Betrachtung grundlegender, für die Thematik dieser Arbeit relevanter Eigenschaften. Weiters werden einige einige spezielle Graphtypen vorgestellt.

In einem *gerichteten Graphen (Directed Graph)* werden Knotenpaare in einer bestimmten Richtung miteinander verbunden. Eine gerichtete Kante verläuft dabei von einem Source-Node zu einem Target-Node. Die Darstellung der Richtung erfolgt meist mittels Pfeilen. Ein directed Graph besteht in der Regel zur Gänze aus gerichteten Kanten. In Ausnahmefällen können jedoch auch einzelne ungerichtete Kanten vorhanden sein. Ein gerichteter Graph verfügt über einige spezielle Merkmale. So kann etwa der *Knotengrad (Node Degree)*, welcher prinzipiell Auskunft über die Anzahl der mit diesem Knoten verbundenen Kanten gibt, in *Ausgangsgrad (Out-Degree)* (= Anzahl ausgehender Kanten) bzw. *Eingangsgrad (In-Degree)* (= Anzahl eingehender Kanten) aufgeteilt werden.

Ein *zusammenhängender Graph (Connected Graph)* stellt einen weiteren Spezialfall dar. Ein solcher Graph besteht aus einer einzigen Komponente, d.h. nur durch Aufbrechen einzelner Kanten können einzelne Knoten bzw. Gruppen von Knoten vom ursprünglichen Graph getrennt und somit weitere Komponenten geschaffen werden. In einem zusammenhängenden Graph existiert demzufolge immer ein Pfad von einem Knoten zu jedem anderen beliebigen Knoten. Als *Pfad (Path)* wird in diesem Fall eine Abfolge mehrerer, mittels Edges verbundener Nodes bezeichnet. Die Konnektivität eines Graphen spielt vor allem in der Berechnung eines geeigneten Layouts eine bedeutende Rolle. Ist der Graph nicht zusammenhängend, können die einzelnen Komponenten unabhängig voneinander angeordnet werden, was u.a. Einfluss auf die Performance des Algorithmus hat.

In einem *gewichteten Graphen (Weighted Graph)* weisen Kanten ein zusätzliches Attribut auf, welches das Kantengewicht bestimmt. Das Gewicht wird durch einen nummerischen Wert beschrieben und repräsentiert, je nach Anwendung etwa Distanzen, Prioritäten, Kosten, Zeitspannen, etc. Für viele Algorithmen stellt das Kantengewicht einen maßgeblichen Input, beispielsweise zur Cluster- oder Layoutberechnung, dar.

Ein *Spannbaum (Spanning Tree)* entsteht durch Transformation eines generellen Graphen in einen Baum. Dieser enthält alle Nodes des ursprünglichen Graphen, wobei nicht benötigte Edges jedoch weggelassen werden. Die Reduktion eines Graphen auf einen Spannbaum und die anschließende Darstellung dieser hierarchischen Struktur ist eine häufig verwendete Visualisierungstechnik. Zur Extraktion eines Spanning Trees sowohl aus gerichteten als auch aus ungerichteten Graphen existiert eine Reihe unterschiedlicher Algorithmen (siehe auch [Munzner, 2000], [Herman et al., 2000] bzw. [Andrews, 2002]).

Die Relation zwischen Knoten- und Kantenanzahl ist ein weiteres Unterscheidungsmerkmal. Diese Relation spielt auch in der Betrachtung der Laufzeitperformance von Layoutalgorithmen bzw. in der Berechnung des Speicherbedarfs für Graphen eine entscheidende Rolle. In einem *vollständigen Graphen (Complete Graph)* steht die Anzahl der Nodes n und die Anzahl der Edges e im Verhältnis $e = n * (n - 1)/2$ zueinander. Jeder Knoten ist mit jedem anderen Knoten durch genau eine Kante verbunden. Erreicht die Kantenanzahl annähernd n^2 so kann von einem *dichten Graphen (Dense Graph)* gesprochen werden. Ein *lichter Graph (Sparse Graph)* weist hingegen eine sehr viel kleinere Kantenanzahl als n^2 auf.

Als *planar* wird ein Graph bezeichnet, wenn dessen Darstellung auf einer Ebene keine Ueberschneidungen der Kanten aufweist. Dies wird beispielsweise durch das Zeichnen von gekrümmten Kanten (Kurven) erreicht. Planarität ist vor allem für die Berechnung des Layouts von kleineren bzw. lichten Graphen von Bedeutung. Diese können beispielsweise durch Extraktion von Subgraphen oder durch Clustering bzw. Filtering großer/dichter Graphen entstehen. Bei einer größeren Anzahl an Nodes bzw. Edges spielen Planarity-Tests oder die Minimierung von Kantenüberschneidungen eine untergeordnete Rolle (siehe dazu auch [Herman et al., 2000] bzw. [Kaufmann and Wagner, 2001]).

Weitere Eigenschaften und Begriffe im Zusammenhang mit Graphen wären Hypergraph, Cycle, Self-Loop bzw. Graph Theoretic Distance. Als *Hypergraph* wird ein Graph bezeichnet, in dem eine Kante mehr als zwei Knoten miteinander verbindet. Ein *Cycle* ist ein einfacher Pfad (= jeder Knoten wird exakt einmal aufgesucht) mit der Eigenschaft, dass ein und derselbe Knoten sowohl den Ausgangs- als auch den Endpunkt darstellt. Eine *Self-Loop* (oder auch Self-Edge), als Sonderfall eines Cycle, verbindet hingegen einen Knoten nur mit sich selbst. Die *Graph Theoretic Distance* zwischen zwei Knoten ist gleich der Länge des kürzesten Pfades, welcher diese beiden Knoten verbindet. Viele Algorithmen verwenden diese Distanz zur Beschreibung von Nähe-Relationen zwischen Knoten ([Prinz, 2006].

Es gibt noch eine Reihe weiterer Merkmale, Sonderfälle und Unterteilungsmöglichkeiten eines Graphen. Im Kontext dieser Arbeit sind in weiterer Folge vor allem gerichtete Graphen mit gewichteten Kanten von Bedeutung. Dazu sei an dieser Stelle auch auf Abschnitt 2.4 verwiesen, welcher sich u.a. mit semantischen Aspekten sowie mit dem Spezialfall eines semantischen Graphen auseinandersetzt. [Kaufmann and Wagner, 2001] [Sugiyama, 2002] [Linz, 2002] [Prinz, 2006]

2.2.3 Graph Drawing

Graph Drawing (bzw. *Graph Layout*) kann, wie bereits erwähnt, als Teilaspekt der Graph Visualisation betrachtet werden. Es gibt verschiedene Möglichkeiten zur Repräsentation bzw. Beschreibung der abstrakten Struktur eines Graphen. Formalistische, textbasierte Varianten, wie beispielsweise in Abschnitt 2.2.6 beschrieben, eignen sich vor allem zur computerunterstützten Weiterverarbeitung oder Speicherung. Graph Drawing beschäftigt sich hingegen mit der Frage, wie die in einer Graphstruktur enthaltenen Informationen in eine für Menschen verständliche und ansprechende Form umgewandelt werden können. Mithilfe verschiedener Layout- bzw. Graph Drawing Algorithmen erfolgt eine Transformation der abstrakten Graphstruktur $G = (V, E)$ in eine räumliche Darstellung. Dabei werden geometrische Ob-

jekte, als Repräsentation der Elemente eines Graphen, im zwei- bzw. dreidimensionalen Raum (*Drawing Space*) abgebildet.

Es existiert eine Vielzahl an verschiedenen visuellen Representationsmöglichkeiten. Je nach Anwendung werden Nodes und Edges in unterschiedlicher Form bzw. mithilfe unterschiedlicher geometrischer Objekte dargestellt. Knoten können beispielsweise durch Punkte, Kreise, Rechtecke oder nur durch ihre Labels repräsentiert werden. Die Darstellung der Kanten kann zum Beispiel mittels Geraden, Kurven oder orthogonalen Pfaden erfolgen. Pfeile bzw. Farbverläufe zeigen zumeist die Orientierung einer Kante in einem gerichteten Graphen an. Weitere knoten- bzw. kantenbezogene Informationen können zum Beispiel mithilfe von Labels oder Farb- bzw. Größenkodierungen (siehe [Kaufmann and Wagner, 2001] Kapitel 1) dargestellt werden.

Die grundsätzliche Problemstellung des Graph Drawings kann wie folgt zusammengefasst werden:

- Berechnung der Positionen sowie Platzierung der Nodes im Drawing Space (*Node Placement*)
- Zeichnen der Edges zwischen verbundenen Nodes (*Edge Routing*)
- Platzierung von Beschriftungen für Nodes und/oder Edges (*Label Placement*)

Eine zentrale Stellung nimmt dabei die Platzierung der Nodes ein. Mathematisch betrachtet ist dies eine Funktion, welche die Menge der Knoten V auf einen m-Dimensionalen Raum abbildet ($f : V \rightarrow R^m$). Die Anordnung der Knoten hat die größte Auswirkung auf das Endergebnis, weshalb der Fokus vieler Layoutalgorithmen auf diesem Task liegt. Einen Überblick über unterschiedliche Algorithmen und Techniken bietet Abschnitt 2.2.5.

Die Darstellung der Relationen zwischen Nodes ist eine weitere wichtige Aufgabe des Graph Drawings. Edges können als Gerade bzw. bei Bedarf auch als gekrümmte Linie oder Pfeil gezeichnet werden. Eines der Hauptprobleme stellt dabei die Vermeidung von Kantenüberschneidungen dar. Weiters sollten Nodes nicht von Edges überlagert werden. Dies setzt jedoch einen ausreichend großen Raum zwischen den einzelnen Nodes voraus. Es gibt verschiedene Ansätze und Techniken zur Lösung dieses Problems. Viele Algorithmen platzieren in einer ersten Phase die Knoten und zeichnen erst im Anschluss daran die entsprechenden Kanten. Die Positionen der Nodes können einerseits fixiert sein, wodurch das Zeichnen der Edges keinerlei Auswirkung auf die ursprüngliche Darstellung hat. Andererseits besteht aber auch die Möglichkeit, die Positionen der Nodes im Zuge des Edge Routings zu adaptieren (siehe dazu auch [yWorks, 2009b] bzw. [Kaufmann and Wagner, 2001] Kapitel 6).

Die Platzierung von Labels spielt vor allem im Hinblick auf die Visualisierung semantischer Graphen eine entscheidende Rolle. Nodes und Edges sind mit zusätzlichen Informationen verknüpft, welche in der Darstellung ebenfalls berücksichtigt werden sollten. Intuitiv wäre eine Positionierung des Labels in der Nähe des beschrifteten Elements am geeignetsten. Teile des Graphen bzw. andere Labels sollten dadurch jedoch nicht verdeckt oder überlagert werden. Weiters ist auf die Lesbarkeit der Labels zu achten. Einige spezielle Label Placement Algorithmen haben einen bereits gezeichneten Graph als Ausgangspunkt. Die Position bzw. die Größe von Nodes und Edges wird durch das Hinzufügen von Labels nicht mehr geändert. In andere Algorithmen ist Label Placement hingegen ein integrierter Teil der gesamten Layoutberechnung. Eine ausführliche Behandlung dieses Themas findet sich auch unter [yWorks, 2009a], [Wagner et al., 2000] bzw. [Kaufmann and Wagner, 2001] Kapitel 10. [Wong et al., 2006] präsentiert einen interessanten Label Placement Ansatz. Labels für Kanten und Knoten werden dabei direkt in die Struktur des Graphen eingebunden. Beispielsweise werden Edges nicht wie üblich durch Linien, sondern durch längliche Schriftzüge repräsentiert.

Node Placement, Edge Routing sowie Label Placement stellen zusammen eine große algorithmische Herausforderung dar. Meist hat daher die Positionierung von Knoten bzw. das Zeichnen von Kanten Priorität gegenüber der Behandlung von Labels. Häufig wird auch gänzlich auf Label Placement verzichtet. [Kaufmann and Wagner, 2001] [Herman et al., 2000] [Prinz, 2006]

2.2.4 Ästhetische Kriterien

Um die menschliche Wahrnehmungsfähigkeit nicht zu überfordern, sollte beim Zeichnen eines Graphen die richtige Balance zwischen Detailgrad und Übersichtlichkeit gefunden werden. Wie in Abschnitt 2.2.3 erläutert, stellt sich dabei die Frage nach einer geeigneten Abbildung bzw. Anordnung der Nodes, Edges und Labels im Drawing Space. Ziel ist es, ein ansprechendes und intuitives Layout zu generieren. Ob eine Darstellung als intuitiv oder ästhetisch betrachtet werden kann, hängt stark von der jeweiligen Anwendung bzw. vom Betrachter selbst ab. Es bedarf aufgrund dessen einiger allgemein gültiger Kriterien, um objektive Aussagen über die Qualität einer Graphdarstellung machen zu können. In diesem Zusammenhang wird häufig von *Ästhetikkriterien* gesprochen. Diese Kriterien umfassen eine Reihe von Konventionen und Regeln das Layout eines Graphen betreffend. [Sugiyama, 2002] unterscheidet dabei zwischen zwingenden Vorgaben (grundlegende Konventionen hinsichtlich Node Placement und Edge Routing) und Ästhetikregeln. Diese können einen semantischen oder einen strukturellen Hintergrund aufweisen.

Semantische Regeln basieren auf der Bedeutung von Knoten und Kanten (z.B. die Priorität eines Knoten oder die Wichtigkeit einer Relation). Die Bedeutung wird dabei entweder durch den Benutzer definiert oder automatisch, beispielsweise aus vorhandenen Labels, abgeleitet. Im Gegensatz dazu betreffen *strukturelle Regeln* ausschließlich graphtheoretische Merkmale (wie etwa Node Degree, Kantenlänge, etc.).

Nachfolgend werden einige der wichtigsten Ästhetikkriterien aufgelistet und diskutiert (siehe auch [Kern, 2005], [Prinz, 2006] sowie [Kaufmann and Wagner, 2001] Kapitel 1). Eine umfangreiche Aufstellung weiterer Konventionen und Regeln findet sich u.a. in [Sugiyama, 2002].

- **Vermeidung von Überschneidungen**
  Grundlegende Regeln betreffen vor allem die Vermeidung bzw. Minimierung von Knoten- und Kantenüberschneidungen. Je größer die Anzahl an Überlappungen von Nodes und Edges, desto schwieriger ist es, Relationen zwischen Nodes zu erkennen.

- **Knotenabstand**
  Der Abstand der Nodes zueinander sollte die Stärke bzw. Art der Relation widerspiegeln. So könnten Knoten, welche eine starke Verbindung aufweisen, näher beieinander platziert werden. Die Stärke sowie die Art der Relation kann zum Beispiel mit Hilfe des Kantengewichts bestimmt werden. Weiters sollte der Abstand zwischen nicht zusammenhängenden Knoten maximiert werden um so Fehlinterpretationen hinsichtlich der Beziehungen von Nodes zu vermeiden.

- **Gruppierung - Clustering**
  Die Behandlung großer bzw. dichter Graphen bringt einige Probleme mit sich. Viele Layoutalgorithmen stoßen an ihre Grenzen, sobald die Anzahl an Nodes und Edges eine gewisse Schwelle überschreitet. Eine zu komplexe und umfangreiche Darstellung des Graphen bzw. seiner einzelnen Element überfordert zudem den Betrachter. Eine Möglichkeit zur Reduktion von Details ist die Gruppierung von zusammengehörenden Teilen. Eine intensivere Behandlung dieser Thematik folgt in Abschnitt 2.3.

- **Form und Symmetrie**
  Wenn einzelne Teile der Darstellung eine spezielle Form aufweisen bzw. symmetrisch angeordnet sind, ist es für den Betrachter wesentlich einfacher, Zusammenhänge zu erfassen sowie Relationen und Muster zu erkennen.

- **Maximierung der Kantenwinkel**
  Größere Winkel zwischen Edges erleichtern das Erkennen bzw. die Unterscheidung einzelner Kanten.

- **Minimierung von Kantenkrümmungen**
  Gekrümmte oder geknickte Linien sind wesentlich schwerer zu verfolgen als eine Gerade. Aus diesem Grund sollte sowohl die Gesamtzahl an Kantenkrümmungen als auch die Anzahl der Krümmungen einer einzelnen Kante so niedrig wie möglich gehalten werden.

- **Minimierung der Zeichenfläche**
  Weitläufige Darstellungen können vom Betrachter nur schwer erfasst werden. Eine Minimierung der Zeichenfläche (bzw. des Drawing Space) erhöht die Informationsdichte.

- **Weitere strukturelle Regeln**
  Neben den bisher vorgestellten grundlegenden strukturellen bzw. semantischen Regeln gibt es noch eine Reihe weiterer ästhetischer Forderungen. So sollten etwa Nodes, welche einen hohen Degree aufweisen, im Zentrum platziert werden ('Sternform'). Isomorphe Subgraphen sollten identisches Layout aufweisen. Weiters wäre eine Minimierung der durchschnittlichen Kantenlänge wünschenswert.

Je nach Anwendungsfall widersprechen sich einzelne Regeln und Konventionen. Die Einhaltung bzw. Beachtung sämtlicher Kriterien ist somit unmöglich. Diverse Usability Studien und Untersuchungen zeigten zudem, dass sich Ästhetikkriterien unterschiedlich auf einen Betrachter auswirken (siehe dazu auch [Purchase, 1997]). Unter Berücksichtigung dieser Erkenntnisse und zur Vermeidung von Abhängigkeiten bzw. Widersprüchen zwischen einzelnen Regeln/Konventionen nehmen Layoutalgorithmen meist eine Priorisierung der Kriterien vor ([Herman et al., 2000] [Sugiyama, 2002]). Weiters ist zu erwähnen, dass viele ästhetische Forderungen, im Speziellen jene struktureller Natur, in erster Linie im Falle kleinerer Graphen zu erfüllen sind. Die Verhinderung von Überschneidungen und Überlagerungen oder die Maximierung von Kantenwinkeln wird hinsichtlich Graphkonstrukten mit weit über 1000 Elementen nur bedingt möglich sein.

2.2.5 Layout-Techniken

Dieser Abschnitt bietet einen generellen Überblick über unterschiedliche Techniken und Algorithmen zur Berechnung eines Graphlayouts. Bei der Wahl eines Layout-Algorithmus spielen Aspekte wie die *Größe* des Graphen, die *Vorhersagbarkeit* des Algorithmus oder die *Zeitkomplexität* eine wichtige Rolle.

Die Anzahl der Elemente eines Graphen hat entscheidenden Einfluss auf die Performance eines Layout-Algorithmus. Die Darstellung großer Graphen ist mit einigen Problemen verbunden und erfordert spezielle Techniken zur Visualisierung (siehe dazu auch Abschnitt 2.2.7.

Vorhersagbarkeit ist eine wichtige Eigenschaft eines Graph-Algorithmus. Zwei unterschiedliche Durchläufe des Algorithmus sollten in annähernd identischen Layouts des selben Graphen resultieren. Ändert sich die Darstellung nach jedem Durchlauf zu stark, kann dies u.a. die Möglichkeiten zur Interaktion bzw. Navigation beeinträchtigen.

Zeitkomplexität ist ein weiterer entscheidender Faktor bei der Auswahl eines Layout-Algorithmus. Interaktion in annähernd Echtzeit ist eine der Grundvoraussetzungen eines Visualisierungssystems. [Herman et al., 2000]

Ob ein Algorithmus zufriedenstellende und brauchbare Ergebnisse liefert, hängt unter anderem auch stark vom jeweiligen Anwendungsgebiet ab. Je nach Anwendung weisen Graphen eine unterschiedliche Bedeutung auf. Ein anwendungsspezifische Bedeutung kann jedoch von Standardalgorithmen in den seltensten Fällen berücksichtigt werden. Diese sind zumeist auf die Einhaltung allgemeiner Ästhetik- bzw. Performancekriterien optimiert, welche mitunter im Gegensatz zu semantischen Forderungen stehen. Aus diesem Grund ist eine Anpassung der Algorithmen notwendig. Ziel ist dabei die Erfüllung sowohl anwendungsspezifischer, semantischer Kriterien als auch die Berücksichtigung allgemeiner struktureller

bzw. technischer Aspekte [Kaufmann and Wagner, 2001]. Im Hinblick auf die Visualisierung semantischer Graphe/Netzwerke sind vor allem Methoden und Algorithmen zur Darstellung von großen gerichteten Graphen von Interesse (siehe auch 2.4.1).

Im Anschluss werden relevante Drawing- bzw. Layout-Techniken vorgestellt. Eine umfangreichere Behandlung dieser Thematik sowie eine detaillierte Betrachtung der einzelnen Algorithmen hinsichtlich Performance, Komplexität und Aufbau findet sich u.a. in [Battista et al., 1999], [Kaufmann and Wagner, 2001] sowie [Sugiyama, 2002].

Hierarchical Drawing

Eine Möglichkeit zur Darstellung einer Graphstruktur wäre mithilfe hierarchischer Layout-Techniken. Durch die Transformation eines generellen Graphen in einen *Spanning Tree* (siehe auch 2.2.2) können hierarchische Techniken zur Visualisierung angewandt werden. Layoutalgorithmen für Bäume haben in der Regel eine geringere Komplexität (linear zur Anzahl der Knoten) und sind einfacher zu implementieren als spezielle Layoutalgorithmen für Graphstrukturen. Viele traditionelle Layout-Techniken eignen sich zudem nur begingt zur Darstellung großer Graphen mit einer hohen Anzahl an Nodes/Edges. Es gibt eine Reihe von Algorithmen zur Extraktion eines Spanning Trees sowohl aus gerichteten als auch aus ungerichteten Graphen. Eine Möglichkeit besteht darin, ausgehend von einem Startknoten (Wurzel), mittels Breitensuche alle Knoten des Graphen zu traversieren und aus der Menge der durchlaufenen Kanten einen Baum zu formen [Herman et al., 2000] [Linz, 2002] [Andrews, 2002]. Eine weitere Variante einer Transformation eines Graphen in eine hierarchische Struktur findet sich in [Lee et al., 2006]. Dort werden nach der Extraktion eines Spanning Trees zusätzliche Kanten, sogenannte Cross Links, hinzugefügt. Für weitere Details bezüglich Spanning Trees siehe auch [Jungnickel, 2008], [Munzner, 2000] sowie [Herman et al., 2000].

Klassische Layout-Techniken für Bäume platzieren Child Nodes unterhalb bzw. rechts vom Parent Node. Ein bekannter Algorithmus wäre jener von *Reingold und Tilford* [Reingold and Tilford, 1981]. Der Baum wird dabei rekursiv von unten nach oben ('Bottom-To-Top') aufgebaut. Eine solche Darstellung spiegelt die hierarchische Struktur der Daten besonders gut wider. Im Laufe der Jahre wurden einige Erweiterungen und Verbesserungen des ursprünglichen Algorithmus publiziert (siehe z.B. [Buchheim et al., 2002]).

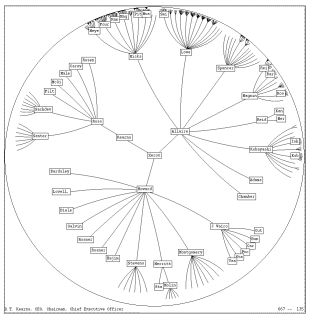
Hyperbolic Layout ist eine Technik zur Darstellung großer Hierarchien. Eine Baumstruktur wird dabei auf eine hyperbolische Ebene projiziert. Objekte erscheinen kleiner bzw. verzerrt, je weiter außerhalb des Projektionszentrums sie sich befinden ('Fisheye'-Effekt). Die hyperbolische Ebene verfügt dadurch über genügend Platz zum Layout der gesamten Hierarchie. Durch den Verzerrungseffekt werden Objekte an den äußersten Rändern mitunter stark verkleinert dargestellt. [Eklund et al., 2002] [Andrews, 2002]

Weitere Tree Layout-Techniken wären beispielsweise *Radial Layout* bzw. *Balloon Layout*. In radialen Darstellungen werden Knoten in Abhängigkeit der Hierarchiestufe entlang konzentrischer Kreise angeordnet. Dabei wird ein Sub-Tree über einen Sektor eines Kreises gelegt. Eine mögliche Erweiterung dieses Konzepts wäre ein gewichtetes Radial Layout wie in [Eklund et al., 2002] beschrieben. In einer sog. Ballondarstellung werden einzelne Knoten bzw. Sub-Bäume kreisförmig um einen zentralen Parent Node angeordnet. Abbildung 2.8 zeigt einige Beispiele der hier erwähnten Tree Layout-Techniken. [Herman et al., 2000] [Andrews, 2002]

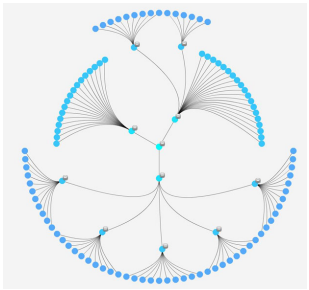
Layered Drawing

Schichtenbasierte (layered) Techniken werden häufig zur Darstellung von gerichteten Graphen verwendet. Die bekannteste Methode stammt von [Sugiyama et al., 1981]. Folgende Ästhetikkriterien haben dabei die höchste Priorität ([Andrews, 2002] [Kern, 2005]):

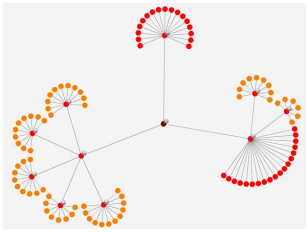
- Edges sollten ausschließlich abwärtsgerichtet sein.



(a) Hyperbolische Darstellung



(b) Radiale Darstellung



(c) Ballondarstellung

Abbildung 2.8: Möglichkeiten zur Darstellung hierarchischer Strukturen. (a) Hyperbolische 2D-Darstellung. (Quelle: [Lamping et al., 1995]; siehe auch ACM Copyright Notice) (b) Radiale Anordnung der Knoten. (Quelle: [Kap, 2009]) (c) Anordnung der Elemente in Form eines Ballons. (Quelle: [Kap, 2009])

- Nodes sollten gleichmäßig verteilt sein.
- Kantenlängen bzw. Kantenüberschneidungen sollten minimiert werden.
- Kanten sollten möglichst gerade gezeichnet werden.

Der Algorithmus zur Optimierung dieser Kriterien läuft dabei in vier Stufen ab:

1. Entfernen aller Zyklen des Graphen.
2. Zuordnung von Nodes zu Schichten (Layer), sodass alle Edges abwärtsgerichtet sind.
3. Reduktion von Kantenüberschneidungen durch Ordnung der Nodes.
4. Horizontale Positionierung der Nodes und anschließendes Zeichnen der Kanten.

Eine detaillierte Beschreibung des Algorithmus sowie weitere Methoden des Layered Drawings finden sich in Kapitel 5 von [Kaufmann and Wagner, 2001] bzw. unter [Sugiyama, 2002].

Physical Analogies Drawing

Im Gegensatz zur Technik des Layered Drawing, welches hauptsächlich auf den strukturellen Eigenschaften eines Graphen basiert, beruhen Algorithmen im Bereich des Physical Analogies Drawing auf physikalischen Modellen. Ein Graph wird dabei als ein physikalisches System mit sich gegenseitig beeinflussenden Objekten (Knoten) betrachtet. Das Layout ergibt sich durch Minimierung der Kräfte bzw. der Energie zwischen den Knoten. In mehreren Iterationsschritten ermittelt ein entsprechender Algorithmus das Gleichgewicht des Systems. Je nach verwendetem Optimierungsalgorithmus kann zwischen kräftebasierten (*Force-Directed)* und energiebasierten (*Energy-Based*) Methoden unterschieden werden (siehe weiters auch [Kaufmann and Wagner, 2001] Kapitel 4).

Gebräuchliche Layout-Techniken wären beispielsweise die als *Kamada Kawai (KK)* bzw. *Fruchterman Reingold (FR)* bezeichneten Algorithmen. Beide Ansätze arbeiten nach dem Prinzip der iterativen Optimierung wobei die Relation zwischen den einzelnen Knoten in Form von Distanzwerten ausgedrückt wird. Die Position eines Knoten ergibt sich durch dessen Reaktion auf die Positionen der anderen Knoten. Beide Algorithmen werden häufig zur Visualisierung von Netzwerken verwendet (siehe [Moody et al., 2005]).

Klassische Force-Directed Methoden weisen eine hohe Laufzeitkomplexität auf, da pro Iterationsschritt alle Knotenpaare besucht werden müssen. Aus diesem Grund eignen sich diese Techniken in erster Linie zur Darstellung kleinerer bzw. lichter Graphen. Im Hinblick auf die Behandlung semantischer Graphen, welche meist aus mehreren tausend Nodes/Edges bestehen, sind kräftebasierte Methoden vor allem in Verbindung mit Subgraphen oder geclusterten Graphen von Bedeutung. [Andrews, 2002] [Herman et al., 2000] [Kern, 2005] [Kaufmann and Wagner, 2001] Darüber hinaus gibt es jedoch eine Reihe von Algorithmen welche sich, basierend auf Force-Directed Techniken, auch zur Visualisierung großer Graphen eignen. Ein Beispiel dafür wäre das in [Harel and Koren, 2002] beschriebene *Multi-Scale* Verfahren.

Interactive Drawing

Die bis jetzt vorgestellten Techniken und Methoden widmen sich hauptsächlich den statischen Aspekten der Graphvisualisierung. Die Darstellung komplexer semantischer Graphen bzw. die Behandlung von Graphen, welche sich im Zeitablauf ändern, erfordert meist zusätzliche Möglichkeiten zur Interaktion bzw. spezielle Layout-Techniken. Die Abschnitte 2.5.2, 2.5.1 bzw. 1.4 beschäftigen sich u.a. mit zentralen Aspekten des dynamic bzw. interactive Drawings. [Kaufmann and Wagner, 2001]

3D Layout

Zusätzlich zu den hier vorgestellten Methoden existiert noch eine Vielzahl weiterer Layout-Techniken bzw. Kombinationen und Variationen einzelner Verfahren. 3D Layout-Techniken bieten weitere Visualisierungsmöglichkeiten. Durch die zusätzliche Dimension soll mehr Platz zur Darstellung komplexer Graphstrukturen geschaffen werden. Einige Methoden und Algorithmen sind dimensionsunabhängig, d.h. sie liefern sowohl im zwei- als auch im dreidimensionalen Bereich brauchbare Ergebnisse (z.B. die meisten Force-Directed Ansätze). Weiters besteht die Möglichkeit, klassische 2D Darstellungen in den dreidimensionalen Raum zu portieren. Beispiele dafür wären 3D-Versionen der zuvor besprochenen radialen bzw. hyperbolischen Darstellungen. Das Hauptaugenmerk dieser Arbeit liegt auf der Darstellung von Graphen im zweidimensionalen Bereich. Bezüglich Techniken und Ansätze zur Visualisierung von Graphen und Netzwerken im 3D Bereich sei an dieser Stelle auf [Herman et al., 2000] verwiesen. Diese Arbeit stellt einen guten Ausgangspunkt für weitere Recherchen auf diesem Gebiet dar.

2.2.6 Speicherungsformen von Graphen

Es existieren verschiedene Möglichkeiten zur Repräsentation bzw. Speicherung einer Graphstruktur. Prinzipiell kann zwischen einer textuellen und einer graphischen (visuellen) Darstellungsform unterschieden werden. Beide Varianten beinhalten die selben Informationen, weisen aber je nach Anwendungsbereich unterschiedliche Vor- und Nachteile auf. Die graphische Repräsentation bietet u.a. einen besseren Überblick über die in einem Graph enthaltenen Informationen und ermöglicht bzw. erleichtert das Erkennen von Beziehungen und Mustern. Diese Variante ist vor allem dann sinnvoll, wenn der Mensch Adressat der Darstellung ist (siehe dazu auch Abschnitt 2.2.3).

Im Gegensatz zur graphischen Repräsentation eignen sich textbasierte Darstellungs- bzw. Speicherungsformen besonders zur maschinellen Verwendung. Ein Programm oder eine spezielle Anwendung kann Text, im Vergleich zu einer visuellen Darstellung, wesentlich besser lesen und weiterverarbeiten. Übliche Repräsentations- bzw. Speicherungsformen wären eine Adjacency Matrix oder eine Adjacency-List (bzw. Edge-List). Beide Varianten haben unterschiedlichen Einfluss auf die Laufzeitperformance diverser Algorithmen und benötigen unterschiedlich viel Speicherplatz.

Eine *Adjacency Matrix* eines Graphen $G = (V, E)$ mit der Anzahl n an Nodes ist eine boolesche Tabelle bestehend aus n Reihen und n Spalten. Jede Zeile und jede Spalte steht für einen Knoten. Der Speicherbedarf beläuft sich daher auf $O(n^2)$. Existiert eine Kante zwischen zwei Knoten, wird in der

entsprechenden Zelle eine 1 (true) andernfalls eine 0 (false) vermerkt. Die Matrix ist symmetrisch für ungerichtete bzw. asymmetrisch für gerichtete Graphen. Eine Adjacency Matrix eignet sich besonders zur Repräsentation/Speicherung dichter Graphen. Da diese Graphen eine hohe Anzahl an Edges aufweisen, wird die Tabelle, im Vergleich zu lichten Graphen, wo viele Zellen unbenutzt bleiben würden, optimal gefüllt. Des weiteren können Verbindungsinformationen schnell und in effizienter Weise abgerufen werden.

Eine *Adjacency List* eines Graphen $G = (V, E)$ mit der Anzahl n an Nodes und der Anzahl e an Edges besteht aus einem Array der Größe n, welches alle Knoten des Graphen enthält. Zusätzlich werden die benachbarten Knoten in Form einer linearen Liste gespeichert. Für gerichtete Graphen ergibt sich daher ein Speicherbedarf von $O(n + e)$. Die Repräsentation in Form einer Adjacency List eignet sich in diesem Fall besonders für lichte Graphen.

Eine *Edge List* stellt eine weitere Variante zur Speicherung lichter Graphen dar. Dabei wird für jede Kante eine Liste mit den durch diese Kante verbundenen Knoten geführt. Da eine Kante in der Regel nur zwei Knoten miteinander verbindet, hat jede Liste eine konstante Größe.

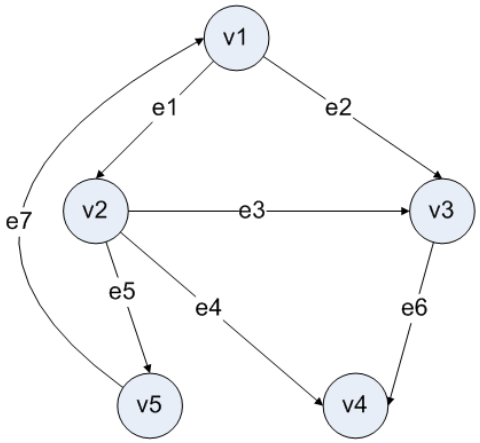
Je nach Anwendungsgebiet finden sich noch weitere Speicherungsformen bzw. Variationen und Kombinationen der hier besprochenen Techniken. Abbildung 2.9 zeigt beispielhaft visuelle bzw. textbasierte Repräsentationsmöglichkeiten anhand eines einfachen gerichteten Graphen. [Linz, 2002] [Haenelt, 2004] [Prinz, 2006]

2.2.7 Größe von Graphen - Skalierbarkeit

Die Größe bzw. Komplexität ist eines der Schlüsselthemen bei der Visualisierung von Graphen. Große Graphen bzw. Netzwerke können u.a. durch die automatisierte Sammlung und Verknüpfung von Daten aus digitalen Quellen entstehen. Komplexe Strukturen wie diese stellen eine Herausforderung für die verschiedenen Teilbereiche der Graph Visualisation dar. Layoutalgorithmen, welche für kleine Graphen gute Ergebnisse liefern, können bei größeren Graphen sehr schnell an ihre Grenzen stoßen. Viele Visualisierungstechniken wurden in erster Linie zur Handhabung kleinerer Graphstrukturen konzipiert. Dies resultiert u.a. in einer schlechten Performance des Algorithmus (Laufzeit, Speicherbedarf, etc.) bzw. in einer ungeeigneten Darstellung des Graphen (Überlagerung, Verdeckung, etc.). Selbst wenn es gelingt, ein Layout für eine große Graphstruktur zu berechnen, ergeben sich etwa im Bereich der Usability oder Readability weitere Probleme. Einerseits können aufgrund eines zu dichten Layouts Nodes und Edges nicht mehr voneinander unterschieden werden. Eine Auswertung oder Analyse der dargestellten Informationen wird dadurch erschwert bzw. gänzlich verhindert. Auch die Lesbarkeit von Labels sowie deren eindeutige Zuordnung zu Nodes/Edges ist in diesem Fall nicht mehr gegeben. Andererseits werden Möglichkeiten zur Interaktion (z.B. Navigation,...) stark beeinträchtigt. Eine komplexe visuelle Darstellung überfordert zudem die menschliche Wahrnehmungsfähigkeit. Ein weiteres Problem stellt der mitunter enorme Ressourcenbedarf (Prozessorleistung, Speicher, Grafikhardware, etc.), welcher mit der Darstellung großer Informationsmengen verbunden sein kann, dar. Beispielsweise kann die Anzahl der Graphelemente die Anzahl der maximal darstellbaren Pixel eines Monitors überschreiten. Auch der zur Verfügung stehende Arbeitsspeicher stellt bei komplexen Visualisierungen einen möglichen Engpass dar.

Größenbegriffe

Nach welchen Kriterien die Größe eines Graphen bestimmt wird, hängt stark von der jeweiligen Betrachtungsweise ab. In der Literatur finden sich dazu unterschiedliche Herangehensweisen. Häufig dient die Anzahl der Knoten und/oder Kanten als Maßstab für Größe. Dies ist jedoch nur ein grobe Abschätzung der tatsächlichen Komplexität eines Graphen. Die Relation zwischen Node- und Edge-Anzahl bzw. das Vorhandensein weiterer graphbezogener Elemente (Labels, Metadaten,...) sollten ebenfalls berücksichtigt werden.



G = (V,E)
V = {v1, v 2, v3, v4, v5}
E = {e1, e2, e3, e4, e5, e6, e7}

(a) visuelle (grafische) Repräsentation

Adjacency Matrix:

	v1	v2	v3	v4	v5
v1	0	1	1	0	0
v2	0	0	1	1	1
v3	0	0	0	1	0
v4	0	0	0	0	0
v5	1	0	0	0	0

Adjacency List:

Nodes	Liste der benachbarten Nodes
v1	[v2, v3]
v2	[v3, v4, v5]
v3	[v4]
v4	[ ]
v5	[v1]

G = (V, E)
V = {v1, v2, v3, v4, v5}
E = {(v1, v2), (v1, v3), (v2, v3), (v2, v4), (v2, v5), (v3, v4), (v5, v1)}

(b) Adjacency Matrix, Adjacency List

Abbildung 2.9: Möglichkeiten zur Repräsentation bzw. Speicherung am Beispiel eines gerichteten Graphen. (a) Darstellung mithilfe geometrischer Elemente (Kreise, Pfeile bzw. Beschriftungen). (b) Repräsentation mittels Adjacency Matrix bzw. Repräsentation mittels Adjacency List.

Auch die Definition der Begriffe klein, mittel, groß bzw. riesig in Verbindung mit Graphen bzw. Netzwerken variiert je nach Publikation. Dies ist einerseits auf die Jahreszahl der Veröffentlichung zurückzuführen. Aufgrund von Weiterentwicklungen, sowohl im Bereich der Hardware als auch auf algorithmischer Ebene, veränderte sich im Laufe der Zeit die Definition der Größenbegriffe. Anderseits sind diese Begriffe auch stark kontextabhängig. Eine exakte Abgrenzung ist demnach nur schwer vorzunehmen. Als generelle Richtlinie bzw. zur besseren Orientierung gelten im Rahmen dieser Arbeit Graphen mit mehr als 10^8 Elementen (Nodes, Edges) als riesig. Kleine Graphen weisen in diesem Fall weniger als hundert Elemente auf.

Vorgehensweisen

Riesige Graphen oder Netzwerke können unter Umständen aus mehreren (hundert) Millionen Elementen bestehen. In solchen Fällen ist es nicht mehr sinnvoll bzw. möglich, die gesamte Struktur auf einmal darzustellen. Zur Lösung dieses Problems bedarf es spezieller Vorgehensweisen und Techniken. Eine Möglichkeit zur Visualisierung besteht darin, die Größe bzw. Komplexität des Graphen in einem Vorverarbeitungsschritt zu reduzieren. Dies kann zum Beispiel mithilfe von Clustering bzw. Aggregierung (siehe Abschnitt 2.3) erreicht werden. Im Anschluss an die Komplexitätsreduktion erfolgt die Darstellung des vereinfachten Graphen unter Verwendung klassischer Layout-Algorithmen.

Eine Alternative dazu wäre, immer nur einen kleinen Ausschnitt des Graphen bzw. des Netzwerks darzustellen. Der User kann durch das Netzwerk navigieren bzw. einzelne Bereiche auswählen. Ein detailliertes Layout wird nur für die fokussierten Teilgraphen berechnet, die Darstellung der umliegenden Bereiche erfolgt überblicksmäßig (Fokus+Context). [Herman et al., 2000] [Andrews, 2002] [Prinz, 2006] [Gansner et al., 2005] [Liu et al., 2005]

Es existieren zahlreiche Publikationen, welche die Visualisierung großer Graphen von verschiedenen Standpunkten aus beleuchten. Die darin präsentierten Konzepte basieren auf unterschiedlichen Techniken und Ansätzen (z.B. Clustering, Level-Of-Detail, Multi-Scaling, Semantic Zooming, Interactive Navigation, etc.). Im Kontext dieser Arbeit sind speziell Techniken zur Aggregierung (Clustering) und anschließender Visualisierung von mittleren bzw. größeren Graphen von Interesse.

In [Abello et al., 2006] wird ein, in diesem Zusammenhang interessantes System zur Visualisierung großer Graphen präsentiert. Dieser Ansatz beruht auf der Kombination von Clustering und Interactive Navigation Techniken. In mehreren Vorverarbeitungsschritten wird dabei u.a. mittels Clustering-Verfahren eine hierarchische Struktur eines beliebigen Input-Graphen erzeugt. Die anschließende Visualisierung des clustered Graph erfolgt als Node-Link Darstellung. Durch Öffnen (Expand) bzw. Schließen (Collapse) von Cluster-Nodes kann durch die Graphstruktur navigiert werden. Das Prinzip der Clustered Graph Navigation wird auch von dem in [Eades and Huang, 2000] vorgestellten Visualisierungssystem aufgegriffen. [Liu et al., 2005] verwendet hingegen eine Level-Of-Detail (LOD) Strategie zur Visualisierung und Navigation großer Graphen. In einer ersten Phase wird der LOD Tree durch Clustering des Input-Graphen aufgebaut. Jeder Knoten dieses Baumes repräsentiert dabei einen Sub-Graphen. Im Anschluss erfolgt das Layout der einzelnen Sub-Graphen mithilfe von Force-Directed Techniken (für weitere Details siehe auch Abschnitt 2.3).

2.3 Dynamische Aggregierung - Clustering

Die zeitgleiche Darstellung vieler oder aller Elemente eines größeren Graphen ist, wie bereits erwähnt, in den meisten Fällen mit erheblichen Problemen verbunden. Nodes, Edges und Labels können sich überlappen und sind dadurch nicht mehr voneinander zu unterscheiden. Elementüberlagerungen oder Kantenüberschneidungen führen zu einer unübersichtlichen, für den menschlichen Betrachter unbrauchbaren Visualisierung. Wird eine gewisse Anzahl an Graphelementen überschritten, so sind die Grenzen des zur Darstellung verfügbaren Raumes erreicht. Folglich existiert nicht mehr ausreichend Platz für die

gleichzeitige Darstellung aller Elemente. Neben Problemen algorithmischer Natur sind ebenfalls Limits im Bereich der Hardware (CPU, GPU, Speicherkapazität, Bildschirmauflösung, etc.) zu beachten. In diesem Fall ist somit eine detaillierte Visualisierung einzelner Nodes, Edges bzw. Labels nicht zielführend. Stattdessen wäre eine generelle Übersichtsdarstellung des Graphen wünschenswert.

Aggregierung

Das Konzept der Aggregierung stellt eine effektive Möglichkeit zur Behandlung größerer Graphen (bzw. allgemein großer Datenmengen) dar. [Chuah, 1998] versteht darunter die Vereinfachung großer Datenmengen durch Zusammenfassung von Datenelementen zu Gruppen. Ein einzelnes Symbol bzw. Icon dient anschließend zur Repräsentation einer Gruppe. Die wichtigsten Merkmale einer dynamischen Aggregation im Überblick ([Chuah, 1998]):

- *Automatische Aggregation* (bzw. De-Aggregation) von Elementen: Der Detailgrad der Aggregierung wird je nach Bedarf automatisch berechnet. Eine flexiblere Variante wäre die Aggregation von Elementen auf Basis einer User-Aktion (beispielsweise die Formulierung einer Abfrage, die Durchführung einer Suche, etc.).
- *Interaktive Kontrolle* der Aggregationsstufe: Mittels Expand- bzw. Collapse-Operationen ist es dem User möglich, die unterschiedlichen Levels der Aggregation zu durchlaufen um so beispielsweise Muster oder Regelmäßigkeiten besser erkennen zu können. Fortlaufende, sanfte Wechsel bzw. Übergänge zwischen den Aggregationsstufen helfen dabei Überblick und Orientierung zu bewahren.
- *Aggregation benachbarter Elemente*: Nahe beieinanderliegende Elemente werden, falls zu wenig Platz vorhanden ist, zusammengefasst und auf ein grafisches Element gemappt. Die Form des Elements soll dabei Rückschlüsse auf den aggregierten Inhalt ermöglichen.
- *Kontexterhaltung* zwischen verschiedenen Aggregationsstufen: Da eine Änderung des Aggregationslevels auch gleichzeitig eine Änderung der visuellen Repräsentation zur Folge hat, kann mitunter der Überblick bzw. die Orientierung verlorengehen. Um dies zu verhindern, können Daten mit einer, von der jeweiligen Aggregationsstufe unabhängigen, grafischen Eigenschaft versehen werden.
- *Feedback hinsichtlich aktueller Aggregationsstufe*: Um Fehlinterpretation bzw. Vergleiche zwischen unterschiedlichen Aggregationsstufen zu vermeiden, sollte der aktuelle Grad der Aggregation klar erkennbar sein. Die kann beispielsweise durch ein Größenkodierung der Icons erreicht werden.
- *Bewahrung von Information*: Jedes aggregierte Element hat Einfluss auf die Gestalt bzw. das Aussehen des Icons. Auf diese Weise wird jedes einzelne Element in der Visualisierung berücksichtigt und ein Verlust von Informationen im Zuge der Aggregation verhindert.

Als Aggregierung (bzw. Aggregation) wird im Rahmen dieser Arbeit prinzipiell das Zusammenfassen bestimmter Nodes und Edges sowie die anschließende Visualisierung bezeichnet. Ziel ist die visuelle Komplexitätsreduktion einer Darstellung. Reduktion bedeutet in diesem Zusammenhang, dass nicht alle Teile des Graphen zur selben Zeit dargestellt werden. Dadurch sollen einerseits Layoutalgorithmen brauchbare Ergebnisse in akzeptabler Zeit liefern, andererseits aber auch die Grenzen der zur Verfügung stehenden Hardware nicht überschritten werden. Voraussetzung ist jedoch, dass die grundlegende Struktur des Graphen weitestgehend unverändert bleibt. Im Zuge der Komplexitätsreduktion dürfen Details nicht verloren gehen, sondern müssen bei Bedarf wieder abgerufen bzw. eingeblendet werden können. Es erfolgt somit eine Vereinfachung des ursprünglichen Graphen mit der Möglichkeit zur interaktiven Anpassung bzw. Modifikation der Abstrahierung. [North and Woodhull, 2001].

Grundsätzlich erfolgt die Aggregierung in zwei Schritten: Zuerst werden Gruppen (Cluster) innerhalb der Graphstruktur identifiziert. Danach erfolgt die weitere Verarbeitung der gefundenen Gruppen. Dazu gehört u.a. das Layout des geclusterten Graphen. Dabei stellt sich auch die Frage nach einer geeigneten Repräsentation der gefundenen Cluster. Wie eingangs bereits erwähnt, bietet sich Darstellung einer Gruppe bzw. eines Clusters in Form eines eigenen Icons (Glyph) an. In weiterer Folge müssen zudem Funktionalitäten zur Interaktion bereitgestellt werden. Die Kombination aus Clustering- und Interaktionstechniken ergibt somit den dynamischen Charakter einer Aggregierung.

Eine Alternative zu diesem zweistufigen Verfahren stellt Aggregierung bzw. Clustering basierend auf speziellen force-directed Methoden dar (siehe dazu auch [Kern, 2005] sowie [Kaufmann and Wagner, 2001] Kapitel 8). In mehreren Iterationsschritten werden dabei Anziehungs- bzw. Abstoßungskräfte zwischen den einzelnen Nodes berechnet und diese in Folge dementsprechend positioniert. Aggregierung bzw. Clustering erfolgt somit im Zuge des Graph Layouts. Für große Graphen mit vielen Nodes eignet sich diese Methode aber nur bedingt, da sie u.a. eine hohe Laufzeitkomplexität aufweist. [Herman et al., 2000] [Wills, 1999] [Huang and Eades, 1998]

Clustering

Clustering ist ein im Kontext dieser Thematik essentielles Konzept. Es stellt eine Möglichkeit zur Reduktion der Komplexität einer Graphvisualisierung dar. Allgemein kann Clustering als Gruppierung bzw. Kategorisierung einer Menge von ungeordneten Objekten definiert werden. Die dadurch entstandenen Gruppen werden als Cluster bezeichnet. Objekte innerhalb einer Gruppe weisen Ähnlichkeiten hinsichtlich bestimmter Kriterien bzw. Eigenschaften auf. Ein Objekt besteht dabei aus einem Vektor. Die Komponenten des Vektors repräsentieren die Objekteigenschaften. [Sabol, 2001].

Graph-Clustering ist eine spezielle Clustering-Variante. Graph-Clustering befasst sich u.a. mit dem Erkennen von dichten Sub-Graphen innerhalb der gesamten Graphenstruktur. Ein Sub-Graph weist intern eine hohe Verbindungsdichte auf. Zwischen den einzelnen Sub-Graphen bestehen jedoch nur wenige Links. [Brandes et al., 2003] [Herman et al., 2000]. Graph-Clustering bzw. die Analyse von Graphen zur Auffindung von Gruppen ist eng mit dem Begriff *Graph-Partitioning* verbunden. Einer der Schwerpunkte von Graph-Partitioning ist die optimale Unterteilung (Partitionierung) eines Graphen unter Berücksichtigung bestimmter Vorgaben. Clustering dient dabei oft als Vorverarbeitungsschritt. Eine weitere Disziplin, welche sich teilweise mit der Analyse bzw. der Gruppierung von Graphen auseinandersetzt, wäre *Pattern-Recognition*. [van Dongen, 2000] [Xu and Wunsch, 2005] [Berkhin, 2002]

Im Zuge des Clustering von Graphen werden Nodes bzw. Edges aufgrund unterschiedlicher Kriterien einzelnen Klassen oder Kategorien zugeordnet und auf Basis dessen zu Cluster zusammengefasst. Kriterien zur Unterteilung wären beispielsweise:

- Bedeutung (Semantik)
- Node-Degree
- Edge-Weight
- Distanz zwischen Nodes bzw. Edge-Length
- Position von Nodes im Raum (Euklidische Distanz)
- Verbindungsdichte
- Ähnlichkeiten hinsichtlich anderer Node- bzw Edge-Eigenschaften (z.B. Name, Größe, Farbe, etc.)

Die dadurch definierten Cluster müssen nicht zwangsläufig genau voneinander abgegrenzt sein. Häufig können sich einzelne Gruppen überschneiden oder überlappen, wodurch die weitere Behandlung (Darstellung, Navigation, etc.) dieser Cluster erschwert wird. Grundsätzlich soll mithilfe von Clustering die

Dichte bzw. Komplexität des dargestellten Graphen verringert und dessen allgemeine Struktur erkennbar werden. Gleichzeitig verbessert sich auch die Performance hinsichtlich Rendering, Layout, Interaktion, etc., da nicht mehr alle Elemente des Graphen einzeln visualisiert werden. Wie bereits erwähnt, ist das Ziel die Reduktion der visuellen Komplexität um so einen kleineren bzw. übersichtlicheren Graphen zu generieren.

Dabei ist vor allem auf die Anzahl der Cluster sowie auf die Anzahl der Nodes/Edges innerhalb dieser Cluster zu achten. Die richtige Balance zwischen der Reduktion von Komplexität einerseits, und der Bewahrung von wichtigen Informationen bzw. Details andererseits, stellt ein wichtiges Qualitätsmerkmal eines Clustering-Verfahrens dar. Neben Clustering gibt es noch eine Reihe weiterer Abstraktions- und Reduktionstechniken, welche im Rahmen dieser Arbeit jedoch nicht näher behandelt werden. [Herman et al., 2000] [Prinz, 2006]

In den nächsten Abschnitten erfolgt eine Betrachtung der für die Fragestellung dieser Arbeit relevanten Aspekte, Techniken, Algorithmen und Konzepte zur Reduktion der visuellen Komplexität des dargestellten Graphen. Zudem werden Möglichkeiten zur Repräsentation von geclusterten Graphen näher erläutert.

2.3.1 Merkmale von Clustering-Techniken

Es gibt unterschiedliche Möglichkeiten zur Einteilung und Differenzierung der einzelnen Clustering-Methoden. Meist dient die Struktur der erzeugten Cluster als Unterscheidungskriterium. Auf Basis dessen kann grundsätzlich zwischen hierarchischen und partitionierenden (nicht-hierarchischen) Techniken unterschieden werden.

Hierarchical Clustering stellt dabei die komplexere der beiden Varianten dar. Prinzipiell werden bei hierarchischen Verfahren Objekte bzw. bereits gebildete Gruppen paarweise zusammengefasst. Das Ergebnis ist eine baumartige Clusterstruktur (Hierarchie) mit der Eigenschaft, dass ein Cluster sowohl aus Objekten als auch aus weiteren Clustern aufgebaut sein kann. Partitionierende Verfahren (u.a. auch als *flat Clustering* bezeichnet) liefern hingegen eine flache Clusterstruktur. Das bedeutet, dass jeder Cluster nur Objekte enthält. Dabei wird die Menge der Objekte in einzelne Teilmengen aufgespaltet. Diese Verfahren sind konzeptionell einfacher und effizienter, weisen aber im Vergleich zu hierarchical Clustering auch einige Besonderheiten bzw. Nachteile auf. So muss beispielsweise die Anzahl der Cluster bereits im Voraus spezifiziert werden. Weiters wird jedes Objekt genau einem Cluster zugeordnet. Kombinationen aus flat und hierarchical Clustering sind ebenfalls möglich. [Sabol, 2001] [Lux, 2004] [Manning et al., 2008a] [Manning et al., 2008b] [Herman et al., 2000] [Neidhart, 2005]

Es finden sich noch eine Reihe weiterer Merkmale bzw. Unterscheidungskriterien. So kann beispielsweise zwischen *deterministischen* Verfahren, welche bei gleicher Ausgangssituation immer die gleichen Ergebnisse liefern, und *stochastischen* Verfahren unterschieden werden. Bei *monothetischen* Methoden erfolgt die Zuordnung eines Objektes zu einem Cluster aufgrund einer einzigen Eigenschaft (im Gegensatz zu *polythetischen* Methoden). *Exklusive* Techniken ordnen jedes Objekt genau einem Cluster zu, wohingegen *überlappende* Verfahren auch Mehrfachzuordnungen erlauben. Die *Skalierbarkeit* eines Algorithmus gibt Aufschluss über dessen Anwendbarkeit auf große Datenmengen. [Sabol, 2001] [Neidhart, 2005]

Speziell im Bereich des Clustering von Graphen kann zudem zwischen structure-based und content-based Techniken unterschieden werden. *Content-based Clustering* basiert auf semantischen Daten eines Graphen. Diese Techniken sind meist für eine spezielle Anwendung bzw. Domäne entwickelt. Voraussetzungen für content-based Verfahren sind daher anwendungsspezifisches Wissen sowie anwendungsspezifische Daten/Informationen.

Eine generellere Methode stellt hingegen *structure-based Clustering* dar (u.a. auch als das Auffinden natürlicher Cluster bezeichnet). Dabei dienen strukturelle Informationen des Graphen als Grundlage für

die Cluster-Bildung. Ein Vorteil dabei ist, dass die Struktur des ursprünglichen Graphen besser erhalten bleibt. Sinnvollerweise können bzw. sollten structure-based und content-based Methoden miteinander kombiniert werden. [Herman et al., 2000]

2.3.2 Clustering-Metrics

Das Auffinden von Clustern kann auf Basis nummerischer Metriken erfolgen. Nodes bzw. Edges werden dabei (meist in einem Vorverarbeitungsschritt) Werte zugewiesen. Mithilfe dieser Werte können die abstrakte Eigenschaften von Nodes/Edges quantifizierbar gemacht werden. In diesem Zusammenhang findet sich häufig der Begriff *Metric bzw. Node-Metric* (wobei es in der Literatur durchaus noch andere Definitionen dieses Terms gibt). Ausgehend von den zugewiesenen Werten kann nun ein Vergleich bzw. ein Ranking der einzelnen Nodes/Edges und in weiterer Folge eine Zuordnung zu bestimmten Gruppen oder Klassen durchgeführt werden.

Metriken können wiederum structure-based oder content-based sein. Ein Beispiel für content-based Metrics wäre die Berechnung von anwendungsspezifischen Gewichtungen von Nodes oder Edges. Die Anzahl der mit einem Node verbundenen Edges (Node-Degree) stellt hingegen ein Beispiel für structure-based Metrics dar. So könnten nur jene Nodes, welche einen gewissen Degree-Schwellwert überschreiten, dargestellt werden. In diesem Fall würden Bereiche des Graphen mit niedriger Konnektivität ausgeblendet werden. Dies stellt eine relativ simple aber effektive Methode zur Reduktion der visuellen Komplexität dar.

Die Kombination mehrerer Metrics kann beispielsweise durch Berechnung eines gewichteten Durchschnitts aller Were erfolgen. Durch die Gewichtung einzelner Metrics besteht die Möglichkeit, deren Wichtigkeit zu verändern und in Folge dessen das Clustering-Ergebnis zu beeinflussen. Ein Beispiel für die Kombination mehrere Metrics wäre der sog. *Degree Of Interest* (Details dazu siehe auch [Furnas, 1986]). Die beiden Parameter Distanz und Level of Detail bilden dabei die Berechnungsgrundlage. [Herman et al., 2000]

Für die Behandlung von Graphen ebenfalls relevant ist die sog. *Strahler Metric*, welche speziell bei baumartigen Strukturen zur Anwendung kommt. Mittels Strahler Metrics wird versucht, die Komplexität bzw. Gestalt eines Baumes in Form von quantitativen Informationen auszudrücken. Dabei weist ein höherer Wert eines Baum-Knoten auf einen komplexeren Sub-Tree hin. In [Herman et al., 1998] werden die ursprünglich für binäre Bäume entwickelten Strahler Metrics um zwei Aspekte erweitert bzw. generalisiert: Einerseits wird in dieser modifizierten Version die Anzahl der Children eines Nodes berücksichtigt. Dies ist für Aussagen bezüglich der Komplexität eines Baumes von Bedeutung. Andererseits können Nodes noch mit einer anwendungsspezifischen Gewichtung versehen werden. Dadurch fließen zusätzlich Aussagen hinsichtlich der Wichtigkeit bzw. Bedeutung einzelner Nodes/Edges in die Berechnung der Strahler Zahlen mit ein. [Herman et al., 1998]

2.3.3 Ausgewählte Clustering-Techniken

Dieser Abschnitt bietet eine Übersicht über einige, vor allem für die praktische Implementierung relevante Clustering-Algorithmen und Techniken. Eine ausführliche Behandlung verschiedener Clustering-Algorithmen findet sich u.a. in [Sabol, 2001] sowie in [Neidhart, 2005]. Kapitel 8 von [Kaufmann and Wagner, 2001] widmet sich speziellen Methoden des Graph-Clustering.

K-Means Clustering

K-Means zählt zu den partitionierenden Clustering-Verfahren. Ein entscheidendes Merkmal dabei ist, dass die Anzahl k der zu bildenden Cluster bereits zu Beginn festgelegt wird. Der Algorithmus besteht prinzipiell aus folgenden vier Schritten:

1. Zunächst erfolgt die Definition von k Clusterschwerpunkten (*Centroids*). Anfangs sollten die Centroids soweit wie möglich voneinander entfernt platziert werden

2. Anschließend wird die Distanz der Datenelemente (Objekte) zu den einzelnen Schwerpunkten berechnet. Jedes Objekt wird infolge der nächstgelegenen Gruppe zugeordnet.

3. Nachdem die Zuweisung aller Objekte abgeschlossen ist, liegt eine Gruppierung vor. Davon ausgehend werden die Clusterschwerpunkte neu berechnet.

4. Die Neupositionierung der Centroids hat zur Folge, dass Distanzberechnung und Zuweisung erneut durchgeführt werden müssen. Schritt 2 und 3 werden nun so oft wiederholt, bis sich die Positionen der Centroids nicht mehr verändern.

Ziel des Algorithmus ist die Minimierung der *Objective Function* im euklidischen Raum. Diese Funktion lautet:

$$f = \sum_{j=1}^{k} \sum_{i=1}^{n} \|\vec{x}_i - \vec{c}_j\|^2$$

$\|\vec{x}_i - \vec{c}_j\|$ steht dabei für die Distanz des Datenelements x_i zum Clusterschwerpunkt c_j. n gibt die Anzahl der Datenelemente, k die Anzahl der Cluster an. Abbildung 2.10 zeigt die Anwendung des K-Means Algorithmus auf einen Datensatz mit $n = 200$ und $k = 4$ (Screenshots eines Java Applets aus [Matteucci, 2009b]).

Die initiale Definition bzw. Positionierung der Centroids hat entscheidenden Einfluss auf das Endergebnis. Aus diesem Grund kann der Algorithmus mit jeweils unterschiedlichen Startparametern mehrmals wiederholt werden.

Weiters hat auch die Anzahl k der Cluster Auswirkung auf das Resultat. Mitunter ist es aber schwer abzuschätzen, wie viele Cluster tatsächlich existieren.

Grundsätzlich weist der K-Means Algorithmus eine Laufzeitkomplexität von $O(kn)$ auf (wobei mehrere Iterationen diesen Wert erhöhen). Die annähernd lineare Laufzeit stellt einen Vorteil gegenüber hierarchischen Clustering-Methoden dar. [Neidhart, 2005] [Sabol, 2001] [Matteucci, 2009c]

Hierarchical Agglomerative Clustering (HAC)

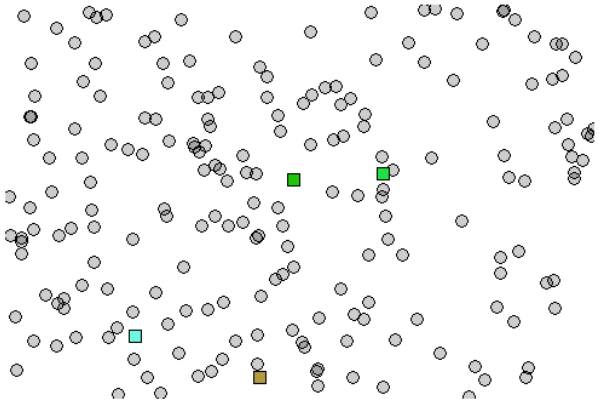
HAC ist eine hierarchische Clustering-Technik, welche nach dem Bottom-Up Prinzip funktioniert. Jedes Objekt (Datenelement) repräsentiert zu Beginn einen Cluster. Während eines Durchlaufs werden ähnliche Cluster sukzessive miteinander kombiniert, wodurch letztendlich ein einziger Cluster entsteht. Dieser Ansatz wird als *agglomerative Clustering* bezeichnet. Im Gegensatz dazu arbeiten Methoden des *divisive Clustering* nach dem Top-Down Prinzip. Dabei befinden sich zu Beginn alle Objekte in einem Cluster. Dieser wird anschließend in kleinere Teile aufgegliedert. Das HAC-Verfahren läuft grundsätzlich in folgenden Schritten ab:

1. Zu Beginn wird jedes der n Objekte als Cluster definiert.

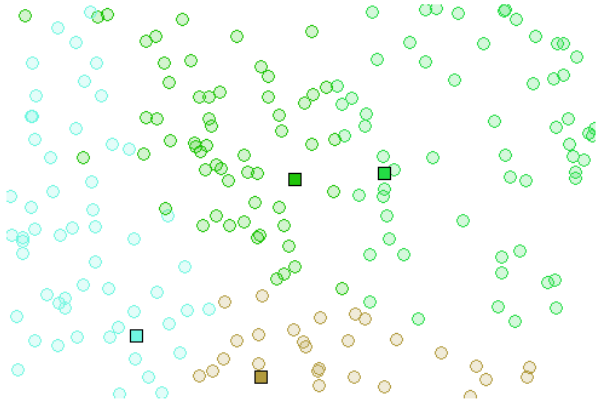
2. Im Anschluss daran werden die zwei Cluster mit der größten Ähnlichkeit (bzw. der geringsten Distanz) zu einem Cluster verschmolzen.

3. Danach erfolgt die Berechnung der Ähnlichkeit (Distanz) zwischen dem neu entstandenen Cluster und den übrigen Clustern.

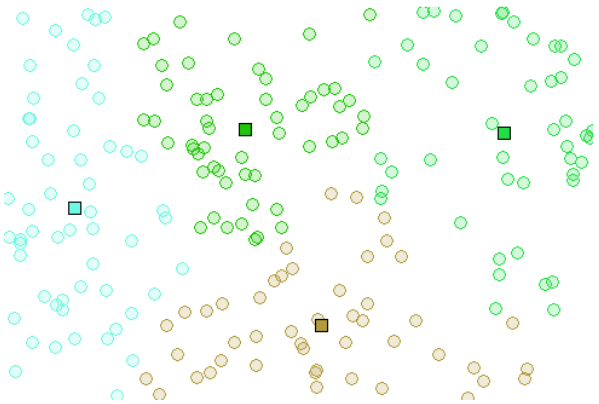
4. Schritt 2 und 3 werden solange wiederholt, bis alle Objekte in ein einem einzigen Cluster der Größe n vereint sind.



(a) Initiale Beispieldaten



(b) Durchlauf 1



(c) Durchlauf 7

Abbildung 2.10: Demonstration des K-Means Algorithmus anhand eines Beispieldatensatzes. (Screenshots eines Java Applets aus: [Matteucci, 2009b]). (a) Zufällige Platzierung der $n = 200$ Datenelemente sowie der 4 Centroids (dargestellt als Quadrate). (b) Zuordnung der Elemente (Farbe des jeweiligen Centroids) nach einem Schritt (Basis: euklidische Distanz). Ein Datenelement wird dabei immer nur genau einem Cluster zugeordnet. (c) Entwicklung nach 7 Schritten. Der Algorithmus terminiert, da sich die Cluster-Positionen nicht mehr ändern.

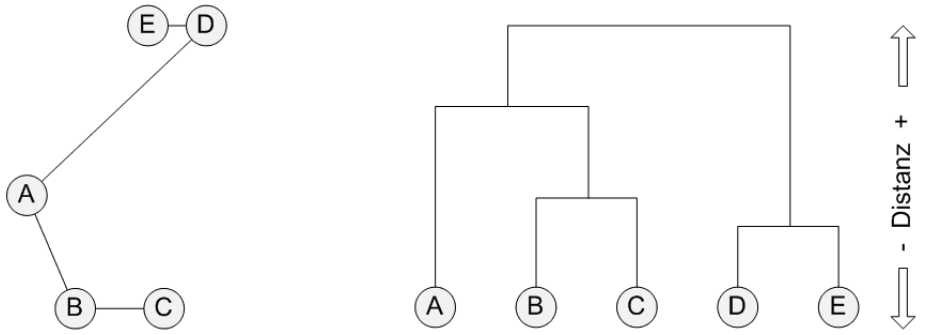


Abbildung 2.11: Hierarchical Agglomerative Clustering: Links: Beispielgraph mit 5 Knoten. Rechts: Darstellung der hierarchischen Clusterstruktur als Dendrogram. Die Verschmelzung erfolgte auf Grundlage der Distanz zwischen den Knoten. (basierend auf: [Sabol, 2001] bzw. [Matteucci, 2009a])

Das Ergebnis ist eine hierarchische Struktur, welche in Form eines Dendrograms dargestellt werden kann (siehe Abbildung 2.11). Je nach Berechnungsweise der Distanz bzw. der Ähnlichkeit zwischen den Clustern, kann zwischen folgenden Methoden unterschieden werden:

- *Single-Link*: Die Distanz zwischen zwei Clustern ergibt sich aus der minimalen Distanz zwischen den Cluster-Mitgliedern (bzw. ergibt sich die Ähnlichkeit zwischen 2 Clustern dementsprechend aus der maximalen Ähnlichkeit zwischen ihren Mitgliedern). Diese Methode weist im Normalfall eine Laufzeitkomplexität von $O(n^2)$ auf.
- *Complete-Link*: Die Distanz zwischen zwei Clustern ergibt sich in diesem Fall aus der maximalen Distanz (bzw. der minimalen Ähnlichkeit) zwischen den Cluster-Mitgliedern. Dies Methode produziert eine große Zahl an kompakten, kleinen Clustern bei einer Laufzeit von etwa $O(n^3)$.
- *Average-Link*: Bei dieser Methode ergibt sich die Distanz zwischen zwei Clustern aus der durchschnittlichen Distanz zwischen den Cluster-Mitgliedern. Dies stellt einen Kompromiss zwischen der Single-Link und Complete-Link Variante dar.

Die hohe Laufzeitkomplexität von zumindest $O(n^2)$ ist ein entscheidender Nachteil gegenüber partitionierenden Clustering-Algorithmen wie zum Beispiel dem K-Means Verfahren. Die hierarchische Clusterstruktur sowie die Tatsache, dass die Anzahl der Cluster nicht zuvor festgelegt werden muss, können unter Umständen Gründe für die Verwendung eines HAC-Algorithmus sein. [Neidhart, 2005] [Matteucci, 2009a] [Lux, 2004] [Sabol, 2001]

Molecular Complex Detection (MCODE)

Dieser in [Bader and Hogue, 2003] beschriebene Algorithmus stammt ursprünglich aus dem Bereich der Bioinformatik. Innerhalb großer Molekularnetzwerke werden mittels Graph-Clustering Verfahren dicht vernetzte Regionen (*Densely Connected Regions*) unter Berücksichtigung gegebener Parameter gefunden bzw. extrahiert. Ein solches Netzwerk kann als Graph betrachtet werden: Nodes (Vertices) repräsentieren dabei die Moleküle, Edges dienen als Darstellung der Interaktion zwischen den Molekülen. Je nach Bedarf kann das Netzwerk als gerichteter bzw. ungerichteter Graph modelliert werden. Mit Hilfe des MCODE Algorithmus können einzelne Cluster separat vom gesamten Netzwerk betrachtet, Feinabstimmungen vorgenommen sowie Zusammenhänge zwischen Clustern untersucht werden. Durch graphtheoretische Abstraktion erhält der Algorithmus einen sehr generellen Charakter und ist somit auf beinahe alle Graphen anwendbar. Dadurch wird dieses Verfahren auch für das Clustering von semantischen Graphen interessant.

Das MCODE Verfahren umfasst drei Stufen:

1. Vertex Weighting
2. Complex Prediction
3. Postprocessing

Vertex Weighting bildet die Basis des Algorithmus. In dieser Phase erfolgt die Gewichtung der einzelnen Vertices (Nodes) in Abhängigkeit ihrer jeweiligen lokalen Network Density. Die Dichte (Density) basiert prinzipiell auf der Verbindungsdichte (Connectivity Level) eines Graphen bzw. dem Abstand der Nodes zueinander. Im Falle des MCODE Vertex Weighting errechnet sich die Dichte aus der Division der tatsächlichen Edge-Anzahl durch die maximale theoretische Edge-Anzahl des Graphen.

Der erste Schritt der Gewichtungsphase ist das Auffinden von Nachbarn der Tiefe 1 für jeden Node (also alle Nodes, welche direkt mit dem betreffenden Node verbunden sind). Aus dieser Menge wird im Anschluss der dichteste zentrale Sub-Graph sowie der sog. highest k-Core Level ermittelt (ein k-Core ist ein Graph mit minimalem Degree k).

Im nächsten Schritt erfolgt die Berechnung des Core-Clustering Koeffizienten aus der Density des zentralen Sub-Graphen. Die finale Gewichtung des Vertex ergibt sich schließlich aus der Multiplikation des Core-Clustering Koeffizienten mit dem highest k-Core Level.

Complex Prediction, als zweite Phase des MCODE Algorithmus, erhält den im vorangegangen Schritt gewichteten Graphen als Input. Ausgehend von einem Seed Vertex (Vertex mit dem höchsten Gewicht) werden rekursiv Vertices, deren Gewichtung einen gewissen Schwellwert (Vertex Weight Percentage) übersteigt, ermittelt und zu einem Cluster (Complex) zusammengefügt. Dieser Schwellwert ist ausschlaggebend für die Dichte des resultierenden Clusters.

Im nächsten Schritt werden die Nachbar-Knoten jedes neu hinzugefügten Knoten (Vertex) dem selben Prozess unterzogen. Jeder Knoten wird jedoch maximal einmal überprüft, wodurch es in dieser Phase zu keinen Cluster-Überlappungen kommt. Können keine weiteren Nodes hinzugefügt werden, startet der Prozess mit dem am nächsthöchst gewichteten, noch nicht überprüften Knoten von Neuem.

Postprocessing, die dritte Stufe des MCODE Verfahrens, dient der Nachbearbeitung der Ergebnisse aus der Complex Prediction. Dabei werden die Cluster auf Basis unterschiedlicher Kriterien bzw. Parameter gefiltert oder modifiziert. Beispielsweise finden jene Cluster, die nicht zumindest einen Graph mit minimalem Degree von 2 enthalten, keine Berücksichtigung im Endergebnis. Weiters kann mit Hilfe der sog. Fluff Option Einfluss auf die Größe der Cluster genommen werden. Ein hoher Fluff-Wert erweitert die Ergebnis-Cluster, wodurch es auch zu Überlappungen kommen kann. Die als Haircut bezeichnete Funktion erlaubt hingegen das Entfernen von nur einfach verbundenen Nodes. Die auf diese Weise ermittelten Cluster werden anschließend auf Basis ihrer Größe und Dichte gewertet und in aufsteigender Reihenfolge sortiert. [Bader and Hogue, 2003]

Flow Simulation - Markov Clustering (MCL)

Die in [van Dongen, 2000] vorgestellte Graph-Clustering Methode verwendet eine *Random Walk* Strategie zur Auffindung dichter Sub-Graphen innerhalb der Graphstruktur. Ein Random Walk bezeichnet allgemein eine Zufallsbewegung bzw. eine statistische Wanderung in eine oder mehrere Dimensionen (siehe dazu auch [Grinstead and Snell, 1997] bzw. [Foell, 2008]). Hinter dem MCL-Algorithmus verbirgt sich folgende Grundidee: Es ist unwahrscheinlich, dass ein Random Walk einen dichten Cluster, in dem er sich gerade befindet, verlässt, ohne nicht vorher viele Nodes (Vertices) dieses Clusters besucht zu haben. Es soll auf diese Weise eine Strömung innerhalb des Graphen simuliert werden (*Flow Simulation*). Die Strömung wird zwischen den Grenzen der einzelnen Sub-Graphen (Gruppen) abgeschwächt,

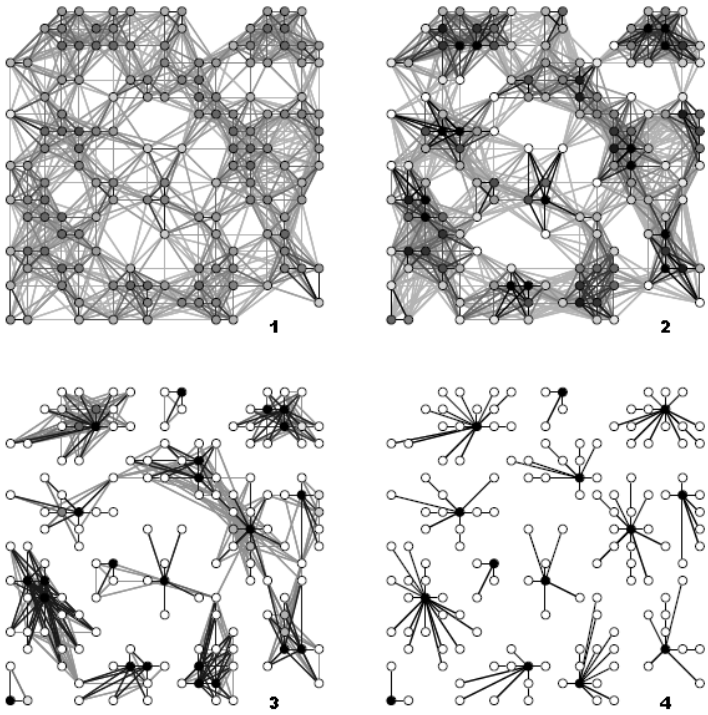


Abbildung 2.12: Flow Simulation mittels Markov Clustering. (Quelle: [van Dongen, 2000])

wodurch eine mögliche Clusterstruktur innerhalb des Graphen identifiziert werden kann. In Abbildung 2.12 sind vier unterschiedliche Iterationsstufen der Flow Simulation anhand eines Beispielgraphen dargestellt. Die Schattierung der Nodes dient dabei als Indikator für die Summe der einfließenden Strömung (dunklere Nodes - größere einfließende Strömung). Die unterschiedlichen Grautöne der Edges stehen wiederum für unterschiedliche Maximalwerte der Strömung (dunklere Edges - größeres Maximum). Beispiel: Schwarze Edges zwischen weißen und schwarzen Nodes bedeuten in diesem Fall, dass das Maximum der Strömung in Richtung der schwarzen Nodes fließt. [van Dongen, 2000] [Brandes et al., 2003]

Level-Of-Detail (LOD)

Der in [Liu et al., 2005] vorgestellte Ansatz bedient sich Elementen des flat bzw. hierarchical Clusterings. Es handelt sich dabei um kein reines Clustering-Verfahren sondern vielmehr um eine Kombination aus Clustering- und Layout-Techniken.

Zu Beginn werden repräsentative Nodes des Graphen ausgewählt. Mittels *Wavefront Algorithmus* erfolgt anschließend die Gruppierung weiterer Nodes um jene auswählten Knoten herum. Der Wavefront Algorithmus durchläuft den Graphen wellenförmig. Dabei werden alle von einem Start-Node aus erreichbaren Knoten mit Hilfe einer *Breadth-First* Suche ermittelt (siehe dazu auch [Siek, 2001] bzw. [Linz, 2002]). Auf Basis dessen erfolgt die Zusammenfassung der Nodes eines Graphen zu mehreren Clustern. Die Nodes eines Clusters bilden in weiterer Folge eigene Sub-Graphen. Der Vorgang wird solange wiederholt, bis die Node-Anzahl jedes Sub-Graphen einen gewissen Schwellwert erreicht hat. Das Ergebnis ist eine hierarchische Baumstruktur (*LOD-Tree*), wobei jeder Knoten dieses Baumes einen Cluster repräsentiert. Im Anschluss erfolgt das Layout der einzelnen Sub-Graphen mittels Force-Directed Techniken. [Liu et al., 2005].

2.3.4 Cluster-Repräsentation/Darstellung

Mit Hilfe der zuvor erwähnten Metrics ist es möglich, signifikante Nodes/Edges des Graphen zu identifizieren und auszuwählen. In Folge stellt sich nun die Frage nach der geeigneten Repräsentation bzw. Darstellung sowie nach der weiteren Behandlung sowohl der ausgewählten als auch der nicht ausgewählten Graphelemente.

Es existieren unterschiedliche Varianten zur Repräsentation einer Gruppe von zusammengehörigen Objekte. Häufig dient der Clusterschwerpunkt (*Centroid*) bzw. ein spezielles Objekt der Gruppe als Repräsentant. Eine weitere Möglichkeit wäre die Repräsentation des Clusters in Form eines Polygons (*Convex Hull*). Die Form des Polygons wird dabei von den äußeren Objekten (hinsichtlich ihrer Platzierung im zwei- oder dreidimensionalen Drawing Space) bestimmt. [Sabol, 2001]

Nachfolgend werden einige im Zusammenhang mit der Darstellung und Behandlung von Clustern bzw. dem Layout eines geclusterten Graphen relevanten Konzepte erläutert.

Skeleton

Ein Skeleton bezeichnet in diesem Fall das Grundgerüst eines Graphen. Es besteht aus jenen Nodes und Edges deren Metric einen gewissen Schwellwert übersteigen. Es handelt sich dabei um eine Untermenge signifikanter Nodes und Edges eines großen Graphen (Sub-Graph). Die Schwierigkeit liegt in der Wahl der passenden Metrics, da diese direkten Einfluss auf das Aussehen und den Aufbau des Skeletons haben. Das Gerüst sollte die grundlegende Struktur des originalen Graphen widerspiegeln. Der Algorithmus besteht prinzipiell aus folgenden vier Hauptschritten:

1. Wahl einer Metric und eines Schwellwertes (*Cutoff Value*) (ausschlaggebend für den Detailgrad des Skeletons).
2. Selektion aller Nodes bzw. Edges des Graphen, welche den Cutoff Value übersteigen.
3. Aufbau und Darstellung des Skeletons aus den zuvor selektierten Nodes/Edges.
4. Darstellung bzw. weitere Behandlung der nicht ausgewählten Nodes/Edges.

Nicht ausgewählte Nodes und Edges (Schritt 4) könnten in der Skeleton-Darstellung einfach ausgespart werden. Diese Möglichkeit wird auch *Hiding* bezeichnet. Eine Alternative dazu wäre eine als *Ghosting* bezeichnete Vorgehensweise. Dabei werden nicht selektierte Elemente in unauffälliger Art und Weise dargestellt. Dies kann beispielsweise durch Verschieben der Nodes/Edges in den Hintergrund, Erhöhung der Transparenz oder Zuordnung einer anderen Farbe erreicht werden.

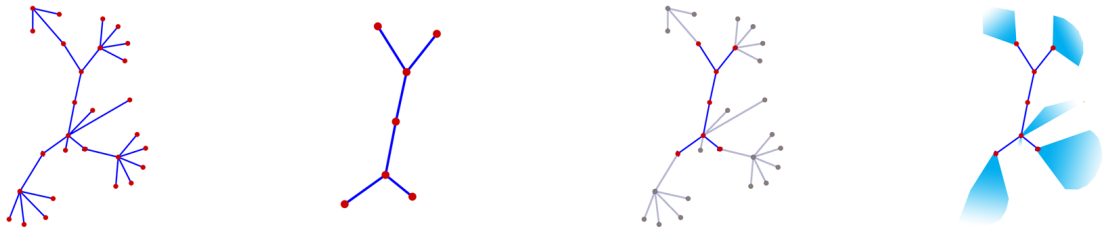
Als dritte Variante bietet sich *Grouping* an. Dabei werden Nodes und Edges zusammengefasst und mittels eines eigenen Icons (auch Glyph genannt) visualisiert. Form, Farbe, Größe, etc. des Icons sollten dabei Rückschlüsse auf die ursprüngliche Struktur des Sub-Graphen bzw. auf die Anzahl der enthaltenen Elemente zulassen. So könnten in der Visualisierung Sub-Graphen etwa durch Polygone, Dreiecke, Halbkreise oder andere geometrische Figuren ersetzt werden. [Herman et al., 1999] [Herman et al., 2000]

In Abbildung 2.13 wird anhand eines Beispielgraphen die Anwendung von Hiding, Ghosting und Grouping demonstriert.

Meta-Nodes

Die mittels Clustering-Methoden identifizierten Gruppen müssen nun in geeigneter Weise visualisiert werden. Wie bereits erwähnt, sollten dabei einige Grundregeln der Aggregierung beachtet werden:

- Die Grundstruktur des originalen Graphen sollte möglichst unverändert bleiben.



(a) Originalgraph **(b)** Hiding **(c)** Ghosting **(d)** Grouping

Abbildung 2.13: Skeleton-Konzept anhand eines Beispielgraphen. (Abbildungen basierend auf [Herman et al., 2000]). (a) Originalgraph (b) Extrahiertes Grundgerüst (Skeleton), nicht selektierte Nodes/Edges wurden ausgeblendet. (c) Darstellung der nicht selektierten Nodes/Edges in grauer Farbe. (d) Darstellung der Sub-Graphen mittels geometrischer Figuren.

- Details dürfen im Zuge der Aggregation nicht verloren gehen.
- Details bzw. aggregierte Elemente müssen bei Bedarf wieder verfügbar sein.

Als *Meta-Node* (oder auch Super-Node) wird ein Knoten eines Graphen bezeichnet, welcher einzelne Nodes bzw. Gruppen von Nodes in sich vereint (aggregiert). Ein Meta-Node stellt somit eine Abstraktion eines Sub-Graphen dar [Morris and Avila-Campillo, 2009]. Zur besseren Übersicht bzw. zur besseren Unterscheidung von Standardknoten werden Meta-Nodes meist in spezieller Weise dargestellt. Dies kann beispielsweise mithilfe entsprechender Icons oder Glyphs, aber auch durch eine unterschiedliche Farb- oder Größenkodierung erreicht werden. Das Konzept der Meta-Nodes erfüllt einerseits die Forderung nach Reduktion der visuellen Komplexität einer Graphdarstellung, andererseits werden aber auch die zuvor erwähnten grundlegenden Aggregationsregeln weitestgehend beachtet. (siehe auch [Eades and Huang, 2000], [Herman et al., 2000] bzw. [Schaffer et al., 1998])

Im Zusammenhang mit dem Konzept von Meta-Nodes sind einige wichtige Aspekte zu beachten. So stellt sich in Folge der Aggregation von Nodes die Frage nach einer geeigneten Behandlung der Edges zwischen neu entstandenen Meta-Nodes. Eine Variante wäre, Edges komplett wegzulassen und stattdessen durch entsprechende Positionierung der Nodes die Verbindungen nur anzudeuten [Herman et al., 2000]. In [Huang and Eades, 1998] wird hingegen auch die Möglichkeit zur Aggregation von Edges (Meta-Edges) sowie eine Methode zur Induktion von Edges zwischen Meta-Nodes vorgestellt. Grundsätzlich müssen Kanten, welche im Zuge der Aggregation reduziert wurden, ebenfalls bei Bedarf wieder eingeblendet werden können.

Form, Größe, Farbe bzw. Position eines Icons sollte Rückschlüsse auf die im jeweiligen Meta-Node aggregierten Elemente zulassen. Wie zuvor im Zusammenhang mit Grouping bereits erwähnt, können verschiedene geometrische Figuren zur Andeutung der Struktur der zusammengefassten Nodes/Edges verwendet werden [Herman et al., 2000] [Herman et al., 1999]. Weiters wäre es wünschenswert, dass aufgrund des Aussehens des Icons Aussagen über die Anzahl der aggregierten Elemente getätigt werden können. Dies kann zum Beispiel durch Farb- bzw. Größenkodierung oder mittels entsprechenden Labels erfolgen. In [Sprenger et al., 1998] werden Meta-Nodes etwa in Form halb-transparenter Icons dargestellt. Für die Darstellung von Meta-Edges gilt Analoges.

Um reduzierte Details oder aggregierte Elemente bei Bedarf wieder darstellen zu können, sollten Meta-Nodes (bzw. auch Meta-Edges) Collapse- und Expand-Funktionalitäten unterstützen. Das Ein-/Ausblenden aggregierter Elemente erfolgt durch Auffächern bzw. Zuklappen der Meta-Nodes. Im Zuge dessen müssen natürlich auch Meta-Edges entsprechend aufgefächert oder zugeklappt werden. Expand/-Collapse kann entweder automatisiert (z.B. in Abhängigkeit des jeweiligen Zoomfaktors) oder interaktiv durch den User erfolgen (beispielsweise durch Klick auf das Icon) [Wills, 1999]. Einzelne Cluster-Hierarchiestufen können so durch Öffnen oder Schließen von Meta-Nodes durchlaufen werden (siehe

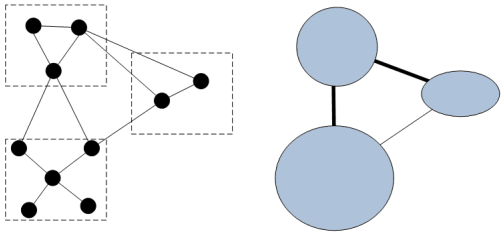


Abbildung 2.14: Links: Originalgraph mit bereits identifizierten Gruppen (gestrichelte Rechtecke). Rechts: Repräsentation der Cluster mittels Meta-Nodes (blaue Ellipsen). Die Größe der Meta-Nodes wird durch die Anzahl der aggregierten Nodes bestimmt. Ursprüngliche Kanten wurden zu Meta-Edges aggregiert. Die Breite der Linien gibt Aufschluss über die Anzahl der enthaltenen Edges. (basierend auf: [Wang and Miyamoto, 1996] bzw. [Kaufmann and Wagner, 2001] Kapitel 4)

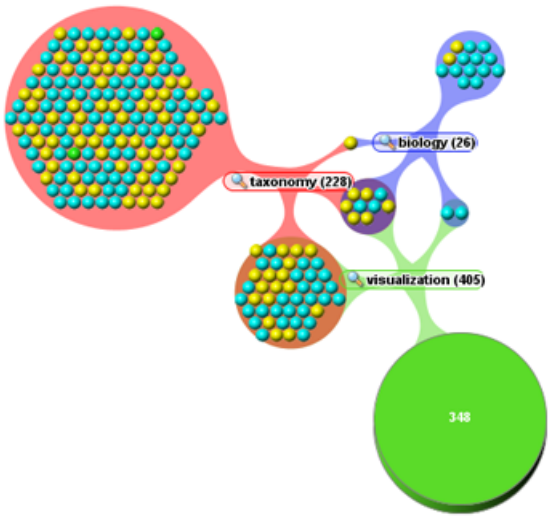


Abbildung 2.15: Cluster-Map: Meta-Nodes werden als transparente Kreise, einzelne Elemente als Spheres dargestellt. (Quelle: [Aduna-Software, 2008])

[Schaffer et al., 1998]). Abbildung 2.14 zeigt die Aggregierung von Nodes und Edges anhand eines Beispielgraphen.

Das in [Aduna-Software, 2008] verwendete Visualisierungsverfahren stellt ein konkretes Beispiel für die Anwendung von Meta-Nodes dar. Die Visualisierung einzelner Elemente einer hierarchischen Struktur erfolgt mittels verschiedenfarbiger Spheres. Zusammengehörende Elemente (Cluster) werden in transparenten Kreisen gruppiert und durch geschwungene Edges verbunden. Dadurch entstehen sog. Cluster-Maps. Farbkodierungen erlauben zudem unterschiedliche Aspekte der Cluster-Maps prägnant darzustellen. Die Größe eines Kreises hängt von der Anzahl der darin aggregierten Elemente ab. Ueberschreitet die Anzahl der aggregierten Elemente eine gewisse Grenze, wird der betreffende Kreise durch einen nicht-transparenten Kreis ersetzt. Ein entsprechendes Label gibt Aufschluss über die Anzahl der darin zusammengefassten Elemente. Weiters bestehen diverse Möglichkeiten zur Interaktion (Expand, Collapse von Meta-Nodes,...). Auf diese Weise können komplexe hierarchische Strukturen übersichtlich und verständlich visualisiert werden (siehe Abbildung 2.15).

2.4 Semantische Aspekte

Ein zentraler Punkt dieser Masterarbeit ist die Visualisierung von Graphstrukturen mit semantischem Hintergrund. In den vorangegangen Abschnitten wurden bereits wichtige Konzepte, Prinzipien und Tech-

niken der Information- bzw. Graph-Visualisation vorgestellt und näher erläutert. Dieser Abschnitt widmet sich speziellen Fragestellungen welche sich im Zuge der Behandlung bzw. Repräsentation semantischer Graphen ergeben. Der Fokus liegt dabei vor allem auf Graphstrukturen im RDF-Format. Diese weisen in der Regel eine hohe Node-Anzahl sowie eine hohe Verbindungsdichte auf.

Die Einbeziehung semantischer Informationen eröffnet unterschiedlichste Möglichkeiten hinsichtlich Darstellung und Interaktion. Verschiedene Anwendungen, etwa aus den Bereichen Wissensrepraesentation, Linguistik, künstliche Intelligenz, Link bzw. Information Analysis, Datenbankmanagement, etc. beruecksichtigen semantische Aspekte bei der Bearbeitung bzw. Analyse vorliegender Informationen. Dabei spielt neben der strukturellen bzw. schematischen Perspektive vor allem die Bedeutung von Knoten und Kanten eine entscheidende Rolle. [Deligiannidis et al., 2007] [Barthelemy et al., 2005]

2.4.1 Semantische Graphen

In Abschnitt 2.2 wurden relevante Eigenschaften allgemeiner bzw. spezieller Graphen sowie Möglichkeiten und Techniken zur Darstellung solcher Graphen diskutiert. Semantische Graphen weisen im Vergleich dazu einige Besonderheiten sowie zusätzliche Merkmale auf. Im Rahmen dieser Arbeit sind folgende drei Charakteristika von essentieller Bedeutung:

- Die Bedeutung (Semantik) von Nodes und Edges ist eindeutig definiert.
- Die Semantik ist ein Bestandteil des Graphen.
- Semantische Informationen existieren in maschinenlesbarer bzw. maschineninterpretierbarer Form.

Grundsätzlich kann ein *semantischer Graph* als ein Verbund heterogener Nodes und Edges (Links) betrachtet werden. Heterogen bedeutet in diesem Fall, dass der Graph aus Nodes und Edges unterschiedlicher Art aufgebaut sein kann. Die Menge der unterschiedlichen Arten (Typen, Klassen) ist in den meisten Fällen sehr viel kleiner als die Menge an vorhandenen Nodes/Edges. Jedem Knoten werden ein Typ (z.B. Person, Fahrzeug, etc.) sowie gegebenfalls weitere Attribute (z.B. Name, Alter, Geschlecht,...) zugeordnet. Attribute dienen der Bereitstellung weiterer Informationen bzw. der näheren Spezifikation. Links können ebenfalls Typen (z.B. 'lebt in', 'ist Freund von', etc.) bzw. Attribute (Gewichtung, Priorität,...) aufweisen. Kanten in einem semantischen Graphen sind gerichtet, d.h. sie besitzen eine Orientierung. Weiters besteht die Möglichkeit mehrfacher Links zwischen einem Knotenpaar.

Die in einem semantischen Graphen enthaltenen Konzepte und Relationen werden mithilfe einer *Ontologie* (oder eines *Schemas*) beschrieben bzw. spezifiziert. Die Menge an möglichen Relationen ist dabei von den vorhandenen Node- bzw. Link-Typen abhängig. Eine Ontologie kann in Form eines sog. *Ontologie-Graphen* dargestellt werden. Abbildung 2.16 zeigt das Beispiel eines semantischen Graphen. Die zugrundeliegende Ontologie umfasst drei Node-Typen (Person, Stadt, Konferenz) sowie vier unterschiedliche Link-Typen ('kennt', 'wohnt in', 'findet statt in', 'nimmt teil an'). Die Ontologie definiert ebenfalls die erlaubten Relationen. So kann eine Person nur eine andere Person, nicht aber beispielsweise eine Stadt kennen. Weiters kann z.B. eine Konferenz nur in einer Stadt stattfinden. Den Nodes und Edges des semantischen Graphen sind die verschiedenen, mithilfe der Ontologie beschriebenen/spezifizierten Typen bzw. Relationen zugeordnet. In diesem Zusammenhang wird häufig auch von Instanzen der spezifizierten Klassen gesprochen (wobei es zum Klassen-Instanzen Konzept von Programmiersprachen einige Unterschiede gibt; siehe auch Unterpunkt 2.4.3).

Ein weiteres zentrales Merkmal solcher Graphkonstrukte ist die Maschinenlesbarkeit bzw. Maschineninterpretierbarkeit der vorliegenden semantischen Informationen. Dadurch sind Programme bzw. Anwendungen in der Lage, die Informationen zu lesen, zu verarbeiten und in Folge deren Bedeutung zu erfassen. Standards bzw. Formate wie *RDF (Ressource Description Framework)* oder *OWL (Web Ontology Language)* bilden dafür die Basis (siehe Abschnitt 2.4.2). Maschinenlesbare, semantische Informationen erleichtern die Integration eines Graphen in unterschiedliche Applikationen und eröffnen zudem

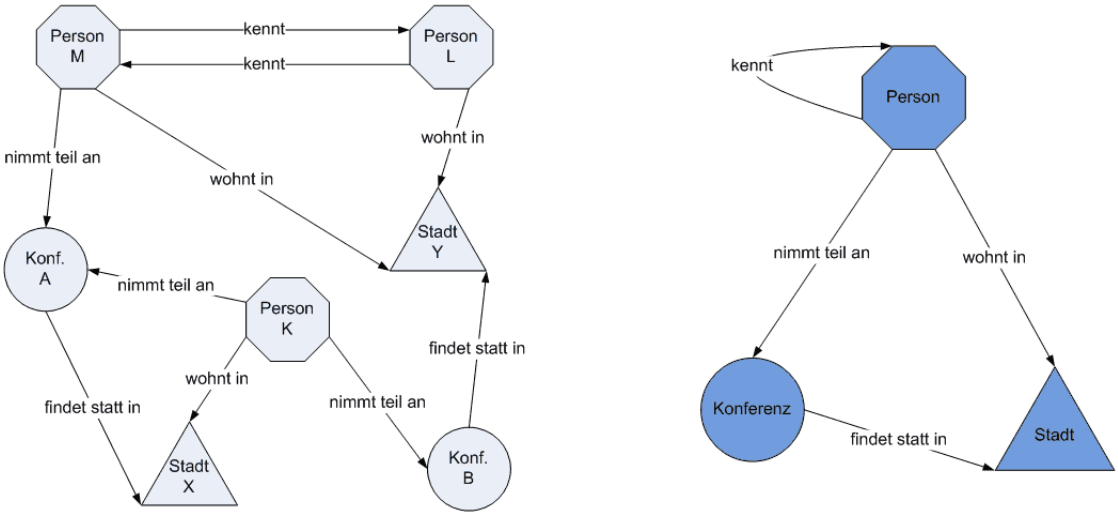


Abbildung 2.16: Beispiel eines semantischen Graphen sowie der zugrundeliegenden Ontologie (Schema). Links: Semantischer Graph. Rechts: Ontologie-Graph. Dieser zeigt erlaubte Relationen sowie die verschiedenen Node- bzw. Link-Typen. (basierend auf: [Barthelemy et al., 2005])

alternative Möglichkeiten hinsichtlich Clustering, Aggregation, Layout, Navigation, etc. Des weiteren finden sich eine Reihe Algorithmen und Techniken, welche sich mit speziellen Problemstellungen in Verbindung mit semantischen Graphen auseinandersetzen (beispielsweise Verfahren zur Untersuchung des Charakters einer Relation oder Algorithmen zur Bestimmung der Relevanz von Links; siehe dazu auch [Barthelemy et al., 2005]).

Es existiert eine Vielzahl an möglichen Einsatzgebieten semantischer Graphen. Konkrete Anwendungen wären etwa Filmdatenbanken (Knoten stehen für Filme, Personen, Auszeichnungen, Studios,...) oder Zitatsammlungen (Knoten bestehen nicht nur aus Artikeln sondern auch aus Autoren, Institutionen, Konferenzen, etc.). Weitere Anwendungsbeispiele wären Applikationen zur Analyse von sozialen Netzwerken, chemischen Verbindungen oder molekularen Netzen (Bioinformatik). Als Datenquellen semantischer Graphen können u.a. Datenbanken, Textdokumente, Online-Enzyklopädien (z.b. Wikipedia), usw. fungieren. [Barthelemy et al., 2005] [Wong et al., 2006] [Spivack, 2007]

Beispiel

Folgendes Beispiel soll die zuvor erwähnten Konzepte semantischer Graphen veranschaulichen: Die Beispiel-Ontologie in Abbildung 2.16 beschreibt Node-Typen (Klassen) und Relationen. (Anmerkung: Eine Ontologie kann u.a. mithilfe der Beschreibungssprachen RDF(-S) oder OWL formal bzw. in maschinenlesbarer Weise definiert werden; siehe dazu Unterpunkt 2.4.3.) Auf Basis dieser Ontologie erstellt Anwendung A einen Graphen, der u.a. die Aussage '*[Parteitagung] [findet statt in] [Wien]*' enthält. Mithilfe dieser Ontologie wird die Bedeutung der Beziehung '*findet statt in*' zwischen den beiden Entitäten '*Parteitagung*' und '*Wien*' eindeutig definiert. Anwendung B kann nun unter Bezugnahme auf die zuvor definierte, sowohl für Anwendung A als auch für Anwendung B verfügbare Ontologie folgende Fakten ableiten sowie die Bedeutung der Relation eindeutig erkennen:

- *'Parteitagung' ist eine Instanz vom Typ 'Konferenz'.*
- *'Wien' ist eine Instanz vom Typ 'Stadt'.*
- *Die Relation 'findet statt in' ist nur zulässig zwischen einer Instanz vom Typ 'Konferenz' und einer Instanz vom Typ 'Stadt'*

- *'findet statt in' beschreibt die Relation zwischen einer 'Konferenz' und einer 'Stadt'*

Auf diese Weise wird der problemlose, verlustfreie Austausch bzw. die korrekte Behandlung und Weiterverarbeitung der von Anwendung A erstellten Informationen durch Anwendung B sichergestellt.

2.4.2 Ressource Description Framework (RDF)

Im Kontext semantischer Graphen sind Informationen, welche von Anwendungen/Programmen gelesen und verarbeitet werden können, von zentraler Bedeutung. Die Bedeutung (Semantik) des Graphen muss eindeutig definiert und in einer maschinenlesbaren Form vorliegen. Um dies zu ermöglichen, bedarf es eines einheitlichen Formats bzw. einer maschinenverständlichen Sprache. Mithilfe solcher Sprachen ist es möglich, semantisch angereicherte Modelle zu generieren. Diese Modelle bestehen aus einer Menge von Triple (Subjekt, Prädikat, Objekt), wobei jedes Triple die Relation zwischen zwei Entitäten beschreibt ([Mutton and Golbeck, 2003]). Beispiele formaler Beschreibungssprachen wären *RDF (Ressource Description Framework)* oder *OWL (Web Ontology Language)*.

RDF ist grundsätzlich eine Sprache/ein Framework zur Repräsentation von Informationen bzw. Metadaten über Web-Resources (z.B. Title, Autor, Erstellungsdatum einer Website). Durch Generalisierung des Begriffs der Web-Resource ist es mittels RDF auch möglich, Informationen über Dinge (*Things, Resources*), welche nicht direkt aus dem Web abrufbar sind, darzustellen. D.h. neben Websites können auch Personen, Orte, Events, etc. beschrieben werden. Somit sind Aussagen (*Statements*) über jede beliebige Resource möglich. Dinge werden mit Hilfe von *URIs (Uniform Resource Identifiers)* eindeutig identifiziert. Daneben wird die auch Verlinkung von Informationen quer durch das Web ermöglicht.

Eine weitere Besonderheit von RDF ist, dass Informationen von Programmen gelesen und verarbeitet werden können. RDF stellt in diesem Zusammenhang ein allgemeines Framework zur Beschreibung der Informationen zur Verfügung. Dadurch wird ein verlustfreier Informationsaustausch (hinsichtlich der Bedeutung von Informationen) zwischen verschiedenen Anwendungen ermöglicht. [Manola and Miller, 2004]

Ein Hauptziel bei der Entwicklung des RDF-Formats war die Schaffung eines simplen Datenmodells bzw. einer formalen Semantik. Dies bietet die Basis, um Schlussfolgerungen hinsichtlich der Bedeutung eines RDF-Ausdrucks zu ziehen (auch als *Reasoning* bezeichnet). Weitere Ziele waren die Verwendung einer XML-basierten Syntax sowie der Gebrauch eines erweiterbaren, URI-basierten Vokabulars zur Formulierung von Aussagen über jede beliebige Resource [Klyne and Carroll, 2004].

Nachfolgend werden die grundlegenden Konzepte von RDF näher erläutert. [Klyne and Carroll, 2004] [Manola and Miller, 2004]

Graph Data Model

Die Grundstruktur eines Ausdrucks (*Statement*) in RDF ist ein *Triple*, bestehend aus *Subject*, *Predicate* und *Object*. Jedes Triple beschreibt die Beziehung zwischen zwei Dingen. Ein Subject kann entweder eine URI-Referenz oder ein Blank Node sein. Ein Object kann eine URI-Referenz, ein Blank Node oder ein Literal sein. Ein Predicate (auch als Property bezeichnet) besteht hingegen nur aus einer URI-Referenz.

Jedes Triple kann mittels eines Node-Edge-Node Diagramms dargestellt werden. Nodes repräsentieren dabei Subjects bzw. Objects, Edges stehen hingegen für Predicates. Die gerichteten Kanten verlaufen dabei immer von Subject zu Object (Subject-Predicate-Object). Eine Menge von Triple wird in Folge als *RDF-Graph* bezeichnet.

URI (Uniform Resource Identifier)

Wie zuvor erwähnt, können Nodes eines RDF-Graphen entweder eine URI-Referenz, ein Literal oder ein Blank Node sein. Edges bestehen hingegen ausschließliche aus einer URI-Referenz. URIs ermöglichen eine eindeutige Identifikation von Nodes bzw. Edges. Sie können sich auf webzugängliche Dinge (z.B. elektronische Dokumente, Homepages, Bilder, etc.), nicht-webzugängliche Dinge (beispielsweise Personen, Organisationen, etc.) sowie auf abstrakte Konzepte (z.B. Erstelldatum, Urheber,...) beziehen. Eine URI-Referenz ist eine Kombination aus einer URI (z.B. `http://www.xyz.org/index.html`) und einer optionalen Fragmentkennung (z.B. `#chapter5`).

Mittels *Literals* werden Werte (z.B. Nummern, Daten, Namen, etc.) von Objects identifiziert. Dies stellt im Vergleich zur Identifikation mittels URIs eine praktischere und intuitivere Variante dar. Ein Literal kann entweder aus einem einfachen String (*Plain Literal*) oder einer Kombination aus einem String und einem URI-Datentyp (*Typed Literal*) bestehen.

Als *Blank Nodes* werden jene Knoten eines RDF-Graphen bezeichnet, welche weder eine URI-Referenz noch ein Literal sind. In diesem Zusammenhang findet sich auch der Begriff *Anonymus Resource*.

XML-based Syntax - XML Schema Datatypes

Die als RDF/XML bezeichnete Syntax basiert weitestgehend auf XML und ist maschinenlesbar. Auf diese Weise kodierte Datenmodelle ermöglichen den Austausch von Informationen zwischen verschiedenen RDF- bzw. XML-Anwendungen.

RDF bietet keine vordefinierten Datentypen (mit Ausnahme eines Datentyps zur Einbettung von XML in RDF). Weiters ist es auch nicht möglich, neue Datentypen zu definieren. Stattdessen werden hauptsächlich XML-Datentypen verwendet.

Simple Facts

Mittels eines RDF-Triples (Statement) sollen einfache Fakten ausgedrückt werden. Ein Statement setzt sich somit aus folgenden drei Komponenten zusammen:

- Einem Ding (Subject), welches beschrieben wird (z.B. eine Website).
- Einer spezifische Eigenschaft (Property bzw. Predicate) des Subjects (z.B. Ersteller, Erstelldatum, Sprache, ...).
- Ein Ding (Object), welches für den Wert dieser Eigenschaft steht (z.B. 'Max Muster', 'März 2005',...).

Ein Beispiel dafür wäre folgende Aussage: '*Die Website* `http://www.xyz.org/index.html` hat einen *Ersteller* dessen Wert ist *Max Muster*'. Dieselbe Aussage in Triple-Schreibweise (Subject-Predicate-Object):

```
<http://www.xyz.org/index.html>   <Ersteller>   <Max Muster>
```

2.4.3 RDF Schema - OWL

RDF-S und OWL sind Schema- bzw.Ontologie-Beschreibungssprachen. *RDF Schema (RDF-S)* ist eine Beschreibungssprache speziell für RDF. Mittels RDF-Statements werden Aussagen über verschiedene Resources getätigt. Voraussetzung zur Formulierung dieser Statements ist jedoch die Definition eines, zum jeweiligen Anwendungsfall passenden Vokabulars. Spezifische Klassen und Eigenschaften

(Properties) dienen zur näheren Beschreibung einer Resource. RDF selbst bietet weder die Möglichkeit zur Definition eigner Klassen oder Eigenschaften, noch existiert ein bereits vordefinierter Sprachschatz. Aus diesem Grund stellt RDF-S Mittel zur Beschreibung eines anwendungsspezifischen Vokabulars zur Verfügung.

Ähnlich dem Typkonzept einer objektorientierten Programmiersprache können Resources in RDF-S Instanzen einer oder mehrerer Klassen sein. Ebenfalls ist es möglich, Klassen hierarchisch zur strukturieren. Der größte Unterschied zum Konzept von Programmiersprachen besteht jedoch darin, dass eine Resource nicht strikt einer Klasse zugeordnet ist bzw. fixe Eigenschaften aufweist. Stattdessen stellen Klassen- und Property-Beschreibungen in RDF-S lediglich Zusatzinformationen über die jeweilige Resource zur Verfügung.

RDF-S Vokabular-Beschreibungen (Schemas) werden in Form von RDF-Statements bereitgestellt. Dadurch sind RDF-fähige Anwendung in der Lage, diese Schemas zu verarbeiten bzw. als gültigen RDF-Graph zu interpretieren. Um jedoch auch die Bedeutung zu 'verstehen' muss zusätzlich zum RDF-Vokabular (erkennbar am Prefix `rdf:`) das RDF-S Vokabular (Prefix `rdfs:`) eingebunden werden. [Manola and Miller, 2004]

Als Klasse gilt in RDF-S grundsätzlich jede Resource, welche eine `rdf:type`-Eigenschaft mit dem Wert `rdfs:Class` aufweist. Eine Eigenschaft (Property) ist eine Instanz der Klasse `rdf:Property`. Beispielsweise würde eine Klasse 'Car' bzw. die Eigenschaft 'Color' demnach mithilfe folgender Statements beschrieben werden:

```
http://www.xyz.org/Car    rdf:type  rdfs:Class
http://www.xyz.org/Color  rdf:type  rdfs:Property
```

Eine ausführliche Auflistung bzw. Beschreibung der einzelnen Klassen und Properties findet sich in [Brickley and Guha, 2004]. [Manola and Miller, 2004] bietet zudem eine gute Einführung sowie weiterführende Details zum Thema RDF Schema.

OWL ist ein weiteres Beispiel einer Ontologie-Sprache. Aufbauend auf XML, RDF bzw. RDF-S wird zusätzliches Vokabular zur Beschreibung von Klassen und Properties bereitgestellt. OWL bietet weitergehende Möglichkeiten um Bedeutung bzw. Semantiken auszudrücken. Dadurch werden die Fähigkeiten anderer Sprachen zur Repräsentation von maschineninterpretierbaren Inhalten erweitert. Für eine ausführliche Behandlung der grundlegenden Konzepte von OWL sowie weitere Details siehe [McGuinness and van Harmelen, 2004].

2.4.4 Ontology-Inference

Als *Inference* wird prinzipiell das Herleiten zusätzlicher Informationen und Fakten aus Instanzdaten bzw. Klassenbeschreibungen/Schemas (OWL, RDFS,...) bezeichnet. Ein *Reasoner* ist ein Programm zur Durchführung einer Inference-Operation. Das Inference-Konzept soll anhand eines Beispiels verdeutlicht werden (entnommen aus: [Jena, 2009b]):

Das folgende Schema definiert ein Property `Elternteil` (mit der Range/Domain `Person`), ein Sub-Property `Mutter` vom Typ `Elternteil` sowie den Integer-Wert `Alter`. Der dazugehöriger Instanzdatensatz beschreibt einen `Teenager` mit dem Namen `Colin`, welcher eine `Mutter` mit dem Namen `Rosy` sowie ein `Alter` von `13` hat.

Schema:

```
<rdf:Description rdf:about="&eg;Mutter">
  <rdfs:subPropertyOf rdf:resource="&eg;Elternteil"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="&eg;Elternteil">
  <rdfs:range  rdf:resource="&eg;Person"/>
  <rdfs:domain rdf:resource="&eg;Person"/>
</rdf:Description>

<rdf:Description rdf:about="&eg;Alter">
  <rdfs:range rdf:resource="&xsd;integer" />
</rdf:Description>
```

Instanz:

```
<Teenager rdf:about="&eg;Colin">
    <Mutter rdf:resource="&eg;Rosy" />
    <Alter>13</Alter>
</Teenager>
```

Ein RDFS-Reasoner kann zum Beispiel nun folgende Fakten zum Typ (`rdf:type`) von `Colin` herleiten:

- `Colin` ist ein `Teenager`: Herleitung aufgrund der direkten Definition.
- `Colin` ist sowohl eine `Resource` als auch eine `Person`: Indirekte Herleitung aufgrund der Tatsache, dass `Colin` eine `Mutter`, respektive einen `Elternteil` hat und `Elternteil` Teil der Domain `Person` ist.

Weitere Details zum Thema Inferencing bzw. Reasoning finden sich unter [Jena, 2009b].

2.4.5 Beispiel RDF

Die zuvor erläuterten Konzepte sollen nun anhand eines konkreten Beispiels verdeutlicht werden. Die Basisdaten stammen dabei aus [Manola and Miller, 2004].

Als Ausgangspunkt dient folgende Gruppe von Feststellungen: '*Es gibt eine Person, welche durch den Ausdruck* `http://www.w3.org/People/EM/contact#me` *eindeutig identifiziert wird. Der volle Name lautet Eric Miller. Die Mailbox Adresse der Person lautet em@w3.org. Der Titel der Person ist Dr.*' [Manola and Miller, 2004]

Dieser Ausdruck kann nun mithilfe von RDF in eine maschineninterpretierbare Form gebracht werden. Die Kodierung mittels RDF/XML hat folgendes Aussehen:

```
<?xml version="1.0"?> <rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
    <contact:Person
    rdf:about="http://www.w3.org/People/EM/contact#me">
        <contact:fullName>Eric Miller</contact:fullName>
        <contact:mailbox rdf:resource="mailto:em@w3.org"/>
        <contact:personalTitle>Dr.</contact:personalTitle>
    </contact:Person>
</rdf:RDF>
```

Der Ausgangstext wurde dabei in vier RDF-Statements gegliedert (URI-Referenzen der Properties in Kurzschreibweise).

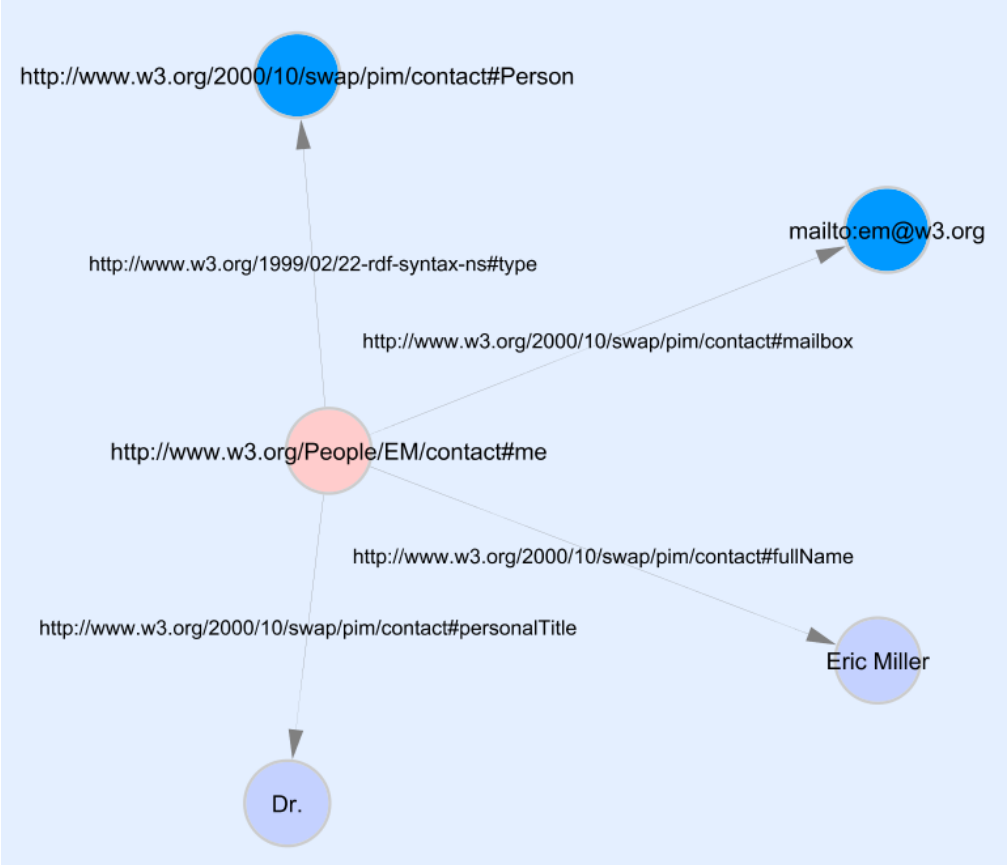


Abbildung 2.17: RDF-Graph (basierend auf: [Manola and Miller, 2004])

Abbildung 2.17 zeigt hingegen die visuelle Repräsentation in Form eines RDF-Graphen. Jede Sourcenode - Edge - Targetnode Relation steht für ein Statement. Der Graph besteht aus einem zentralen Node, welcher das Subject repräsentiert. Die vier davon ausgehenden Edges stehen für die jeweiligen Predicates. Targetnodes dienen zur Darstellung der Objects. Hellblaue Knoten stehen in diesem Fall für Literals, dunkelblaue hingegen für URI-Referenzen.

Anhand dieses Beispiels lässt sich auch die Verwendung von URIs demonstrieren. Folgende Resources werden dadurch eindeutig identifiziert [Manola and Miller, 2004]:

- Individuen (z.B. Eric Miller): `http://www.w3.org/People/EM/contact#me`
- Typen/Klassen (z.B.: Person): `http://www.w3.org/2000/10/swap/pim/contact#Person`
- Properties (z.B. Mailbox): `http://www.w3.org/2000/10/swap/pim/contact#mailbox`
- Werte der Properties (z.B. Wert von 'mailbox' oder 'personalTitle'): `mailto:em@w3.org` bzw. `Dr.` (String-Literal).

Es gibt verschiedene Möglichkeiten zur Kodierung bzw. Speicherung von RDF-Daten. Bei allen Varianten handelt es sich um Plaintext Formate, wodurch es für Menschen relativ schwierig ist, diese zu lesen bzw. zu interpretieren. Unterschiede finden sich vor allem in der Anordnung der einzelnen Statements bzw. der Schreibweise von Subject, Predicate und Object. Dadurch soll u.a. eine Verbesserung der Lesbarkeit erzielt werden.

Zuvor wurde bereits die Möglichkeit zur Kodierung der Daten in Form von *RDF/XML* besprochen. Üblicherweise werden diese Inhalte in Dateien mit der Endung '.rdf' gespeichert. Weitere gängige

Formate wären *N-Triples* (Endung '.nt') sowie *Notation3* (Endung '.n3'). Siehe dazu auch [Grant and Beckett, 2004] bzw. [Berners-Lee, 2006].

2.5 Theorie der Visualisierung - Teilaspekte

In diesem Abschnitt werden ausgewählte Aspekte aus den verschiedenen Themenbereichen näher betrachtet. Einige der Punkte wurden in den vorangegangen Abschnitten bereits kurz angeschnitten bzw. überblicksmäßig erläutert. Relevante Themen werden an dieser Stelle nochmals aufgegriffen bzw. essentielle Konzepte näher erläutert.

2.5.1 Interaktivität

Zur Visualisierung großer Graphen bzw. Netzwerke bedarf es interaktiver Techniken und Konzepte. Bestimmte Tasks wie etwa die Analyse einer Graphstruktur oder das Erkennen von Mustern, Regelmaeßigkeiten und Relationen sind mit Hilfe einer rein statischen Repräsentation kaum durchführbar. Der User muss die Möglichkeit haben, mit der Darstellung zu interagieren um beispielsweise den Betrachtungswinkel zu ändern, das Layout anzupassen oder Details bei Bedarf ein- bzw. auszublenden. Die Auswirkungen einer Aktion (beispielsweise einer Parameteränderung) auf die Graphdarstellung sollten unmittelbar erkennbar sein. Weitere Merkmale einer interaktiven Visualisierung wären u.a. die Unterstützung von Navigations- bzw. Explorations-Funktionalitäten sowie das Vorhandensein von Techniken zur intuitiven Bearbeitung/Manipulation des dargestellten Graphen bzw. einzelner Graphelemente. Wie in Abschnitt 2.1.1 bereits erwähnt, ist neben der grafischen Repräsentation somit die Möglichkeit zur Interaktion mit der Darstellung von entscheidender Bedeutung und eine Grundvoraussetzung jeder computerunterstützten Visualisierung. [Andrews, 2002] [Sabol, 2001]

Im Kontext interaktiver Visualisierungen sind kritische Faktoren wie Zeit bzw. Raum zu beachten. Graph-Layout sowie Cluster-Berechnungen sind unter Umständen äußerst zeitaufwändig, wodurch ein Update der Darstellung mitunter zu lange dauern könnte. Ein Visualisierungssystem muss somit Mechanismen zur Gewährleistung der interaktiven Performance bereitstellen. Die Handhabung bzw. Visualisierung großer Graphen erfordert zudem einen ausreichend großen Speicher sowie ausreichendst Platz/Raum (z.B. Display-Größe) zur visuellen Darstellung. [Abello et al., 2006]

Nachfolgend werden essenzielle Möglichkeiten zur Interaktion mit einer Graphdarstellung sowie einige zentrale Begriffe und Ideen aus den Bereichen Interaktivität bzw. Dynamik vorgestellt und näher erörtert. Die vorgestellten Methoden und Techniken gehen mit den in Abschnitt 2.1.1 erläuterten Grundprinzipien der Information Visualisation konform.

Zooming - Panning

Zooming bzw. Panning (*ZUI* - Zoomable User Interface) sind wesentliche Features einer interaktiven Visualisierung. Im Zuge einer Zoom-In Operation wird ein kleiner ausgewählter Bereich vergrößert bzw. detaillierter dargestellt. Im Gegensatz dazu bewirkt Zoom-Out, dass sich wieder ein größerer Bereich der Darstellung im Blickfeld befindet. Panning bedeutet in diesem Zusammenhang das Schwenken bzw. Verschieben des Blickfeldes. Sowohl Zooming als auch Panning sollten stufenlos erfolgen (*Smooth Zoom*).

Vom Konzept des *Geometric Zooming* wo, wie zuvor beschrieben, Teile der Graphdarstellung einfach vergrößert werden, ist der Begriff des *Semantic Zooming* zu unterscheiden. Semantic Zooming Techniken ändern in Abhängigkeit des Zoomfaktors den Inhalt der Darstellung. Dabei werden Details der Graphstruktur, einzelne Sub-Graphen bzw. zusätzliche Informationen ein- oder ausgeblendet. In diesem Zusammenhang sind die in Abschnitt 2.3 vorgestellten Clustering- bzw. Aggregierungstechniken von Bedeutung.

Mittels semantischem Zoom werden Einzelheiten sichtbar, welche durch eine simple Vergrößerung der Graphelemente nicht ersichtlich wären. Auf diese Weise wird u.a. auch die Navigation bzw. Exploration einer komplexen, semantischen Graphstruktur ermöglicht. Weiters können mittels Semantic Zooming große semantische Netzwerke/Graphen visualisiert werden, indem immer nur einzelne Teile (beispielsweise der aktuelle Zoombereich) dargestellt werden (siehe dazu auch [Andrews, 2002] bzw. [Mueller, 2005]). In vielen Fällen ist eine Kombination aus semantischen und geometrischen Zooming-Techniken sinnvoll. [Herman et al., 2000] [Sabol, 2001]

Navigation

Es existiert eine Reihe von Techniken zur Navigation bzw. Exploration von Graphen und Netzwerken. In diesem Zusammenhang findet sich oft auch der Begriff des Graph-Browsings. Vor allem hinsichtlich der Handhabung semantischer Graphen, welche meist aus einer großen Anzahl von Nodes bestehen bzw. eine hohe Verbindungsdichte aufweisen, spielen diese Konzepte eine entscheidende Rolle.

Mithilfe *inkrementeller Navigations-Methoden* wird immer nur ein kleiner, als *logical Frame* bezeichneter Ausschnitt des ganzen Graphen detailliert dargestellt. In Abhängigkeit des Zoomfaktors bzw. verschiedener User-Interaktionen ändert sich der Fokus, wodurch ein neuer logical Frame erzeugt und dessen Inhalt in weiterer Folge visualisiert wird. Die Graphstruktur wird auf diese Weise schrittweise erforscht. Die Visualisierung dieser kleineren Teilbereiche bzw. Sub-Graphen erfordert nur einen Bruchteil an Rechenaufwand, Arbeitsspeicher und Darstellungsraum. Zentrale Aspekte betreffen dabei die Definition des Inhalts eines logical Frames sowie die Wahl einer geeigneten Layout-Technik zur Darstellung der Sub-Graphen. Beispielsweise können logical Frames aus einem zentralen Knoten sowie dessen unmittelbaren Nachbarknoten bestehen. Das Layout der kleineren Sub-Graphen kann zum Beispiel mit Hilfe von Force-Directed Methoden erfolgen.

Inkrementelle Verfahren eignen sich speziell zur Exploration komplexer Strukturen, in denen einzelne Bereiche des Graphen bzw. des Netzwerks noch nicht bekannt sind. Zudem besteht die Möglichkeit, den Graphen nicht nur auf geometrische Weise (z.B. mittels Scrollbars) bzw. aufgrund der geometrischen Graphrepräsentation, sondern auf Basis logischer bzw. semantischer Informationen (z.B. Relationen, Hyperlinks,...) zu durchwandern (siehe dazu auch *Semantic Zooming*). [Herman et al., 2000] [Kaufmann and Wagner, 2001], Kapitel 8

Die inkrementelle Exploration einer Graphstruktur wird u.a. in [Lee et al., 2006] erläutert. Ein speziell zur visuellen Exploration von RDF-Daten entwickelter Algorithmus findet sich in [Dokulil and Katreniakova, 2008]. [Deligiannidis et al., 2007] präsentiert eine Technik zur inkrementellen Exploration und Visualisierung von semantischen Informationen im RDF-Format. Zudem findet sich dort auch eine Auflistung weiterer Explorations- und Navigations-Methoden.

Eine weitere Möglichkeit zur Graph-Navigation stellen interaktive Clustering- bzw. Aggregierungstechniken dar. Durch iteratives Öffnen bzw. Schließen einzelner Cluster kann eine komplexe Graphstruktur erkundet werden. Eine ausführlichere Behandlung der Themen Aggregation bzw. Clustering findet sich in Abschnitt 2.3. [Abello et al., 2006] [Eades and Huang, 2000] [Kaufmann and Wagner, 2001], Kapitel 8

In Abschnitt 2.1.1 wurde bereits auf die besondere Bedeutung des Fokus+Kontext Prinzips hingewiesen. Im Zusammenhang mit Navigations- bzw. Zooming/Panning-Operationen besteht die Gefahr, dass der User den Überblick über die Darstellung bzw. die Orientierung verliert. Um dies zu verhindern, darf im Zuge der Fokussierung auf einzelne Details der allgemeine Kontext nicht verlorengehen. Dies kann beispielsweise durch den Einsatz von Distortion-Techniken (z.B. *Fisheye Views*) oder die Kombination verschiedener Ansichten (z.B. *Overview+Detail*) erreicht werden. [Herman et al., 2000] [Abello et al., 2006]

Weitere Interaktionsmöglichkeiten

Neben den bisher erläuterten Konzepten kann eine interaktive Visualisierung noch über eine Reihe weiterer Merkmale und Features verfügen. Beispiele dafür wären:

- **Editing**
  Techniken zur Editierung ermöglichen u.a. die Bearbeitung, Manipulation bzw. Modifikation der Graphstruktur. Beispiele für Editierungsoperationen wären die Erstellung bzw. das Löschen von Knoten oder Kanten, das Hinzufügen von Sub-Graphen, die Modifikation von Attributen einzelner Graphelemente oder auch die Möglichkeit, Teile des Graphen manuell zu platzieren bzw. zu verschieben (siehe auch Frasincar et al. [2005], Ellson et al. [2003] bzw. Abschnitt 2.5.2). Eine Editierungsoperation sollte unmittelbare Auswirkungen auf den dargestellten Graphen haben (Gewährleistung der interaktiven Performance).

- **Details on Demand**
  Bei Bedarf müssen Details bzw. zusätzliche Informationen über Nodes, Edges bzw. Cluster abrufbar sein (z.B. Namen, Gewichtungen, Werte,...). Diese Details können im Zuge einer *Selection*- bzw. *Mouse-Over*-Aktion beispielsweise in einem eigenen Fenster angezeigt werden.

- **Filter**
  Filtering ermöglicht das Ausblenden jener Graphelemente, welche gewisse Parameter bzw. Kriterien erfüllen (im Gegensatz zur *Suche*, wo jene Elemente hervorgehoben werden). Einzelne Nodes, Edges, Labels oder Gruppen können zeitweise aus der Darstellung entfernt werden. Vor allem bei einer großen Anzahl an Elementen führt dies zu einer besseren Übersicht. Bei Bedarf müssen ausgeblendete bzw. entfernte Elemente wieder eingeblendet werden können.

- **Queries**
  Die Möglichkeit zur Formulierung von Abfragen ist speziell für Such- bzw. Filteroperationen von Bedeutung. Die Abfrageformulierung sollte durch Bereitstellung verschiedener Features (z.B. Eingabemasken, etc.) erleichtert werden.

Für weitere Details bzw. Anwendungsmöglichkeiten interaktiver Techniken siehe auch [Abello et al., 2006], [Sabol, 2001] sowie [Herman et al., 2000].

2.5.2 Dynamische Aspekte - Dynamic Graph Drawing

Wie aus den bisher behandelten Themen und Szenarien klar hervorgeht, können rein statische Darstellungstechniken den komplexen Anforderungen kaum gerecht werden. Ein interaktives Visualisierungssystem für größere, semantische Graphen muss somit einige Zusatzfeatures inkludieren. Statische Ansätze gehen von der Annahme aus, dass Aufbau und Layout des gesamten Graphen unveränderlich sind. Die Behandlung von Graphen, die sich im Zeitablauf ändern, erfordert meist zusätzliche Möglichkeiten zur Interaktion bzw. spezielle Layout-Techniken und Algorithmen. Der Begriff *Dynamik*, bzw in weiterer Folge der Begriff *Dynamic Graph Drawing* kann in diesem Kontext als das wiederholte Zeichnen des Graphen aufgrund häufiger Änderungen der Graphstruktur definiert werden (siehe [Kaufmann and Wagner, 2001], Kapitel 9).

Strukturänderungen können unterschiedliche Ursachen haben. So kann eine interaktive Visualisierung Funktionen zur manuellen Modifikation der Graphstruktur unterstützen. Der Benutzer hat dabei etwa die Möglichkeit Nodes und Edges hinzuzufügen oder zu entfernen bzw. zusätzliche Layoutparameter zu spezifizieren. Die Struktur eines Graphen kann jedoch auch durch Navigations-Operationen verändert werden. Wie in den Abschnitten 2.3 und 2.5.1 bereits erwähnt, können im Zuge dessen Sub-Graphen bzw. Cluster aufgefächert oder zusammengeklappt werden. Im Falle großer Graphen oder Netzwerke besteht auch die Möglichkeit, immer nur kleine Teilbereiche bzw. Ausschnitte darzustellen. Weiters kann eine

an sich dynamische Struktur mithilfe eines Graphen visualisiert werden. Der Graph ändert sich dabei kontinuierlich, indem permanent Nodes und Edges hinzugefügt oder entfernt werden. Beispiele für dynamische Strukturen wären Computernetzwerke, Web-Sites auf einem Server oder Datenstrukturen in einem laufenden Programm. [Tatzmann, 2004] [North and Woodhull, 2001] [Kaufmann and Wagner, 2001], Kapitel 9

Vorgehensweisen

Die einfachste Methode zur Berücksichtigung dynamischer Aspekte besteht darin, das Layout des Graphen mithilfe statischer Techniken nach jeder Änderung von Grund auf neu zu berechnen. Weist der Graph eine moderate Größe auf, kann diese Methode durchaus zielführend sein. Nichtsdestotrotz gehen damit einige Probleme einher. Beispielsweise ist dieser Ansatz, vor allem bei einer größeren Anzahl an Graphelementen, äußerst ineffizient. Selbst bei minimalen Änderungen der Graphstruktur wird das gesamte Layout neu berechnet. Statische Layout-Algorithmen sind meist zur globalen Optimierung einer Graphstruktur ausgelegt. Dies hat zur Folge, dass bereits geringe Änderungen zu einem im Vergleich zur vorhergehenden Darstellung, komplett unterschiedlichen Layout führen. Dadurch muss sich der Betrachter nach jeder Änderung neuerlich mit der Darstellung vertraut machen.

In diesem Zusammenhang findet sich häufig der Begriff *Mental Map*. Der User betrachtet die Darstellung, interagiert mit ihr und versucht deren Bedeutung zu verstehen. Nach und nach entsteht somit ein geistiges Modell, die sog. mentale Karte oder Mental Map. Mit jeder Änderung der Darstellung muss dieses Modell angepasst bzw. gänzlich neu aufgebaut werden. Eine Visualisierung sollte den Aufwand dafür weitestgehend reduzieren und somit die Beibehaltung des geistigen Modells (*Maintaining the Mental Map*) unterstützen.

Zum einen kann dies durch das Hervorheben von Änderungen und die Animation von Übergängen zwischen einzelnen Darstellungen erreicht werden (*Animated Transitions*). Andererseits besteht die Option, die Änderungen zwischen altem und neuem Layout zu minimieren. Spezielle *Dynamic Graph Drawing* Algorithmen optimieren die Anpassung des Layouts nach Änderungen der Graphstruktur. Dabei werden nur jene Teile des Graphen neu gezeichnet, welche auch tatsächlich von einer Modifikation betroffen sind. Der Rest des Graphen bleibt unverändert. Dies wird u.a. auch als *dynamische Stabilität* bezeichnet. Im Zuge dessen muss ein Kompromiss zwischen traditionellen Ästhetikkriterien (siehe Abschnitt 2.2.4) und Kriterien zur Bewahrung der Mental Map bzw. der dynamischen Stabilität gefunden werden. In [Ellson et al., 2003] wird beispielsweise eine inkrementelle Version eines Sugiyama Layout-Algorithmus zur Berücksichtigung dynamischer Aspekte verwendet. Die einzelnen Phasen des an sich statischen Algorithmus werden dabei den Anforderungen eines dynamischen Graph-Layouts angepasst.

In vielen Fällen besteht ein dynamischer Graph aus einer Sequenz einzelner Graphen (sog. SnapShots). Die meisten Ansätze zur Visualisierung dieser dynamischen Graphen basieren entweder auf einer statisch-kumulativen Darstellung dieser Sequenz (*Cumulative View*) oder auf dynamischen Animationen (*Animated/Sequential View*). [Gaertler and Wagner, 2005] präsentiert hingegen ein hybrides Modell, welches sowohl kumulative als auch sequentielle Techniken vereint. Neben Änderung der Graphstruktur kann mit dieser Methode auch die Entwicklung bzw. Änderung von Knoten- oder Kantengewichten visualisiert werden.

Für eine genauere Betrachtung weiterer Dynamic Graph Drawing Algorithmen sei an dieser Stelle auf [Kaufmann and Wagner, 2001], Kapitel 9 bzw. auf [Sugiyama, 2002] verwiesen. Weitere Aspekte bezüglich Mental Map bzw. Animated Transitions finden sich u.a. in [Tatzmann, 2004] bzw. [Ellson et al., 2003].

Animation

Vor dem Hintergrund von Interaktivität bzw. Dynamik stellt sich unter anderem die Frage, wie Übergänge (*Transitions*) zwischen einzelnen Layouts realisiert werden. Änderungen an der Graphstruktur sowie

Auswirkungen von User-Interaktionen sind durch die Verwendung animierter Übergänge leichter nachvollziehbar bzw. besser erkennbar. Entscheidend dabei ist, wie lange der Wechsel von einer Darstellung zur nächsten dauert. Zu lange Übergänge vermitteln einerseits das Gefühl eines langsamen Systems. Anderseits führen zu schnelle Wechsel dazu, dass die Bewegungen einzelner Objekte nicht mehr nachvollziehbar sind und der Betrachter die gesamte Darstellung somit neuerlich erfassen muss. Grundsätzlich sollten die Bewegungen von Nodes und Edges möglichst gut nachverfolgbar sein. Weiters wäre im Laufe der Animation ebenfalls auf die Einhaltung ästhetischer Graph-Layout Regeln zu achten. [Andrews, 2002] [Munzner, 2000] [Tatzmann, 2004]

Kapitel 3

Visualisierungssysteme

In diesem Kapitel werden aktuell existierende Softwarelösungen (Toolkits, Frameworks, Pakete) zur Visualisierung von Graphen näher betrachtet. Neben den hier diskutierten Systemen existiert noch eine Vielzahl weiterer Pakete zur Graphvisualisierung sowie eine Reihe von Frameworks und Tools zur Visualisierung von Wissen bzw. Informationen. Die im Rahmen dieser Masterarbeit näher beleuchteten Visualisierungssysteme wurden u.a. auf Basis folgender Kriterien ausgewählt:

- Möglichkeit zur Visualisierung größerer semantischer Graphen
- Verfügbarkeit (Open-Source, Lizenzen)
- Möglichkeit/Potenzial zur Erweiterung

Aufgrund der großen Zahl an verfügbaren Softwarelösungen ist es schwierig, eine adequate Auswahl zu treffen bzw. ein geeignetes System zu finden, welches spezifische Anforderungen erfüllt. Ziel dieser Evaluierung ist es, den derzeitigen Stand der Entwicklung auf dem Gebiet der Graph Visualisation zu eruieren und in Folge eine kompakte Auflistung geeigneter, aktuell verfügbare Toolkits und Frameworks zu erstellen. Im Zuge der Evaluierung werden allgemeine sowie spezifische Features ausgewählter Pakete betrachtet. Anschließend erfolgt eine Bewertung der relevanten Vor- und Nachteile der einzelnen Visualisierungssysteme. Im Hinblick auf die praktische Implementierung wird zudem das Potenzial für Erweiterungen bzw. Modifikationen analysiert. Der Fokus liegt dabei auf der Implementierung eines Systems zur interaktiven Visualisierung von semantischen Graphen im RDF-Format, mit der Möglichkeit, Graphelemente dynamisch zu aggregieren. Den Abschluss der Evaluierung bildet ein kurzes Fazit sowie eine Zusammenfassung der Ergebnisse.

Eine Auflistung inklusive Kurzbeschreibung weiterer Tools, Pakete und Frameworks mit dem Fokus semantische bzw. große Graphen findet sich auch unter [Bergman, 2009b], [Bergman, 2008], [Bader, 2009b] sowie [Beckett, 2005].

3.1 Themenschwerpunkte

Den Ausgangspunkt der Evaluierung bildet die Eingrenzung der betrachteten Aspekte bzw. die Definition der Themenschwerpunkte. Eine zentrale Rolle spielen dabei die Möglichkeit zur Visualisierung semantischer Datensätze sowie die Handhabung größerer Graphen. Ausgewählte Graphvisualisierungs-Pakete werden anhand dieser Punkte miteinander verglichen und in weiterer Folge deren Relevanz für die anschließende praktische Implementierung bewertet. Viele der verfügbaren Systeme unterstützen, zusätzlich zu elementaren Visualisierungsfunktionen, weitere Features wie etwa verschiedene Moeglichkeiten zur Interaktion mit der Graphdarstellung, Animation, 3D-Layout, Import/Export verschiedener

Formate, usw. Neben den grundlegenden Fähigkeiten zur visuellen Repräsentation einer Graphstruktur liegt das Hauptaugenmerk der Evaluierung u.a. auf folgenden Punkten:

- **Unterstützung semantischer Graphen**
  Möglichkeiten und Techniken hinsichtlich Import, Speicherung, Verarbeitung und Visualisierung semantischer Graphstrukturen bzw. die Berücksichtigung der mit einem Graph assoziierten semantischen Informationen. Der Fokus liegt dabei vor allem auf Graphen im RDF-Format.

- **Skalierbarkeit**
  Fähigkeit zur Behandlung großer Datenmengen sowie Methoden und Techniken zur Visualisierung von Graphen mit einer größeren Anzahl an Nodes und Edges. In diesem Zusammenhang spielt auch die Performance (Ladezeiten, Verzögerungen bei der Interaktion,...) eine wichtige Rolle. Neben der Unterstützung semantischer Graphen stellt Skalierbarkeit ein Schlüsselkriterium im Rahmen der Evaluierung dar.

- **Reduktion der visuellen Komplexität**
  Betrachtung von bereitgestellten Aggregierungs- bzw. Clustering-Features. Dazu zählt u.a. die Möglichkeit, bestimmte Teile des Graphen zusammenzufassen und durch ein einzelnes Icon zu ersetzten. In diesem Zusammenhang sind Cluster-Algorithmen zur Identifikation von Gruppen, Mustern, etc., aber auch Filtering-Techniken von Interesse.

- **Analyse des Graphen**
  Im Zuge von Aggregierung bzw. Clustering werden ebenfalls Methoden zur Untersuchung der Graphstruktur sowie Algorithmen zur Berechnung verschiedener Parameter bzw. graphtheoretischen Eigenschaften des Graphen (Degree, Edge Length,...) betrachtet. Die Möglichkeit zur Analyse des Graphen aufgrund semantischer Informationen stellt dabei einen weiteren zentralen Aspekt dar.

- **Interaktivität**
  Betrachtung der unterstützten Features zur Interaktion mit der Darstellung, wie etwa Navigation, Zooming, Editing, Filtering, grafische User-Interfaces (GUIs), Queries, usw.

- **Dynamische Aspekte**
  Betrachtung dynamischer Features, beispielsweise die Unterstützung dynamischer Layoutalgorithmen, der Umgang mit veränderlichen Datensätzen sowie das Handling von Strukturänderungen (animierte Übergänge zwischen Darstellungen,...).

- **Darstellungsformen, Layout Algorithmen**
  Betrachtung der implementierten Visualisierungstechniken bzw. der verfügbaren Darstellungsformen (z.B. Baumdarstellung, Maps, 2D-3D,...) sowie der verwendeten Algorithmen zur Berechnung eines Graphlayouts. Auch die Art und Weise, wie einzelne Nodes, Edges bzw. Labels dargestellt werden (Farbe, Form, Ausrichtung,...) ist in diesem Zusammenhang von Interesse.

- **Lizenzierung, Verfügbarkeit**
  Überprüfung unter welcher Lizenz die Software veröffentlicht wurde, respektive Betrachtung der Einzelheiten zur Verfügbarkeit (unterstützte Betriebssysteme, Source Code oder Binaries, kommerzielles oder nicht-kommerzielles Tool,...).

- **Technische Details**
  Nähere Betrachtung technischer Gesichtspunkte wie etwa:

  - Unterstützte Datenformate zum Import/Export von Graphstrukturen
  - Software-Plattformen (z.B. Java, C++, C,...)

  - Verwendete Rendering Technologien (z.B. Java2D, Java3D, JOGL, DirectX, OpenGL,...)
  - Requirements (zusätzliche Libraries, etc.)
  - Architektur, Aufbau,...

- **Weitere Aspekte**
  Zusätzlich zu den bisher erläuterten Punkten werden je nach Bedarf weitere allgemeine sowie spezifische Aspekte und Features betrachtet. Dazu zählen beispielsweise das Vorhandensein bzw. Umfang der Dokumentation, die Aktualität der Software (Datum der letzten Releases, Updates,...) oder die verschiedenen Einsatzgebiete.

Durch die Eingrenzung der betrachteten Aspekte soll eine bessere Übersicht bzw. eine bessere Vergleichbarkeit der diskutierten Software-Pakete erreicht werden. Unter anderem stellt die nachfolgende Evaluierung bestehender Visualisierungssysteme den Ausgangspunkt der praktischen Implementierung dar.

3.2 Evaluierung

In diesem Abschnitt erfolgt die nähere Betrachtung ausgewählter Graphvisualisierungs-Systeme unter besonderer Berücksichtigung der zuvor erläuterten Themenschwerpunkte. Ergänzend dazu sollen Screenshots von Beispielvisualisierungen einen tieferen Einblick in das jeweils betrachtete System gewaehren. Die hier getroffene Auswahl spiegelt nur einen kleinen Teil der enormen Zahl an verfügbaren Software-Lösungen wider.

3.2.1 Graphviz

Graphviz ist eine Sammlung von Libraries und Tools zur Darstellung, Betrachtung und Manipulation von Graphen. Grundsätzlich handelt es sich dabei um verschiedene Command-Line Anwendungen. Das Kernstück bilden sog. Layout-Engines, welche mittels Library-Interfaces, stream-basierten Kommandozeilen-Werkzeugen oder grafischen Benutzeroberflächen angesprochen werden. Editoren zur Manipulation des Graphen bzw. Viewer zur Betrachtung des Outputs werden über zusätzliche Packages bereitgestellt. [Graphviz, 2009a] [Ellson et al., 2003] [Prinz, 2006]

Graphviz ist eines der ältesten und am weitest verbreiteten Graph-Visualisierungssysteme. Obwohl der Kern aus den frühen 90iger Jahren stammt, wird die Software noch immer weiterentwickelt und in den verschiedensten Bereichen eingesetzt (z.B. zur Darstellung von Klassenhierarchien, biologischen Strukturen, etc.). Viele der derzeit verfügbaren Visualisierungssysteme nutzen Funktionalitäten von Graphviz bzw. verwenden Graphviz-Libraries (Zugriff mittels Graphviz-API; siehe dazu auch [Gansner, 2007]). Es existiert eine große Zahl an Dokumentationen, Online-Referenzseiten, User-Guides, Manuals, Source-Dokus, Tutorials, FAQs, Vorlagen usw. zu den verschiedenen Tools und Bibliotheken. Weiters finden sich diverse Artikel und Paper, welche sich mit Graphviz beschäftigen. [Ellson et al., 2003] [Gansner and North, 1999]

Unterstützung semantischer Graphen:
Es existiert keine direkte Unterstützung für semantische Graphen. RDF-Graphen müssten zuerst in das Graphviz Inputformat (DOT-Format) konvertiert werden. Eine etwas umständliche Möglichkeit besteht darin, das RDF-File zuerst mittels eines externen Programmes/Konverters in das GXL-Format (Graph Exchange Language) und anschließend mithilfe des Graphviz Kommandozeilen-Tools *gxl2gv* in das DOT-Format umzuwandeln. In weiterer Folge werden semantische Informationen, beispielsweise im Zuge des Layouts, jedoch nicht weiter berücksichtigt. Es gibt eine Reihe spezieller, auf Graphviz basierender, Visualisierungssysteme für RDF-Graphen. Diese sind in der Lage, Graphen im RDF-Format

direkt zu importieren (siehe zum Beispiel 3.2.2). [Ellson et al., 2003]

Skalierbarkeit:
Trotz der prinzipiellen Skalierbarkeit der Graphviz Tools, liegt das Hauptaugenmerk in erster Linie auf der ästhetisch ansprechenden Darstellung kleinerer Graphen ('modest-sized graphs'). Größere Graphen können somit hinsichtlich Input, Layout, bzw. Output zu Problemen führen. [Ellson et al., 2003]

Reduktion der visuellen Komplexität/Analysemöglichkeiten:
Das Graphviz Toolkit selbst implementiert keine Clustering- Analyse bzw. Aggregierungs-Features. Zu diesem Zweck müssten zusätzliche Viewer oder Editoren eingebunden werden.

Interaktivität:
Das Graphviz Toolkit selbst bietet keine Möglichkeit zur Interaktion mit der Darstellung. Erst durch die Verwendung von interaktiven Editoren/Viewern werden Features zur Navigation, Editierung und Analyse des Graphen bereitgestellt. Ein Beispiel dafür wäre der im Graphviz Download-Paket enthaltene Graph Editor bzw. Graph Viewer (*Dotty*). Dieser bietet grundlegende interaktive Features wie Zooming, Editierung, Suche, etc. (siehe dazu [Koutsoos and North, 1996]). Eine Reihe weiterer Editoren und Viewer findet sich unter [Graphviz, 2009b].

Dynamische Aspekte:
Der Fokus der in Graphviz implementierten Algorithmen liegt auf statischen Layouts. *Dynagraph*, als Erweiterung von Graphviz, bietet jedoch Algorithmen (z.B. eine inkrementelle Version des Sugiyama-Algorithmus) und interaktive Programme für inkrementelles, dynamisches Layout. Dadurch können Graphen, welche sich im Zeitablauf ändern, dargestellt werden. [Dynagraph, 2009] [Ellson et al., 2003]

Darstellungsformen, Layout Algorithmen:
Grundsätzlich bietet Graphviz neben generellen Algorithmen (beispielsweise zur Reduktion von Ueberlappungen) einige allgemein gebräuchliche Layout-Techniken im zweidimensionalen Bereich. Dazu zählen u.a. *hierarchical Layouts* gerichteter Graphen (Algorithmus basierend auf Sugiyama), *spring-based symmetric Layouts* größerer bzw. geclusterter Graphen (Algorithmus nach Kamada Kawai bzw. Fruchterman Reingold) sowie *radial* bzw. *circular Layouts*. Zusätzlich gibt es noch die Möglichkeit, grafische Attribute (Farben, Formen, Schriftarten, Linientypen, etc.) von Nodes, Edges und Label näher zu spezifizieren (siehe auch Abbildung 3.1). [Graphviz, 2009a] [Gansner, 2007]

Lizenzierung, Verfügbarkeit:
Graphviz ist auf Open Source Basis (*Common Public License (CPL)*) lizenziert. Sowohl Source Code (stable bzw. development) als auch executable Packages für Linux, Windows und Mac sind frei verfügbar. [Graphviz, 2009c]

Technische Details:
Datenformate: Graphen werden mithilfe einer textbasierten Sprache (*DOT Language*) beschrieben. Dateien dieses Formats dienen als Input für die jeweilige Layout-Engine. Es besteht die Möglichkeit zur Konvertierung anderer Formate in das DOT-Format (z.B von GXL (einem XML-Dialekt zur Repraesentation von Graphen als Text) nach DOT). Die von Graphviz direkt unterstützten Output-Formate sind u.a. *SVG* (Scalable Vector Graphics), *Postscript*, diverse Rastergrafiken (*GIF, JPEG, PNG*) bzw. auch *VRML* (Virtual Reality Modeling Language) sowie verschiedene textuelle Formate (*GXL, plain, plain-ext*. [Gansner and North, 1999] [Gansner, 2007]

Architektur/Komponenten: Graphviz weist eine schichtenbasierte Architektur auf. Den Kern bildet eine Sammlung von Libraries zur Bereitstellung des grundlegenden Graphmodells. Darauf aufbauend gibt

es eine Schicht, in welcher Layout-Algorithmen sowie Input- und Output-Routinen gekapselt sind. Die oberste Schicht bilden diverse Applikationen zur Betrachtung und Editierung des Graphen. Das gesamte Graphviz-Toolkit besteht somit aus folgenden Komponenten ([Gansner and North, 1999] [Ellson et al., 2003]):

- Libraries (*Libgraph* als Basisbibliothek für Lese- und Schreib-Operationen bzw. zur Manipulation des abstrakten Graphmodells; *Dynagraph* als Erweiterung zur Darstellung von dynamischen Graphen)
- Tools bzw. Engines zur Berechnung des Layouts (z.B. *Dot, Neato,...*)
- Graphische Tools zur Betrachtung bzw. Editierung (z.B. *Dotty, Webdot, Grappa,...*)
- Graph-Filters zum Lesen bzw. zur Verarbeitung von Input-Streams sowie zur Erzeugung von Output-Streams (z.B. *Gpr*).

Software-Plattform: Die Grundkomponenten von Graphviz wurden in C implementiert. Die Graphviz API eignet sich somit hauptsächlich zur Verwendung in C bzw. C++ Programmen. Zudem gibt es eine Reihe von Language Bindings zur Verwendung von Graphviz in Verbindung mit anderen Sprachen bzw. Systemen (z.B. Java, C#, Python,...). [Graphviz, 2009b] [Prinz, 2006]

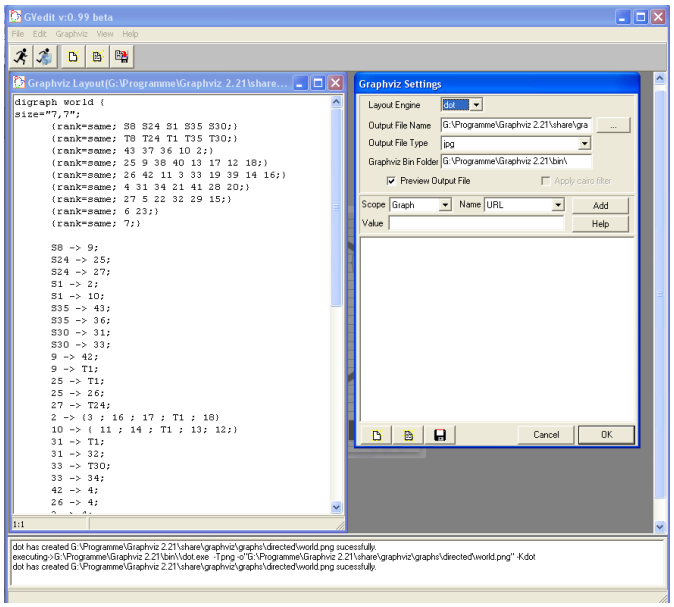
Fazit:
Graphviz bietet den Vorteil einer umfangreichen und guten Dokumentation. Aktuelle Weiterentwicklungen, eine Vielzahl an verfügbaren Tools auf Graphviz-Basis (siehe dazu [Graphviz, 2009b]) sowie eine Open Source Lizenz sind weitere positive Aspekte. Zudem kann eine Erweiterung bzw. die gesamte Implementierung eines umfangreichen Visualisierungssystems auf Grundlage der Graphviz-Bibliothek (Zugriff via API) erfolgen.

Graphviz selbst ist in erster Linie ein Toolkit zur statischen Darstellung von Graphen (Graph Drawing) und weniger ein System zur interaktiven Visualisierung. Graphdefinitionen werden als Textfiles eingelesen und unter Berücksichtigung der gewählten Layout-Engine als 'Bild' ausgegeben. Moeglichkeiten zur Interaktion bieten sich nur in Verbindung mit zusätzlichen Graph Editoren/Viewern. Darüber hinaus kann die Konvertierung der Datenformate (beispielsweise von RDF nach DOT) mitunter sehr aufwendig sein. Eingeschränkte Möglichkeiten zur Darstellung großer Graphen sowie die fehlende Einbindung semantischer Informationen sind weitere Argumente die gegen die Verwendung von Graphviz im Rahmen der praktischen Implementierung sprechen.

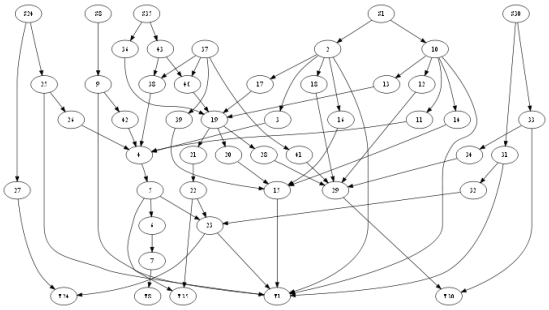
3.2.2 IsaViz

IsaViz ist eine visuelle Umgebung speziell zur Darstellung und Bearbeitung von RDF-Graphen. Das u.a. auf der Graphviz Bibliothek aufbauende, frei verfügbare Java-Tool bietet eine intuitive grafische Benutzeroberfläche (bestehend aus mehreren separaten Fenstern) sowie Features zur Behandlung RDF-spezifischer Aspekte. Das erste Release stammt aus dem Jahr 2002, die letzte Development Version wurde im Mai 2007 zum Download freigegeben. Neben einer Auflistung bekannter Probleme bzw. Adressen diverser Mailing-Lists finden sich auf der offiziellen Website auch Installationsanweisungen sowie gut dokumentierte User-Manuals. [Pietriga, 2007]

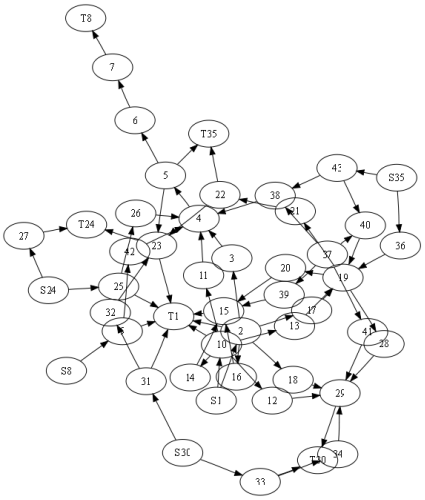
Unterstützung semantischer Graphen:
RDF-Modelle in den Formaten *RDF/XML, N3* bzw. *N-Triples* können direkt importiert werden. RDF-Resources werden dabei als Nodes, Properties als Edges dargestellt. Zudem erfolgt ein Labeling von Knoten und Kanten. In weiterer Folge bietet Graphviz umfangreiche Möglichkeiten und Features speziell zur Darstellung und Bearbeitung von RDF-Modellen. Die visuelle Repräsentation des RDF-Modells



(a) Graphviz Editor



(b) Hierarchical Layout



(c) Spring-based Layout

Abbildung 3.1: Graphviz Toolkit ([Graphviz, 2009a]) + unterschiedliche Layouts eines kleinen (ca. 50 Nodes) gerichteten Beispielgraphen. (a) Oberfläche des Basiseditors. Links ist die textuelle Graphdefinition (DOT-Format) zu sehen. Auf der rechten Seite befindet sich das Fenster zur Wahl des Output-Formats bzw. der Layout-Engine. (b) Hierarchical Layout (*Dot Layout-Engine*) (c) Spring-based Kamada Kawai Layout des selben Datensatzes (*Neato Layout-Engine*)

wird durch eine übersichtliche textuelle Aufbereitung der semantischen Informationen (Auflistung von Statements, Property-Types, Resources, URIs,...) ergänzt. [Pietriga, 2005a]

Skalierbarkeit:
IsaViz eignet sich in erster Linie zur Visualisierung kleinerer Graphen. Die implementierten Layout-Algorithmen liefern für Graphen ab ca. 100 Nodes unübersichtliche Ergebnisse. Siehe dazu auch die Visualisierung des Beispielgraphen aus Abbildung 3.2 (825 Nodes bzw. 1126 Edges). [Frasincar et al., 2005] [Sayers, 2004]

Interaktivität:
IsaViz implementiert verschiedene interaktive Features. Dazu zählen u.a. Geometrical Zooming, Panning, Overview, Selektion oder textbasierte Suche. Zudem besteht die Möglichkeit zur Editierung/Manipulation sowohl des visualisierten Graphen als auch des RDF-Modells (Kopieren, Einfügen, Löschen, Hinzufügen, Verschieben von Elementen). [Pietriga, 2005a]

Reduktion der visuellen Komplexität/Analysemöglichkeiten:
Die Einbindung von *Graph Stylesheets (GSS)* ermöglicht u.a. das Aus-/Einblenden, das Gruppieren, bzw. die separate Darstellung ausgewählter Nodes und Edges. Dabei werden vor allem semantische bzw. RDF-spezifische Aspekte berücksichtigt. (GSS ist eine RDF-basierte Erweiterung von CSS (Cascading Stylesheets). IsaViz inkludiert ein grafisches Front-End zur Editierung der Stylesheets.)

Darüber hinaus existieren jedoch keine Möglichkeiten zur Reduktion der visuellen Komplexität (wie beispielsweise Aggregierung, Clustering, Level-Of-Detail,...) bzw. keine Features zur Berechnung von Graph- bzw. Netzwerkeigenschaften. [Pietriga, 2004a]

Dynamische Aspekte:
Das Layout des Graphen erfolgt nur nach dem Einlesen bzw. auf dezidierten Wunsch des Users. Somit eignet sich IsaViz nur bedingt zur Visualisierung von dynamischen Graphstrukturen.

Darstellungsformen, Layout-Algorithmen:
Das Layout gerichteter Graphen wird mithilfe der Graphviz Library erstellt (zweidimensionale Node-Link Repräsentation; siehe Abschnitt 3.2.1). Durch Einbindung der jeweiligen *Layout-Engine* (via Angabe des Pfades) könne somit hierarchische, symmetrische, radiale oder kreisförmige Darstellungen generiert werden. Zusätzlich besteht die Möglichkeit, mithilfe von Stylesheets (GSS) den Graphelementen verschiedene visuelle Attribute (z.B. Farbe, Linienstärke, Icons, etc.) zuzuordnen. Das Aussehen von Nodes und Edges kann auch direkt via Editierung verändert werden. [Pietriga, 2005a] [Pietriga, 2004a]

Lizenzierung, Verfügbarkeit:
IsaViz ist eine Open Source Software (*W3C Software License*). Plattformunabhängige Binaries sowie Java Source Code (development bzw. stable Versionen) sind frei verfügbar. [Pietriga, 2007]

Technische Details:
Datenformate: RDF-Modelle können in den Formaten *RDF/XML* (*RDFS, OWL*), *Notation3 (N3)* und *N-Triples* importiert bzw. auch exportiert werden. Zusätzlich besteht die Möglichkeit, die aktuelle Ansicht als *PNG* bzw. den gesamten Graphen als *SVG* zu exportieren. Ein optionales *Sesame*-Plugin ermöglicht den direkten Zugriff auf serverseitig gespeicherte RDF-Daten (Sesame ist ein Java Framework zur Speicherung und Abfrage von RDF-Daten; siehe [Aduna, 2008]). [Pietriga, 2005a] [Pietriga, 2007]

Software-Plattform/Zusatzpakete: IsaViz wurde in Java implementiert und ist plattformunabhängig. Neben *JVM*(Java Virtual Machine) werden noch einige zusätzliche Libraries/Packages benötigt. Das *Jena Semantic Web Toolkit* ermöglicht den Import/Export bzw. die Manipulation von RDF-Modellen (siehe

[Jena, 2009a]). Die *Graphviz* Library dient dem Layout des eingelesenen Graphen. Das GUI von IsaViz basiert auf dem *ZVTM*-Toolkit (Zoomable Visual Transformation Machine) bzw auf *Java2D*. Weiters wird ein *Apache Xerces* XML-Parser verwendet. Alle benötigten Zusatzbibliotheken (mit Ausnahme von Graphviz) sind im Downloadpaket bereits inkludiert. [Pietriga, 2004b]

Fazit:
Der Fokus von IsaViz liegt in erster Linie auf speziellen Features zur Behandlung von RDF-Modellen. Der direkte Import von Dateien im RDF-Format, die Einbindung semantischer Informationen in die Graphdarstellung sowie umfangreiche textbasierte Analysemöglichkeiten von RDF-Daten stellen die größten Vorteile des Tools dar. Weitere positive Aspekte wären der frei verfügbare Java Source Code und die intuitive grafische Benutzeroberfläche (GUI).

Demgegenüber stehen die eingeschränkte Skalierbarkeit (Verwendung von Graphviz Layout- Algorithmen) sowie fehlende Möglichkeiten zur Reduktion der visuellen Komplexität. Im Hinblick auf eine mögliche Erweiterung des Tools hätte somit die Implementierung von Aggregations-/Clustering-Features bzw. die Verbesserung der grafischen Darstellungsmöglichkeiten oberste Priorität.

3.2.3 Tulip

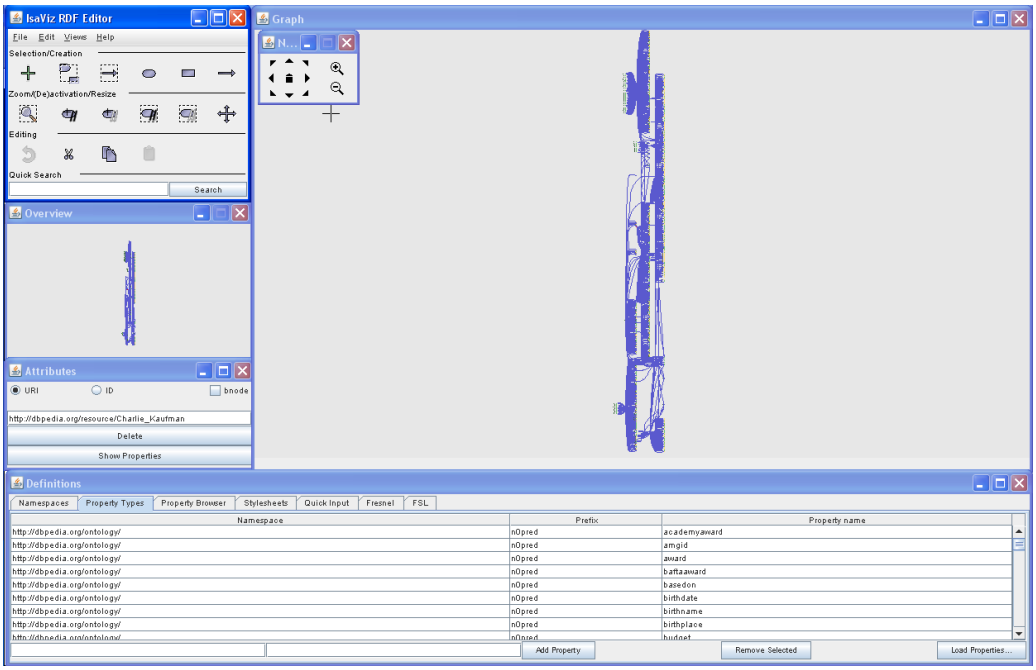
Tulip ist ein umfangreiches Framework zur Visualisierung großer Graphstrukturen. Spezielle Features sind u.a. ein Plugin Support für Erweiterungen, 3D-Visualisierungen, diverse Interaktionsmöglichkeiten sowie Methoden zur Behandlung großer Graphen (Clustering, etc.) [Tulip, 2007b]. Tulip bietet eine umfangreiche Dokumentation sowohl für Anwender (User Manual) als auch für Entwickler (Developer Handbook, Libraries-Doku, Tutorials). Die Software wird aktuell weiterentwickelt - das letzte Release wurde 2009 zum Download bereitgestellt [Sourceforge, 2009]). Anwendungsgebiete wären u.a. die Visualisierung von Molekülen, RNA-Strukturen, sozialen Netzwerken, Dateihierarchien oder 3D-Meshes [Tulip, 2007c]. Abbildung 3.3 zeigt die grafische Oberfläche von Tulip sowie die Darstellung eines Beispielgraphen.

Unterstützung semantischer Graphen:
Es existiert in Tulip keine Möglichkeit, RDF-Modelle direkt zu importieren. Somit müsste beispielsweise erst eine Konvertierung der RDF-Datei in das *DOT*- bzw. das *GML*-Format (siehe Unterpunkt *Datenformate*) vorgenommen werden. Infolgedessen finden sich auch keinerlei Features zur Behandlung semantischer bzw. RDF-spezifischer Informationen.

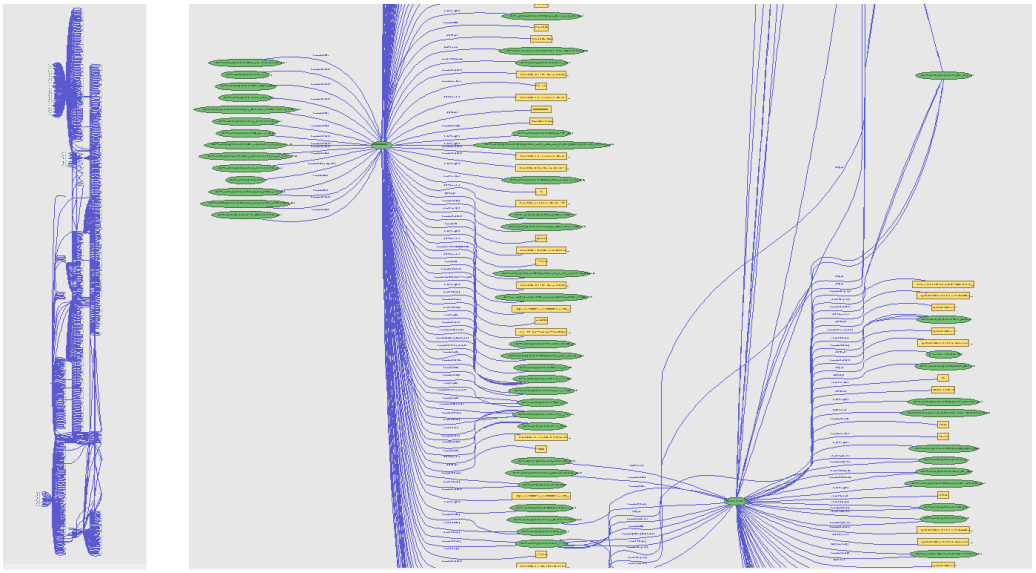
Skalierbarkeit:
Das Framework wurde speziell zur Behandlung von großen Graphen entwickelt. Somit können sowohl kleine Graphen, als auch Graphen mit bis zu 10^6 Elementen (lt. Angabe der Tulip Entwickler) auf einem Standard-PC visualisiert werden. [Tulip, 2007b] [Bergman, 2009b] [Prinz, 2006]

Abbildung 3.3 zeigt die Darstellung eines Beispielgraphen bestehend aus 30000 Nodes und 20000 Edges (Force-Directed Layout).

Interaktivität:
Tulip stellt eine breite Palette an Interaktions-Features zur Verfügung. Beispiele dafür wären u.a. die Navigation mittels geometrischer Operationen, Hinzufügen, Entfernen, Kopieren, Einfügen und Verschieben von Nodes/Edges, 3D-Overview, Suche bzw. spezielle Algorithmen zur Modifikation des Graphen (Spanning Tree,...). Zudem implementiert Tulip neben standardmäßigen Selektionsmöglichkeiten auch noch spezielle Algorithmen, welche Nodes bzw. Edges aufgrund von Grapheigenschaften markieren (z.B. Loop-Selection, Reachable Sub-Graph, etc.). Details bzw. weitere Funktionalitäten finden sich im Tulip User Manual [Tulip, 2007d]



(a) IsaViz GUI



(b) Graphdarstellung

Abbildung 3.2: IsaViz ([Pietriga, 2007])) + Visualisierung eines RDF-Graphen aus [DBpedia, 2009]. (a) IsaViz GUI. Im Zentrum befindet sich die Graphdarstellung. Die weiteren Fenster beinhalten die textuelle Repräsentation der RDF-Daten (Auflistung von Statements, Namespaces, Properties,...) bzw. Features zur Interaktion (Overview, Zoom, Edit,...). (b) Visualisierung eines RDF-Graphen bestehend aus 825 Nodes bzw. 1126 Edges (Übersichts- bzw. Detaildarstellung eines gezoomten Bereichs). Das hierarchische Layout stellt in diesem Fall eine suboptimale Lösung dar.

Reduktion der visuellen Komplexität:
Das Framework bietet einige Features um die Komplexität der dargestellten Graphen zu reduzieren. Dazu zählen u.a. die Extraktion von Subgraphen, Filtering oder diverse Clustering-Techniken (z.B. Convolution-, Hierarchical-, Quotient-, oder Strength-Clustering; für Details dazu siehe [Tulip, 2007d]). Das Source Code Package von Tulip umfasst zudem die Möglichkeit, Clustering auf Basis zuvor berechneter Statistiken durchzuführen. [Tulip, 2007b] [Tulip, 2007d]

Analyse des Graphen:
Tulip ermöglicht die Berechnung von diversen Grapheigenschaften bzw. Metrics (z.B. Degree, Clustering Coefficient, Strahler Metrics, Path-Length, Edge-Strength,...). Darüber hinaus besteht die Möglichkeit, den Graphen auf spezifische Merkmale hin zu überprüfen (beispielsweise ob der Graph planar, azyklisch oder gerichtet ist). Für weitere Details hinsichtlich der implementierten Analysealgorithmen siehe [Tulip, 2007d].

Dynamische Aspekte:
Das Layout erfolgt im Anschluss an das Einlesen des Graphen bzw. nur nachdem der User einen neuen Layout-Vorgang initialisiert. Die Visualisierung von Graphstrukturen, welche sich im Zeitablauf kontinuierlich ändern, wird von Tulip nicht unterstützt.

Darstellungsformen, Layout Algorithmen:
Standardmäßig stellt Tulip eine Reihe von Layout-Algorithmen (2D/3D) zur Verfügung. Diese können in folgende sechs Kategorien unterteilt werden:

- Planar (3-Connected, Mixed Model)
- Tree (z.B. Bubble Tree, Cone Tree, Tree Map, Radial Tree,...)
- Basic (z.B. Circular, Random)
- Misc (z.B. Scatterplot,...)
- Force Directed (HDE, GEM)
- Hierarchical

Zudem besteht die Möglichkeit, die Farbe bzw. Größe von Nodes und Edges mithilfe von Color-Mapping bzw. Auto-Sizing Algorithmen automatisch zu berechnen und den Elementen zuzuweisen. Größe, Farbe, Aussehen, etc. von Nodes und Edges können vom User auch direkt geändert werden. [Tulip, 2007d]

Lizenzierung, Verfügbarkeit:
Tulip ist unter der *GNU General Public License (GPL)* veröffentlicht. Sowohl Source Code als auch Binaries für Windows und Linux sind frei verfügbar. [Sourceforge, 2009]

Technische Details:
Datenformate: Graphen können in den Formaten *GML* (Graph Modelling Language - eine textbasierte Sprache zur hierarchischen Beschreibung von Graphen) und *DOT* (siehe 3.2.1) bzw. in Form einer *Adjacency Matrix* importiert werden. Der visualisierte Graph kann wiederum als GML-Datei exportiert bzw. als Bild (Bitmap, JPEG, PNG,...) abgespeichert werden. [Tulip, 2007d]

Plugin-Support: Der modulare Aufbau des Frameworks ermöglicht die Einbindung von Erweiterungen ohne Recompilation der gesamten Software. Mithilfe eines grafischer Editors können fertige Plugins hinzufügt oder entfernt werden. [Tulip, 2007a] [Tulip, 2007d] [Prinz, 2006]

Software-Plattform/Libraries: Das Framework wurde in C++ unter Verwendung der *Standard Template Library* implementiert. Das zwei- bzw. dreidimensionale Rendering der Graphen wird mittels *OpenGL* realisiert. OpenGL ermöglicht zudem die Verwendung von Texturen und 3D-Objekten [Tulip, 2007a] bzw. erlaubt effizientes, schnelles Rendering. Die Benutzeroberfläche basiert auf der *Qt Library*. Durch die Verwendung von Qt bzw. OpenGL wird Plattformunabhängigkeit gewährleistet. [Tulip, 2007a] [Prinz, 2006]

Fazit:
Tulip ist ein äußerst umfangreiches Framework welches sich besonders zur Visualisierung von größeren Graphstrukturen eignet. Eine Vielzahl an unterschiedlichen Layout-Algorithmen (2D/3D), umfangreiche Interaktions- und Analysemöglichkeiten sowie einige bereits implementierte Clustering-Features machen Tulip zu einem mächtigen Visualisierungssystem. Im Vergleich zu anderen Tools bietet Tulip somit einen enorm großen Funktionsumfang. Weitere Vorteile sind die Einbindung von OpenGL (schnelles, effizientes Rendering), die Möglichkeit zur einfachen Erweiterung via Plugins sowie die Aktualität der Software.

Das Framework bietet jedoch keine Funktionen zum Import bzw. zur Behandlung/Analyse von RDF-Modellen. Ein Plugin zum Import von Dateien im RDF-Format wäre somit eine sinnvolle Erweiterung des Frameworks. Zusätzlich könnten weitere Features zur Berücksichtigung von semantischen Informationen im Zuge des Layouts bzw. zur Analyse von RDF-Daten hinzugefügt werden. Hinsichtlich der praktischen Implementierung im Rahmen dieser Masterarbeit wäre jedoch eine Lizenzierung der Software in Form der *GNU Lesser General Public License (LGPL)* anstelle der GPL wünschenswert.

3.2.4 Welkin

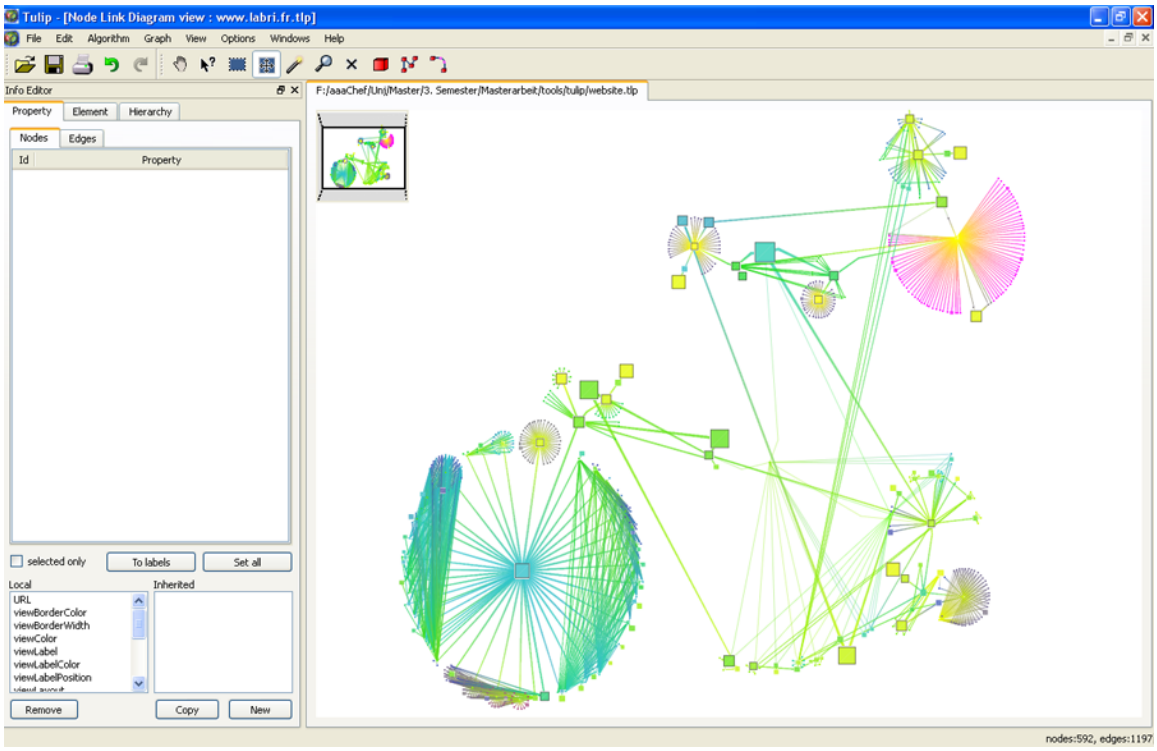
Welkin ist ein graph-basiertes Lightweight-Visualisierungstool für RDF-Modelle. Es soll dem User in erster Linie einen besseren Überblick über die Daten bzw. die zugrundeliegende Ontologie verschaffen. Details (beispielsweise die Betrachtung einzelner Statements) werden dabei bewusst vermieden. Das Haupteinsatzgebiet wäre die Daten- bzw. Metadatenanalyse. Der Fokus von Welkin liegt auf folgenden Punkten:

- Schnelle und unkomplizierte Visualisierung von RDF-Modellen
- Identifikation der globalen Struktur eines RDF-Datensatzes
- Vermitteln eines mentalen Modells der Daten bzw. der Ontologien
- Erkennen von Bereichen mit speziellen graphtheoretischen Eigenschaften

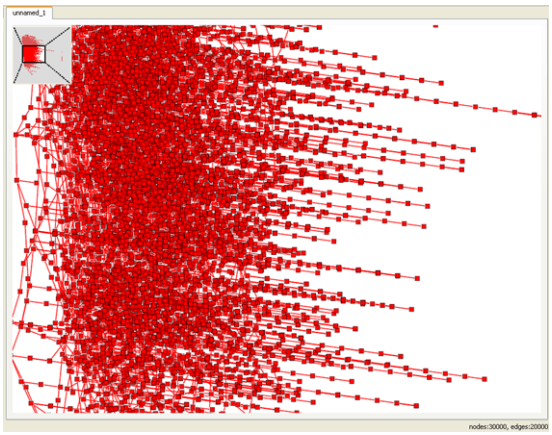
Das Tool bietet eine überschaubare Zahl an Features sowie ein intuitives User Interfaces (siehe Abbildung 3.4). [Mazzocchi and Ciccarese, 2008a] [Mazzocchi and Ciccarese, 2008b]

Unterstützung semantischer Graphen:
Welkin unterstützt eine Reihe von Features zur Behandlung semantischer Graphen im RDF-Format, darunter:

- Import von RDF-Files
- Visualisierung der RDF-Modelle als gerichteter Graph



(a) Tulip GUI



(b) Graphdarstellung

Abbildung 3.3: Tulip ([Tulip, 2007b])) (a) Grafische Oberfläche von Tulip. Der dargestellte Graph besteht aus 592 Nodes und 1197 Edges (Bubble Tree Layout). (b) Force-Directed Layout eines größeren Beispielgraphen (30000 Nodes bzw. 20000 Edges)

- Auflistung/textuelle Darstellung von RDF-spezifischen Aspekten (Ontologie, Predicates, Resources, etc.)
- Analyse bzw. Filtering aufgrund semantischer Informationen

Skalierbarkeit:
Die Software eignet sich in erster Linie zur Visualisierung von kleinen bis mittelgroßen Graphen. Ab ca. 1000 Nodes (bzw. ab einer Dateigröße von mehr als 1MB) ist folglich mit Performanceeinbußen zu rechnen. Laut Projekt-Homepage sollen neuere Versionen der Software die effiziente Behandlung größerer Graphen ermöglichen. [Mazzocchi and Ciccarese, 2008a]

Einlesen bzw. initiales Layout größerer Datensätze ist jedoch ohne Probleme bzw. innerhalb kürzester Zeit möglich. Abbildung 3.5 zeigt die Visualisierung eines Graphen mit ca. 13500 Nodes und 14000 Edges (Dateigröße 6MB).

Interaktivität:
Welkin implementiert einige Features zur Interaktion u.a. eine textbasierte Suche (+ Highlighting der Ergebnisse), Filtering oder ein Fish-Eye View. Darüber hinaus bietet das Tool jedoch keine gängigen Interaktionsmöglichkeiten (wie z.B. Zooming- oder Panning). Weiters können nur einzelne Nodes selektiert bzw. verschoben, nicht jedoch gelöscht, kopiert oder eingefügt werden. [Mazzocchi and Ciccarese, 2008b]

Reduktion der visuellen Komplexität:
Es besteht die Möglichkeit, Teile der Graphdarstellung ein- bzw. auszublenden. So können etwa die gesamten Nodes und/oder Edges (bzw. deren Pfeilspitzen) ein- oder ausgeblendet werden.

Welkin implementiert zudem einige Filter-Funktionalitäten. Gruppen von Nodes und Edges können aufgrund RDF-spezifischer Informationen aus der Visualisierung entfernt bzw. wieder hinzugefügt werden. So besteht u.a. die Möglichkeit, jene Nodes, welche mit einem bestimmten Predicate oder einer bestimmten Resource verbunden sind, zu filtern. Weiters kann auch auf Basis graphtheoretischer Aspekte (Degree, Clustering Coefficient) gefiltert werden. [Mazzocchi and Ciccarese, 2008b]

Abbildung 3.5 zeigt die Visualisierung eines größeren Graphen vor bzw. nach Anwendung einer Filter-Operation.

Analyse des Graphen:
Die Software umfasst sowohl Funktionen zur semantischen Analyse der RDF-Daten als auch zur Berechnung und Darstellung von graphtheoretischen Eigenschaften (In-Degree, Out-Degree, Clustering Coefficient). [Mazzocchi and Ciccarese, 2008b]

Dynamische Aspekte:
Welkin unterstützt, im Gegensatz zu den bisher betrachteten Systemen, animierte Übergänge. Der User hat die Möglichkeit, verschiedene Parameter (Attraction, Repulsion, Delay,...) zu justieren bzw. den Animationsmodus ein- oder auszuschalten. Das Layout (force-directed) wird in Echtzeit angepasst. User-Interaktionen (z.B. das Verschieben eines Nodes) haben unmittelbare Auswirkungen auf die Animation. Die Performance verschlechtert sich je nach Größe des Datensatzes.

Darstellungsformen, Layout Algorithmen:
Nach dem Import des RDF-Modells werden Subjects und Objects als Punkte bzw. Predicates als dünne Linien dargestellt und per Zufall in der Ebene platziert. In weiterer Folge besteht die Möglichkeit folgende Layouts auf den Graphen anzuwenden:

- Circle (radiale Darstellung)

- Scramble (zufällige Anordnung der Nodes)
- Shake (leichte Änderung der Node-Positionen)

Zusätzlich wird im Animationsmodus ein force-directed Layout verwendet. [Mazzocchi and Ciccarese, 2008b]

Lizenzierung, Verfügbarkeit:
Welkin ist Open Source (*BSD - Berkeley Software Distribution License*) und kann in einer Prepackaged Version von der Projekt-Website heruntergeladen werden. Der aktuelle Development Snapshot ist via SVN verfügbar (siehe [Mazzocchi and Ciccarese, 2008a]). Das Tool kann auch als Applet mittels Java WebStart direkt im Web-Browser gestartet werden (siehe [Mazzocchi and Ciccarese, 2008c]).

Technische Details:
Datenformate: Welkin ist ein reiner Viewer und unterstützt somit keine Export-Funktionen. Datenformate für den Import wären *RDF/XML (RDFS, OWL)* bzw. *TURTLE/N3*. [Mazzocchi and Ciccarese, 2008a]

Software-Plattform/Libraries: Welkin wurde vollständig in Java implementiert und ist plattformunabhaengig. Die *Jena RDF API* ([Jena, 2009a]) ermöglicht den Import/Export bzw. die Manipulation von RDF-Modellen. Weiters wird der *Apache Xerces* XML-Parser verwendet. Alle benötigten Libraries sind als JAR-Files im Download-Package enthalten.

Fazit:
Welkin ist ein übersichtliches und schlankes Tool, welches speziell zur Behandlung von RDF-Daten einige nützliche Features bereitstellt. Die Funktionalität ist auf das Wesentliche - der schnellen und überblicksmäßigen Visualisierung von mittelgroßen RDF-Graphen - beschränkt. Obwohl keine Dokumentation des Source Codes verfügbar ist, würde die Einarbeitungszeit (hinsichtlich der Implementierung möglicher Erweiterungen) aufgrund der überschaubaren Anzahl an Klassen im Vergleich zu anderen, umfangreicheren Systemen (wie etwa Tulip) relativ kurz sein. Die größten Nachteile von Welkin sind die eingeschränkten Interaktionsmöglichkeiten, eine fehlende Export-Funktion, sowie die schlechte Performance bei größeren Graphen (hinsichtlich Übersichtlichkeit, Interaktion, Animation,...). Auch im Bezug auf die visuelle Darstellung weist das Tool einige Defizite auf (kein Labeling von Nodes oder Edges, eingeschränkte Layout-Möglichkeiten,...). Nichtsdestotrotz stellt Welkin ein interessantes Visualisierungssystem mit einigem Erweiterungspotenzial dar.

3.2.5 RDF Gravity

RDF Gravity ist ein Tool zur Visualisierung von gerichteten Graphen im RDF- bzw. OWL-Format. Es implementiert eine Reihe von Features zur Interaktion sowie zur Analyse von RDF-Daten. Weiters können Graphstrukturen, welche sich in verschiedenen RDF-Files befinden, in einer Ansicht visualisiert werden. Abbildung 3.6 zeigt die Oberfläche von RDF Gravity sowie die Darstellung eines kleinen RDF-Graphen. [Goyal and Westenthaler, 2009a]

Unterstützung semantischer Graphen:
Die Software ermöglicht das Einlesen von RDF-Modellen und die Visualisierung in Form eines gerichteten Graphen. Weitere Features wären die Berücksichtigung der RDF-Syntax bei der Darstellung von Nodes und Edges, sowie Filtering bzw. Formulierung von Queries auf Basis RDF-spezifischer Informationen. [Goyal and Westenthaler, 2009a]

Skalierbarkeit:
Bei der Visualisierung von Graphen ab ca. 150 Nodes kommt es aufgrund von Labels sehr schnell zu

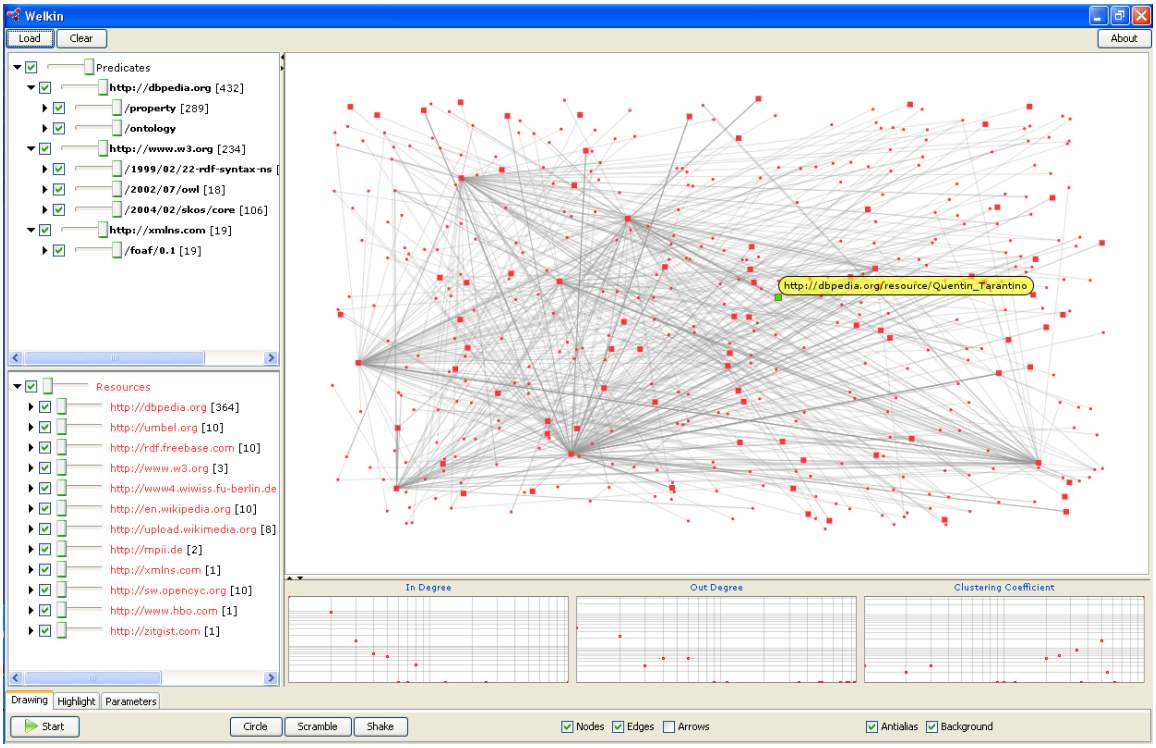


Abbildung 3.4: Oberfläche von Welkin ([Mazzocchi and Ciccarese, 2008a]). Die beiden linken Fenster beinhalten die textuelle Darstellung RDF-spezifischer Informationen (Predicates, Resources). Im Zentrum befindet sich die Graph-Visualisierung. Dabei handelt es sich um den selben RDF-Datensatz (825 Nodes, 1126 Edges; aus [DBpedia, 2009]), welcher auch schon mittels IsaViz (Unterpunkt 3.2.2) visualisiert wurde. Bei Klick auf einzelne Nodes werden Details angezeigt (gelbes Oval).

Überlagerungen (siehe Abbildung 3.7). Größere Graphen führen weiters zu einer schlechteren Performance hinsichtlich der Animation des Layouts bzw. der verschiedenen Interaktionsmöglichkeiten. Es empfiehlt sich daher, nach dem Einlesen eines großen Graphen, einen Filter anzuwenden bzw. eine Query auszuführen um so eine reduzierte Version des visualisierten Graphen zu generieren.

Interaktivität:
RDF Gravity implementiert eine Reihe von Interaktions-Mechanismen, darunter Geometrical Zoom, Multiple Node- bzw. Edge-Selection, Mouse-Over Events, textbasierte Suche, Formulierung von Queries sowie Filtering. Nodes und Edges können verschoben, nicht jedoch hinzugefügt oder entfernt werden, d.h. es gibt keine Möglichkeit zur Manipulation der grundlegenden Graphstruktur. [Goyal and Westenthaler, 2009a]

Reduktion der visuellen Komplexität/Analyse:
Nodes/Edges können einzeln bzw. via Filter-Operationen ein- oder ausgeblendet werden. Globale bzw. lokale Filter ermöglichen das Ausblenden jener Edges bzw. Nodes, welche eine bestimmte RDF-Relation repräsentieren (beispielsweise Ausblenden aller Relationen vom Typ 'subClassOf'). Zudem können mittels Queries Sub-Graphen extrahiert werden. [Goyal and Westenthaler, 2009a]

Dynamische Aspekte:
Das Layout wird nur nach einer, vom User veranlassten, Neuinitialisierung angepasst. Dies erfolgt mithilfe einer Animation.

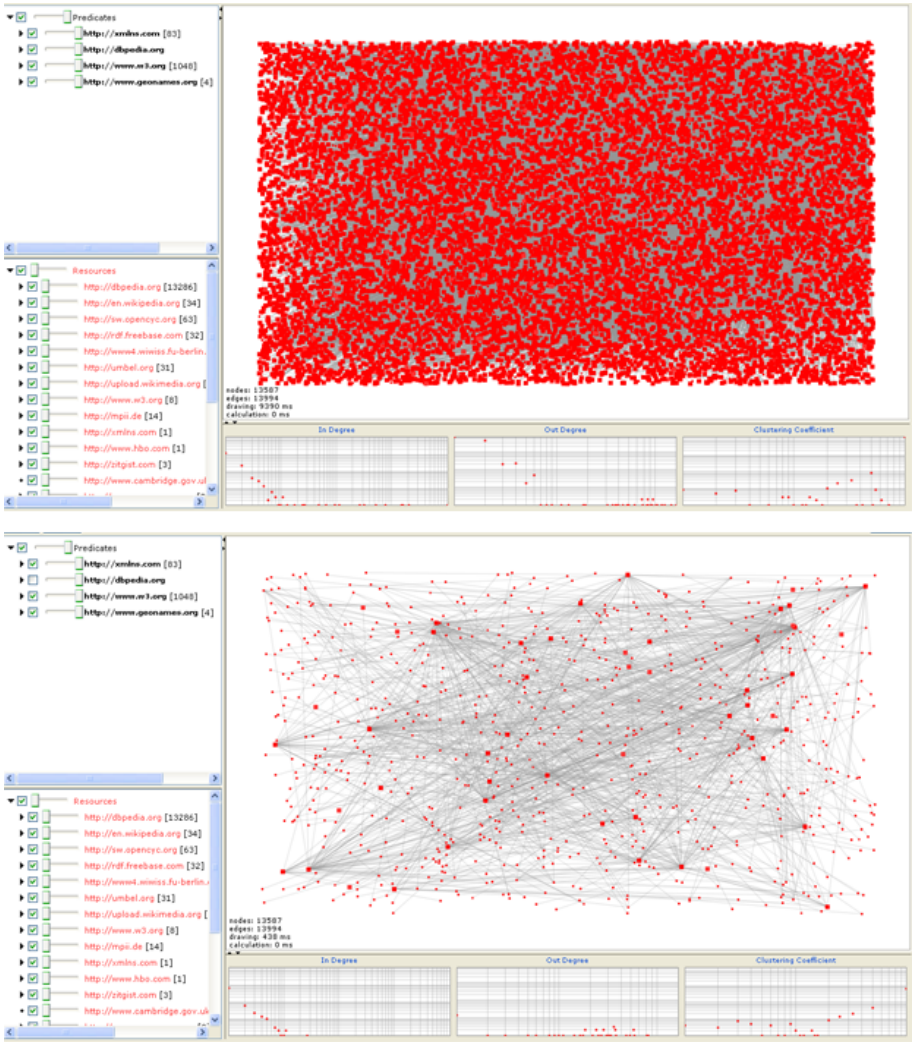


Abbildung 3.5: Visualisierung eines größeren Graphen (ca. 13500 Nodes, 14000 Edges; Quelle: [DBpedia, 2009]) mithilfe von Welkin ([Mazzocchi and Ciccarese, 2008a]). Die beiden Bilder zeigen den Graphen vor bzw. nach dem Filter-Prozess. Im Zuge des Filter-Prozesses wurden einzelne Predicate-Gruppen ausgeblendet.

Darstellungsformen, Layout Algorithmen:
Das Graph-Layout erfolgt mittels Algorithmen, welche von der *Jung Graph API* bereitgestellt werden (siehe [JUNG, 2009]).

Einzelne Nodes und Edges werden unter Berücksichtigung der RDF-Syntax dargestellt. So gibt es unterschiedliche Symbole bzw. Pfeiltypen zur Repräsentation von Klassen, Literals, URIs und Properties (siehe Abbildung 3.6). [Goyal and Westenthaler, 2009a]

Lizenzierung, Verfügbarkeit:
Eine Online-Version ist frei verfügbar. Der Source Code + Dokumentation kann via Email angefordert werden (siehe dazu [Goyal and Westenthaler, 2009b])

Technische Details:
Datenformate: Dateien können im *RDF*- bzw. *OWL*-Format importiert werden. Es besteht keine Möglichkeit zum Export von Dateien.

Software-Plattform/Libraries: Die Software wurde in Java implementiert. Das Graph-Layout wird mit-

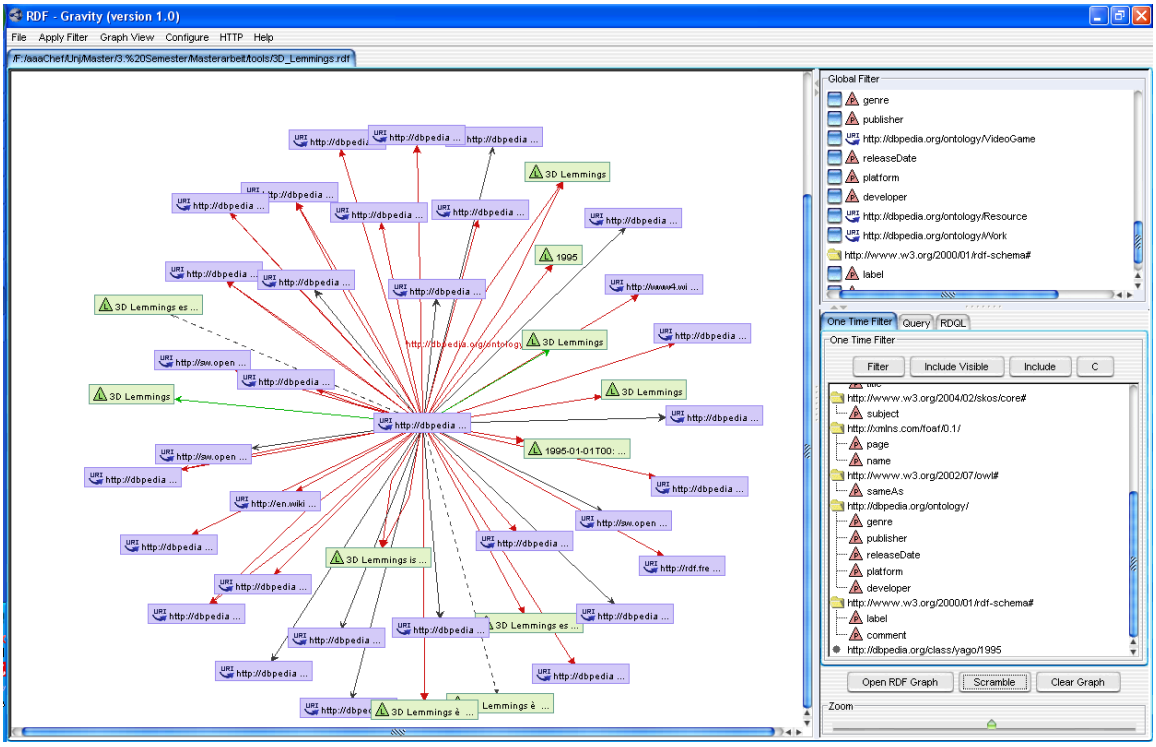


Abbildung 3.6: RDF Gravity ([Goyal and Westenthaler, 2009a]): Oberfläche + Visualisierung eines kleinen RDF-Graphen (44 Nodes, 51 Edges; aus [DBpedia, 2009])

hilfe der *JUNG Graph API* realisiert. Das *Jena Semantic Web Toolkit* ermöglicht das Handling der RDF-Modelle. [Goyal and Westenthaler, 2009a]

Fazit:
RDF Gravity eignet sich besonders zur Visualisierung kleinerer Graphen. Durch den Einsatz diverser Filter-Mechanismen ist jedoch auch das Handling von Graphen mit einer höheren Node-Anzahl möglich. Die umfangreiche Unterstützung RDF-spezifischer bzw. semantischer Aspekte sowie die intuitive Bedienung sind die größten Vorteile dieser Software. Allerdings ist der Source Code (+ Dokumentation) nicht direkt verfügbar. Dieser kann nur auf Anfrage und möglicherweise gegen Gebühr bezogen werden. Die frei verfügbare Online-Version stammt zudem bereits aus dem Jahr 2004.

3.2.6 Pajek

Pajek ist ein Programm zur Visualisierung und Analyse von großen Netzwerken bzw. Graphen (z.B. Collaboration Networks, Internet Networks, Citation Networks, etc.) [Batagelj and Mrvar, 2009a] [Batagelj and Mrvar, 2009b]. Diese für gewöhnlich lichten Netzwerke können aus bis zu mehreren Millionen Knoten bestehen. In diesem Zusammenhang liegt das Hauptaugenmerk von Pajek auf der Bereitstellung von effizienten Visualisierungswerkzeugen und Analysealgorithmen sowie der Unterstützung von Techniken zur Abstraktion bzw. Reduktion eines großen Netzwerks.

Unterstützung semantischer Graphen:
Pajek bietet keine direkten Features zur Visualisierung oder Analyse semantischer Graphen. Beispielsweise können RDF-Files nicht direkt importiert/gelesen werden. Dazu ist erst eine Konvertierung in das von Pajek verwendete Ascii-Format *net* notwendig. Das auf der Projekt-Website verfügbare Konvertierungstool (siehe [Pfeffer, 2009]) ermöglicht die Umwandlung eines Textfiles (z.B. *N-Triples, Notation3, CSV,...*) in eine *net*-Datei. Die Konvertierung funktioniert jedoch nicht immer problem- bzw. verlustlos.

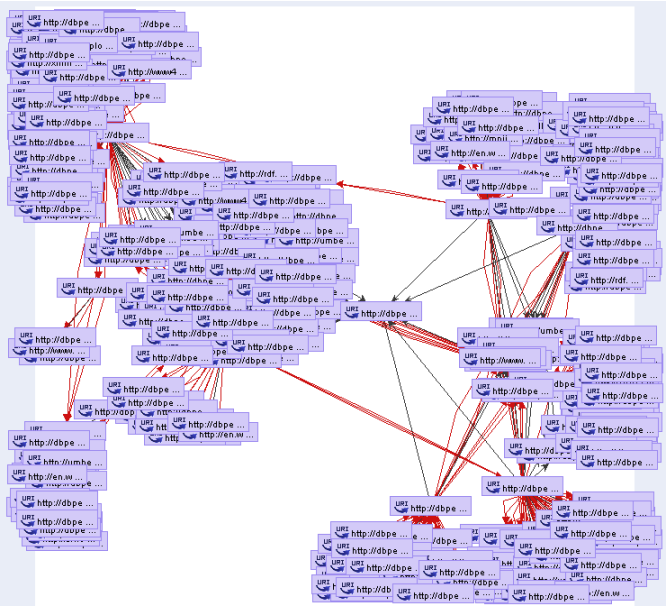


Abbildung 3.7: Visualisierung des bereits unter Punkt 3.2.2 bzw. 3.2.4 verwendeten RDF-Datensatzes (825 Nodes, 1126 Edges; aus [DBpedia, 2009]). Die Überlagerung von Nodes bzw. Labels führt zu einem unübersichtlichen Layout.

Der in Abbildung 3.8 dargestellte Graph wurde mithilfe dieses Tools vom *RDF*- in das *net*-Format konvertiert.

Skalierbarkeit:
Mithilfe effizienter Visualisierungstechniken sowie verschiedener Abstraktions- bzw. Reduktionstechniken ist Pajek in der Lage, Graphen/Netzwerke mit bis zu mehreren Millionen Knoten darzustellen bzw. zu analysieren. [Batagelj and Mrvar, 2009a]

Interaktivität:
Das Tool unterstützt eine Reihe interaktiver Features wie beispielsweise Zooming, Editierung, Fish-Eye View, Kontext-Menüs, Verschieben von Nodes/Edges sowie die Möglichkeit die Graphdarstellung entlang der x-, y- und z-Achse zu drehen. Für weitere Details siehe Pajek User Manual [Batagelj and Mrvar, 2009a]

Reduktion der visuellen Komplexität/Analysemechanismen:
Pajek implementiert u.a. folgende Features zur Komplexitätsreduktion eines großen Netzwerks:

- Identifikation von Clustern
- Extraktion und separate Darstellung von Knoten, welche dem selben Cluster angehören (*Detailed Local View*)
- Aggregierung von Knoten in Clustern und Darstellung von Relationen zwischen diesen Clustern (*Global View*)

Zudem umfasst das Programm noch eine Reihe von Features zur graphtheoretischen Analyse des Netzwerks. [Batagelj and Mrvar, 2009a]

Dynamische Aspekte:
Pajek unterstützt neben einer Reihe normaler Netzwerke/Graphen (gerichtet, ungerichtet, etc.) auch *Tem-*

poral Networks, d.h. Netzwerke bzw. Graphstrukturen, welche sich im Zeitablauf ändern. Zusätzlich werden Layouts animiert. [Batagelj and Mrvar, 2009a]

Darstellungsformen, Layout Algorithmen:
Neben der Möglichkeit, Farbe, Form und Größe der Nodes und Edges manuell anzupassen, umfasst Pajek eine Reihe von Graph Drawing Algorithmen, darunter *energy-based*- (Kamada Kawai, Fruchterman Reingold; sowohl 2D als auch 3D), *circular*- bzw. *eigenvalue-based* Layouts. [Batagelj and Mrvar, 2009a]

Lizenzierung, Verfügbarkeit:
Das Programm kann für nichtkommerzielle Zwecke als Windows Binary von der Pajek-Website ([Batagelj and Mrvar, 2009b]) frei heruntergeladen werden. Allerdings ist kein Source Code verfügbar.

Technische Details:
Datenformate: Netzwerke können als Ascii-Datei in verschiedenen Formaten (z.B. als File mit der Endung *.net*) gelesen bzw. gespeichert werden. Export-Formate wären u.a. *GraphML*, *EPS/PS*, *SVG* oder *Bitmap*. Zudem besteht die Möglichkeit, die Darstellung im *X3D* bzw. *VRML*-Format zu exportieren.

Software-Plattform/Aktualität: Das Programm wurde in *Delphi* (bzw. *Pascal*) implementiert. Die Entwicklung von Pajek startete Ende 1996, die aktuelle Version stammt aus dem Jahr 2009. [Batagelj and Mrvar, 2009b]

Fazit:
Pajek bietet die größte Auswahl an Clustering- bzw. Analysemechanismen der im Zuge der Evaluierung näher betrachteten Visualisierungssysteme. Die Möglichkeit zur Visualisierung von größeren, komplexeren Graphen ist ein weiterer Vorteil. Aufgrund des großen Funktionsumfangs bedarf es jedoch einiger Einarbeitungszeit bzw. Vorkenntnisse auf dem Gebiet der Netzwerkanalyse.

Die fehlende Unterstützung semantischer Graphen sowie der Umstand, dass der Source Code nicht verfügbar ist, stellen die größten Nachteile von Pajek im Kontext dieser Masterarbeit dar. Hinsichtlich der praktischen Implementierung ist diese Softwarelösung somit von geringerer Bedeutung.

3.2.7 ZGRViewer

ZGRViewer ist eine Software zur Betrachtung von Graphstrukturen. Es wurde in Java implementiert und beruht auf den Graphviz Layout-Engines *Dot, Neato, Circo, Twopi*. Das GUI basiert wie bei IsaViz auf *ZVTM*(Zoomable Visual Transformation Machine) bzw. der *Java2D* Bibliothek. Es können ausschließlich Files im DOT-Format geladen bzw. die aktuelle Darstellung als Bild in den Formaten PNG bzw. SVG exportiert werden. ZGRViewer implementiert somit keine Features zur direkten Visualisierung bzw. Analyse von semantischen Graphen.

Die Software bietet eine Reihe von Features zur Betrachtung bzw. Navigation des dargestellten Graphen (Zooming, Panning, Distortion Lenses, Focus+Context Magnification,...). Da es sich um einen Viewer handelt, werden keinerlei Editierungsfunktionalitäten unterstützt. Laut Projektbeschreibung auf [Pietriga, 2009], ist die Software zur Darstellung bzw. Behandlung großer Graphen in der Lage.

Der Source Code (Lizenz: *LGPL*) kann von der Projekt-Website bzw. direkt vom SVN Repository heruntergeladen werden (aktuelle Version: April 2009). Daneben ist auch eine Applet-Version verfügbar. [Pietriga, 2009] [Pietriga, 2005b]

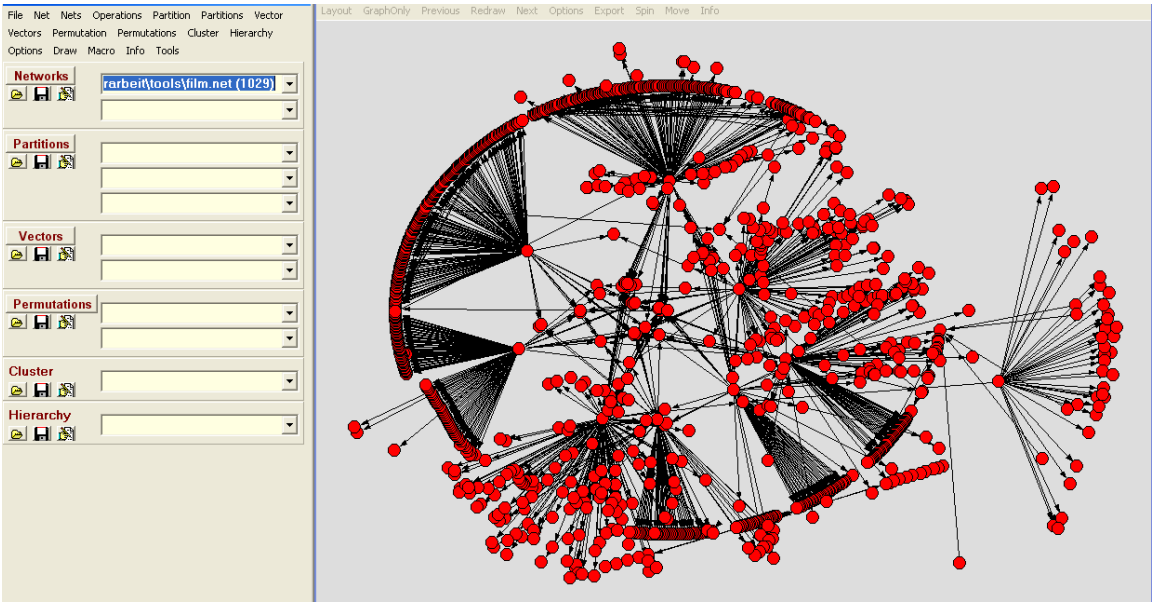


Abbildung 3.8: Pajek ([Batagelj and Mrvar, 2009b]): Oberfläche + Visualisierung (Kamada Kawai-Layout) des bereits in den Abschnitten 3.2.2, 3.2.4 und 3.2.5 verwendeten RDF-Graphen. In diesem Fall erfolgte jedoch zuvor eine Konvertierung vom RDF- in das *net*-Format. Die Knoten- bzw. Kanten-Labels wurden in dieser Darstellung zur besseren Lesbarkeit ausgeblendet.

3.2.8 Cytoscape

Cytoscape ist eine Open Source Plattform für die Analyse und Visualisierung von komplexen Netzwerken. Die Software wurde ursprünglich für Anwendungen aus dem Bereich der Bioinformatik entwickelt, so zum Beispiel zur Visualisierung von molekularen Interaktionsnetzwerken oder biologischen Pfaden. Im Laufe der Zeit entwickelte sich Cytoscape zu einem generellen Graphvisualisierungs- bzw. Analyseframework. Das erste Release wurde 2002, das aktuelle 2009 veröffentlicht. Die Weiterentwicklung von Cytoscape wird von einem Basis-Entwicklerteam sowie einer aktiven Community vorangetrieben. [Cytoscape, 2009d] [Cytoscape, 2009g]

Cytoscape ist ein äußerst mächtiges und umfangreiches Softwarepaket, welches eine breite Palette an nützlichen Features zur Verfügung stellt. Trotz des enormen Potenzials bzw. der vielfältigen Möglichkeiten weist Cytoscape einen vergleichsweise geringen Bekanntheitsgrad auf. Einer der Hauptgründe dafür liegt wohl im ursprünglichen Verwendungszweck bzw. der Klassifizierung der Software als Bioinformatik-Visualisierungsplattform (siehe Cytoscape-Homepage [Cytoscape, 2009d] bzw. [Bergman, 2009a]). Die wichtigsten Merkmale und Funktionalitäten dieses Frameworks können folgendermaßen zusammengefasst werden:

- Visualisierung, Analyse und Manipulation von Graphen bzw. Netzwerken verschiedenen Typs (jedoch keine direkte Unterstützung von semantischen Graphen)
- Skalierbarkeit (je nach vorhandener Hardware können Graphen mit bis zu 10^6 Elementen visualisiert werden)
- Umfangreiche Interaktionsmöglichkeiten (Navigation, Editierung, Zoom, Suche, Query,...)
- Vielzahl an Layout-Algorithmen
- Möglichkeit zur Komplexitätsreduktion großer Graphen (Filter, Query, Extraktion, etc.)
- Import/Export mehrerer unterschiedlicher Formate

- Open Source (*LGPL* - GNU Lesser General Public License)
- Plattformunabhängige Java-Implementierung; Binaries für Windows sowie Source Code sind frei verfügbar
- Frei verfügbare und leicht zu integrierende Plugins zur Bereitstellung weiterer Funktionalitäten
- Übersichtlich strukturierter bzw. gut dokumentierter Source Code
- Erweiterbar (mittels Plugins bzw. direkter Modifikation des Source Codes)
- Umfangreiche Dokumentation (User Manual, Wiki, Tutorials, Discussion-Groups)
- Aktualität bzw. permanente Weiterentwicklung

Eine ausführlichere Behandlung der speziellen Eigenschaften von Cytoscape sowie eine nähere Betrachtung der implementierten Features und Algorithmen findet sich im nächsten Kapitel.

Cytoscape bietet den in Summe größten Funktionsumfang aller betrachteten Frameworks und Tools. Jedoch können trotz der Vielzahl an unterstützten Netzwerktypen bzw. Formaten RDF-Graphen nicht direkt gelesen werden. Die RDF-Triple müssen zuerst in ein *CSV*-File geparst und dann mittels der *Network from Table*-Funktion importiert werden (siehe dazu auch [Cytoscape, 2009g] bzw. [Bergman, 2009a]).

3.3 Schlussfolgerungen

Es existiert eine große Anzahl an unterschiedlichen Systemen bzw. Tools zur Visualisierung von Graphen und Netzwerken. Neben der Möglichkeit zur visuellen Repräsentation der Graphstruktur finden sich auch häufig Features zur Analyse, Interaktion bzw. Manipulation. Unterschiede ergeben sich u.a. hinsichtlich der Nutzung (frei bzw. kommerziell), dem Verwendungszweck (generelle bzw. spezielle Graphen) sowie dem prinzipiellen Software-Typ (Forschungsprojekt, Prototyp, aktive Weiterentwicklung,...).

Die im Rahmen der Evaluierung betrachteten Visualisierungspakete können grundsätzlich zwei unterschiedlichen Gruppen zugeordnet werden: Die erste Gruppe umfasst Tools und Frameworks zur Visualisierung bzw. Analyse von generellen Graphen und Netzwerken. Dazu zählen *Graphviz, Tulip, Pajek, ZGRViewer* und *Cytoscape*. Der Fokus liegt dabei vor allem auf allgemeinen Graph Drawing Aspekten (z.B. Layout-Algorithmen, graphtheoretische Analyse, etc.). Diese Pakete sind auch zur Behandlung größerer Graphen in der Lage. Jedoch werden kaum (bzw. nur wenige) Features zur Berücksichtigung semantischer Aspekte bereitgestellt.

Zur zweiten Gruppe zählen Tools speziell zur Verarbeitung von RDF-Graphen. Beispiele dafür sind *IsaViz, Welkin* und *RDF Gravity*. Diese Pakete bieten spezielle Features zur Darstellung, Navigation und Analyse von semantischen Graphen bzw. RDF-Modellen. Ein Schwachpunkt ist jedoch deren schlechte Skalierbarkeit (unübersichtliches Layout, schlechte Performance,...).

Unterschiede ergeben sich auch im Bezug auf den Funktionsumfang der einzelnen Pakete. *Cytoscape* und *Tulip* bieten beispielsweise die größte Auswahl an Layout-Algorithmen sowie umfangreiche Interaktionsmöglichkeiten. *Pajek* ist wiederum das Tool mit der größten Anzahl an Analysemechanismen bzw. Features zur Reduktion der visuellen Komplexität. Hinsichtlich der Möglichkeiten zur Darstellung, Navigation und Analyse von RDF-Graphen stellt hingegen *IsaViz* das Referenztool dar. In Tabelle 3.1 bzw. in Tabelle 3.2 werden die Ergebnisse der Evaluierung zur besseren Übersicht nochmals zusammengefasst.

Idealerweise würde ein Visualisierungssystem einerseits effiziente Mechanismen und Algorithmen zur Darstellung bzw. Manipulation größerer Graphen bereitstellen. Andererseits sollten aber auch Features zur Behandlung semantischer Graphen (RDF-Graphen) unterstützt werden (Import, Darstellung, Navigation, Analyse, etc.). Ein Hybrid aus *Cytoscape* und *IsaViz* würde somit einen Großteil dieser Forderungen erfüllen.

	Graphviz	IsaViz	Tulip	Welkin
Import von RDF-Modellen	nein	ja	nein	ja
Berücksichtigung semantischer (RDF) Informationen (z.B. bei Analyse, Layout, etc.)	nein	ja	nein	ja
Skalierbarkeit	kleinere Graphen	kleinere Graphen	größere Graphen	mittlere Graphen
Möglickeiten zur Reduktion der visuellen Komplexität	nein	eingeschränkt	ja	ja
Features zur Analyse des Graphen	nein	nein	umfangreich	ja
Möglichkeiten/Techniken zur Interaktion	mit zusätzlichen Viewern/Editoren	ja	umfangreich	ja
Berücksichtigung dynamischer Aspekte	mittels Erweiterungen	nein	nein	teilweise (animierte Übergänge)
Lizenzierung	Open Source (CPL)	Open Source (W3C Software License)	Open Source (GPL)	Open Source (BSD)
Layout-Algorithmen/Darstellungsformen	5 Layouts; 2D	Layouts von Graphviz; 2D	ca. 19 Layouts; 2D/3D	ca. 2-3 Layouts; 2D
Datenformate Import	DOT	RDF	GML, DOT	RDF
Datenformate Export	DOT, Bildformate, Textformate	RDF, Bildformate	GML, Bildformate	kein Export-Feature
Software-Plattform	hauptsächlich C	Java	C++, OpenGL, Qt	Java
Betriebssystem/verfügbare Packages	Windows, Linux, Mac; Binaries, Source Code	plattformunabhängig; Source Code; Jar-Files	Windows, Linux; Binaries, Source Code	plattformunabhängig; Source Code
Dokumentation	umfangreich; User/Developer	User	User/Developer	kurzer User Guide
letztes Release/Aktivität	2009	2007	2009	2008
Erweiterungsmöglichkeiten/Relevanz für praktische Implementierung	*	* *	*	* *

Tabelle 3.1: Zusammenfassung der Evaluierungsergebnisse I (*Graphviz, IsaViz, Tulip, Welkin*)

	RDF Gravity	Pajek	ZGRViewer	Cytoscape
Import von RDF-Modellen	ja	nein	nein	indirekt
Berücksichtigung semantischer (RDF) Informationen (z.B. bei Analyse, Layout, etc.)	ja	nein	nein	nein
Skalierbarkeit	kleine bis mittlere Graphen	große Netzwerke	mittlere Graphen	große Graphen
Möglickeiten zur Reduktion der visuellen Komplexität	ja	umfangreich	nein	teilweise
Features zur Analyse des Graphen	ja	umfangreich	nein	ja
Möglichkeiten/Techniken zur Interaktion	ja	ja	ja	umfangreich
Berücksichtigung dynamischer Aspekte	nein	ja	nein	nein
Lizenzierung	Binary frei verfügbar; kein Open Source	Binary frei verfügbar; kein Open Source	Open Source (LGPL)	Open Source (LGPL)
Layout-Algorithmen/Darstellungsformen	eingeschränkte Layout-Möglichkeiten	circular, energy-based eingenvalue-base	Layouts von Graphviz; 2D	ca. 30 Layouts; 2D
Datenformate Import	RDF	Ascii-Formate (net)	DOT, SVG	umfangreich
Datenformate Export	kein Export-Feature	Ascii-Formate, Bildformate	SVG, PNG	umfangreich
Software-Plattform	Java	Delphi	Java	Java
Betriebssystem/verfügbare Packages	plattformunabhängig; nur Online-Version - Source Code auf Anfrage	Windows Binary; kein Source Code	plattformunabhängig; Source Code; Jar-Files	plattformunabhängig; Source Code; Jar-Files; Windows Binary
Dokumentation	Intro auf Website	User Manual	kurzes Intro auf Website	umfangreich; User/Developer
letztes Release/Aktivität	2004	2009	2009	2009
Erweiterungsmöglichkeiten/Relevanz für praktische Implementierung	*	*	*	* * *

Tabelle 3.2: Zusammenfassung der Evaluierungsergebnisse II (*RDF Gravity, Pajek, ZGRViewer, Cytoscape*)

3.4 Konsequenzen und weitere Vorgehensweise

Nach der genaueren Betrachtung der im Kontext dieser Masterarbeit relevanten Features und Eigenschaften sowie der Abwägung der Vor- und Nachteile der einzelnen Tools, kann *Cytoscape* als das umfangreichste und kompletteste Softwarepaket bezeichnet werden. Hinsichtlich der Implementierung eines Systems bzw. einer Komponente zur interaktiven Visualisierung von semantischen Graphen im RDF-Format, welches zudem Features zur dynamischen Aggregierung von Graphelementen unterstützt, stellt Cytoscape somit das ideale Framework dar. Nachfolgend werden die Gründe bzw. Hauptargumente für die Implementierung dieser Visualisierungskomponente auf Basis von Cytoscape aufgelistet.

- Cytoscape unterstützt bereits viele der relevanten Features (wie etwa, Skalierbarkeit, Interaktivität, Analyse, diverse Layout-Algorithmen, etc.).
- Der Source Code ist frei unter der *GNU Lesser General Public License* verfügbar.
- Eine flexible Plugin-Architektur ermöglicht die Einbindung neuer Komponenten, ohne dass der gesamte Cytoscape-Source neu kompiliert werden muss.
- Der gut dokumentierte bzw. übersichtlich strukturierte Source Code erleichtert die Implementierung von Erweiterungen/Plugins.
- Cytoscape wird permanent weiterentwickelt. Zudem existiert eine Reihe von frei verfügbaren Plugins (die meisten davon Open Source) zur Erweiterung des Funktionsumfangs.
- Die plattformunabhängige Java-Implementierung von Cytoscape stellt einen weiteren positiven Aspekt dar.

Das nächste Kapitel beschäftigt sich eingehend mit Cytoscape bzw. den zur Verfügung gestellten Funktionalitäten und Mechanismen. Auf Basis dessen werden im Anschluss die Requirements für den praktischen Teil der Masterarbeit definiert.

Kapitel 4

Cytoscape Framework

Im vorangegangen Kapitel wurden existierende, für diese Arbeit relevante Graph- Visualisierungspakete näher betrachtet und bewertet. *Cytoscape* stellte sich im Zuge dessen als das am besten geeignete Framework zur Implementierung einer neuen Visualisierungskomponente heraus. Aus diesem Grund widmet sich dieses Kapitel der genaueren Betrachtung von Cytoscape. Einleitend werden im Rahmen eines Überblicks allgemeine sowie technische Aspekte des Frameworks behandelt. Im Anschluss erfolgt die Diskussion der wichtigsten bzw. für diese Arbeit relevanten Funktionalitäten, Eigenschaften und Mechanismen. Abschließend werden Einschränkungen sowie fehlende Funktionalitäten von Cytoscape beleuchtet und auf Basis dessen mögliche Erweiterungen diskutiert bzw. grobe Anforderungen an die nachfolgende praktische Implementierung festgelegt.

4.1 Überblick

Dieser Abschnitt bietet einen generellen Überblick über Cytoscape. Im Zuge dessen wird der Hintergrund bzw. die Entwicklungsgeschichte kurz beleuchtet und grundlegende technische Aspekte bzw. Lizenzierungsfragen erörtert.

4.1.1 Allgemein

Cytoscape ist eine Open Source Plattform für die Analyse und Visualisierung von komplexen Netzwerken. Die Software wurde 2002 zur grafischen Darstellung von molekularen Interaktionsnetzwerken und biologischen Pfaden entwickelt. Die Veröffentlichung der aktuellen Version 2.6.3 erfolgte im dritten Quartal 2009. Die Entwicklung von Version 3.0 ist derzeit im Gange und soll in naher Zukunft abgeschlossen werden. Das Haupteinsatzgebiet liegt nach wie vor in der biologischen Forschung. Cytoscape wird auch seitens der Entwickler als Bioinformatik-Software klassifiziert (siehe offizielle Website [Cytoscape, 2009d]). Nichtsdestotrotz kann Cytoscape als ein generelles Framework zur Visualisierung von großen Graphen bzw. komplexen Netzwerken betrachtet werden.

Cytoscape ist ein Gemeinschaftsprojekt mehrerer Universitäten und Forschungseinrichtungen, darunter dem *Institute for Systems Biology*, der *University of California* (San Francisco, San Diego) und der *University of Toronto*. Finanzielle Unterstützung erfährt das Projekt von verschiedenen Gesundheitsinstitutionen (siehe dazu [Cytoscape, 2009d]).

4.1.2 Verfügbarkeit, Requirements

Cytoscape ist eine in Java geschriebene Open Source Software (GNU Lesser General Public License). Es existieren automatische Installationspakete für Windows, Linux und Mac OSX. Der Source Code

kann entweder als Archiv oder direkt von einem Subversion Repository heruntergeladen werden. Das Download-Package enthält neben Beispieldaten und Manuals auch eine Reihe von third-party Java Libraries, wie etwa *GINY* (Graph Interface Library; siehe [GINY, 2009]), *Piccolo* oder *Xerces XML Parser*. Grundsätzlich erfordert die Verwendung von Cytoscape (ab Version 2.5) mindestens *Java SE 5*. Die Anforderungen an die Hardware variieren je nach Größe des zu visualisierenden Netzwerkes bzw. Graphen. Der Speicherbedarf ist dabei von der Anzahl der Graphelemente (Nodes und Edges) sowie deren Attribute (z.B. Name, Typ, Gewichtung,...) abhängig. Als Faustregel gilt: Die Visualisierung und Analyse von Konstrukten mit bis zu 150.000 Elementen benötigt in etwa 1GB Arbeitsspeicher sowie einen Prozessor mit einer Taktfrequenz von mindesten 1GHz. Größere Netzwerke erfordern dementsprechend leistungsfähigere CPUs und Grafikkarten sowie mindestens 2GB RAM. Nähere Informationen dazu finden sich auch unter [Cytoscape, 2009g].

4.1.3 Implementierungsaspekte

Die Software besteht aus einem Kern, welcher grundlegende Funktionalitäten zur Verfügung stellt. Eine flexible Plugin-Architektur erlaubt die einfache Erweiterung des Kerns um zusätzliche Features. Aufgrund des modularen, flexiblen Aufbaus können neue Komponenten und Funktionalitäten problemlos und ohne Recompilation des gesamten Source Codes eingebunden werden. Dies resultiert in einer großen Zahl frei verfügbarer Plugins, beispielsweise zur Netzwerkanalyse, zum Import verschiedener Datenformate oder zur Gruppierung von Graphelementen (siehe [Cytoscape, 2009c]). Eine nähere Betrachtung relevanter Plugins findet sich zudem in Abschnitt 4.3.

Grundsätzlich können Features in zwei Gruppen eingeteilt werden. Einerseits implementiert Cytoscape eine Reihe grundlegender *Core-Features*. Diese können als Core-Plugins, Core-Libraries oder Core-Source im Cytoscape-Package enthalten sein. Andererseits besteht die Möglichkeit, zusätzliche Funktionalitäten mittels optionaler, externer *Plugins* hinzuzufügen. Die Trennung in Core- bzw. Plugin-Features soll die Modularität, Flexibilität und Wiederverwendbarkeit des Frameworks sicherstellen. So enthält der Core-Source Code von Cytoscape keinerlei biologischer Semantiken. Spezielle Bioinformatik-Features werden ausschließlich via Plugins eingebunden. Cytoscape stellt somit ein generelles Framework zur Graphvisualisierung dar. [Cytoscape, 2009a]

Die Entwicklung des Frameworks wird von einem Core Development Team sowie einer aktiven Community vorangetrieben. Neben einem umfangreichen User Manual existiert speziell für Entwickler ein eigenes Wiki, mehrere Tutorials und Discussion-Groups sowie eine API-Dokumentation des Cytoscape Source Codes. [Cytoscape, 2009g] [Cytoscape, 2009d]

Da der Source Code frei verfügbar ist, kann auf annähernd alle der nachfolgend beschriebenen Features auch programmiertechnisch (als Library bzw. Source) zugegriffen werden. Dies ermöglicht die Erweiterung der ursprünglichen Funktionalität sowie die Anpassungen an spezifische Problemstellungen. Der Fokus dieses Kapitels liegt in erster Linie auf der Beschreibung der Features aus Anwendersicht. Das nächste Kapitel widmet sich hingegen der näheren Betrachtung relevanter Konzepte vom Entwicklerstandpunkt aus. Im Zuge dessen werden verschiedene Aspekte der Implementierung näher behandelt.

4.1.4 Aufbau, Architektur

Cytoscape wurde unter Berücksichtigung objektorientierter Prinzipien komplett in Java implementiert. Abbildung 4.1 zeigt eine Übersicht der wichtigsten Komponenten und deren Beziehungen. Nachfolgend werden die grundlegenden Konzepte kurz umrissen.

`Cytoscape` ist eine abstrakte statische Klasse, welche u.a. für die Erzeugung, die Modifikation bzw. das Löschen eines Netzwerks (`CyNetwork`) zuständig ist.

`CyNetwork` stellt die zentrale Komponente zur Generierung eines Netzwerks bzw. Graphen in Cytoscape dar. Sie basiert auf der *GINY Library* (siehe [GINY, 2009]) und hält das grundlegende Netzwerk- bzw.

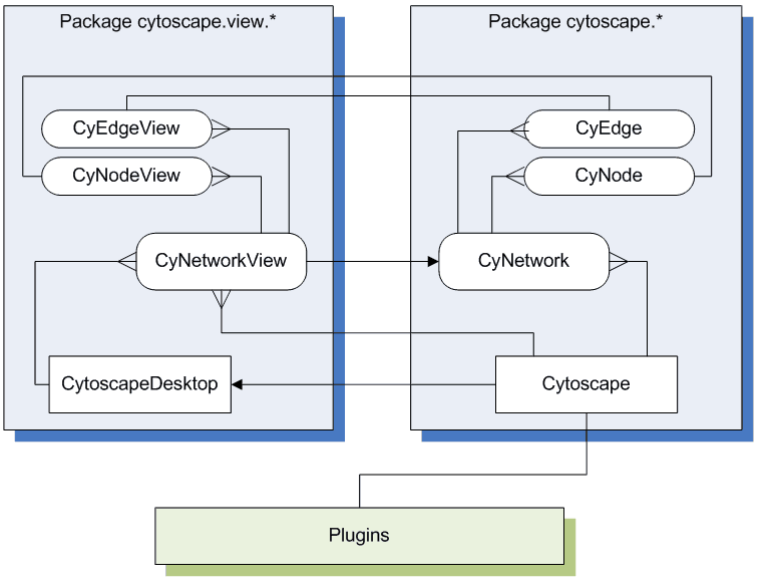


Abbildung 4.1: Übersicht über die grundlegende Architektur von Cytoscape (basierend auf [Cytoscape, 2009b]) (Anmerkung: keine UML-Notation)

Graph-Modell. Ein Netzwerk besteht aus einer Menge von Nodes (`CyNode`) und einer Menge von Edges (`CyEdge`). Nodes und Edges werden durch einen eindeutigen String identifiziert. Edges verbinden immer zwei Nodes miteinander und können sowohl gerichtet als auch ungerichtet sein.

`CyNetworkView` dient prinzipiell der Visualisierung einer Instanz von `CyNetwork`. Pro Node bzw. Edge im Graph-Modell existiert ein `CyNodeView` bzw. ein `CyEdgeView`. Ein `CyNetworkView` wird wie ein `CyNetwork` mithilfe der statischen Klasse `Cytoscape` erzeugt bzw. auch wieder gelöscht. Es können gleichzeitig mehrere Instanzen von `CyNetworkView` bzw. `CyNetwork` existieren.

`CytoscapeDesktop` verwaltet die verschiedenen `CyNetworkView`-Objekte. Es existiert immer nur eine Instanz von `CytoscapeDesktop`, die Erzeugung erfolgt wiederum mithilfe der statischen `Cytoscape`-Klasse. [Cytoscape, 2009h] [Cytoscape, 2009f] [Cytoscape, 2009b]

Eine genauere Betrachtung der für die praktische Implementierung relevanten Klassen, Konzepte und Strukturen findet sich im nächsten Kapitel.

4.2 Features

Cytoscape ist ein mächtiges und umfangreiches Framework zur Visualisierung, Manipulation und Analyse von komplexen Netzwerken und Graphen. Eine große Anzahl an Features wird von Cytoscape bereits standardmäßig zur Verfügung gestellt. Plugins erweitern diesen Funktionsumfang um ein Vielfaches. Im vorangegangen Kapitel wurden die wichtigsten Funktionalitäten und Merkmale bereits überblicksmäßig präsentiert. In diesem Abschnitt sollen nun essentielle Funktionen sowie die für die anschließende Implementierung besonders relevanten Features näher betrachtet werden.

4.2.1 Oberfläche

Cytoscape bietet eine interaktive grafische Benutzeroberfläche. Das Standard-Design des Interface ermoeglicht eine intuitive Nutzung der wichtigsten Funktionen. Abbildung 4.2 zeigt das GUI mit den wichtigsten Features und Funktionalitäten. Die einzelnen Komponenten im Überblick:

- **A**: Panel zur Graphdarstellung

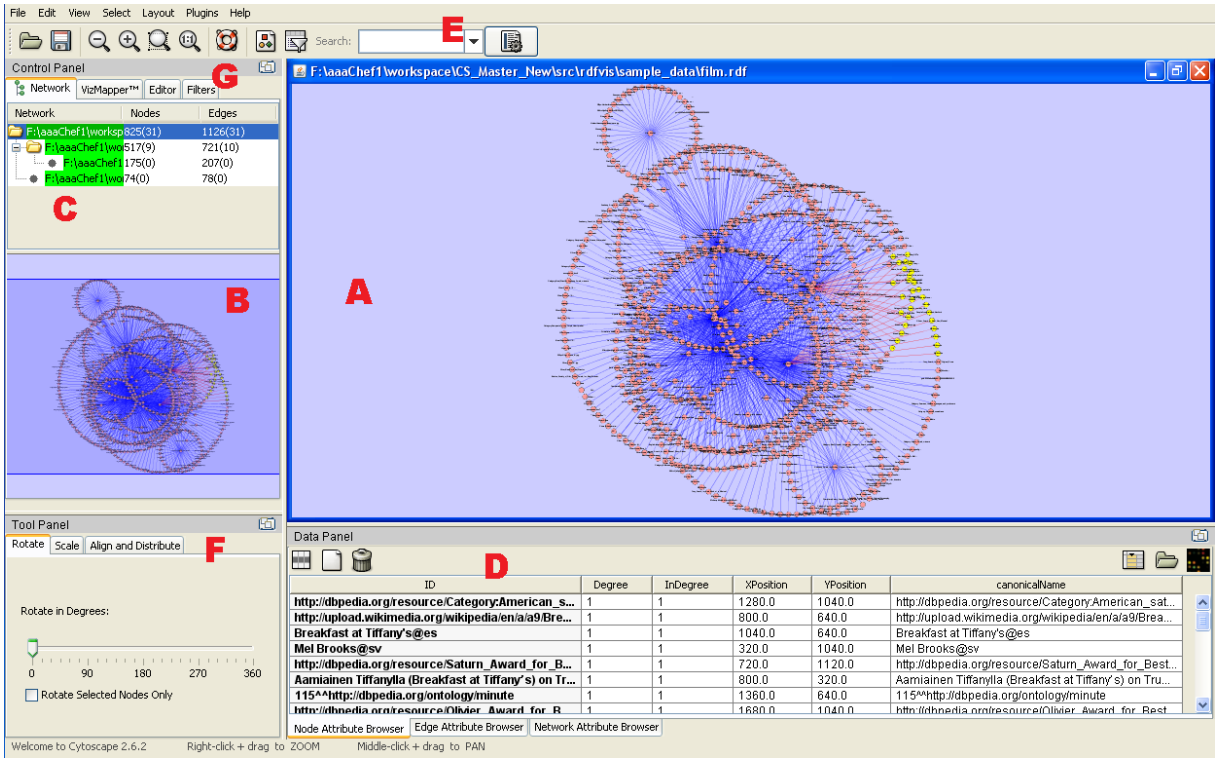


Abbildung 4.2: Die grafische Benutzeroberfläche von Cytoscape (Screenshot; [Cytoscape, 2009d])

- **B**: Overview-Window
- **C**: Network-Panel (nähere Informationen zu den aktuell geladenen Graphen; hierarchische Anordnung der einzelnen Sub-Graphen)
- **D**: Attribute-Browser (Zusatzinformationen zu selektierten Nodes und Edges)
- **E**: Eingabemaske für textbasierte Suche
- **F**: Tool-Panel zum manuellen Layout des Graphen
- **G**: Registerkarten für Filter- und Editierungs-Tools

4.2.2 Datenintegration

Es existieren folgende Möglichkeiten zum Import und Export sowie zur Generierung eines Graphen bzw. eines Netzwerkes [Cytoscape, 2009g]:

- **Import/Export von Graph- bzw. Netzwerk-Files**
  Cytoscape kann verschiedene vordefinierte Datenformate direkt importieren bzw. auch exportieren. Neben speziellen Formaten aus dem Bereich der Bioinformatik (z.B. *Simple Interaction File (SIF), BioPax,...*) wären dies *GML (Graph Markup Language)* bzw. *XGMML (Extensible Graph Markup and Modelling Language)*. Bei allen Formaten handelt es sich um Textfiles. Es können sowohl lokal als auch remote gespeicherte Dateien (via URL) importiert werden.
- **Import von Netzwerken/Graphen aus Tabellen**
  Ab Version 2.4 wird der Import von Tabellen in Form von *delimited Textfiles* (Trennung der Spalten

u.a. mittels Tabulator, Komma, etc.; beispielsweise *CSV*-Files) und *Excel-Sheets (.xls)* unterstützt. Mittels Interface lassen sich verschiedene Parsing-Parameter konfigurieren. So muss u.a. festgelegt werden, welche Spalten der Tabelle Source-Nodes, Target-Nodes und Interactions (=Edges) repräsentieren.

- **Laden/Speichern von Session-Files**
  Zum Speichern bzw. Laden einer Cytoscape-Session wird das *.cys*-Format verwendet. Neben dem Netzwerk bzw. der Graphstruktur selbst, enthält eine solche Datei u.a. auch Node- und Edge-Attribute (siehe Unterpunkt 4.2.3).

- **Netzwerk-Erstellung**
  Netzwerke/Graphen können auch durch das manuelle Hinzufügen von Nodes und Edges, durch Klonen des aktuellen Netzwerks sowie durch Extraktion von Sub-Graphen generiert werden.

- **Import via Web Services**
  Dieses Feature erlaubt den Import von Graphen/Netzwerken sowie deren Attributen aus externen Datenbanken. Mittels *Web Service Clients* kann auf Remote-Datenquellen zugegriffen werden. Derzeit werden jedoch nur Web Services aus dem Bioinformatikbereich unterstützt (Details siehe [Cytoscape, 2009i]).

- **Image Export**
  Neben dem Export von Graph-Formaten besteht weiters die Möglichkeit, die Darstellung als Bild, u.a. in den Formaten *PDF, EPS, SVG, PNG, JPEG* und *BMP* abzuspeichern.

Nach dem Import eines Netzwerks befinden sich alle relevanten Informationen (Nodes, Edges, evtl. auch Attribute) im Speicher. Danach besteht die Möglichkeit, eine grafische Darstellung (= *View)* des Netzwerks zu generieren. Es können auch mehrere Views desselben Netzwerks erstellt bzw. wieder zerstört werden. Auf diese Weise ist es zum Beispiel möglich, mehrere unterschiedliche Layouts des selben Graphen miteinander zu vergleichen. Cytoscape ist in der Lage, mehrere Netzwerke gleichzeitig zu laden bzw. diese hierarchisch anzuordnen. So wird beispielsweise ein extrahierter Sub-Graph als Child des ursprünglichen Netzwerks behandelt. Zudem werden in einem *Network-Panel* Details wie etwa die Anzahl der Nodes und Edges, der Name des geladenen Files oder die hierarchische Anordnung der Netzwerke ausgegeben (siehe Abbildung 4.2 sowie Abbildung 4.3). [Cytoscape, 2009g]

4.2.3 Attribute

Cytoscape bietet die Möglichkeit, Zusatzinformationen zu Nodes, Edges bzw. dem gesamten Netzwerk zu definieren. Neben einer eindeutigen ID jedes Elements werden auf diese Weise Metadaten wie etwa Name, Typ, Node-Degree, Edge-Weight, URLs, usw. mit den einzelnen Nodes und Edges assoziiert. Diese zusätzlichen Informationen (= Attribute) können in weiterer Folge u.a. für Layout- oder Analysezwecke verwendet werden (siehe dazu auch Unterpunkt 4.8 bzw. 4.2.8). Weiters besteht die Möglichkeit zum Import bzw. Export von Attributen als einfache Textfiles (Formatspezifikationen siehe [Cytoscape, 2009g].

Zusatzinformationen zu markierten Nodes und Edges werden mithilfe des in Cytoscape integrierten *Attribute-Browsers* dargestellt (siehe Abbildung 4.2). Der Browser ermöglicht darüber hinaus auch das Kopieren, Löschen bzw. die Erstellung von Attributen. Diese können vom Typ *Integer, Float, String* oder *Boolean* sein.

4.2.4 Skalierbarkeit

Das Framework ist in der Lage, große Graphen bzw. Netzwerke zu visualisieren. Mit zunehmender Objektanzahl (Nodes, Edges,...) wächst auch der Ressourcenbedarf. Beim Starten von Cytoscape kann mit-

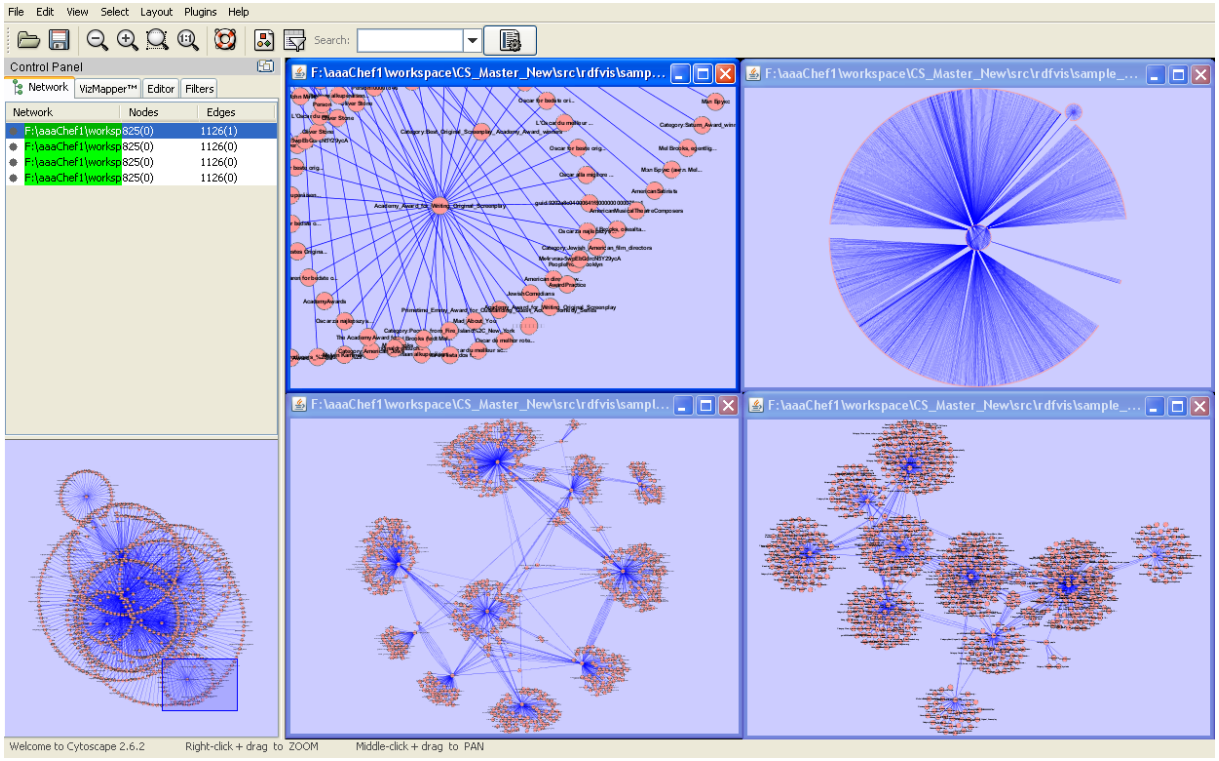


Abbildung 4.3: Mehrere Views desselben Graphen mit jeweils unterschiedlichen Layouts (Screenshot; [Cytoscape, 2009d]). Das Network-Panel rechts oben liefert Informationen zur Objektanzahl bzw. zeigt die hierarchische Anordnung der einzelnen Graphen. Das Overview-Window darunter zeigt die gesamte Graphstruktur. Das blaue Rechteck gibt Aufschluss über den aktuellen Zoombereich.

tels Commandline-Argument der maximal zur Verfügung stehende Arbeitsspeicher sowie der maximale Stack Space (u.a. für Layout-Operationen) vergrößert werden. Bei einem ausreichend schnellen Prozessor, einer leistungsstarken Grafikkarte, genügend Arbeitsspeicher sowie der richtigen Konfiguration aller Memory-Parameter können bis zu 10^6 Objekte visualisiert werden. Abbildung 4.4 zeigt die Cytoscape-Visualisierung eines Netzwerks bestehend aus 500.000 Nodes und 500.000 Edges (Voraussetzung: 5GB RAM, Java 64-Bit Version).

In Cytoscape werden Graphen und Netzwerke eingelesen und in den Speicher geladen. Ist die Anzahl der Objekte innerhalb eines (konfigurierbaren) Bereichs, erfolgt die automatische Generierung der Graphdarstellung (= View). Bei einer größeren Objektanzahl wird der Graph zur weiteren Bearbeitung vorerst im Speicher gehalten. Somit können als Vorverarbeitungsschritt Operationen zur Komplexitätsreduktion durchgeführt oder Sub-Graphen extrahiert werden. [Cytoscape, 2009g]

Test größerer Graphen

In Abbildung 4.5 finden sich weitere, mithilfe von Cytoscape generierte Darstellungen größerer Graphen. Die dabei visualisierten Datensätze I bis III sind im Cytoscape Download-Package als Beispielgraphen inkludiert. Die restlichen Datensätze stammen aus [DBpedia, 2009]. Tabelle 4.1 beinhaltet statistische Informationen zu den einzelnen Datensätzen (darunter die Anzahl der Graphelemente) sowie die zum Import und Layout (*yFiles Organic Layout*) benötigte Zeit. Insgesamt wurde der Import und die Darstellung von sechs unterschiedlich großen Graphen getestet. Im Schnitt benötigte Cytoscape für die Visualisierung (Import und Layout) von Graphen mit bis zu ca. 100.000 Elementen (Nodes + Edges) 22 Sekunden (Konfiguration: Intel Dual-Core 1.66GHz, 1GB RAM; 512MB Speicherzuweisung, 5MB Stack Size; Cytoscape Version 2.6.2). Aus der Tabelle geht auch hervor, dass eine größere Zahl an Attributen pro

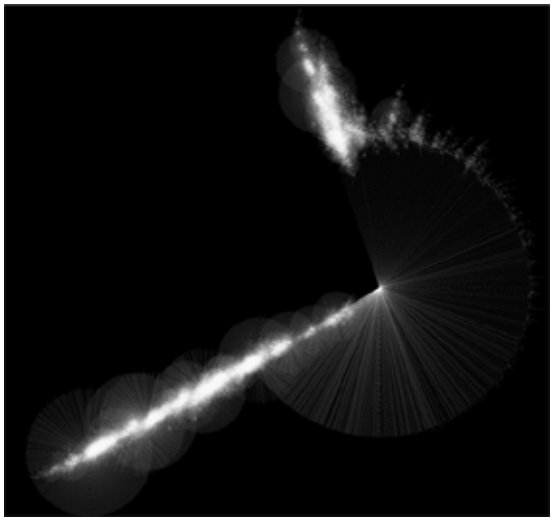


Abbildung 4.4: Visualisierung eines großen Graphen (10^6 Elemente) mithilfe von Cytoscape (Quelle: [Cytoscape, 2009g])

	Datensatz I	Datensatz II	Datensatz III	Datensatz IV	Datensatz V	Datensatz VI
# Nodes	20.000	24.000	21.000	16.000	120.000	590.000
# Edges	32.000	62.000	70.000	22.000	75.000	392.000
# Attribute pro Node/Edge	2	2	2	15	3	3
Labels	nein	nein	ja	ja	ja	ja
Import ohne View	6 Sek.	15 Sek.	20 Sek.	21 Sek.	10 Sek.	30 Sek.
Import mit View (Grid Layout)	20 Sek.	24 Sek.	24 Sek.	23 Sek.	78 Sek.	-
Organic Layout	36 Sek.	40 Sek.	35 Sek.	39 Sek.	140 Sek.	-

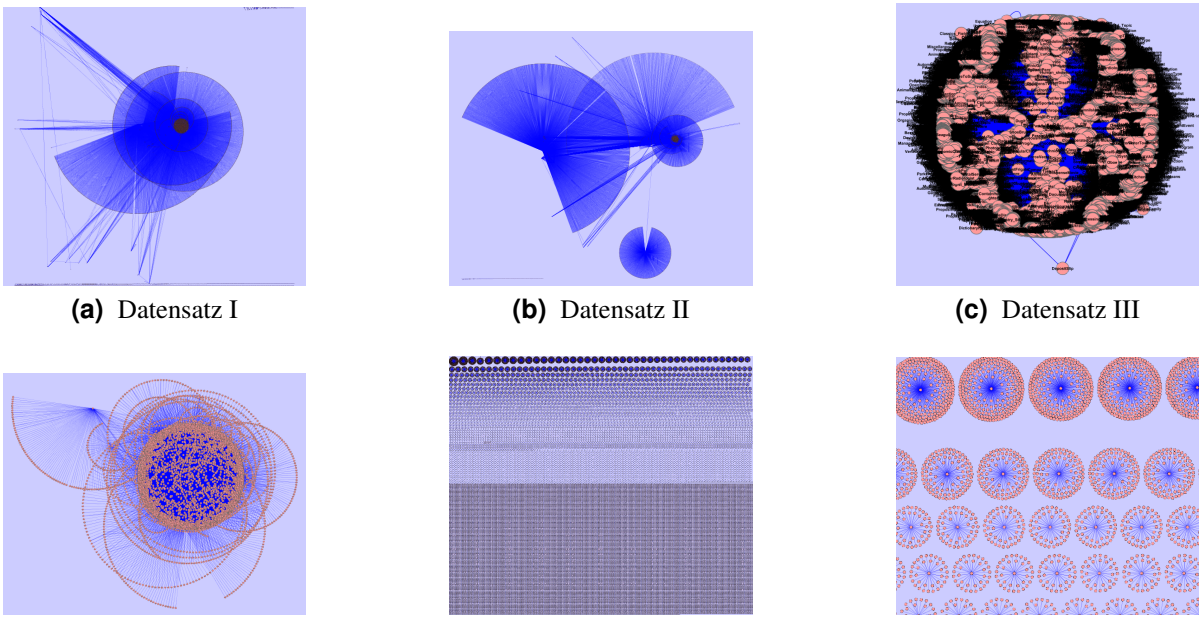
Tabelle 4.1: Visualisierung größerer Graphen in Cytoscape. Konfiguration: Intel Dual-Core 1.66GHz, 1GB RAM; 512MB Speicherzuweisung, 5MB Stack Size; Cytoscape Version 2.6.2)

Node bzw. Edge die Performance verschlechtert (siehe Datensatz IV). Datensatz VI (ca. 1.000.000 Elemente) konnte zwar importiert werden, die Generierung eines Views war jedoch mit dieser Konfiguration bzw. den vorhandenen Hardwareressourcen nicht möglich.

Wie in Abbildung 4.5 ersichtlich, kommt es aufgrund der höheren Anzahl an Elementen bei der Graphdarstellung zu Überlagerungen. Einzelne Nodes und Edges sind mitunter nicht mehr erkennbar. Speziell die Berücksichtigung von Labels (siehe Datensatz III) führt zu unübersichtlichen Ergebnissen.

4.2.5 Rendering Engine

Ab Version 2.3 verfügt Cytoscape über eine Rendering Engine, welche die effiziente, interaktive Darstellung großer Graphen mit mehr als 10.000 Knoten ermöglicht. Die Engine basiert auf einer Level-Of-Detail Technik (*LOD*) und stellt eine effektive Möglichkeit zur Reduktion der visuellen Komplexität dar. Die aktuelle Detailstufe ist dabei von der Anzahl der darzustellenden Nodes und Edges abhängig. Bei Überschreitung gewisser Grenzwerte (LOD-Werte), wird eine gröbere Detailstufe gewählt. Je niedriger die LOD-Werte, desto besser die Performance bei Layout, Interaktion, etc. Die Grenzwerte sind standardmäßig konfiguriert, können jedoch vom User geändert werden. Folgende LOD-Werte bestimmen



(a) Datensatz I **(b)** Datensatz II **(c)** Datensatz III

(d) Datensatz IV **(e)** Datensatz V (gesamter Graph) **(f)** Datensatz V (Ausschnitt)

Abbildung 4.5: Beispiele für Visualisierungen größerer Graphen mithilfe von Cytoscape ([Cytoscape, 2009d]). (a) 20.000 Nodes, 32.000 Edges (b) 24.000 Nodes, 62.000 Edges (c) 21.000 Nodes, 70.000 Edges (d) 16.000 Nodes, 22.000 Edges (e) 120.000 Nodes, 75.000 Edges; (f) Zoom auf eine bestimmte Region des Graphen

den aktuellen Detailgrad und beeinflussen dadurch die Visualisierung:

- *Detail Threshold*: Überschreitet die Summe aus Nodes und Edges diesen Wert (Default: 2000), werden Nodes nicht in ihrer aktuellen Form, sondern als Quadrate bzw. Edges ausschließlich als gerade Linien dargestellt. Zudem wird die Anti-Aliasing Funktion deaktiviert.

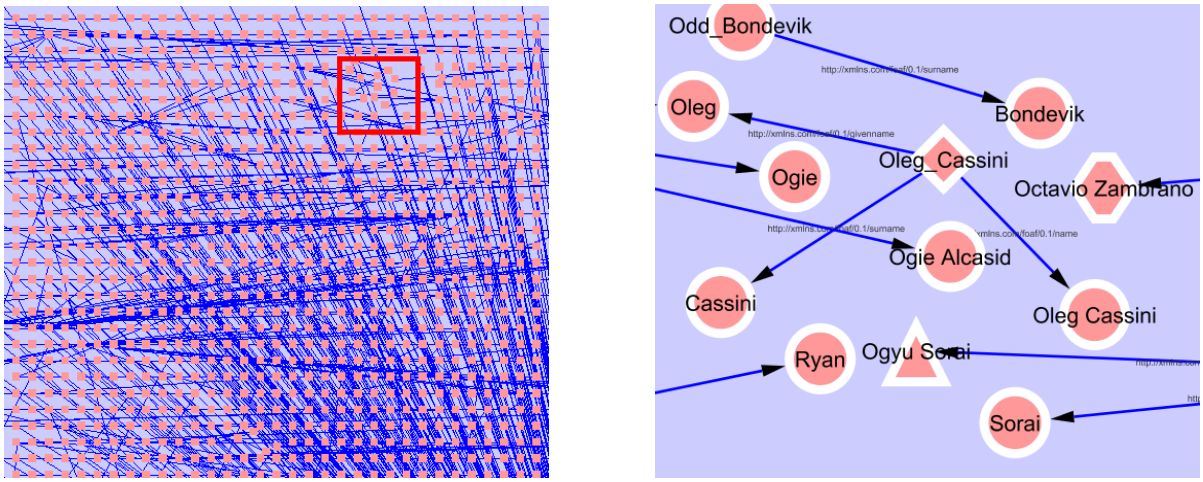
- *Node-/Edge-Label Threshold*: Ist die Node- bzw. Edge-Anzahl größer als dieser Wert, wird auf die Anzeige von Node- und/oder Edge-Labels verzichtet. Da die Darstellung von Text bzw. die Platzierung von Labels ein vergleichsweise aufwändiger Vorgang ist, liegt der Defaultwert bei 100 Nodes bzw. 150 Edges.

- *Node-Border Threshold*: Bei Überschreitung (Anzahl der Nodes) werden die Ränder der Nodes nicht mehr dargestellt.

- *Edge-Arrow Threshold*: Übersteigt die Anzahl an Edges diesen Wert, werden Kanten nicht mehr mittels Pfeilen sondern als einfache, gerade Linien dargestellt.

LOD-Werte haben entscheidenden Einfluss auf die Performance. Bei der Arbeit mit größeren Graphen empfiehlt es sich daher diesen Parametern möglichst kleine Werte zuzuordnen. Die Level-Of-Detail Berechnungen können bei Bedarf auch deaktiviert werden, wodurch unabhängig von der Objektanzahl alle Details des Graphen angezeigt werden. [Cytoscape, 2009g]

Abbildung 4.6 demonstriert die Verwendung der LOD-Technik anhand eines Beispielgraphen. Aufgrund der hohen Objektanzahl erfolgt in der groben Darstellung die Repräsentation der Nodes als Quadrate. Weiters werden Labels und Edge-Arrows ausgeblendet. Nach Zoom auf einen kleinen Teilbereich des Graphen werden auch Details, wie etwa Node-Shapes oder Labels, dargestellt.



(a) Grobe Detailstufe **(b)** Feine Detailstufe

Abbildung 4.6: Demonstration der Level-Of-Detail Technik von Cytoscape ([Cytoscape, 2009d]). (a) Grobe Darstellung eines größeren Graphen aufgrund Überschreitung von LOD-Grenzwerten. (b) Erst nach Zoom auf den rot markierten Bereich (= weniger darzustellende Objekte) werden Details sichtbar (unterschiedliche Node-Shapes, Node-Borders, Node- bzw. Edge-Labels, Edge-Arrows).

4.2.6 Interaktion

Cytoscape implementiert eine breite Palette an Features zur Interaktion (siehe auch [Cytoscape, 2009g]). Die wichtigsten davon sind:

- **Zoomable User Interface**
  Zur Navigation bzw. Betrachtung des Graphen bzw des Netzwerks werden sowohl Zooming- als auch Panning-Techniken unterstützt (Zoom auf ausgewählte Region, Zooming mittels Mouse-Gesten,...).

- **Edit**
  Nodes,Edges sowie ganze Netzwerke können hinzugefügt, entfernt, kopiert (geklont) bzw. verschoben werden. Weiters werden Features zur Manipulation der visuellen Repräsentation von Objekten (z.B. Farbe, Größe, Form,...) sowie zur manuellen Erstellung eines Layouts bereitgestellt.

- **Overview**
  Ein Overview-Window (*Bird's Eye-View*) zeigt den gesamten Graphen in einer Uebersichtsdarstellung (siehe Abbildung 4.2 sowie Abbildung 4.3). Darüber hinaus wird mit dessen Hilfe die Navigation einer großen Graphdarstellung erleichtert.

- **Selection**
  Es können sowohl einzelne als auch mehrere Nodes und Edges gleichzeitig markiert werden. Neben den standardmäßigen Selektionsmechanismen können Objekte auch auf Basis von Attributen bzw. topologischen Eigenschaften ausgewählt werden (z.B. Selektion von Nodes/Edges aufgrund des Namens, Selektion aller durch eine spezielle Kante verbundenen Knoten, Selektion aller an einen speziellen Knoten angrenzenden Kanten, etc.). Siehe dazu auch Unterpunkt 4.2.8;

- **Context**
  Bei Rechts-Klick auf ein Objekt (Node oder Edge) werden zusätzliche Informationen in einem Kontextmenü angezeigt.

4.2.7 Visuelle Repräsentation

Das Framework umfasst eine Vielzahl verschiedener Features zur Manipulation der visuellen Graph- bzw. Netzwerk-Repräsentation. Die meisten Mechanismen werden von Plugins, welche in neueren Versionen von Cytoscape standardmäßig inkludiert sind, bereitgestellt.

Automatische Layout-Algorithmen

Cytoscape unterstützt eine Reihe von Algorithmen und Techniken zur automatisierten Darstellung eines Graphen im zweidimensionalen Raum. Die Algorithmen stammen aus drei unterschiedlichen Quellen und werden dementsprechend gruppiert. Die einzelnen Implementierungen unterschieden sich, abgesehen von den verschiedenen Darstellungsformen, vor allem hinsichtlich der Performance beim Layout größerer Graphen. Die nachfolgende Auflistung (sortiert nach Herkunft) bietet einen Überblick über die wichtigsten Layout-Algorithmen (siehe auch [Cytoscape, 2009g]). Insgesamt umfasst Cytoscape an die 30 verschiedenen Layouts.

- **Cytoscape Layouts**
  Diese Algorithmen wurden speziell für Cytoscape implementiert und sind voll in das Framework integriert. Die Implementierung erlaubt die Modifikation/Anpassung verschiedener Parameter. Layouts können entweder auf alle oder nur auf markierte Nodes/Edges angewandt werden. Weiters besteht bei den meisten Layouts die Möglichkeit, Node- oder Edge-Attribute zu berücksichtigen. Der Source Code jedes Algorithmus ist im Cytoscape-Package enthalten. Nachfolgend werden einige der Layouts kurz vorgestellt.

  Das *Grid Layout* ist eine der simpelsten aber auch effizientesten Darstellungsformen. Nodes werden dabei in einem quadratischen Raster angeordnet. Nach dem Einlesen eines Graphen/eines Netzwerks wird standardmäßig dieses Layout berechnet.

  Das *Force-Directed Layout* basiert auf einem Algorithmus aus dem Prefuse-Toolkit (siehe [Heer, 2008b]) und ermöglicht ein schnelles Layout selbst größerer Graphen. Weiters können Edge-Attribute als Gewichtung verwendet werden.

  Das *Spring-Embedded Layout* ist eine Variation des Kamada- Kawai Algorithmus. Das Layout kann auf das gesamte Netzwerk oder nur auf ausgewählte Teile angewandt werden.

  Weitere Cytoscape Layouts wären u.a. *Attribute Circle Layout, Group Attributes Layout, Hierarchical Layout, Circular Layout, Inverted Self-Organising Map Layout, etc.*

- **JGraph Layouts**
  Cytoscape beinhaltet ebenfalls eine Auswahl an JGraph Layout-Algorithmen, darunter *Circle Layout, Radial Tree Layout, Sugiyama Layout, Spring-Embedded Layout,* etc. JGraph ist eine Java Open Source Software zum Layout bzw. zur Visualisierung von Graphen (Details siehe [JGraph, 2009]). JGraph Layouts eignen sich in erster Linie zur Darstellung kleinere Graphen da die Algorithmen mitunter schlecht skalieren.

- **yFiles Layouts**
  yFiles ([yWorks, 2009c]) ist eine kommerzielle Layout-Bibliothek. Die Layouts können innerhalb von Cytoscape frei verwendet werden, jedoch ist kein Source Code verfügbar. Die Nutzung im Rahmen einer Implementierung (z.B. als Bibliothek bzw. Source Code) ist somit nicht möglich. Im Vergleich zu JGraph Layouts und Cytoscape Layouts liefern Algorithmen dieser Bibliothek die beste Performance (u.a. hinsichtlich Berechnungszeit, Skalierbarkeit,...). Die wichtigsten Darstellungsformen wären *Circular Layout*, *Hierarchical Layout* und *Organic Layout* (eine Spring-Embedded Variation zur Darstellung der Clusterstruktur eines Graphen).

Abbildung 4.7 zeigt verschiedene Layouts eines Beispielgraphen (ca. 2000 Elemente). Während die Berechnung von yFiles bzw. Cytoscape Layouts innerhalb weniger Sekunden durchgeführt wurde, benötigte das JGraph Circle Layout bei gleicher Ausgangslage in etwa 6 Minuten. (Konfiguration: Intel Dual-Core 1.66GHz, 1GB RAM; 512MB Speicherzuweisung, 5MB Stack Size; Cytoscape Version 2.6.2)

Manuelles Layout

Neben automatisch generierten Layouts existiert auch die Möglichkeit, Nodes manuell anzuordnen bzw. die Darstellung von Edges zu modifizieren (z.B. anstatt einer Geraden eine gekrümmte Linie zu zeichnen). Wie zuvor bereits erwähnt, können Nodes und Edges hinzugefügt, kopiert, gelöscht und verschoben werden. Zudem finden sich Features zur horizontalen oder vertikalen Ausrichtung selektierter Nodes sowie Tools zur Rotation bzw. Skalierung des gesamten Netzwerks (siehe Abbildung 4.2). Nähere Informationen dazu finden sich im User Manual [Cytoscape, 2009g].

Visual Styles

Attribute von Nodes bzw. Edges (z.B. Name, Typ, Grad, Gewichtung, etc.) können als visuelle Eigenschaften (Farbe, Form, Größe, Transparenz, etc.) kodiert werden. Datenattribute werden somit auf visuelle Attribute gemappt. Die Zusammenfassung mehrerer visueller Kodierungen ergibt einen *Visual Style*. Ein spezielles Cytoscape-Tool ermöglicht die Erstellung, Modifikation und Zuweisung solcher Visual Styles. U.a. können folgende visuelle Eigenschaften kontrolliert werden:

- *Nodes*: Farbe, Größe, Transparenz, Form, Ränder, etc.
- *Edges*: Farbe, Linientyp, Linienbreite, Transparenz, Aussehen/Farbe von Source- bzw. Target-Arrows, etc.
- *Labels*: Farbe, Schriftart, Größe, Position, etc.
- *Global*: Hintergrundfarbe, Selektionsfarbe, etc.

Mittels Visual Styles lassen sich unterschiedliche Aspekte des Netzwerks hervorgehoben, ausklammern bzw. zusätzliche Informationen kodieren. Beispiele: Unterschiedliche Node-Shapes stehen für unterschiedliche Klassen; verschiedene Linientypen repräsentieren unterschiedliche Relationen; der Node-Degree bestimmt die Größe der Nodes; das Kantengewicht bestimmt die Größe oder Farbe von Labels, usw.

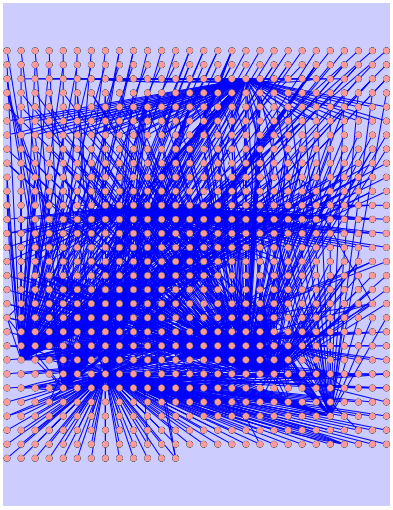
Abbildung 4.8 zeigt einen Graphen, welchem unterschiedliche Visual Styles zugeordnet wurden. [Cytoscape, 2009g]

4.2.8 Analyse

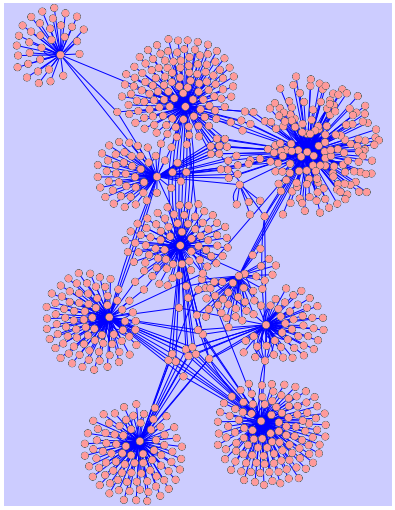
Standardmäßig implementiert Cytoscape Filter- und Such-Funktionalitäten sowie einige grundlegende Features zur Analyse der Graphstruktur. Eine Vielzahl von Analyse-Plugins erweitert den Funktionsumfang. Einige relevante Plugins zur Netzwerk- bzw. Graphanalyse werden in Abschnitt 4.3 vorgestellt.

Filter

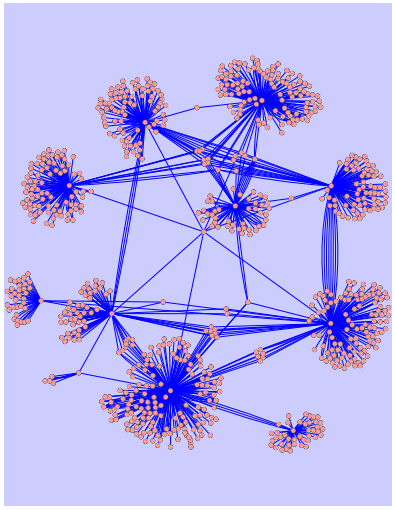
Mittels Filter lassen sich jene Nodes und Edges auswählen, welche spezielle Attribute aufweisen, einem spezifischen Muster entsprechen oder Werte innerhalb eines definierten Bereichs besitzen. Selektierte Objekte können in Folge ausgeblendet oder aber extrahiert und anschließend in einem neuen View



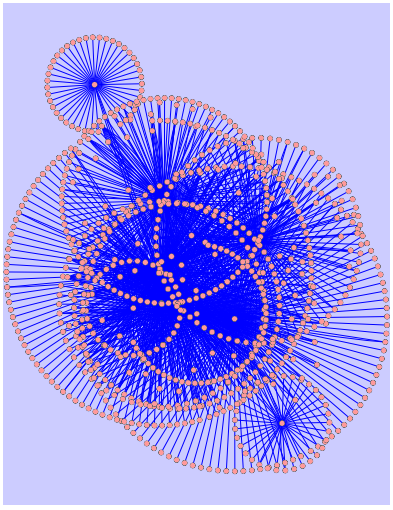
(a) Grid Layout



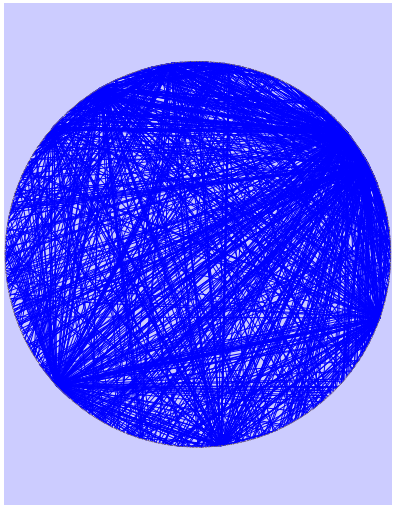
(b) Force-Directed Layout



(c) Spring-Embedded Layout

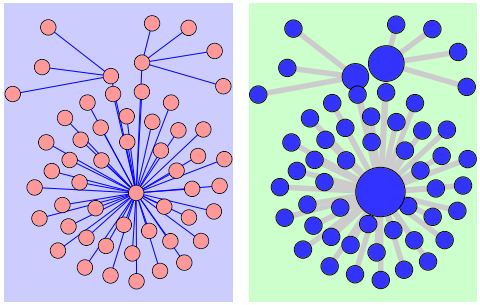


(d) Organic Layout



(e) Circle Layout

Abbildung 4.7: Verschiedene in Cytoscape verfügbare Layouts am Beispiel eines Graphen mit ca. 2000 Elementen (Screenshots; [Cytoscape, 2009d]) (a) Standard Grid Layout (Cytoscape Layout); Berechnungszeit: weniger als 1 Sekunde (b) Force-Directed Layout mit Kantengewichtung (Cytoscape Layout); Berechnungszeit: ca. 5 Sekunden (c) Spring-Embedded Layout (Cytoscape Layout); Berechnungszeit: ca. 17 Sekunden (d) Organic Layout (yFiles Layout); Berechnungszeit: weniger als 1 Sekunde (e) Circle Layout (JGraph Layout); Berechnungszeit: ca. 360 Sekunden



(a) Visual Style I **(b)** Visual Style II

Abbildung 4.8: Unterschiedliche Darstellung der Graphelemente mithilfe von Visual Styles. (a) Standard Visual Style (b) Benutzerdefinierter Visual Style (u.a. Mapping von Node-Degree auf Node-Size bzw. alternativer Hintergrund)

dargestellt werden. Ein GUI erleichtert die Definition, Anwendung und Verwaltung von Filtern. Die Anwendung eines Filters erfolgt bei kleineren Graphen (Default: 1000 Elemente) dynamisch, bei größeren Graphen mittels *Apply*-Button. Grundsätzlich wird zwischen vier Filter-Typen unterschieden ([Cytoscape, 2009g]):

- *Simple/Complex Filter*: Filtering aufgrund von Node- oder Edge-Attributen (AND- bzw. OR-Verknüpfung)
- *Topology Filter*: Auswahl von Nodes, welche eine spezifische Anzahl an Nachbar-Nodes innerhalb einer gewissen Distanz aufweisen
- *Interaction Filter*: Auswahl von Nodes/Edges auf Basis der Eigenschaften von benachbarten Nodes/Edges
- *Basic Filter*: u.a. String-, Numerical- und Boolean-Filter

Suche

Mittels *QuickFind*-Feature (siehe Abbildung 4.2) kann textbasiert nach Nodes oder Edges gesucht werden. Weiters besteht die Möglichkeit zur Konfiguration verschiedener Parameter. Gefundene Nodes/Edges werden markiert und anschließend herangezoomt.

Weitere Analysemöglichkeiten

Wie bereits erwähnt, verfügt Cytoscape über einige einfache Features zur Analyse der Graphstruktur. Beispielsweise können die unmittelbaren Nachbar-Nodes bzw. die angrenzenden Kanten von aktuell markierten Nodes ermittelt werden. Weiters besteht die Möglichkeit alle, durch eine ausgewählte Kante verbundenen Knoten zu suchen.

4.3 Plugins

Es existiert eine Vielzahl von Plugins zur Erweiterung des Funktionsumfangs von Cytoscape. Die meisten Plugins sind via Website-Download (siehe [Cytoscape, 2009c]) als *jar*-File bzw. als gezipptes Source Code Package verfügbar. Die Einbindung in das Framework erfolgt durch Kopieren des jar-Files in den Plugin-Ordner der Cytoscape-Installation. Alternativ können Plugins auch mithilfe des *Plugin-Managers* geladen, aktualisiert oder gelöscht werden. Der Nachteil dieser Methode ist, dass nicht alle auf der Website angeführten Erweiterungen abrufbar sind.

Die Cytoscape-Website listet derzeit 76 verschiedene Plugins, geordnet nach den Kategorien *Analysis, Network I/O, Network Inference, Functional Enrichment, Communication/Scripting* und *Others* [Cytoscape, 2009c]. Nachfolgend werden einige relevante, frei verfügbare Erweiterungen vorgestellt. Eine genauere Betrachtung der im Zuge der praktische Implementierung verwendeten Plugins findet sich im nächsten Kapitel.

MetaNodePlugin2

Mithilfe dieser Erweiterung können markierte Nodes zu einem Meta-Node zusammengefasst werden. Weiters unterstützt das Plugin benutzergesteuerte Expand- und Collapse-Funktionalitäten sowie eine Reihe von Konfigurationsmöglichkeiten. Meta-Nodes werden von Cytoscape wie normale Nodes behandelt, d.h. sie besitzen beispielsweise dasselbe Aussehen, können Attribute aufweisen oder werden von Layout-Algorithmen als Standard-Nodes interpretiert. Mehrere Meta-Nodes können wiederum zu einem neuen Meta-Node zusammengefasst werden, wodurch die Bildung einer hierarchischen Struktur des Netzwerks ermöglicht wird.

Das Plugin ist Open Source (LGPL), sowohl jar-File als auch der aktuelle Source Code sind frei verfügbar (siehe [Morris and Avila-Campillo, 2009], [Morris, 2009] bzw. [Cytoscape, 2009c]). Das Meta-Node Plugin bildet in weiterer Folge die Basis zur Implementierung einer interaktiven, dynamischen Aggregierungskomponente. Details dazu finden sich im nächsten Kapitel. Das grundlegende Konzept von Meta-Nodes wurde in Kapitel 2, Abschnitt 2.3 behandelt.

ClusterMaker

Das Plugin implementiert mehrere Clustering-Techniken, darunter *Hierarchical-, K-Means-, Force-* sowie *MCL-Clustering*. Verschiedene Parameter können mittels Interface konfiguriert werden. Weiters finden sich Features zur Visualisierung der Clustering-Ergebnisse. Der aktuelle Source Code sowie jar-Archive sind frei verfügbar (LGPL). [Morris et al., 2009] [Cytoscape, 2009c]

MCODE

MCODE (Molecular Complex Detection) ist ein Plugin zur Auffindung von Regionen mit einer hohen Verbindungsdichte. Der implementierte Clustering-Algorithmus wurde bereits in Kapitel 2, Abschnitt 2.3 beschrieben. Das Plugin erlaubt die Konfiguration verschiedener Parameter zur Feinabstimmung des Clustering-Prozesses. Die gefundenen Sub-Netzwerke werden in einem eigenen Fenster grafisch dargestellt und können nachträglich angepasst bzw. auch exportiert werden.

Die Software ist unter LGPL verfügbar und kann via SVN (aktueller Source Code) oder als jar-Archiv von der Cytoscape-Plugin Website heruntergeladen werden (siehe [Cytoscape, 2009c] bzw. [Bader, 2009a]). Weitere Details zu MCODE finden sich unter [Bader and Hogue, 2003].

NetworkAnalyzer

Das Plugin dient zur Berechnung und Visualisierung graphtheoretischer bzw. netzwerkspezifischer Parameter, Eigenschaften und Statistiken. Berechnungen können sowohl für gerichtete als auch ungerichtete Graphen durchgeführt werden. Darüber hinaus besteht auch die Möglichkeit zur Modifikation des Netzwerks (z.B. Entfernen aller Self-Loops,...). Beispiele für Parameter wäre u.a.: *Degree, Clustering-Koeffizient, Dichte, Anzahl verbundener Komponenten, Radius, durchschnittliche Anzahl an Nachbar-Nodes, Anzahl an Self-Loops, Degree Distribution, topologische Koeffizienten,* usw.

Die Software kann direkt von der Projekt-Website als jar-Archiv heruntergeladen werden. Es ist jedoch kein Source Code verfügbar. [Assenov et al., 2009]

RDFScape

RDFScape ist ein Plugin, welches speziell zur Analyse und Visualisierung von Ontologien aus dem Bereich des Semantic Web entwickelt wurde. Ontologien können als *RDF(S)*- oder *OWL*-Files geladen und anschließend analysiert werden. Der Fokus von RDFScape liegt auf folgenden Features:

- *Ontologie-Queries*: Unterstützung von SPARQL, RDQL (Abfragesprachen für RDF), Visual Queries (Abfrage auf Basis visueller Muster) sowie Class Queries (Abfrage auf Basis von Klassenzugehörigkeit)
- *Reasoning*: Anwendung von RDFS/OWL-Ableitungsregeln (Inference Rules)
- *Visualisierung*: Berücksichtigung semantischer Aspekte bei der Darstellung der Abfrageergebnisse
- *Ontologie-Mapping*: Abbildung einer Ontologie auf ein Cytoscape-Netzwerk

RDFScape ist Open Source Software (LGPL; Source Code, jar-Files). Weitere Details, User Manuals und Downloadmöglichkeiten finden sich unter [Splendiani, 2008].

4.4 Fazit und weitere Schritte

Cytoscape ist eine mächtige Visualisierungs- und Analysesoftware. Der große Funktionsumfang (hinsichtlich Interaktion, Analyse,...), effiziente Layout-Algorithmen, Möglichkeiten zur Behandlung großer Graphen bzw. Features zur Reduktion der visuellen Komplexität (LOD, Filter,...), Aktualität/permanente Weiterentwicklung, frei verfügbarer Java Source Code sowie eine erweiterbare Architektur sind einige der größten Pluspunkte. In Verbindung mit einigen der zuvor präsentierten Plugins stellt Cytoscape bereits ein umfangreiches Framework zur Visualisierung von großen Graphen und Netzwerken dar.

4.4.1 Limitations - Requirements

Einige der Funktionalitäten von Cytoscape wurden jedoch gezielt für Problemstellungen aus dem Bereich der Bioinformatik entwickelt. Dies betrifft vor allem Features zur Analyse von Graphen sowie Import-/Export-Möglichkeiten. Auch ein großer Teil der verfügbaren Plugins widmet sich in erster Linie Aspekten aus der biologischen Forschung (beispielsweise Darstellung und Analyse von Protein- oder Gen-Interaktionsnetzwerken, etc.).

Im Hinblick auf die Darstellung bzw. der weiteren Behandlung semantischer Graphen im RDF-Format sind Cytoscape somit gewisse Grenzen gesetzt. Diesbezüglich fehlende Features werden auch von den verschiedenen Plugins nicht, oder nur teilweise unterstützt. Im Anschluss erfolgt die Diskussion einiger Grenzen bzw. Nachteile von Cytoscape. Im Zuge dessen ergeben sich auch die wichtigsten Anforderungen an eine neue Visualisierungskomponente.

Import von RDF-Dateien

Wie in Abschnitt 4.2.2 bereits erwähnt, unterstützt Cytoscape eine Reihe von Datenformaten zum Import von Netzwerken und Graphen. Es besteht jedoch keine direkte Möglichkeit Dateien im RDF-Format (*RDF/XML, N-Triple, Notation3, RDF-S, OWL,...*) einzulesen. Mithilfe des *Table Import* Features (siehe 4.2.2) können RDF-Files indirekt importiert werden. Dazu müssen u.a. Parsing-Parameter (z.B. Delimiter) und Spaltenzuweisungen (Source-Node, Edge, Target-Node) manuell festgelegt werden. Diese Variante ist mitunter jedoch fehleranfällig (falscher Delimiter, Probleme beim Parsen, etc.) und erfordert einiges an Konfigurationsaufwand bzw. Hintergrundwissen. Oberste Priorität hat somit die Implementierung einer Funktion zum direkten, fehlerfreien und einfachen **Import von RDF-Daten** in Cytoscape.

Reduktion der visuellen Komplexität

Semantische Graphen bestehen oft aus einer großen Anzahl an Nodes, Edges und Labels. Obwohl Cytoscape u.a. dank skalierbarer Layout-Algorithmen und einer effizienten internen Behandlung der Netzwerkstruktur zur Visualisierung großer Graphen in der Lage ist, stellen unübersichtliche Darstellungen aufgrund von Überlagerungen einzelner Objekte (Nodes, Edges, Labels) nach wie vor ein Problem dar (siehe dazu auch Abbildung 4.5). Auch Performanceeinbußen (Verzögerungen bei Interaktion, lange Berechnungszeiten von Layouts,...) sind die Folge einer zu großen Zahl an darzustellenden Objekten.

Aus diesem Grund bedarf es Möglichkeiten zur Verringerung der visuellen Komplexität eines großen Graphen. Cytoscape unterstützt bereits Filtering bzw. eine LOD-Technik (siehe Abschnitt 4.2.8 bzw. 4.2.5). Darauf aufbauend sollte die neue Visualisierungskomponente folgende Features implementieren:

- In einem Vorverarbeitungsschritt sollten mittels **Aggregations- bzw. Clustering-Algorithmen** Muster, Ähnlichkeiten, Zusammenhänge bzw. Gruppen identifiziert werden. In diesem Zusammenhang stellt das unter Punkt 4.3 vorgestellte *ClusterMaker*-Plugin einige nützliche Funktionalitäten zur Verfügung.
- Anschließend sollte eine interaktive Darstellung mit **dynamischer (zoomabhängiger) Aggregation** generiert werden. Das *MetaNode*-Plugin (siehe Punkt 4.3) würde sich in diesem Fall als idealer Ausgangspunkt anbieten. Dieses müsste u.a. um eine zoomabhängige Expand- bzw. Collapse-Funktion erweitert werden.

Berücksichtigung semantischer Informationen

Prinzipiell können in Cytoscape mithilfe der in Abschnitt 4.2.3 vorgestellten Attribut-Funktionalität semantische Informationen zu Nodes und Edges abgespeichert werden. Dadurch besteht die Möglichkeit, Nodes und/oder Edges RDF-Typen, URIs oder Literal-Values als String- oder Integer-Attribute zuzuordnen. Darüber hinaus implementiert Cytoscape jedoch keine Features zur Darstellung, Analyse bzw. näheren Betrachtung semantischer (RDF-spezifischer) Aspekte. (Anmerkung: Das in 4.3 vorgestellte *RDFScape*-Plugin dient in erster Linie dem Querying bzw. Reasoning von RDF- bzw. OWL-Ontologien und ist im Kontext dieser Arbeit von geringerer Relevanz)

Folgende Requirements können somit hinsichtlich der Unterstützung semantischer Graphen definiert werden:

- Möglichkeit zur **Aggregation aufgrund semantischer Informationen**.
- **Berücksichtigung RDF-spezifischer Informationen** bei der Darstellung der Graphelemente bzw. beim Graph-Layout.
- **Bereitstellung nähere Informationen** zum aktuell importierten RDF-Datensatz.

View Coordination

Wie bereits erwähnt, ermöglicht Cytoscape die Generierung mehrerer Darstellungen (Views) desselben Graphen. Auf diese Weise können unterschiedliche Aspekte eines Datensatzes dargestellt und verglichen werden (z.B. verschiedene Layouts, etc.; siehe auch Abbildung 4.3). Die einzelnen Views sind jedoch nicht koordiniert, d.h. Änderungen in einem View haben keine Auswirkungen auf die übrigen Darstellungen. Eine **Koordination der einzelnen Cytoscape Views** bzw. eine **Koordination mit externen Visualisierungen** hinsichtlich User-Navigation (Zoomstufe, aktuell dargestellter Ausschnitt, etc.) und Zustand der dargestellten Objekte (Farbe, Selektion, etc.) stellt somit eine weitere sinnvolle, wenn auch nicht zwingend notwendige Anforderung im Rahmen der praktischen Implementierung dar.

4.4.2 Zusammenfassung

Das Ziel des praktischen Teils dieser Masterarbeit ist die Implementierung einer Komponente zur interaktiven Visualisierung und dynamischen Aggregierung von semantischen Graphen im RDF-Format. Wie aus der Diskussion der Grenzen von Cytoscape hervorgeht, sind dahingehend einige Modifikationen bzw. Erweiterungen notwendig. Die Anforderungen an diese, auf Cytoscape basierende neue Visualisierungskomponente können wie folgt zusammengefasst werden (die Reihenfolge der Auflistung spiegelt dabei die Implementierungspriorität wider):

- Import von RDF-Daten in Cytoscape
- Reduktion der visuellen Komplexität:
  - Preprocessing mittels Aggregations- bzw. Clustering-Techniken
  - Interaktive Darstellung mit dynamischer (zoomabhängiger) Aggregation
- Unterstützung semantischer Features:
  - Aggregation aufgrund semantischer Informationen
  - RDF-spezifische Darstellung
  - Bereitstellung von Informationen zum aktuellen RDF-Datensatz
- Koordination mehrerer Visualisierungen (optional)

Das nächste Kapitel widmet sich den Details der Umsetzung dieser Requirements und beleuchtet ausgewählte Aspekte der praktischen Implementierung.

Kapitel 5

Implementierung

Das Kapitel befasst sich mit den verschiedenen Aspekten der Implementierung einer neuen Visualisierungskomponente im Rahmen dieser Masterarbeit. Die theoretischen Grundlagen dazu wurden bereits in Kapitel 2 behandelt. Zu Beginn dieses Kapitels wird das implementierte Visualisierungstool kurz vorgestellt, gefolgt von einer Definition bzw. Erklärung der grundlegenden Ziele und Anforderungen. Den Hauptteil bildet die Betrachtung technischer Aspekte. Im Zuge dessen werden Architektur und Design, Aufbau und Komponenten, typische Abläufe, sowie ausgewählte Details der Implementierung näher beleuchtet. Abschließend werden die Ergebnisse bzw. besprochenen Themen dieses Kapitels in Form einer Gegenüberstellung von Requirements und implementierten Funktionalitäten nochmals zusammengefasst. In Anhang A findet sich weiters ein Benutzerhandbuch, welches die Bedienung bzw. die grundlegenden Features erläutert sowie Auskunft hinsichtlich Requirements, Installation und Start der Visualisierungssoftware gibt.

5.1 VIOSEG: Einleitung und Überblick

VIOSEG ist eine Open Source Software (*GNU Lesser General Public License*) zur interaktiven Visualisierung von semantischen Graphen im RDF-Format, mit der Möglichkeit zur dynamischen Aggregierung von Graphelementen. Der Name VIOSEG ist ein Akronym, zusammengesetzt aus den Anfangsbuchstaben der Bezeichnung ***VI****sualisation* ***O****f* ***SE****mantic* ***G****raphs*. Die Software basiert auf *Cytoscape*, einer Open Source Plattform für die Analyse und Visualisierung von komplexen Netzwerken und Graphen (siehe Kapitel 4 bzw. [Cytoscape, 2009d]). VIOSEG kann sowohl als eigenständige, plattformunabhängige Applikation gestartet, als auch in Form eines Plugins (jar-File) in Cytoscape integriert werden. Details dazu finden sich auch in Anhang A. Abbildung 5.1 zeigt die grafische Benutzerschnittstelle sowie einen mithilfe von VIOSEG visualisierten Graphen.

Die Software wurde vollständig in *Java* (Version 6) implementiert. Java ist eine objektorientierte Programmiersprache deren Syntax sich an jener von *C++* anlehnt (siehe [Sun, 2009]). Hauptgründe für die Wahl von Java waren einerseits dessen Konzept der Plattformunabhängigkeit, d.h. Programme können auf verschiedenen Systemen ohne weitere Anpassung oder Modifikation ausgeführt werden. Anderseits legte der, ebenfalls in Java geschriebene Source Code von Cytoscape die Verwendung dieser Programmiersprache im Rahmen der Implementierung von VIOSEG nahe. Als Entwicklungsumgebung diente *Eclipse*, Version 3. Eclipse ist ein freies, plattformunabhängiges Programmierwerkzeug, welches eine breite Palette an nützlichen Features für die Entwicklung von Java-Software bereitstellt (siehe [Eclipse, 2009]).

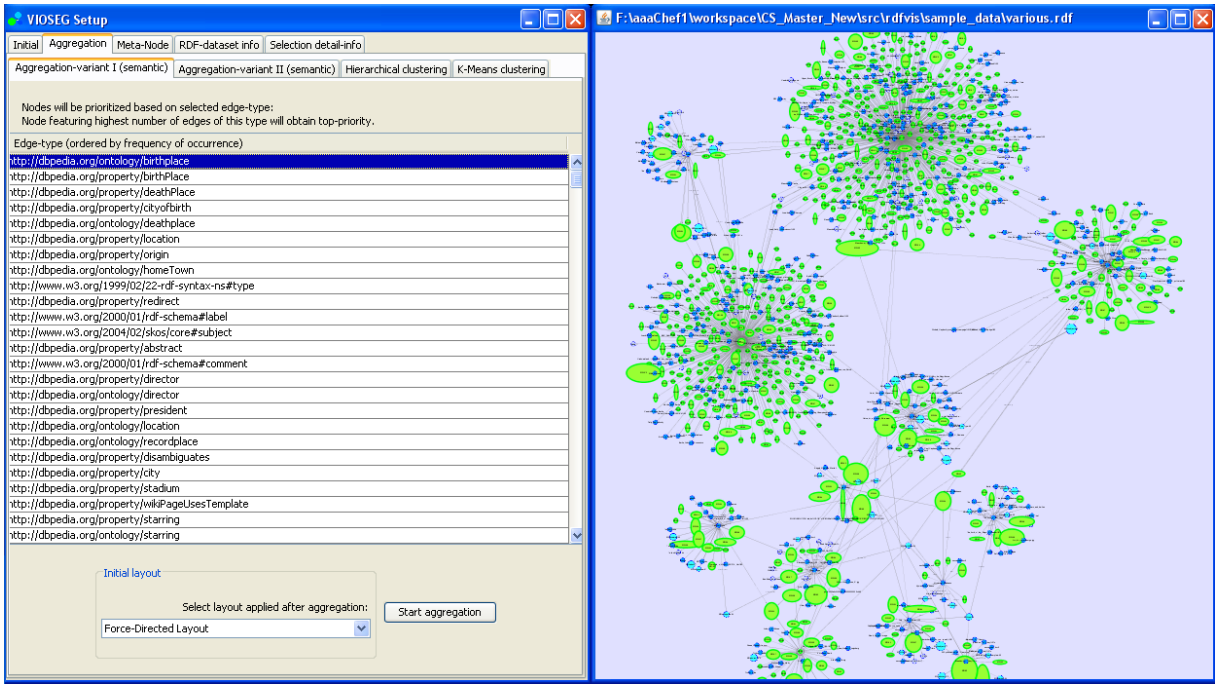


Abbildung 5.1: VIOSEG - Visualisation Of Semantic Graphs

5.1.1 Ziele und Anforderungen

Ziel des praktischen Teils dieser Masterarbeit war prinzipiell die Entwicklung eines Systems bzw. eines Tools zur Visualisierung von semantischen Graphen im RDF-Format mit zusätzlichen Features zur Reduktion der visuellen Komplexität von größeren Graphstrukturen. Im Zuge der Evaluierung bereits existierender Frameworks und Toolkits in Kapitel 3 stellte sich Cytoscape als mächtiges und umfangreiches Softwarepaket zur Visualisierung von Graphen und Netzwerken heraus. Nach Bewertung der Vor- und Nachteile bzw. des Erweiterungspotenzials jedes Visualisierungstools, wurde das Open Source Framework Cytoscape als Basis für die Entwicklung einer neuen Visualisierungskomponente ausgewählt. Kapitel 4 widmete sich der genaueren Betrachtung der Features und speziellen Eigenschaften des Softwarepakets. Nach Analyse von Stärken und Schwächen sowie Untersuchung der Grenzen und Einschränkungen von Cytoscape, erfolgte schließlich die Definition der grundlegenden Features, welche von einer neuen Visualisierungskomponente unterstützt werden sollten. Im Zuge dessen konnten folgende Requirements identifiziert werden:

- **RDF-Import**: RDF-Modelle sollten direkt und ohne weiteren Konfigurationsaufwand eingelesen und auch wieder exportiert (z.B. als Graph im *GML*-Format) bzw. in einem RDF-Datenformat abgespeichert werden können. RDF-relevante Datenformate wären *RDF/XML (.rdf), N-Triple (.nt)* sowie *Notation3 (.n3)*.

- **Visualisierung**: RDF-Modelle sollten nach dem Einlesen als Cytoscape Netzwerk interpretiert und in weiterer Folge in Form eines Graphen in ansprechender und übersichtlicher Weise visualisiert werden.

- **Interaktivität**: Die Visualisierungskomponente sollte grundlegende Funktionalitäten zur Interaktion (Navigation, Zoom, Pan, Edit, Suche,...) implementieren.

- **Reduktion der visuellen Komplexität**: Um auch auf herkömmlichen Rechnern mit einer Standard-Hardwarekonfiguration größere, komplexere Graphen darstellen zu können (u.a. hinsichtlich Echtzeitinteraktion, Übersichtlichkeit, Berechnungszeiten,...), sollte die Komponente Funktionen zur

Verringerung der visuellen Komplexität bereitstellen. Grundlegende Features wären dabei die Unterstützung von Filter bzw. Level-Of-Detail Techniken. Weiters sollten in einem Vorverarbeitungsschritt mittels Aggregations- bzw. Clustering-Algorithmen Muster, Ähnlichkeiten, Zusammenhänge bzw. Elementgruppen innerhalb des Graphen identifiziert werden. Auf Basis dessen sollte anschließend die Generierung einer interaktiven Darstellung mit dynamischer Aggregation erfolgen. Der auf diese Weise vereinfachte Graph könnte somit problemlos visualisiert werden.

- **Berücksichtigung semantischer Informationen**: Semantische Informationen aus den importierten RDF- Modellen sollten den einzelnen Nodes und Edges des Graphen zugeordnet werden können. In weiterer Folge sollten auch Möglichkeiten zur Speicherung bzw. Weiterverarbeitung dieser Informationen bereitgestellt werden. Aus diesem Grund wären u.a. Mechanismen zur Aggregation von Objekten aufgrund semantischer Informationen zu implementieren. Weiters sollten RDF-spezifische Informationen bei der Darstellung der einzelnen Graphelemente sowie dem Layout des gesamten Graphen berücksichtigt werden. So könnten beispielsweise unterschiedliche RDF-Typen auf unterschiedliche Node-Symbole gemappt bzw. Layouts unter Berücksichtigung semantisch gewichteter Kanten berechnet werden. Darüber hinaus müssten nähere Informationen zum aktuell geladenen RDF-Datensatz bereitgestellt werden (z.B. tabellarische Auflistung aller Properties, Literals,...).

- **GUI**: Ein grafische Benutzeroberfläche sollte die intuitive Bedienung sowie die einfache Nutzung der verschiedenen Features ermöglichen.

- **Optional**: Bei Bedarf und ausreichend Zeit könnten zudem noch eine View Coordination Funktion, spezielle Layout-Algorithmen, oder Features zur Behandlung dynamischer Graphstrukturen implementiert werden.

Ausgehend von diesen Anforderungen bzw. Features wurde VIOSEG entwickelt. In den nachfolgenden Abschnitten werden Details der Umsetzung der einzelnen Requirements sowie weitere wichtige Aspekte der Implementierung, wie etwa Funktionsweise, Konzepte, Aufbau und Struktur, sowie ausgewählte Details näher erläutert.

5.2 Architektur und Design

Die VIOSEG Implementierung besteht aus einer Reihe von Kernklassen sowie einigen erweiterten bzw. adaptierten Plugin-Klassen. Cytoscape dient als Framework, welches grundlegende Funktionen zur Verwaltung und Visualisierung von Netzwerken bzw. Graphen bereitstellt. Zusätzlich benötigte Funktionalitäten werden von externen Libraries zur Verfügung gestellt. Das modulare, objektorientierte Design soll den Überblick über die grundlegende Struktur der Implementierung ermöglichen bzw. das Verständnis des Zusammenspiels der einzelnen Komponenten erleichtern. Zudem sollten zukünftige Erweiterungen problemlos und ohne umfangreiche Modifikationen durchgeführt werden können.

Das System kann prinzipiell in folgende drei Hauptkomponenten unterteilt werden:

- VIOSEG Kern
- Cytoscape Framework
- Plugin/Library-Komponente

Der Kern bildet dabei die zentrale Einheit zur Steuerung der Abläufe sowie zur Unterstützung der Hauptfunktionen von VIOSEG. Das Cytoscape Framework dient als Basis und stellt grundlegende Mechanismen und Funktionen zur Behandlung von Graphen bereit. Die dritte Komponente umfasst erweiterte bzw. modifizierte Cytoscape-Plugins sowie Third-Party Libraries. Abbildung 5.2 zeigt die schema-

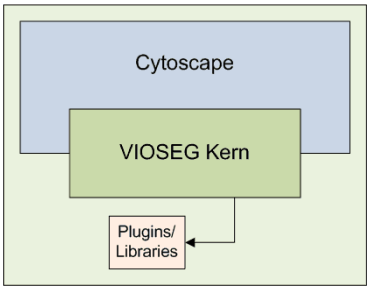


Abbildung 5.2: Grundstruktur bzw. Hauptkomponenten von VIOSEG

tische Darstellung der Hauptkomponenten bzw. die Beziehung zwischen Kern, Cytoscape und Libraries/Plugins. Abbildung 5.3 bietet eine Übersicht der involvierten Klassen und Komponenten sowie der wichtigsten Relationen der Implementierung. Die Ableitungshierarchie der einzelnen Klassen findet sich in Abbildung 5.4.

Nachfolgend werden die einzelnen Hauptkomponenten, ihre Aufgaben sowie deren interne Struktur näher betrachtet.

5.2.1 VIOSEG Kern

Diese Komponente bildet die zentrale Einheit der Implementierung. Der Kern stellt die Main-Features von VIOSEG zur Verfügung und hat somit folgende Hauptaufgaben:

- Steuerung der grundlegenden Abläufe und internen Vorgänge
- Einlesen bzw. Abspeichern von RDF-Modellen
- Durchführung von Aggregierungs- bzw. Clustering-Berechnungen
- Unterstützung einer dynamischen (zoomabhängigen) Aggregierung
- Spezifische visuelle Repräsentation des Graphen bzw. einzelner Graphelemente (u.a. hinsichtlich semantischer Informationen und Aggregationsstufe)
- Bereitstellung eines grafischen User-Interfaces (GUI) zur Nutzung bzw. Kontrolle der Main-Features von VIOSEG

Der Kern besteht aus einer Reihe von Klassen und Sub-Komponenten zur Umsetzung der einzelnen Aufgaben bzw. zur Bereitstellung der Hauptfunktionen von VIOSEG. Diese greifen auf Funktionen des Cytoscape Frameworks bzw. auf externe Bibliotheken und Plugins zu. Abbildung 5.5 zeigt den grundlegenden Aufbau sowie die einzelnen Sub- bzw. Kernkomponenten und deren Beziehungen.

Initializator

Der Initializator ist für den korrekten Start bzw. die Initialisierung des VIOSEG Tools verantwortlich. Der Prozess läuft dabei in mehreren Schritten ab: Nach der Initialisierung von Cytoscape bzw. dem Laden zusätzlicher Plugins erfolgt der Start der zentralen Kontrolleinheit (Controller). Involvierte Klassen sind u.a. `SemanticGraphVisualizer` und `RDFPluginAction` (siehe dazu auch Abschnitt 5.3.1).

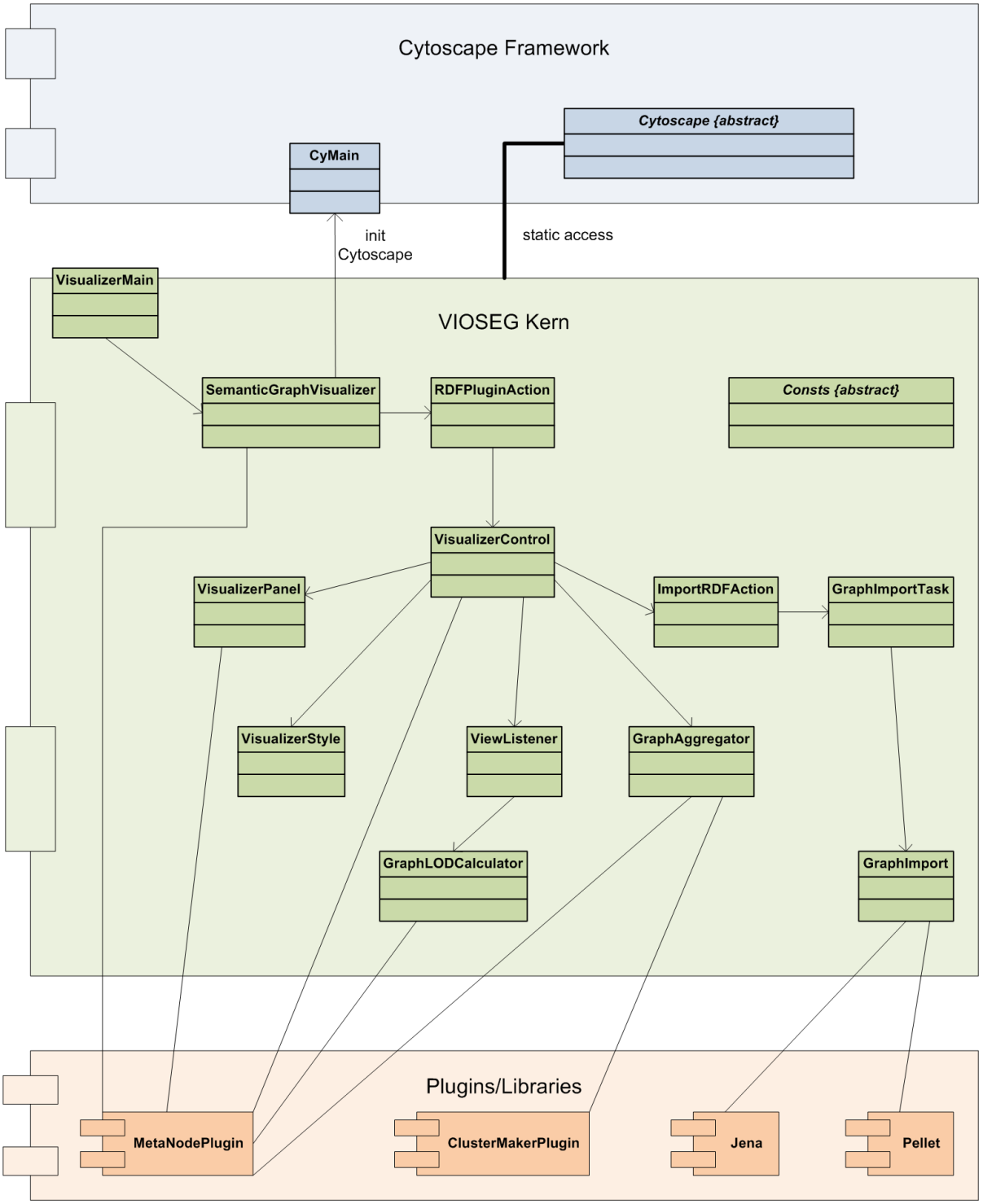


Abbildung 5.3: VIOSEG: Klassen, Komponenten und Relationen. Anmerkungen: `CyMain` ist die Klasse zum Start der Cytoscape Umgebung. Die statischen Methoden der abstrakten Klasse `Cytoscape` ermöglichen den Zugriff auf Cytoscape-Funktionalitäten (z.B. Erzeugung eines Netzwerks durch Aufruf `Cytoscape.getNetwork(...)`). Die abstrakte Klasse `Consts` definiert statische Methoden und Konstanten der VIOSEG Kernklassen.

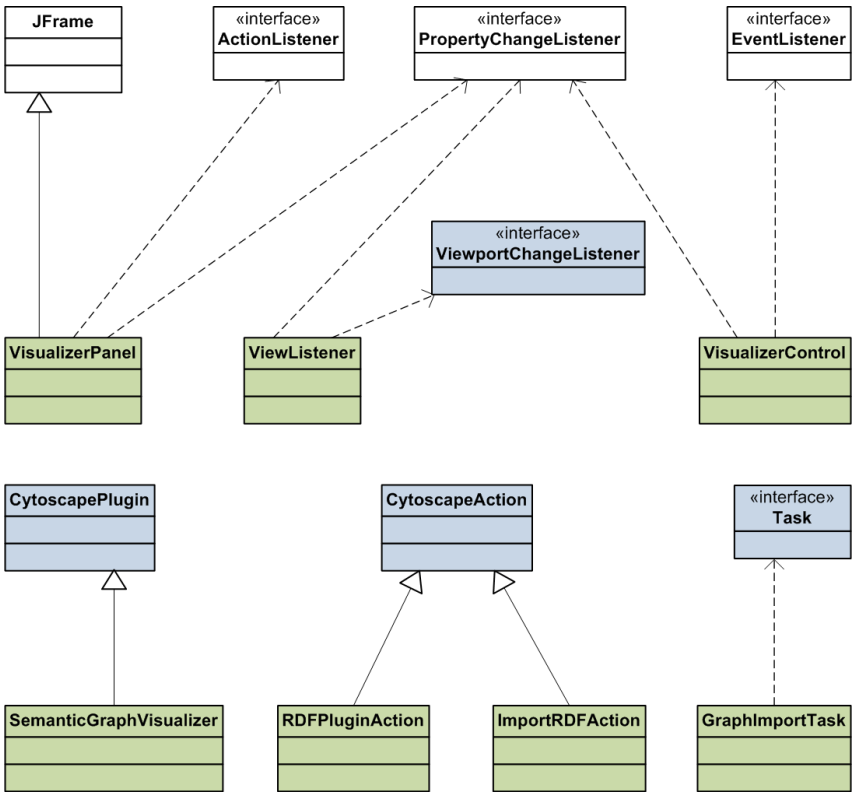


Abbildung 5.4: VIOSEG: Ableitungshierarchie (grün: VIOSEG Kernklassen; blau: Cytoscape Klassen)

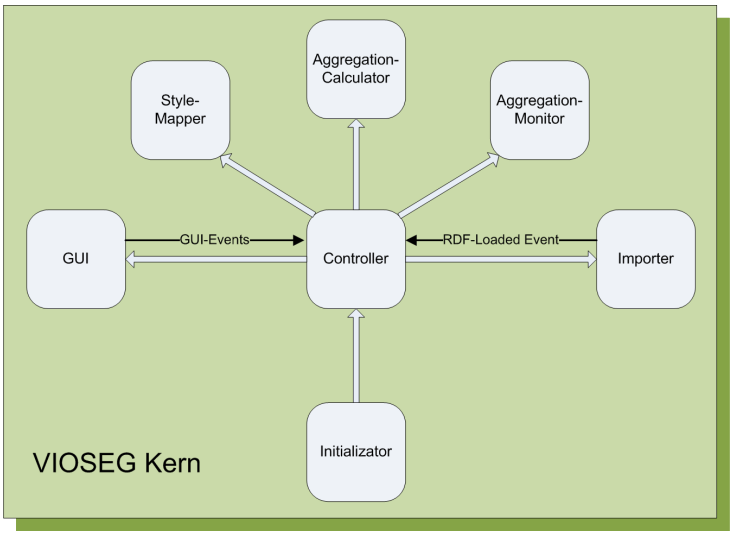


Abbildung 5.5: Aufbau, Struktur und Komponenten des VIOSEG Kerns

Controller

Der Controller stellt die zentrale Kontrolleinheit des VIOSEG Kerns dar. Die Hauptklasse des Controllers (`VisualizerControl`) implementiert *Listener*-Interfaces und ist somit in der Lage, auf *Events*, welche u.a von der GUI-Komponente, der Importer-Komponente oder von Cytoscape ausgesendet werden, zu reagieren. Die Hauptaufgaben des Controllers sind somit:

- Initialisierung der weiteren Kernkomponenten
- Start der GUI-Komponente
- Behandlung von Events
- Berechnung bzw. Update von Node- und Edge-Attributen
- Übermittlung von Informationen des importierten RDF-Datensatzes an die GUI-Komponente

Nachdem das GUI gestartet wurde, wartet der Controller auf ankommende Events. Je nach Typ bzw. Herkunft des Events werden unterschiedliche Prozesse und Abläufe gestartet.

GUI

Diese Komponente implementiert die grafische Benutzerschnittstelle in Form eines Java *Swing Frames*. Die Hauptklasse des GUIs ist `VisualizerPanel`. Die einzelnen Funktionen sind zur besseren Übersicht auf mehrere Registerkarten (Tabs) aufgeteilt. Abbildung 5.6 zeigt die initiale Registerkarte des User-Interfaces. Details zur Bedienung sowie die Erklärung der einzelnen Tabs finden sich im User Guide in Anhang A. Das GUI überträgt die Kontrolle an den User und ermöglicht diesem die Nutzung bzw. Konfiguration der Main-Features von VIOSEG. Hauptaufgaben der grafischen Benutzerschnittstelle sind somit:

- Konfiguration verschiedener Parameter für den Import/Export von RDF-Files (Datenformat, Reasoner,...)
- Auswahl und Konfiguration eines entsprechenden Aggregierungs- bzw. Clustering-Verfahrens (Semantic Aggregation, Hierarchical bzw. K-Means-Clustering)
- Konfiguration der Parameter für die dynamische Aggregierung (Aktivierung/Deatkivierung, Zoomstufe,...)
- Möglichkeiten zur interaktiven Kontrolle der Darstellung (manuelles Expand/Collapse von Meta-Nodes, Aktivierung/Deaktivierung von Labels,...)
- Darstellung näherer Informationen zum aktuellen RDF-Datensatz (Überblick, Auflistung von Subjects, Predicates, Objects, Literals,...)
- Darstellung näherer Informationen zum aktuell selektierten Meta-Node (Anzahl und Name der aggregierten Elemente,...)

Das GUI wird unmittelbar nach der Initialisierung der Cytoscape Umgebung durch die Kontrolleinheit (Controller) gestartet und bildet den Ausgangspunkt für weitere Aktionen. Die Kommunikation mit dem Controller erfolgt mithilfe von Java *Events*. Je nach User-Aktion werden relevante Informationen in ein entsprechendes Event verpackt und dem Controller (implementiert *Listener*-Interfaces) übermittelt. Dieser leitet anschließend die entsprechenden Schritte ein bzw. startet die notwendigen Prozesse. Beispiel: Nachdem der Benutzer eine Clustering-Technik ausgewählt, konfiguriert und den 'Start Clustering'-Button betätigt hat, werden relevante Informationen mithilfe eines Events ausgesendet. Der

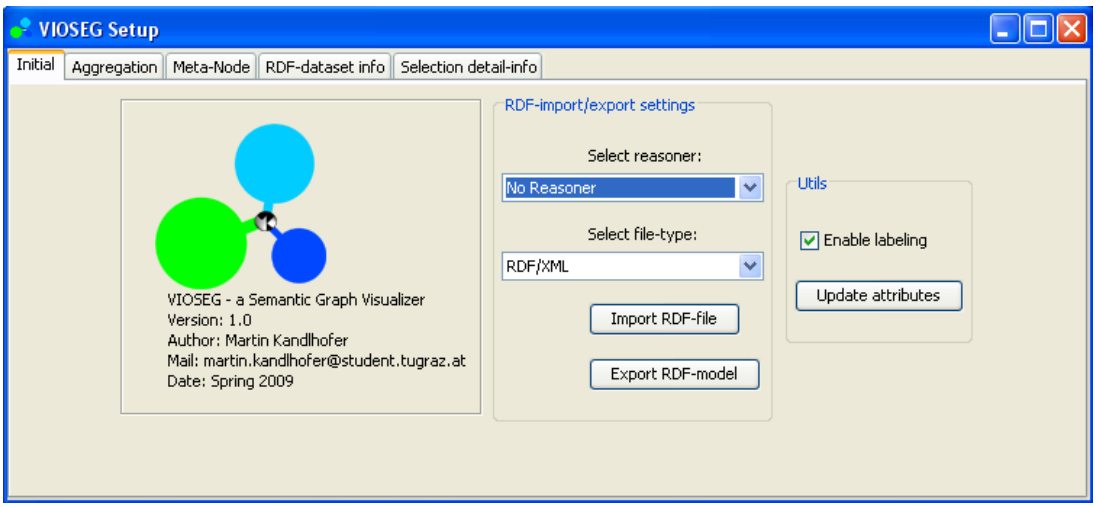


Abbildung 5.6: Grafische Benutzerschnittstelle des VIOSEG Tools

Controller reagiert auf dieses Event indem er die entsprechende Routine der Aggregation-Calculator-Komponente aufruft. Die Beschreibung weiterer relevanter Abläufe und Kommunikationsvorgänge findet sich in Abschnitt 5.3.

Importer

Diese Komponente ist in erster Linie für das Einlesen von RDF-Files verantwortlich. Zusätzlich können bereits geladene RDF-Modelle auch wieder exportiert werden. Die unterstützten Datenformate für Import/Export sind RDF/XML, N-Triple und Notation3. Der gesamte Importvorgang läuft in einem eigenen Thread ab. Ein Progress-Bar zeigt dabei den Fortschritt an. Zum Einlesen und Parsen des RDF-Files sowie zum Export des Modells wird das *Jena Semantic Web Framework* verwendet ([Jena, 2009a]). Diese externe Bibliothek (Zugriff via API)) bietet neben Read-/Write-Operationen für RDF auch die Möglichkeit zur Einbindung eines *Ontology-Reasoners* (siehe dazu Abschnitt 5.2.2). Weitere Aufgaben des Importers sind das Parsen der einzelnen RDF-Statements, die Erzeugung eines Cytoscape-Netzwerks aus Subjects (Source-Nodes), Predicates (Edges) und Objects (Target-Nodes), sowie die initiale Berechnung von Node- und Edge-Attributen. Zudem übernimmt der Importer die automatische Generierung einer visuellen Darstellung des Netzwerks (= *View*). Im Zuge dessen wird auf Funktionalitäten des Cytoscape Frameworks sowie auf die externen Bibliotheken *Jena* und *Pellet* (ein Open Source OWL-Reasoner; siehe [Clark and Parsia, 2009] bzw. Abschnitt 5.2.2) zugegriffen. Die Kernklasse `GraphImport` implementiert die Hauptfunktionalität des Importers. Weitere involvierte Klassen von VIOSEG sind `GraphImportTask` bzw. `ImportRDFAction`.

Der Start der Importer-Komponente erfolgt, nachdem der Controller ein entsprechendes Import-Event von der GUI empfangen hat. Nach erfolgreichem Import eines RDF-Datensatzes wird wiederum der Controller mittels Event informiert. Einzelheiten zum Ablauf des Importvorgangs finden sich Abschnitt 5.3.2. Abschnitt 5.4.1 beleuchtet zudem weitere Details des RDF-Imports.

Aggregation-Calculator

Diese Komponente implementiert die essentiellen Algorithmen zur Berechnung der Aggregierung. Das Ziel ist die Verringerung der visuellen Komplexität eines Graphen. Hauptverantwortliche Klasse ist `GraphAggregator`.

Ein Aggregierungsprozess umfasst grundsätzlich drei Schritte: In einem Vorverarbeitungsschritt werden ähnliche Elemente bzw. Elementgruppen (Nodes oder Edges) identifiziert (*Analyse-Phase*). Im An-

schluss wird für jede Gruppe von Elementen ein Meta-Node erzeugt (*Aggregations-Phase*). D.h. zusammengehörende Nodes werden zu speziellen Meta-Node bzw. Edges zu Meta-Edges zusammengefasst. Dabei wird u.a. das für diese Zwecke erweiterte *MetaNode*-Plugin verwendet (siehe Abschnitt 5.2.2). Im letzten Schritt werden zusätzliche Meta-Node bzw. Meta-Edge Attribute berechnet (Größe, Gewichtung,...).

Der Vorverarbeitungsschritt kann einerseits mithilfe eines Clustering-Algorithmus realisiert werden (*Hierarchical Clustering*, oder *K-Means Clustering*). Basis für die Berechnung ist dabei die euklidische Distanz zwischen den einzelnen Nodes. Die Aggregation-Calculator Komponente greift zur Durchführung von K-Means bzw. Hierarchical Clustering auf das *ClusterMaker*-Plugin zurück (siehe Abschnitt 5.2.2). Die zweite Möglichkeit besteht darin, die Identifikation der Elementgruppen aufgrund semantischer Informationen (z.B. Edge-Type) durchzuführen.

Der Aggregation-Calculator implementiert folgende Möglichkeiten/Mechanismen zur Aggregierung:

- *Aggregation By Semantics* (2 Varianten)
- *Aggregation By K-Means Clustering*
- *Aggregation By Hierarchical Clustering* (2 Varianten)

Die einzelnen Möglichkeiten unterscheiden sich in erster Linie hinsichtlich des Vorverarbeitungsschrittes (Analyse-Phase). Details zu den einzelnen Aggregierungs-Algorithmen finden sich in Abschnitt 5.4.2. In Abschnitt 5.3.3 wird zudem ein typischer Aggregierungsvorgang skizziert.

Die Aufgabe des Aggregation-Calculators ist somit die Identifikation von Gruppen bzw. die Berechnung von Clustern und die darauf basierende Erstellung von Meta-Nodes. Die dynamische, zoomabhaengige Kontrolle dieser Meta-Nodes (expand/collapse) obliegt hingegen der Aggregation-Monitor Komponente.

Aggregation-Monitor

Diese Komponente ist für die Kontrolle der dynamischen Aggregierung verantwortlich. Dabei werden die vom Aggregation-Calculator erstellten Meta-Nodes in Abhängigkeit des aktuellen Zoomfaktors bzw. des aktuellen Ausschnitts des Betrachtungsfensters (= Viewing-Rectangle oder Viewing-Window) automatisch aufgefächert oder zugeklappt (expand/collapse).

Der Monitor wird im Zuge des Startprozesses vom Controller initialisiert. Die Komponente registriert mithilfe eines implementierten *Listener*-Interfaces Änderungen des Viewports (z.B. hervorgerufen durch Zooming- oder Panning-Operationen) und führt bei Bedarf ein Expand bzw. ein Collapse der betreffenden Meta-Nodes durch. Weiters besteht die Möglichkeiten zur Deaktivierung der dynamischen Aggregierung bzw. zum manuellen Expand/Collapse selektierter Meta-Nodes. Hauptverantwortliche Klassen sind `ViewListener` und `GraphLODCalculator` sowie die erweiterte Klasse `MetaNode` des MetaNode-Plugins.

Die beiden Komponenten Aggregation-Calculator und Aggregation-Monitor bilden zusammen mit der Level-Of-Detail bzw. Filter-Funktionalität von Cytoscape (siehe Kapitel 4) die wichtigsten Mechanismen von VIOSEG zur Reduktion der visuellen Komplexität eines Graphen.

Style-Mapper

Der Style-Mapper ist für die RDF-spezifische Darstellung der einzelnen Graphelemente verantwortlich. Dazu wird das Konzept der *Visual Styles* aus Cytoscape verwendet (siehe Kapitel 4, Abschnitt 4.2.7). Die Initialisierung des Style-Mappers (Definition des neuen Visual Styles) erfolgt im Zuge des Startvorgangs durch den Controller. In weitere Folge wird automatisch dieser Visual Style zur Darstellung von Nodes,

Edges und Labels verwendet. Nach Abschluss einer Aggregierungsberechnung (Abschnitt 5.3.3) führt der Controller zudem ein manuelles Update der einzelnen visuellen Attribute durch.

Prinzipiell bildet die Komponente Node- bzw. Edge-Attribute auf visuelle Attribute ab (= Mapping). Die wichtigsten Mappings bzw. visuellen Kodierungen:

- Abbildung des Node-Types (Meta-Node, Default-Node, Literal-Node) auf die Node-Füllfarbe
- Abbildung des Node-Types auf die Node-Form
- Abbildung der Anzahl der zusammengefassten (aggregierten) Nodes auf die Größe einer Meta-Node
- Abbildung der Anzahl der aggregierten Edges auf die Linienstärke einer Meta-Edge
- Zuweisung von Node- und Edge-Labels

Die Berechnung von Node- bzw. Edge-Attributen erfolgt in erster Linie durch den Importer (nach dem Einlesen eines Datensatzes) bzw. durch den Controller (im Zuge der Behandlung verschiedener Events). Zusätzliche Attribute werden auch bei der Erstellung von Meta-Nodes durch den Aggregation-Calculator berechnet.

5.2.2 Plugin/Library-Komponente

Diese Einheit umfasst sowohl externe Bibliotheken (Third-Party Libraries) als auch erweiterte bzw. adaptierte Cytoscape-Plugins. Dadurch werden zusätzlich benötigte Funktionen, welche das Cytoscape Framework nicht direkt unterstützt, zur Verfügung gestellt. Die Kernkomponente greift unmittelbar auf die einzelnen Klassen zu. Nachfolgend werden die wichtigsten Libraries/Plugins sowie deren Funktionen kurz erläutert.

Jena, Pellet

Jena ist ein Java Open Source Framework zur Erstellung von Semantic Web Anwendungen [Jena, 2009a]. Das gesamte Framework kann frei von der Projekt-Website heruntergeladen werden. Es umfasst u.a. eine RDF-API, eine OWL-API, sowie Query- bzw. Inference-Engines. Im Rahmen der VIOSEG Implementierung sind vor allem die Library-Funktionen zum Lesen bzw. Schreiben von RDF in den Formaten RDF/XML, NT und N3 von Bedeutung. Diese werden als jar-Archiv in VIOSEG integriert und im Zuge des Imports bzw. Exports von RDF-Daten verwendet.

Pellet ist ein Java Open Source Ontology-Reasoner für OWL [Clark and Parsia, 2009]. Die Integration in VIOSEG erfolgt wiederum als jar-Archiv. Bei Bedarf wird im Zuge des Importprozesses ein entsprechender Reasoner generiert und in die Jena Inference Engine eingebunden.

Als Inference wird prinzipiell das Herleiten zusätzlicher Informationen und Fakten aus Instanzdaten oder Klassenbeschreibungen (Schemata) bezeichnet. Ein Reasoner ist ein Programm zur Durchführung dieses Tasks [Jena, 2009b]. Weitere Details zum Thema finden sich in Kapitel 2, Abschnitt 2.4.4 sowie unter [Jena, 2009b].

MetaNode-Plugin

Das Cytoscape-Plugin ist ein Java Open Source Programm zur Generierung von Meta-Nodes und Meta-Edges (siehe auch Abschnitt 4.3). Der frei verfügbare Quellcode wurde im Rahmen der Implementierung adaptiert bzw. erweitert und als Source Code Package in das VIOSEG Tool integriert. Auf Features des Plugins wird einerseits bei der Berechnung der Aggregierung zugegriffen. Das Zusammenfassen von

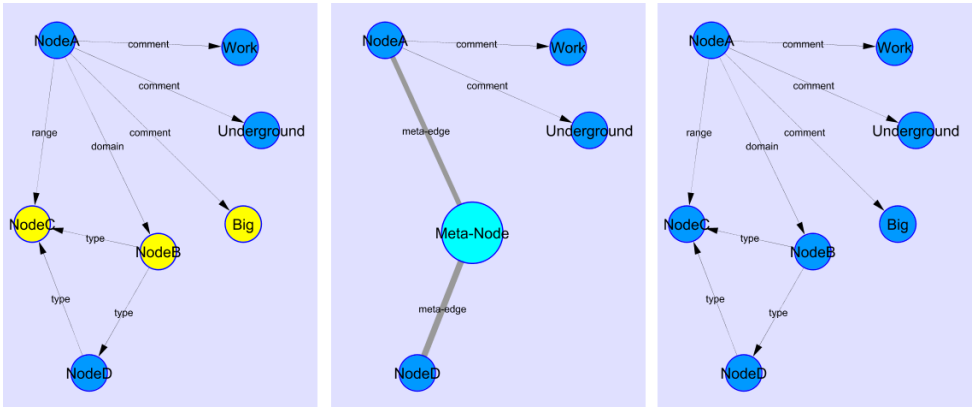


Abbildung 5.7: Demonstration von Collapse und Expand einer Meta-Node. Gelb markierte Nodes (linkes Bild) werden durch einen Meta-Node ersetzt (*Collapse*). Im Zuge dessen werden auch die entsprechenden Edges zu einer Meta-Edge zusammenfasst (mittleres Bild). Das rechte Bild zeigt den Graph nach Durchführung einer *Expand*-Operation

Elementgruppen zu Clustern wird dabei mittels Meta-Nodes realisiert (siehe Abschnitt 5.4.2). Andererseits dienen die Expand-/Collapse-Funktionalitäten zur Umsetzung einer dynamischen, zoomabhängigen Aggregierung (Abschnitt 5.4.3).

Die Klasse `MetaNode` stellt einen Wrapper für die Cytoscape-interne Klasse `CyGroup` zur Verfuegung. `MetaNode` implementiert die Hauptfunktionalitäten des Plugins. Dazu zählen u.a. Mechanismen zur Erstellung, Verwaltung und internen Speicherung von Meta-Nodes. Das ursprüngliche Plugin unterstützt benutzergesteuerte Expand- und Collapse- Funktionalitäten. Diese wurden zur Realisierung einer automatischen, von der jeweiligen Zoomstufe abhängigen Aggregierung erweitert.

Ein Meta-Node repräsentiert eine Gruppe von zusammengehörenden Nodes bzw. Edges. Bei der Erzeugung eines Meta-Nodes wird diese Gruppe (`CyGroup`) einem neu erstellten Node zugeordnet. Ein Meta-Node wird innerhalb von Cytoscape somit als normaler Knoten (Klasse `CyNode`) behandelt (Analoges gilt für Meta-Edges). Im Zuge einer *Collapse*-Operation werden die Nodes einer Gruppe ausgeblendet und durch einen entsprechenden Meta-Node ersetzt. Betreffende Edges werden infolgedessen durch Meta-Edges ersetzt. Eine *Expand*-Operation bewirkt das Gegenteil: Entsprechende Meta-Nodes bzw. Meta-Edges werden aus- und die darin aggregierten Nodes bzw. Edges wieder eingeblendet. Abbildung 5.7 zeigt beispielhaft einen Collapse-/Expand-Vorgang.

ClusterMaker-Plugin

ClusterMaker ist ein Java Open Source Plugin (siehe auch Abschnitt 4.3 in Kapitel 4). Es implementiert eine Reihe von Clustering-Techniken (*Hierarchical-, K-Means-, Force-, MCL-Clustering*, welche mittels verschiedener Parameter konfiguriert werden können (*Distance Metric, Linkage-Method, Number of Iterations,* et.). Im Zuge der Aggregierungs-Berechnungen *Aggregation By Hierarchical Clustering* und *Aggregation By K-Means Clustering*) greift VIOSEG direkt auf Methoden der einzelnen Plugin-Klassen zu. ClusterMaker analysiert mithilfe des gewählten Clustering-Algorithmus die Graphstruktur. Identifizierte Gruppen (Cluster) werden als Objekte der Cytoscape-Klasse `CyGroup` unter einem eindeutigen Namen zur weiteren Behandlung in eine Cytoscape-interne Liste gespeichert.

5.2.3 Cytoscape Framework

Das Cytoscape Framework stellt grundlegende Funktionen zur Verwaltung und Visualisierung von Graphen bereit. Unterstützte Features und Mechanismen wurden in Kapitel 4 vorgestellt und diskutiert.

Abbildung 4.1 zeigt eine Übersicht der wichtigsten Komponenten und deren Beziehungen. Nachfolgend werden die für die Implementierung von VIOSEG relevanten Klassen bzw. Interfaces des Frameworks kurz erläutert (siehe dazu auch Abschnitt 4.1.4).

- **`CyMain`** stellt die Hauptklasse zur Initialisierung bzw. zum Start der Cytoscape Umgebung dar. Durch Übergabe von Parametern kann Cytoscape auf verschiedene Weise bzw. mit unterschiedlichen Konfigurationen gestartet werden (z.B. GUI- oder Text-Version, explizite Zuweisung von Arbeitsspeicher, etc.).

- **`Cytoscape`** bildet die primäre Klasse der Cytoscape API. Sie stellt statische Methoden u.a. zur Erzeugung von Nodes, Edges und Netzwerken (`CyNode, CyEdge, CyNetwork`), zur Generierung einer grafischen Darstellung des Netzwerks (`CyNetworkView`), sowie zur Erzeugung und Zuweisung von Node- und Edge-Attributen (`CyAttributes`) zur Verfügung. Ein Objekt der Klasse `CyNode` wird beispielsweise durch den Aufruf `Cytoscape.getCyNode(nodeID)` instanziert.

- **`CyNetwork`** stellt die zentrale Komponente zur Generierung und Verwaltung eines Netzwerks bzw. eines Graphen dar. Ein solches Netzwerk besteht aus einer Menge von Objekten des Typs `CyNode` sowie aus einer Menge von Objekten des Typs `CyEdge`.

- **`CyNode`, `CyEdge`** sind die grundlegenden Klassen zur Repräsentation von Nodes und Edges in Cytoscape. Nodes und Edges werden durch eine ID (String) eindeutig identifiziert.

- **`CyNetworkView`** ist für die Generierung einer grafischen Darstellung der Netzwerkstruktur verantwortlich. `CyNetworkView` ist somit die Visualisierung eines `CyNetwork`s.

- **`CyAttributes`** erlaubt das Setzen/Löschen von - bzw. den Zugriff auf - Node- und Edge-Attributes innerhalb von Cytoscape. Jedem Node bzw. jeder Edge können ein oder mehrere Attribute zugeordnet werden. `CyAttributes` verwendet eine *Multi Hash-Map* zur Speicherung der Attribute. Erlaubte Datentypen sind String, Integer, Double und Boolean.

- **`CyGroup`** implementiert Methoden zur Verwaltung von Node-Gruppen. Die Nodes der betreffenden Gruppe werden in einer Liste gespeichert. Ein spezieller Node aus dieser Liste dient dabei als Repräsentant der Gruppe (= Group-Node).

- **`VisualMappingManager`** ist für die Kontrolle und Verwaltung der visuellen Darstellung von Nodes und Edges (= *Visual Styles*) bzw. das Mapping von Datenattributen auf visuelle Attribute verantwortlich.

- **`CytoscapePlugin`** stellt die Basisklasse zur Einbindung eines Plugins in Cytoscape dar. Alle Cytoscape-Plugins werden von dieser Klasse abgeleitet.

Detaillierte Beschreibungen weiterer Klassen und Konzepte sowie eine API Javadoc finden sich unter [Cytoscape, 2009h] bzw. [Cytoscape, 2009e].

5.3 Abläufe

Dieser Abschnitt erläutert die wichtigsten Abläufe und Kommunikationsvorgänge sowie involvierte Komponenten bzw. Klassen des VIOSEG Systems. Die Erklärung der einzelnen Kernkomponenten findet sich in Abschnitt 5.2.1.

5.3.1 Start und Initialisierung

Für den Startvorgang bzw. die Einbindung in das Cytoscape Framework ist in erster Linie die Initializator-Komponente verantwortlich. Der Prozess gliedert sich grundsätzlich in folgende Schritte:

1. Instanzierung eines Objekts der Klasse `SemanticGraphVisualizer` (z.B. durch Main- Methode der Klasse `VisualizerMain`).
2. Initialisierung und Start der Cytoscape Umgebung (u.a. Benutzeroberfläche, etc.) durch Erzeugung einer Instanz der Cytoscape-Klasse `CyMain`. Die Klasse `CyMain` dient somit zur 'Inbetriebnahme' von Cytoscape.
3. Initialisierung zusätzlicher Plugins (u.a. MetaNode-Plugin).
4. Registrierung in Cytoscape (als Eintrag im 'Plugin'-Menü).
5. Start der Controller-Komponente durch Instanzierung der Klasse `VisualizerControl`. Der Controller implementiert *Listener*-Interfaces und reagiert bei Bedarf auf spezielle Events (u.a. von der GUI, vom Importer und von Cytoscape).
6. Start der grafischen Benutzerschnittstelle (GUI-Komponente) sowie Initialisierung weiterer Kernkomponenten durch den Controller.

Nach Abschluss dieses Vorgangs wartet der Controller auf weitere Events. Diese werden in erster Linie von der GUI-Komponente bzw. von Cytoscape selbst ausgesandt. Der ordnungsgemäße Abschluss des hier beschriebenen Startprozesses ist Voraussetzung für alle weiteren Prozesse und Vorgänge.

5.3.2 Import von RDF-Daten

Ein Importvorgang umfasst prinzipiell folgende Schritte:

1. Die GUI-Komponente sendet ein Import-Event aus, welches u.a. Informationen zu gewähltem Datenformat bzw. gewähltem Reasoner enthält.
2. Der Controller reagiert auf das Event und übergibt die relevanten Informationen durch Aufruf der entsprechenden Methode an den Importer.
3. Erzeugung eines eigenen Threads für den Importprozess (Klasse `GraphImportTask` der Importer-Komponente).
4. Einlesen der Daten aus dem File und Erzeugung des RDF-Modells mittels Methoden aus der Jena Library. Bei Bedarf wird ein entsprechender Ontology-Reasoner generiert (Pellet Library) und somit ein sog. *inferenced* RDF-Modell erzeugt (siehe dazu auch Punkt 5.2.2).
5. Parsen der einzelnen Statements des RDF-Modells (Subject, Predicate, Object) mittels Jena Library.
6. Erzeugung eines Cytoscape-Netzwerks bzw. Graphen. Subjects werden dabei als Source-Nodes, Predicates als Edges und Objects als Target-Nodes interpretiert.
7. Berechnung RDF-spezifischer sowie weiterer initialer Informationen (Name, Literal, Label, Typ,...) durch Erzeugung und Zuweisung von Node- und Edge-Attributen.
8. Generierung einer visuellen Darstellung des Cytoscape-Netzwerks.

Nach erfolgreichem Abschluss des Importvorgangs informiert die Importer-Komponente den Controller (via Event). Dieser berechnet im Anschluss weitere graphtheoretische Parameter (u.a. Edge-Length, Node-Degree,...) und weist diese den jeweiligen Nodes und Edges als Attribute zu. Abschließend uebermittelt der Controller nähere Informationen des importierten RDF-Datensatzes zwecks tabellarischer Darstellung an die GUI-Komponente

5.3.3 Berechnung der Aggregierung

Nachfolgend werden die wesentlichen Schritte der Aggregierungs- bzw. Cluster-Berechnung erläutert. Bevor eine Aggregierung durchgeführt werden kann, müssen Start- und Importvorgang abgeschlossen worden sein.

1. Die GUI sendet ein Event mit Informationen zur gewählten Aggregierungs-Technik samt Parameterkonfiguration.
2. Der Controller führt ein Update der Node- und Edge-Attribute (Node-Position, Edge-Length,...) durch und startet anschließend die entsprechende Methode des Aggregation-Calculators (in Abhaengigkeit der gewählten Aggregierungstechnik). Die nachfolgenden Schritte werden somit von der Aggregation-Calculator Komponente gesetzt.
3. Identifikation von Elementgruppen (*Analyse-Phase*).
4. Zusammenfassen von Elementen einer Gruppe durch Erzeugung von Meta-Nodes bzw. Meta-Edges (*Aggregations-Phase*).
5. Zuklappen (Collapsing) aller auf diese Weise erstellter Meta-Nodes.
6. Berechnung von zusätzlichen Meta-Node bzw. Meta-Edge Attributen (Größe, Gewichtung,...).
7. Bei gewählter semantischer Aggregierungs-Technik (*Aggregation By Semantics*) wird das Layout des Graphen unter Berücksichtigung der Kantengewichtung neu berechnet.

Nach Abschluss dieses Vorgangs führt der Controller erneut ein Update der Attribute sowie der visuellen Darstellung der einzelnen Graphelemente durch (mittels Aufruf der entsprechenden Methode des Style-Mappers).

5.4 Ausgewählte Details der Implementierung

In diesem Abschnitt erfolgt die nähere Betrachtung ausgewählter Konzepte, Techniken und Mechanismen der Implementierung. Im Zuge dessen werden wichtige Details bzw. technische Aspekte der Hauptfunktionalitäten diskutiert.

5.4.1 Import von RDF-Graphen

Der Vorgang des Imports sowie die daran beteiligten Komponenten wurden bereits in den Abschnitten 5.2.1 und 5.3.2 behandelt. Dieser Abschnitt widmet sich der näheren Betrachtung des Einlesens und Parsens eines RDF-Files mittels Methoden der Jena Library sowie der Erstellung eines Netzwerks/Graphen mithilfe der von Cytoscape zur Verfügung gestellten Mechanismen. Die einzelnen Funktionalitäten werden zum größten Teil von der Klasse `GraphImport` implementiert.

Dem Einlesen der Daten aus einer Datei folgt zunächst die Erzeugung des originalen RDF-Modells *Raw Model*. Je nach gewähltem Reasoner (RDFS- oder OWL-Reasoner von Jena bzw. OWL-Reasoner

von Pellet) kann aus diesem Modell zusätzlich noch ein *Inferenced Model* abgeleitet werden. Die einzelnen Statements (RDF-Triples) des erstellten Modells werden in einer Liste gespeichert. Diese Liste wird im Zuge des Parsing-Vorgangs durchlaufen und jedes Statement in die Bestandteile Subject, Predicate und Object aufgespaltet.

Im nächsten Schritt erfolgt die Erstellung einer Graph- bzw. Netzwerkstruktur. Cytoscape stellt die dafür benötigten Mechanismen und Klassen zur Verfügung. Zuerst wird die URI des Subjects und des Predicates sowie die URI (bzw. der Literal-Value) des Objects ermittelt. Dieser String dient in weiterer Folge als ID zur eindeutigen Identifizierung der einzelnen Nodes und Edges. Im Anschluss wird jeweils für Subject und Object eine Instanz der Cytoscape-Klasse `CyNode` sowie für das Predicate eine Instanz der Klasse `CyEdge` mit der entsprechenden ID erzeugt. Die Konstruktion des Netzwerks/Graphen erfolgt indem Subject-Node (= Source-Node vom Typ `CyNode`), Predicate (= Interaction-Edge vom Typ `CyEdge`) und Object-Node (= Target-Node vom Typ `CyNode`) einer `CyNetwork`-Instanz zugeordnet werden. `CyNetwork` stellt auch sicher, dass das Netzwerk nicht aus mehreren identischen Nodes besteht. Das auf diese Weise erstellte Netzwerk befindet sich nun im Speicher. Es existiert jedoch noch keine grafische Darstellung (= View). Erst nachdem sichergestellt wurde, dass die Elementanzahl (Nodes+Edges) einen gewissen Schwellwert nicht übersteigt, wird Cytoscape angewiesen, ein View des Graphen zu generieren.

Im Zuge des hier beschriebenen Importvorgangs werden zudem RDF-spezifische Informationen aus den einzelnen Subjects, Predicates und Objects extrahiert und als Node- bzw. Edge-Attribute gespeichert (u.a. URI, Literal Value, Namespace, Local Name,...).

5.4.2 Aggregierung/Clustering

Der allgemeine Ablauf einer Aggregierung sowie die beteiligten Komponenten wurden in Abschnitt 5.3.3 bzw. in Abschnitt 5.2.1 bereits skizziert. An dieser Stelle sollen nun die Details der einzelnen Aggregierungstechniken bzw. die verwendeten Algorithmen betrachtet werden. Die Implementierung der Funktionalitäten findet sich in der Klasse `GraphAggregator`.

Eine Aggregierung in VIOSEG läuft prinzipiell in drei Schritten ab: In einem Preprocessing-Step wird der Graph analysiert um ähnliche Elemente (Nodes, Edges) bzw. Gruppen von Elementen zu identifizieren. Dieser Vorgang wird im Kontext der Implementierung als (*Analyse-Phase*) bezeichnet. Die Definition bzw. Berechnung der Ähnlichkeit ist dabei u.a. von der jeweils gewählten Clustering-Technik abhängig. Im zweiten Schritt wird für jeden gefundenen Cluster bzw. jede identifizierte Elementgruppe ein Meta-Node mit einer eindeutigen ID generiert. Dies erfolgt durch Instanzierung eines Objekts der Klasse `MetaNode`. Nodes einer Gruppe werden somit in einem Meta-Node bzw. Edges in einer Meta-Edge aggregiert. Der Prozess wird demzufolge als *Aggregations-Phase* bezeichnet. Der abschließende Schritt widmet sich der Ermittlung zusätzlicher Attribute der neu erzeugten Meta-Nodes/Meta-Edges (Edge-Weight, Labels, Node-Size,...) bzw. der Berechnung eines initialen Layouts.

Die implementierten Aggregierungstechniken unterscheiden sich vor allem hinsichtlich der Analyse-Phase. Aufgrund dessen kann prinzipiell zwischen Aggregierung aufgrund semantischer Informationen (*Aggregation By Semantics*; 2 Varianten) und Aggregierung auf Basis von Clustering-Techniken (*Aggregation By K-Means Clustering* bzw. *Aggregation By Hierarchical Clustering*; 2 Varianten) differenziert werden.

Aggregation By Semantics Methoden entfernen zu Beginn alle bisher erstellten Meta-Nodes bzw. Cluster. Eine zuvor vorgenommene Aggregierung hat somit keine Auswirkung auf nachfolgende Berechnungen. Im Gegensatz dazu entfernen *Aggregation By Clustering* Methoden nur jene Cluster/Meta-Nodes, welche aufgrund von Clustering-Techniken erstellt wurden. Auf Basis semantischer Informationen erzeugte Meta-Nodes bleiben erhalten. Das Ergebnis einer zuvor ermittelten semantischen Aggregierung fließt somit in die *Aggregation By Clustering* Berechnung mit ein. Die einzelnen Aggregierungstechniken können somit beliebig oft, bzw. in willkürlicher Reihenfolge angewendet werden. Hinsichtlich

einer optimalen Komplexitätsreduktion empfiehlt es sich aber, Elemente zuerst mittels *Aggregation By Semantics* zu aggregieren und die verbleibenden Nodes/Edges mittels *Aggregation By Clustering* zusammenzufassen.

Aggregation By Semantics

Diese Methode nützt semantische Informationen des importierten RDF-Graphen zur Identifikation und Aggregierung von Nodes und Edges. Die entscheidende semantische Information bzw. das entscheidende Kriterium stellt dabei der Kantentyp (= Predicate oder Edge-Type) dar. Der Kantentyp wird während des Imports als Edge-Attribut gespeichert und beschreibt grundsätzlich die Art der Relation zwischen zwei Nodes ([Source-Node] - [Edge-Type] - [Target-Node]; Beispiel: [Film A] - [starring] - [Actor X]). Im Zuge des Aggregierungsprozesses werden für jeden Knoten die ausgehenden Kanten gleichen Typs ermittelt und anschließend zusammengefasst.

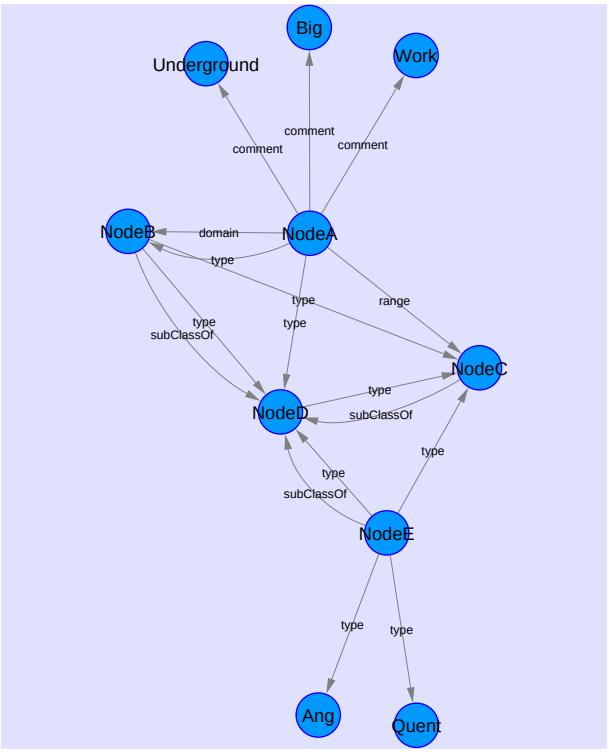
Der Algorithmus gliedert sich grundsätzlich in die zwei Hauptphasen *Analyse* und *Aggregierung*. In der *Analyse-Phase* werden nacheinander alle Nodes des Graphen besucht. Dabei werden die vom aktuell betrachteten Knoten ausgehenden Kantentypen sowie die Häufigkeit der einzelnen Typen ermittelt. Danach erfolgt die Berechnung der Knoten-Priorität. Diese Priorität legt in der anschließenden Aggregierungs-Phase die Reihenfolge der Node-Behandlung fest. Die Priorität eines Knoten ist umso höher, je größer die Häufigkeit eines bestimmten Edge-Types ist. Dieser, als *Node-Priority Criterion* bezeichneter Edge-Type wird vom User zu Beginn des Aggregierungsprozesses via GUI bestimmt. Die Ergebnisse der Analyse-Phase sind somit: **a)** Eine Liste aller ausgehenden Kantentypen jedes Knoten (= *Edge-Map*; die Liste enthält keine doppelten Kantentypen). **b)** Eine Liste der Knoten-Prioritäten (= *Priority-Map*).

Abbildung 5.8 zeigt einen simplen RDF-Graphen sowie als Beispiel die Edge-Maps zweier Knoten. Darin finden sich die Kantentypen der ausgehenden Edges sowie deren Häufigkeit. Die Priorität der Knoten in der Priority-Map ergibt sich aufgrund des gewählten Node-Priority-Criterion (in diesem Fall der Kantentyp *type*).

In der *Aggregierungs-Phase* wird durch die Liste der Nodes gemäß der zuvor ermittelten Prioritäten iteriert. In Folge wird die Edge-Map jedes Nodes durchlaufen. Dabei wird zuerst überprüft, ob die aktuell betrachtete Kante bereits mit einem Meta-Node bzw. einer Meta-Edge in Verbindung steht. Ist dies nicht der Fall, wird der Target-Node dieser Kante ermittelt. Falls es sich dabei nicht um einen Meta-Node handelt, wird ein neuer Meta-Node mit den entsprechenden Attributen (ID, Label) erstellt. Abschließend wird der Target-Node zu diesem Meta-Node hinzugefügt. D.h. ausgehende Kanten des selben Typs werden zu Meta-Edges zusammengefasst, indem deren Target-Nodes zu Meta-Nodes aggregiert werden. Für jeden Kantentyp in einer Edge-Map wird, unter der Voraussetzung dass die erwähnten Vorbedingungen erfüllt sind, somit ein Meta-Node erzeugt.

Nach Abschluss der Aggregierung werden alle neu erzeugten Meta-Nodes zusammengeklappt und zusätzliche Attribute wie Meta-Node Size, Meta-Edge Weight und Meta-Edge Label berechnet. Die Größe der Meta-Node ergibt sich dabei aus der Anzahl der aggregierten Nodes. Das Kantengewicht (Meta-Edge Weight) ist die Anzahl des am häufigst vorhandenen Kantentyps innerhalb der Meta-Edge. Abschließend wird dem auf diese Weise visuell reduzierten Graphen ein, vom User zu Beginn des Aggregierungsprozesses festgelegtes Layout zugewiesen. Bei der Berechnung des Layouts wird das zuvor ermittelte Edge-Weight Attribut berücksichtigt. Abbildung 5.9 zeigt den Graphen aus Abbildung 5.8 nach Abschluss des Aggregierungsprozesses. Der Pseudocode des hier erläuterten Algorithmus findet sich unter Algorithm**1**.

Die zweite Variante des *Aggregation By Semantics* Algorithmus unterscheidet sich lediglich in der Berechnung der Knoten-Priorität. Diese ergibt sich nicht aus der Häufigkeit eines bestimmten Edge-Types sondern aus dem Node-Degree. Nodes mit einem höheren Degree haben höhere Priorität. Der User kann via GUI zwischen In-Degree (incoming edges), Out-Degree (outgoing edges) und Degree



(a) RDF-Beispielgraph

Edge-Maps:

NodeA	Edge-Type	Häufigkeit
	comment	3
	type	2
	domain	1
	range	1

NodeE	Edge-Type	Häufigkeit
	type	4
	subClassOf	1

Priority-Map:

Knoten	Häufigkeit des Kantentyps "*type*"
NodeE	4
NodeA	2
NodeB	2
NodeD	1
NodeC	0
Big	0
Work	0
Underground	0
Ang	0
Quent	0

(b) Edge-Maps

Abbildung 5.8: Aggregation By Semantics (a) RDF-Beispielgraph (b) Edge-Map von *NodeA* und *NodeE*; Priority-Map (Node-Priority-Criterion: *type*)

Algorithm 1 Aggregation By Semantics

```
remove all previously created meta-nodes
/* Analysis-Phase */
for all nodes do
    calculate edge-types of outgoing edges
    put edge-types into edge-map
    calculate frequency of occurrence of each edge-type
    calculate node-priority based on frequency of occurrence
end for
/* Aggregation-Phase */
for all nodes ordered by priority do
    for all edge-types contained in edge-map do
        if current edge-type is already associated with meta-edge then
            continue
        end if
        get target-node of current edge-type
        if target-node is already associated with meta-node then
            continue
        end if
        create new meta-node
        add target-node to meta-node
    end for
end for
/* Final-Phase */
collapse all created meta-nodes
create view
for all created meta-nodes do
    calculate meta-node attributes (size)
    for all meta-edges do
        calculate meta-edge attributes (weight, label)
    end for
end for
apply layout
```

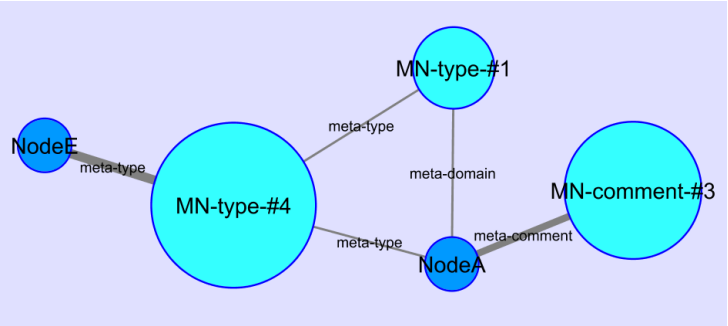


Abbildung 5.9: Aggregation By Semantics: Graph aus Abbildung 5.8 nach Abschluss des Aggregierungsvorgangs (Force-Directed Layout). Türkise Knoten repräsentieren Meta-Nodes (Größe in Abhängigkeit der aggregierten Nodes), graue Linien ohne Pfeile repräsentieren Meta-Edges (Linienstärke als Indikator für die Anzahl der aggregierten Edges). Meta-Node Labels setzen sich aus einem Prefix (*MN*), dem aggregierten Edge-Type (z.B. *comment*) sowie der Anzahl der enthaltenen Nodes (*#3*) zusammen. Meta-Edge Labels bestehen aus einem Prefix (*meta*) und dem Kantentyp einer darin zusammengefassten Kante.

(incoming + outgoing edges) entscheiden.

Aggregation By K-Means Clustering

Diese Aggregierungsmethode verwendet einen K-Means Clustering-Algorithmus (Details dazu siehe Kapitel 2, Abschnitt 2.3.3) zur Identifikation von Elementgruppen. In der *Analyse-Phase* (in diesem Zusammenhang auch als *Clustering-Phase* bezeichnet) wird der K-Means Algorithmus aus dem ClusterMaker-Plugin gestartet. Die Anzahl der Iterationen sowie die Anzahl der Cluster werden vom User via GUI bestimmt. Das Clustering erfolgt auf Basis der euklidischen Distanz (X- und Y-Position der Nodes). Die Node-Positionen werden zu Beginn berechnet bzw. aktualisiert und als Node-Attributes gespeichert. Dies setzt voraus, dass bereits eine visuelle Darstellung des Graphen (View) generiert wurde.

Das ClusterMaker-Plugin speichert gefundene Gruppen unter einem speziellen Namen als Instanzen der Cytoscape-Klasse `CyGroup` in eine Liste, welche im Zuge der *Aggregations-Phase* durchiteriert wird. In weiterer Folge wird für jede gefundene Gruppe ein Meta-Node mit entsprechenden Attributen (ID, Label, Größe, Anzahl aggregierter Elemente) erzeugt. Die Größe (Höhe und Breite) der Meta-Node ergibt sich dabei aus dem Bounding-Polygon der aggregierten Nodes.

Nach dem Collapsing aller neu erzeugten Meta-Nodes erfolgt im letzten Schritt die Berechnung und Zuweisung von Meta-Edge Attributen für Label und Gewichtung (Meta-Edges wurden im Zuge der Meta-Node Erstellung generiert). Die Kantengewichtung ergibt sich dabei aus der Summe der aggregierten Edges. Der hier beschriebene Aggregierungsvorgang findet sich als Pseudocode unter Algorithm**2**.

Aggregation By Hierarchical Clustering

Diese Aggregierungstechnik verwendet Hierarchical Clustering zur Auffindung von Elementgruppen (Details zu HAC siehe Kapitel 2, Abschnitt 2.3.3). Der Vorgang umfasst im Wesentlichen dieselben Schritte wie der zuvor beschriebene *Aggregation By K-Means* Algorithmus. Der größte Unterschied liegt dabei in der *Analyse-* bzw. *Clustering-Phase*, wo der Hierarchical Clustering-Algorithmus des ClusterMaker-Plugins zum Einsatz kommt. Die Parameter für die Ähnlichkeitsberechnung (*Single-Link, Complete-Link, Average-Link*) sowie die Anzahl der Cluster sind via GUI konfigurierbar. Das Clustering erfolgt wiederum auf Basis der euklidischen Distanz (XY-Position der Nodes).

In der *Aggregations-Phase* wird die vom Algorithmus erstellte hierarchische Cluster-Struktur (bestehend aus Objekten der Klasse `CyGroup`) von oben nach unten durchlaufen. Für jede Gruppe (`CyGroup`)

Algorithm 2 Aggregation By K-Means Clustering

```
Require: view created
  remove all meta-nodes created by clustering-aggregation methods
  /* Analysis-Phase */
  calculate xy-position of nodes
  perform k-means clustering
  /* Aggregation-Phase */
  for all groups found by k-means algorithm do
    create meta-node for each group
  end for
  /* Final-Phase */
  collapse all created meta-nodes
  for all created meta-nodes do
    for all meta-edges do
      calculate meta-edge attributes
    end for
  end for
```

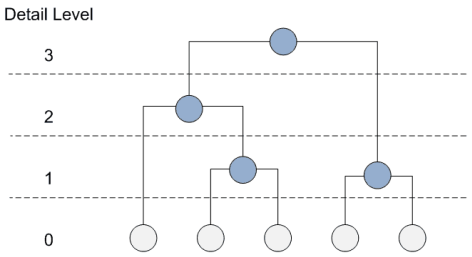


Abbildung 5.10: Hierarchische Cluster-Struktur mit Nummerierung der einzelnen Stufen (*Detail Level*)

wird dabei ein Meta-Node erstellt. Der Prozess stoppt, wenn die vorgegebene Anzahl an Meta-Nodes bzw. Cluster erreicht wurde oder der Algorithmus bis zur obersten Stufe der hierarchischen Struktur (nur mehr ein einziger Cluster) gelangt ist. Die restlichen Abläufe sind analog zu *Aggregation By K-Means*.

Die zweite Variante des *Aggregation By Hierarchical Clustering* Verfahrens unterscheidet sich dadurch, dass die genaue Anzahl der Cluster nicht bekannt ist. Der User konfiguriert lediglich den gewuenschten Detailgrad durch Setzen eines sog. *Detail Level* Parameters. Dies ist ein Integer-Wert beginnend bei 1. Je höher dieser Wert, desto größere Meta-Nodes/Cluster (hinsichtlich der Anzahl der darin aggregierten Nodes) werden erzeugt. Ein kleinerer Wert hat hingegen kleinere Cluster zur Folge. Im Anschluss an die Analyse-Phase wird somit jeder Stufe der hierarchischen Cluster-Struktur ein *Detail Level* Wert zugeordnet (siehe Abbildung 5.10). Die unterste Stufe erhält den Wert 0, die nächste den Wert 1, usw. Im Zuge der Aggregations-Phase wird die Baumstruktur von unten nach oben durchlaufen und für jede Gruppe ein Meta-Node erstellt. Der Prozess stoppt bei Erreichung des gewählten *Detail Levels*.

5.4.3 Dynamische Aggregierung

Im vorangegangenen Punkt wurden die implementierten Algorithmen zur Auffindung bzw. Bildung von Clustern erläutert. Die im Zuge dieses Vorgangs generierten Meta-Nodes sollen nun gemäß den Requirements in Abhängigkeit des Betrachtungsausschnitts bzw. der aktuellen Zoomstufe automatisch aufgefächert bzw. zusammengeklappt werden. Für die Realisierung dieser dynamischen Aggregierung sind die beiden VIOSEG Kernklassen `ViewListener` und `GraphLODCalculator` verantwortlich. Die Klasse `ViewListener` empfängt mithilfe eines implementierten *Listener*-Interfaces sog. *Viewport-*

Change Events, welche von Cytoscape u.a. als Folge von Zooming- oder Panning-Operationen ausgesendet werden. Nach Auftreten eines solchen Events wird der Prozess zur Ermittlung einer allfälligen Expand- oder Collapse-Operation der entsprechenden Meta-Nodes gestartet (`GraphLODCalculator`).

Im Zuge dieses Prozesses wird durch die Liste aller Meta-Nodes iteriert. Dabei wird für jeden Meta-Node überprüft, ob sich dessen XY-Positonskoordinaten innerhalb des aktuellen Betrachtungsfensters (= *Viewing-Rectangle* oder *Viewing-Window*) befinden. Ist dies der Fall, erfolgt anschließend die Berechnung der Relation zwischen Größe des Viewing-Rectangles (abhängig von der aktuellen Zoomstufe) und Höhe bzw. Breite der Meta-Node. Diese Relation ist ausschlaggebend dafür, ob der aktuelle Meta-Node aufgefächert oder wieder zugeklappt wird. Die Berechnung im Detail (Algorithm**3**):

Algorithm 3 Expand/Collapse Calculation

listen until viewport-change event occurs
if meta-node is within viewing-rectangle **and**
viewing-rectangle.width $<$ meta-node.width $*$ *threshold* **and**
viewing-rectangle.height $<$ meta-node.height $*$ *threshold* **then**
  expand meta-node
else
  collapse meta-node
end if

Threshold hat Einfluss darauf, ab welcher Zoomstufe ein Expand durchgeführt wird. Ein hoher Wert führt dazu, dass Meta-Nodes erst bei einem hohen Zoomfaktor aufgefächert werden bzw. vice versa. Der Parameter kann vom User via GUI konfiguriert werden. Zum Auffächern eines Meta-Nodes wird die entsprechende Expand-Methode der Klasse `MetaNode` aufgerufen. Diese stellt sicher, dass sich die aufgefächerten Nodes innerhalb des aktuellen Viewing-Rectangles befinden. Zur besseren Übersicht werden diese Nodes zudem optisch hervorgehoben.

5.5 Zusammenfassung

Zum Abschluss dieses Kapitels sollen zur besseren Übersicht bzw. als Zusammenfassung, die zu Beginn im Abschnitt 5.1.1 definierten Requirements den implementierten Funktionalitäten und Mechanismen gegenübergestellt werden. Zusätzlich finden sich Verweise auf die entsprechenden Kapitelabschnitte. Das nächste Kapitel widmet sich schließlich der Demonstration der einzelnen Features der VIOSEG Implementierung. Im Zuge dessen werden verschiedene Anwendungsszenarien durchgespielt sowie Beispieldaten zur Vermittlung der Funktionalitäten des Tools visualisiert.

- **RDF-Import**: Das Einlesen und Parsen von RDF-Files sowie Generierung eines RDF-Graphen wird in erster Linie von der *Importer*-Komponente des VIOSEG Kerns realisiert. Die Beschreibung dieser Komponente findet sich in Abschnitt 5.2.1. Abschnitt 5.3.2 skizziert den Ablauf eines Importvorgangs, während sich Abschnitt 5.4.1 Details der Graph-Erstellung widmet.
- **Visualisierung**: Die grundlegenden Visualisierungs-Features und Mechanismen werden vom Cytoscape Framework zur Verfügung gestellt (Abschnitt 5.2.3 bzw. Kapitel 4). Die Erstellung eines Graphen aus RDF-Daten erfolgt im Rahmen des Import. Details dazu finden sich in Abschnitt 5.4.1).
- **Interaktivität**: Standard-Funktionalitäten zur Interaktion mit der Darstellung (Navigation, Zoom, Pan, Edit, etc.) werden direkt von Cytoscape unterstützt (siehe u.a. Kapitel 4).
- **Reduktion der visuellen Komplexität**: Neben den von Cytoscape unterstützen Level-Of-Detail und Filter-Features (siehe Kapitel 4, Abschnitt 4.2.5 bzw. Abschnitt 4.2.8) implementieren die

beiden VIOSEG Kernkomponenten *Aggregation-Calculator* und *Aggregation-Monitor* die Hauptfunktionalitäten zur Identifikation und Erstellung von Clustern sowie zur dynamischen Aggregierung. Diese Komponenten werden in Abschnitt 5.2.1 beschrieben. Der allgemeine Ablauf einer Cluster- bzw. Aggregierungsberechnung findet sich in Abschnitt 5.3.3. Die Abschnitte 5.4.2 und 5.4.3 widmen sich Details (Algorithmen, Implentierungsaspekte,...) des Aggregierungsprozesses.

- **Berücksichtigung semantischer Informationen**: Semantische Informationen aus den RDF- Files werden im Rahmen des Importprozesses als Attribute den Nodes und Edges zugeordnet (Abschnitt 5.4.1). Auf diese Weise haben die einzelnen Komponenten zu jedem Zeitpunkt darauf Zugriff. Die *GUI*-Komponente (Abschnitt 5.2.1) bietet die Möglichkeit zur Darstellung näherer Informationen des RDF-Datensatzes (RDF-Statistik, Auflistung von Subjects, Predicates, Objects, etc.). In weiterer Folge werden semantische Informationen auch im Zuge des *Aggregation By Semantics* Prozesses berücksichtigt (siehe *Aggregation-Calculator* Komponente in Abschnitt 5.2.1 bzw. Abschnitt 5.4.2). Die *Style-Mapper* Komponente ist für die RDF-spezifische Darstellung der einzelnen Graphelemente verantwortlich (Abschnitt 5.2.1).

- **Grafische Benutzeroberfläche (GUI)**: Diese Forderung wird u.a. von der *GUI*-Komponente des VIOSEG Kerns (Bedienung bzw. Nutzung der Main-Features von VIOSEG; Abschnitt 5.2.1) bzw. von der Cytoscape Umgebung (siehe Abschnitt 4.2.1) erfüllt.

Kapitel 6

Case Study

Das Kapitel widmet sich der Demonstration der Features und Merkmale der im Rahmen der Masterarbeit entwickelten Visualisierungssystems VIOSEG. Die Software implementiert eine Reihe von Funktionalitäten um die zu Beginn dieser Arbeit erörterte Problemstellung zu adressieren. In Kapitel 5 wurden der grundsätzliche Aufbau sowie die technischen Details der Implementierung behandelt. Im Laufe dieses Kapitels werden verschiedene Anwendungsszenarien durchgespielt sowie Beispieldaten zur Vermittlung der Funktionalitäten des Tools visualisiert. Den Hauptteil bildet somit die Evaluierung der Software mittels verschiedener Testdatensätze.

6.1 Visuelle Kodierung

Die Visualisierung der einzelnen Elemente eines RDF-Graphen erfolgt mithilfe verschiedener visueller Kodierungen. Dabei kommen die von Cytoscape zur Verfügung gestellten Standard-Shapes (einfache geometrische Objekte) zum Einsatz. Abbildung 6.1 zeigt einen Überblick über die nachfolgend erläuterten Kodierungen.

Subjects und Objects eines RDF-Triples werden als einfache Nodes mithilfe von Kreisen einheitlicher Größe dargestellt. Object-Nodes, welche ausschließlich aus einem Literal-Value (also einem konkreten Wert und keiner URI-Referenz) bestehen, weisen zur besseren Identifikation eine hellblaue Füllfarbe auf. Die übrigen Nodes werden mittels dunkelblauer Kreise dargestellt. Zur Beschriftung der Nodes (= Node-Label) dient der letzte Teil der URI-Referenz (= *Local-Name*), da die vollständige URI mitunter

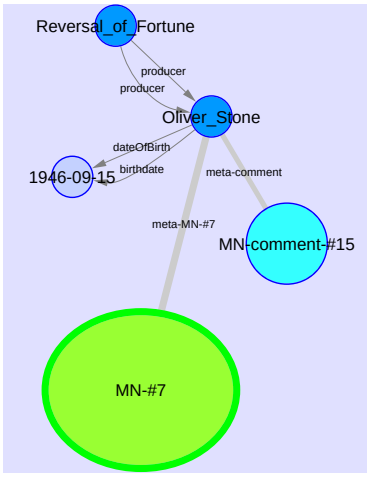


Abbildung 6.1: Visuelle Kodierungen in VIOSEG

zu viel Platz einnehmen würde. Bei Literal-Nodes wird der jeweilige Wert (String oder Integer) als Label verwendet. Aus Gründen der besseren Übersichtlichkeit sind Labels auf 20 Zeichen limitiert. Predicates des RDF-Triples werden als Edges in Form von grauen Pfeilen dargestellt. Die Pfeilspitze zeigt dabei auf den Target-Node. Edge-Labels bestehen wiederum aus dem Local-Name der URI-Referenz.

In Zuge der *Aggregation By Semantics* Methode (siehe Abschnitt 5.4.2) erstellte Meta-Nodes werden als türkise Kreise dargestellt. Die Größe variiert je nach Anzahl der aggregierten Nodes. Meta-Node Labels setzen sich aus einem Prefix (*MN*), dem aggregierten Edge-Type (z.B. *comment*) sowie der Anzahl der aggregierten Nodes (z.B. *#3*) zusammen. Die Darstellung von Meta-Edges erfolgt mittels grauer Linien (ohne Pfeilspitzen). Die Linienstärke dient dabei als Indikator für die Anzahl der darin aggregierten Edges des selben Typs. Meta-Edge Labels bestehen aus einem Prefix (*meta*) sowie dem Namen des Kantentyps, welcher am häufigsten innerhalb der Meta-Edge vorkommt (z.B. *comment*). Siehe dazu auch Abbildung 5.9.

Die Darstellung der im Zuge der *Aggregation By Clustering* Methode entstandenen Meta-Nodes erfolgt in Form von grünen Kreisen mit variabler Höhe und Breite. Höhe bzw. Breite ergeben sich dabei aus dem Bounding-Polygon der aggregierten Nodes, wodurch mitunter elliptische Formen auftreten können. Node-Labels bestehen in diesem Fall aus einem Prefix und der Anzahl der aggregierten Nodes, Edge-Labels aus einem Prefix (*meta*) und der ID der jeweiligen Meta-Node. Diese Kodierung soll eine eindeutige Unterscheidung zwischen Meta-Nodes, welche auf Basis semantischer Informationen, und Meta-Nodes, welche aufgrund von K-Means bzw. Hierarchical Clustering-Verfahren (Basis: Euklidische Distanz) erstellt wurden, ermöglichen.

Nach dem Expand eines Meta-Nodes werden die darin enthaltenen Knoten mit einem roten Rand dargestellt. Mithilfe dieser optischen Hervorhebung sollen soeben aufgefächerte Nodes vom User sofort identifiziert und von anderen Nodes unterschieden werden können.

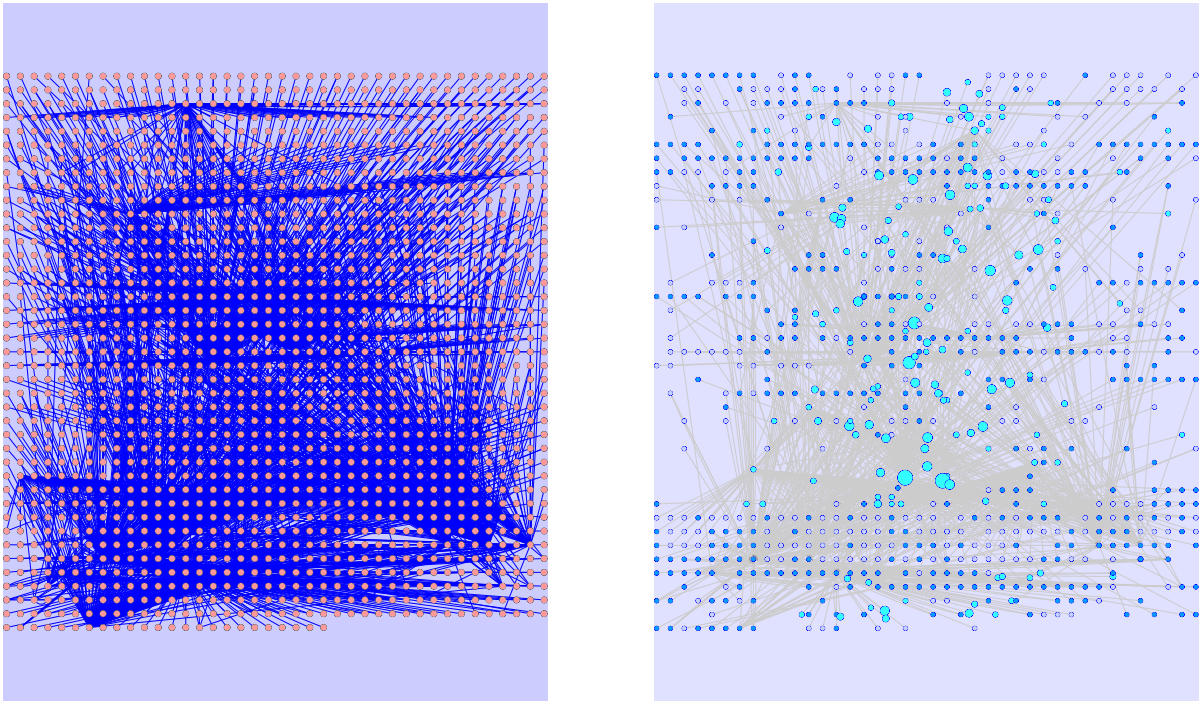
6.2 Demonstration der Features

An dieser Stelle sollen zunächst die Features und speziellen Funktionen des VIOSEG Tools zur Visualisierung und dynamischen Aggregierung anhand eines RDF-Graphen demonstriert werden. Dieser besteht aus einer Kombination von 29 RDF-Datensätzen, welche jeweils unterschiedliche Dinge (Resources) beschreiben, darunter Personen, Filme, Berufe, Videospiele, Konzepte, etc. In Summe umfasst der Datensatz in etwa 2100 Statements (RDF-Triple).

6.2.1 Aggregation By Semantics

Abbildung 6.2 (Bild **a**) zeigt den RDF-Graphen unmittelbar nach dem Import (standardmäßiges Cytoscape Grid-Layout bzw. Default Visual Style). Aufgrund der hohen Node- bzw. Edge-Dichte kommt es zu Überschneidungen der einzelnen Kanten, wodurch sich kaum brauchbare Informationen aus dieser Visualisierung ableiten lassen. Auf Bild **b** ist derselbe Graph nach Anwendung der *Aggregation By Semantics* Methode im VIOSEG spezifischen Visual Style (siehe Abschnitt 6.1) zu sehen. Zum besseren Vergleich wird der Graph vorerst noch im Grid-Layout dargestellt. Im Zuge des Aggregierungsprozesses wurden, wie in Abschnitt 5.4.2 beschrieben, ausgehende Kanten des selben Typs zu Meta-Edges bzw. die Target-Nodes dieser Kanten zu Meta-Nodes zusammengefasst. Als Node-Priority Criterion diente dabei der innerhalb des Datensatzes am häufigst auftretende Kantentyp *subject*. Auf diese Weise konnte die Gesamtelementanzahl um ca. 50% gegenüber der ursprünglichen Darstellung verringert werden.

Abbildung 6.3 zeigt den Graphen nach Berechnung eines Force-Directed Layouts. Dabei wurde die im Zuge des Aggregierungsvorgangs ermittelte Kantengewichtung berücksichtigt. Dichte Regionen können mithilfe dieser Darstellung auf einen Blick erkannt werden. Eine solche Region repräsentiert dabei meist eine bestimmte Resource. Aufgrund der gewählten Zoomstufe werden Einzelheiten wie beispielsweise Node- und Edge-Labels mithilfe des Level-Of-Detail Features von Cytoscape ausgeblendet.



(a) Originalgraph **(b)** Aggregierung

Abbildung 6.2: Aggregation By Semantics: (a) Graph direkt nach Import (Grid-Layout, Default Visual Style) (b) Graph nach semantischer Aggregierung (Grid-Layout, VIOSEG Visual Style)

Weiters sind sämtliche Meta-Nodes (türkise Knoten) zusammengeklappt.

6.2.2 Aggregation By Clustering

Bild **a** in Abbildung 6.4 zeigt den Graphen aus 6.3 nach Anwendung der *Aggregation By K-Means Clustering* Methode (Konfiguration: 2 Iterationen, 29 zu identifizierende Cluster). *Aggregation By Hierarchical Clustering* (Konfiguration: Pairwise Average-Linkage, 29 zu identifizierende Cluster) liefert annähernd das selbe Ergebnis, ist jedoch in Verbindung mit größeren Graphen weniger performant. Wie in Abschnitt 5.4.2 beschrieben, dient beiden Methoden die euklidische Distanz zwischen den einzelnen Knoten als Basis zur Cluster-Identifikation. Die in Abbildung 6.3 gut erkennbaren dichten Regionen können aufgrund dessen zu Cluster bzw. Meta-Nodes zusammengefasst werden. Die visuelle Komplexität des dargestellten Graphen wird auf diese Weise entscheidend verringert.

Bild **b** der Abbildung 6.4 zeigt das Ergebnis der zweiten Variante des implementierten *Aggregation By Hierarchical Clustering* Verfahrens. Die Anzahl der Cluster wird dabei von einem vom User gewählten *Detail Level* Parameter bestimmt (Details dazu siehe Abschnitt 5.4.2). Diese Variante ist vor allem dann sinnvoll, wenn der User die exakte Zahl der zu identifizierenden Cluster nicht kennt. In diesem Beispiel wurde ein Detail Level von 2 gewählt, d.h. alle Cluster bis zur 2. Stufe der im Zuge des Hierarchical Clusterings erstellten Cluster-Baumstruktur werden als Meta-Nodes dargestellt. Je höher dieser Integer-Wert, desto mehr Nodes werden in den einzelnen Meta-Nodes zusammengefasst. Der maximale Detail Level-Wert hat somit einen einzigen Meta-Node (aggregiert alle Nodes des Graphen) zur Folge. Hierarchical Clustering (Variante II; Detail Level 2) in Verbindung mit der zuvor durchgeführten semantischen Aggregierung reduziert somit die Gesamtelementanzahl im Vergleich zum ursprünglichen Graphen um 82%.

Die einzelnen Aggregierungstechniken können beliebig oft, bzw. in willkürlicher Reihenfolge an-

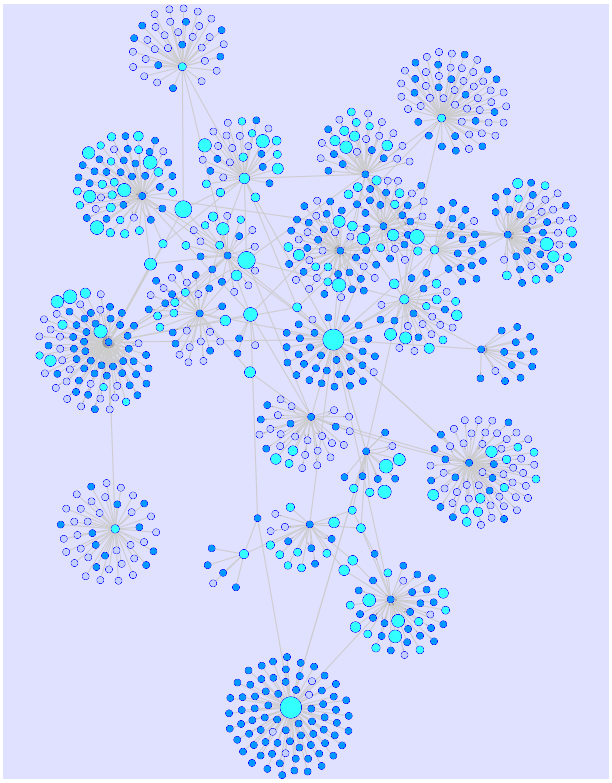
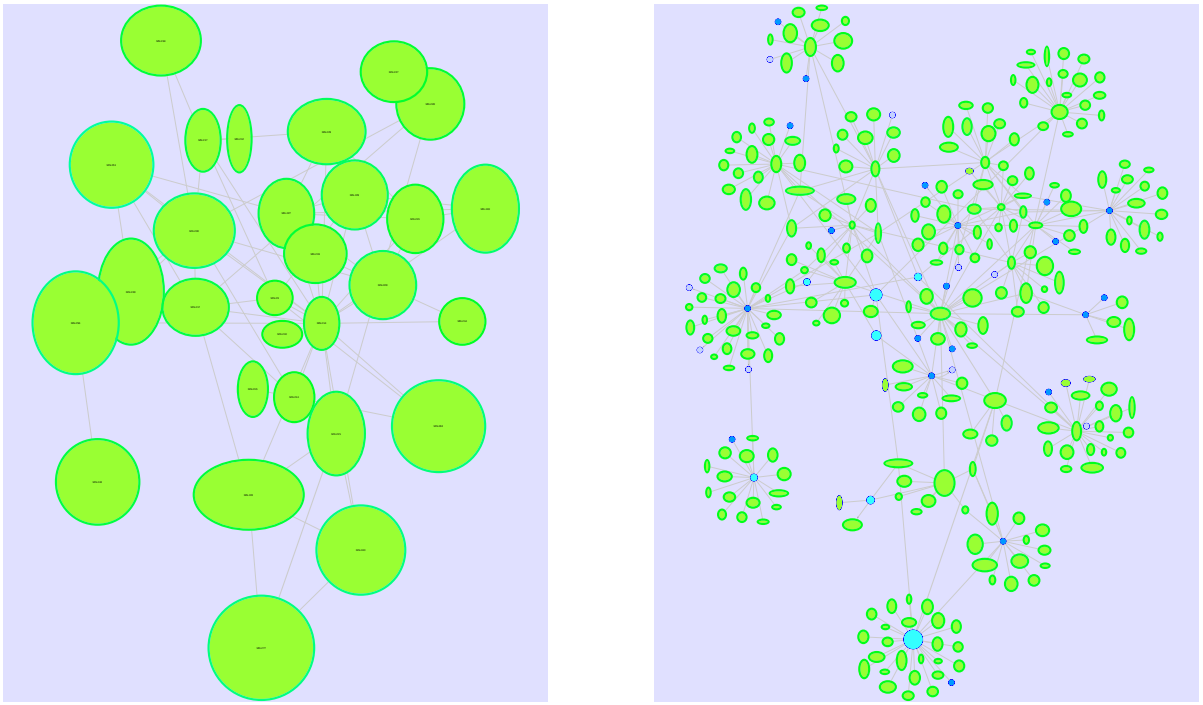


Abbildung 6.3: Force-Directed Layout (Berücksichtigung der Kantengewichtungen) des Graphen aus Abbildung 6.2



(a) K-Means **(b)** Hierarchical (Variante II)

Abbildung 6.4: Aggregation By Clustering: Mittels K-Means bzw. Hierarchical Clustering (Variante II) aggregierter Graph aus Abbildung 6.3 (a) Graph nach K-Means Clustering (29 fixe Cluster, 2 Iterationen) (b) Graph nach Hierarchical Clustering, Variante II (Detail Level 2, Pairwise Average-Linkage)

gewendet werden. Hinsichtlich einer optimalen Komplexitätsreduktion empfiehlt es sich aber, Elemente zuerst mittels *Aggregation By Semantics* zu aggregieren und die verbleibenden Nodes/Edges mittels *Aggregation By Clustering* zusammenzufassen. Der auf diese Weise entscheidend vereinfachte Graph kann anschließend ohne Probleme visualisiert bzw. Interaktionen (Zoom, Pan,...) können ohne merkliche Verzögerungen durchgeführt werden. Mithilfe des dynamischen Aggregierungs-Features lassen sich bei Bedarf Details ein- und wieder ausblenden.

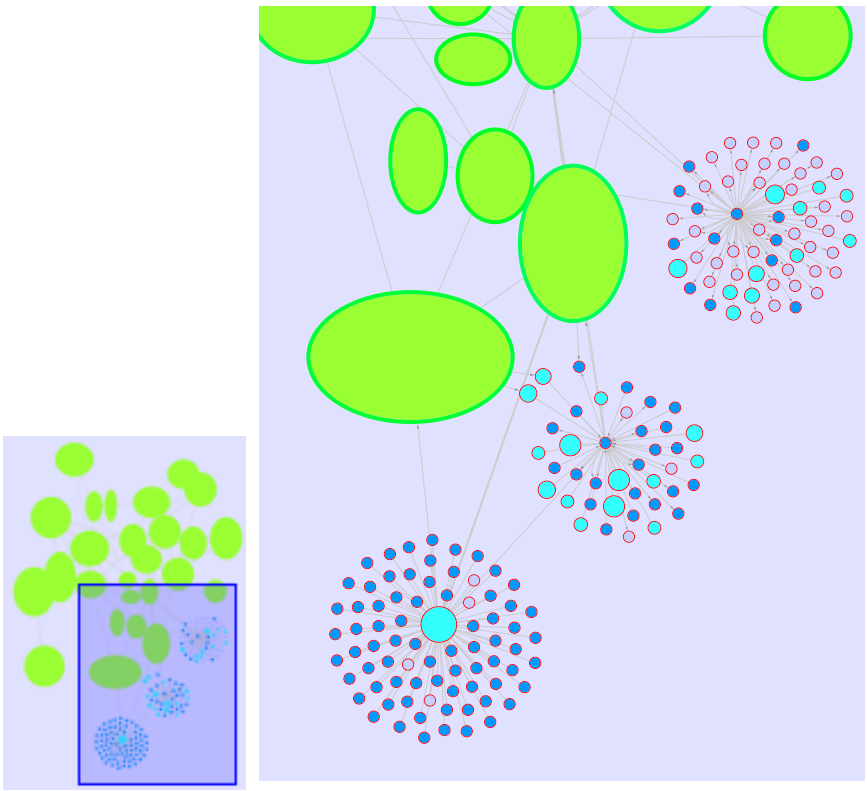
6.2.3 Dynamische Aggregierung

Die im Zuge dieses Aggregierungsvorgangs erzeugten Meta-Nodes werden in Abhängigkeit des Betrachtungsausschnitts bzw. der aktuellen Zoomstufe automatisch aufgefächert bzw. zusammengeklappt (Expand/Collapse; Implementierungsdetails siehe Abschnitt 5.4.3). Bild **b** aus Abbildung 6.5 zeigt einen vergrößerten Ausschnitt des Graphen aus 6.4. Aufgrund des höheren Zoomfaktors werden die größeren Cluster-Meta-Nodes (grün) innerhalb des Viewing-Rectangles automatisch aufgefächert und die darin aggregierten Elemente sichtbar. Meta-Nodes außerhalb des Bildausschnitts bzw. zu weit 'entfernte' Meta-Nodes (hinsichtlich Relation Knotengröße - Zoomfaktor) bleiben zugeklappt. Wird das Viewing-Rectangle verschoben oder eine niedrigere Zoomstufe (Zoom-Out) gewählt, so erfolgt wiederum ein Collapse der entsprechenden Meta-Nodes. Bild **c** der Abbildung 6.5 zeigt den Graph nach einer weiteren Zoom-In Operation. Aufgrund der hohen Zoomstufe werden nun auch Details wie etwa Node- und Edge-Labels dargestellt.

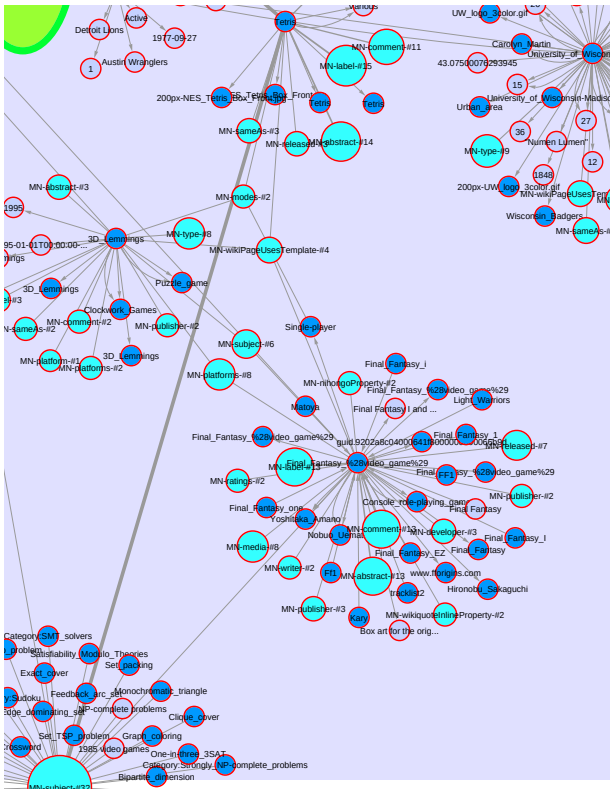
In Abbildung 6.6 ist ein weiteres Beispiel für das dynamische Aggregierungs-Feature von VIOSEG zu sehen. Dabei wird nach einem Zoom-In Schritt der zentrale Meta-Node aus Bild **a** automatisch aufgefächert. Bild **b** zeigt den Graphen nach dieser Expand-Operation. Die soeben aufgefächerten Knoten werden zur besseren Unterscheidung bzw. Identifikation mit einem roten Rand versehen. Außerdem wird sichergestellt, dass alle Nodes innerhalb des aktuellen Viewing-Rectangles sichtbar sind. Dadurch kann es mitunter zu Überlappungen einzelner Nodes kommen. Abbildung 6.7 demonstriert ebenfalls die dynamische Aggregierungsfunktionalität anhand eines weiteren Beispielgraphen.

6.3 Datenquelle

Die Testdaten zur Visualisierung stammen von *DBpedia* [DBpedia, 2009]. Dabei handelt es sich um eine Wissensdatenbank, deren Inhalte unter der *GNU Free Documentation License* im Web verfügbar sind. DBpedia extrahiert strukturierte Informationen von *Wikipedia* [Wikipedia, 2009] und kombiniert diese in einer bereichsübergreifenden Wissensbasis. Das Open Community Projekt wurde von der Freien Universität Berlin, der Universität Leipzig und dem Softwarehersteller OpenLink Software ins Leben gerufen. Die Datenbank umfasst derzeit in etwa 247 Millionen RDF-Triple, extrahiert aus 14 verschiedensprachigen Versionen von Wikipedia. Die Multi-Domain Ontology von DBpedia setzt sich aus ca. 170 Klassen bzw. Sub-Klassen zusammen (u.a. *Place, Person, Organisation, Work, Event, Species,...)*. Insgesamt werden 2,6 Millionen Dinge (Things bzw. Resources) beschrieben, darunter Personen, Orte, Musikalben, Filme, Companies, etc. (Stand: November 2008). Jedes Ding wird durch eine URI-Reference eindeutig identifiziert und mithilfe verschiedener Eigenschaften (Properties) beschrieben (z.B. Kurzbeschreibung, Link auf Wikipedia-Page bzw. Bild, Beschreibung in verschiedenen Sprachen, Klassifizierung, etc.). Daten von DBpedia sind mit einer Reihe von externen Datenquellen verlinkt (z.B. *WordNet, Freebase, Flickr Wrappr, OpenCyc, Project Gutenberg,...*). Die einzelnen DBpedia Kerndatensätze sind im N-Triple- bzw. CSV-Format, Datensätze aus externen Quellen ausschließlich als N-Triples verfügbar. Die Beschreibung einer Resource (z.B. einer Stadt, eines Landes, einer Person, usw.) ist eine Kombination aus DBpedia Kerndatensätzen und externen Datensätzen und kann in den Formaten RDF/XML bzw. Notation3 heruntergeladen werden.

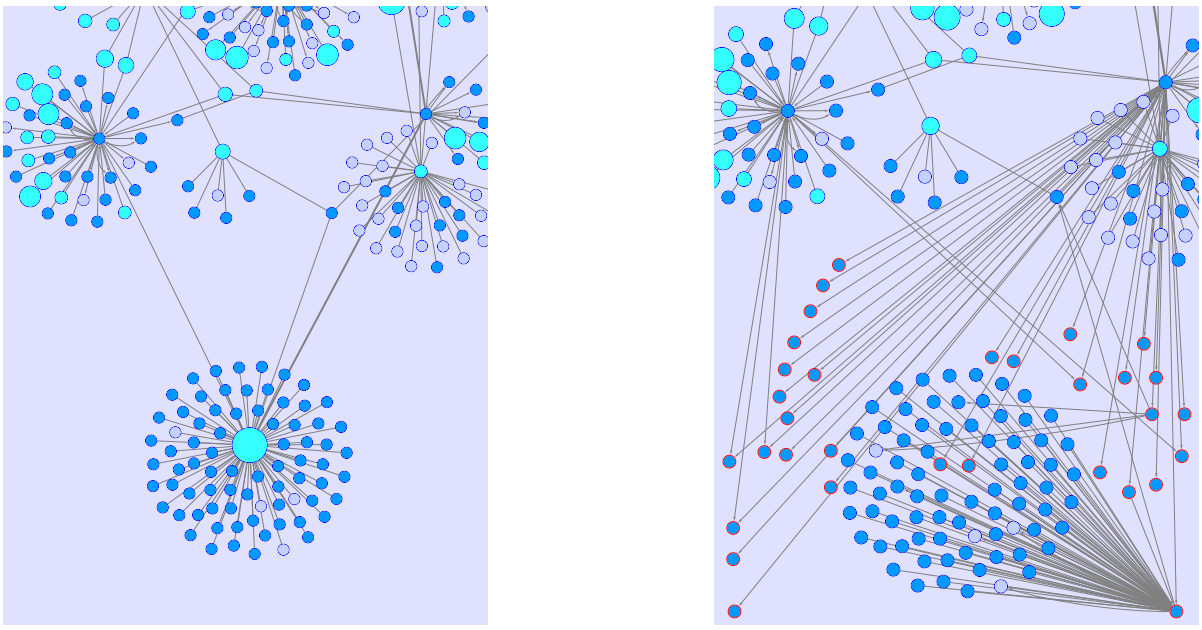


(a) Übersicht **(b)** Zoom+



(c) Zoom++

Abbildung 6.5: Dynamische Aggregierung: Demonstration des automatischen Expand/Collapse von Meta-Nodes anhand des Graphen aus Abbildung 6.4. (a) Overview-Window mit aktuellem Viewing-Rectangle (b) Zoom auf einen Ausschnitt des Graphen. Die größten drei Meta-Nodes innerhalb des Viewing-Rectangles wurden aufgefächert. (c) Erneutes Zoom-In zeigt Details der Darstellung (u.a Labels)



(a) Collapse **(b)** Expand

Abbildung 6.6: Dynamische Aggregierung: Automatisches Expand/Collapse von Meta-Nodes (a) Graph vor dem Expand der zentralen Meta-Node (türkiser Knoten; 32 aggregierte Nodes) (b) Graph nach dem Expand. Aggregierte Nodes (roter Rand) bzw. Edges werden innerhalb des Bildausschnitts dargestellt.

6.4 Evaluierung

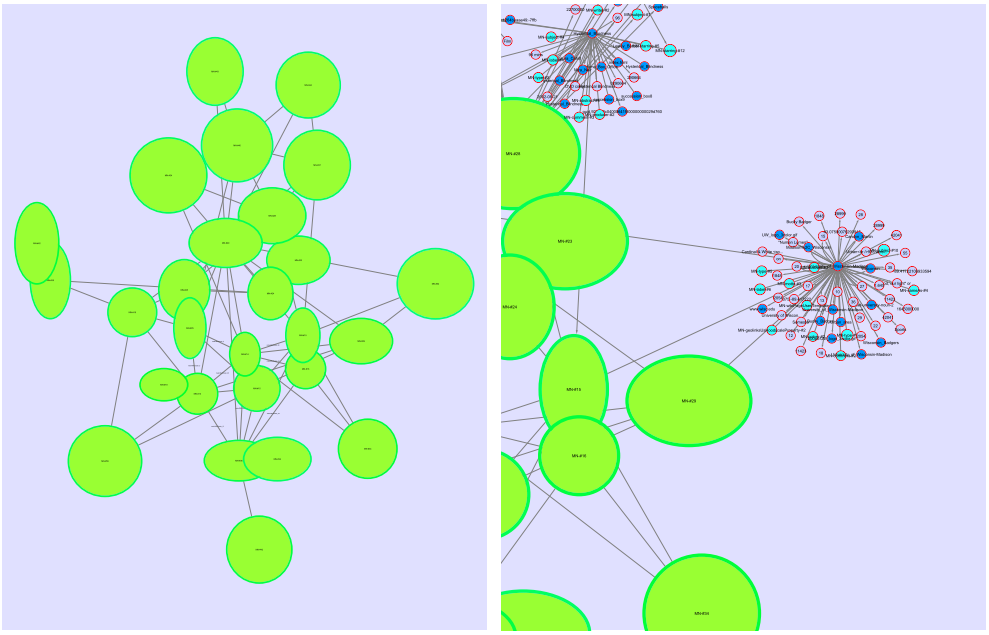
Im Rahmen der Evaluierung bzw. der Präsentation des VIOSEG Tools kommen RDF-Daten aus der DBpedia Wissensdatenbank zum Einsatz. Einzelne Beschreibungen von unterschiedlichen Resources werden dabei zu größeren Testdatensätzen kombiniert und im RDF/XML-Format gespeichert. Ein Überblick über die getesteten Datensätze findet sich in Tabelle 6.1. Die jeweiligen Testfälle bzw. Anwendungsbeispiele sollen die zu Beginn dieser Arbeit erwähnten Problemstellungen widerspiegeln sowie die von VIOSEG implementierten Features zur Bewältigung ebendieser Probleme und Herausforderungen demonstrieren.

Die Evaluierung von VIOSEG (Version 1.0) erfolgte auf einem Intel Dual-Core 1.66GHz mit 1GB Arbeitsspeicher und einer NVIDIA GeForce Go 7400 Grafikkarte. Cytoscape in der Version 2.6.2 wurde mit einer Speicherzuweisung von 512MB sowie einem Stack Size von 5MB initialisiert.

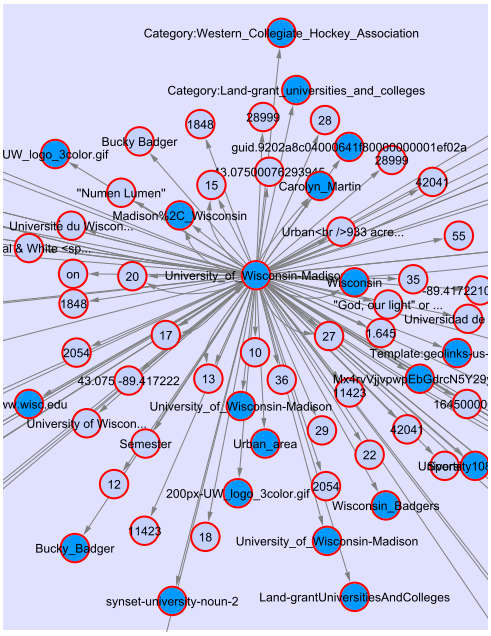
6.4.1 Ablauf

Der Testablauf folgt prinzipiell einem vordefinierten Schema mit jeweils gleichen bzw. ähnlichen Parametern für alle Testfälle. Kleinere Variationen der Konfiguration ergeben sich aus den Unterschieden zwischen den einzelnen Datensätzen. Die Evaluierung umfasst grundsätzlich die nachstehend angeführten Schritte. Die Ergebnisse der einzelnen Schritte werden gesammelt und anschließend gegenübergestellt.

1. Import und Ermittlung der Ausgangssituation
2. Layout des originalen Graphen
3. Aggregierung auf Basis semantischer Informationen (*Aggregation By Semantics*)
4. Layout des auf diese Weise vereinfachten Graphen



(a) Ausgangssituation **(b)** Zoom+



(c) Zoom++

Abbildung 6.7: Dynamische Aggregierung: Demonstration des automatischen Expand/Collapse von Meta-Nodes. (a) Alle Meta-Nodes sind zugeklappt. (b) Zoom auf rechten Ausschnitt des Graphen. Die größten hellgrünen Meta-Nodes innerhalb des Viewing-Rectangles werden aufgefächert. (c) Weiteres Zoom-In. Nun werden auch alle türkisen Meta-Nodes aufgefächert.

	Testfall I	Testfall II	Testfall III	Testfall IV	Testfall V	Testfall VI	Testfall VII
# Nodes	1.624	16.857	37.521	10.306	9.071	825	4.555
# Edges	2.082	27.428	55.346	18.913	12.580	1.126	6.754
# Attribute pro Node	20	20	20	20	20	20	20
# Attribute pro Edge	11	11	11	11	11	11	11
Dateigröße	551 KB	6.166 KB	13.450 KB	4.037 KB	2.639 KB	303 KB	1.564 KB
# kombinierter Datensätze	29	28	180	1	1	10	14
# Statements (RDF-Triple)	2.082	27.428	55.377	18.913	12.580	1.126	6.759
# Literals	690	2.644	6202	113	113	395	888
Importzeit (Grid Layout) in Sekunden	5,2	16,5	29	10	9	4	6
Anmerkung	kleinere Datensätze (max. 251 Statements/Daten satz); verschiedene Kategorien	mittlere und größere Datensätze (bis zu 6700 Statements/ Datensatz); Kategorie *Country*	kleine, mittlere und große Datensätze; verschiedenen Kategorien	Einzeldatensatz *Australia*	Einzeldatensatz *Spain*	kleinere Datensätze; Kategorie *Film*	kleine und mittlere Datensätze; verschiedene Kategorien

Tabelle 6.1: Details zu den einzelnen Testfällen

5. Aggregierung der verbleibenden Elemente des reduzierten Graphen mittels Clustering-Techniken (*Aggregation By Clustering*)

Nach dem Import eines Testdatensatzes werden im ersten Schritt einige Informationen zum originalen Graphen, wie etwa die Anzahl der vorhandenen Nodes, Edges bzw. RDF-Statements, ermittelt (= Ausgangssituation). Danach erfolgt das Layout des originalen Graphen mithilfe des *Force-Directed* sowie des *Organic*- bzw. *Circular*-Algorithmus (Schritt 2). Die visuelle Repräsentation sowie die Zeit zur Berechnung des jeweiligen Layouts dient, neben der ursprünglichen Node- und Edge-Anzahl, der anschließenden Gegenüberstellung von Ausgangssituation und Situation nach Durchführung der verschiedenen Aggregierungsverfahren von VIOSEG. Auf diese Weise soll ein objektiver Vergleich bzw. eine objektive Bewertung der Ergebnisse ermöglicht werden.

In der nächsten Phase (Schritt 3) werden Nodes und Edges des originalen Graphen mithilfe der *Aggregation By Semantics* Methode zu Meta-Nodes bzw. Meta-Edges zusammengefasst. Neben der veränderten Anzahl an Elementen wird dabei auch die Zeit zur Berechnung der Aggregierung ermittelt. Die Aggregierung wird dabei mit 4 unterschiedlichen Konfigurationen jeweils unabhängig voneinander durchgeführt. Der auf Basis dieser Verfahren vereinfachte Graph dient als Ausgangspunkt für die nachfolgenden Schritte 4 und 5. Die einzelnen Verfahren unterscheiden sich hinsichtlich des Node-Priority Criterions (siehe auch 5.4.2):

- *Edge-Type*: Der im gesamten Graphen am häufigst auftretende Kantentyp dient als Prioritaetskriterium. D.h. jene Nodes, welche die größte Zahl an ausgehenden Kanten dieses Typs aufweisen, erhalten eine hohe Priorität.
- *Out-Degree*: Nodes mit der größten Anzahl an ausgehenden Kanten werden priorisiert.
- *In-Degree*: Nodes mit der größten Anzahl an eingehenden Kanten werden priorisiert.
- *Degree*: Nodes mit der größten Anzahl an ein- und ausgehenden Kanten werden priorisiert.

Die auf diese Weise reduzierte Anzahl an Nodes und Edges wird der ursprünglichen Elementanzahl gegenübergestellt. Im Zuge dessen wird auch die Reduktion der Node-, Edge- und Gesamtelementanzahl im Vergleich zur Elementanzahl des originalen Graphen berechnet. (Anmerkung: Meta-Nodes und Meta-Edges werden auf die selbe Weise wie normale Nodes und Edges gezählt. Beispiel: Der Originalgraph besteht aus 2 Nodes, welche durch 2 unterschiedliche Edges miteinander verbunden sind. Im Zuge der Aggregierung dieser beiden Edges zu einer Meta-Edge werden aufgrund der Algorithmus-Implementierung 2 Meta-Nodes mit jeweils nur einem darin aggregierten Node erzeugt. Der vereinfachte Graph besteht somit aus 3 Elementen: 2 Nodes (beide Meta-Nodes) und einer Meta-Edge)

In Schritt 4 erfolgt das Layout des aufgrund der semantischen Aggregierung vereinfachten Graphen (Force-Directed Layout mit Berücksichtigung des Kantengewichts sowie Organic- bzw. Circular Layout). Die Zeit für die Berechnung des Layouts wird wiederum in Relation zur unter Schritt 2 ermittelten Layout-Berechnungszeit des originalen Graphen gesetzt. Weiters erfolgt eine Gegenüberstellung der visuellen Repräsentation vor und nach der Aggregierung bzw. nach der Anwendung des VIOSEG spezifischen Visual Styles.

Abschließend werden im 5. Schritt verschiedene *Aggregation By Clustering* Verfahren auf den mittels semantischer Aggregation vereinfachten Graphen (Schritt 3) angewendet. Dabei wird das Force-Directed Layout aus Schritt 4 beibehalten. Wie in Schritt 3 erfolgt ein Vergleich der auf diese Weise reduzierten Anzahl an Nodes und Edges mit der ursprünglichen Elementanzahl. Folgende Clustering-Varianten werden unabhängig voneinander getestet:

- *K-Means Clustering*: Fixe Anzahl von Cluster; 2 Iterationen
- *Hierarchical Clustering*: Fixe Anzahl von Cluster; Pairwise Average Linking
- *Hierarchical Clustering (Variante II)*: Variable Anzahl von Cluster (Detail Level 2); Pairwise Average Linking

6.4.2 Testfall I

Dieser Testfall ist eine Kombination aus 29 kleineren DBpedia-Datensätzen (maximal 260 RDF- Statements) welche u.a. Dinge aus den Kategorien bzw. Klassen *Person, Actor, Politician, Video Game* und *Film* beschreiben. In Summe umfasst der Testdatensatz 2.082 Statements (RDF-Triple). Der originale Graph besteht somit aus 1.624 Nodes und 2.082 Edges. Pro Node finden sich 20, pro Edge 11 Attribute. Die gesamte RDF-Datei hat eine Größe von 551KB (siehe Tabelle 6.1). Der selbe Datensatz wurde bereits in Abschnitt 6.2 zur Demonstration der VIOSEG Main-Features verwendet.

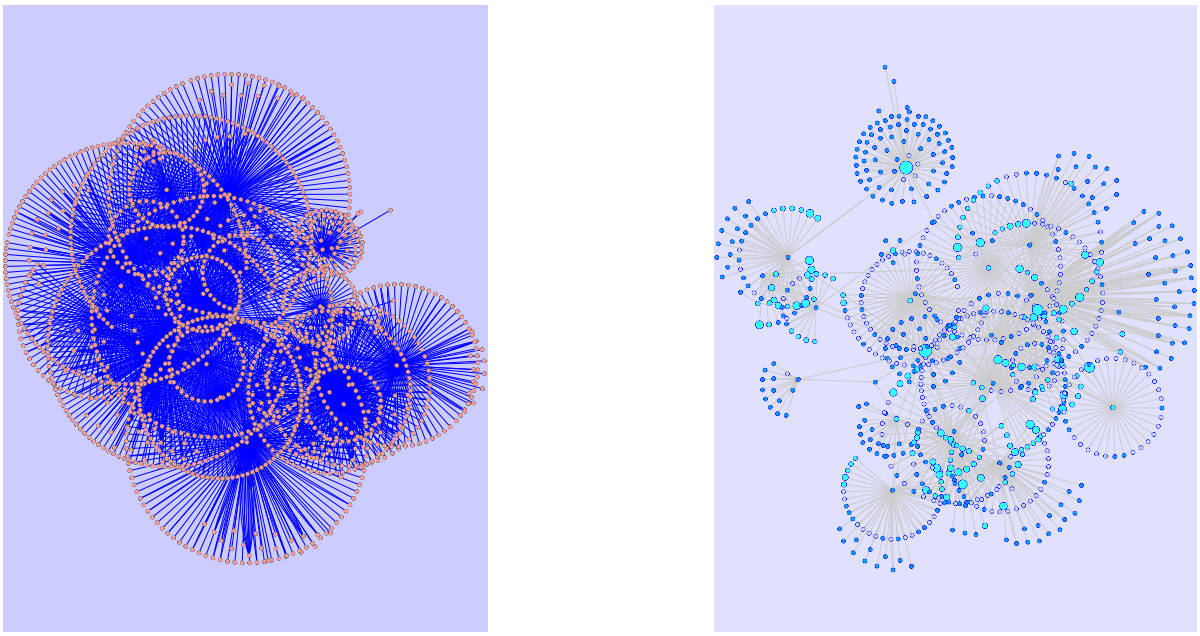
Die Durchführung der semantischen Aggregierung bewirkte eine Reduktion der Gesamtelementanzahl um ca. 50% gegenüber dem ursprünglichen Graphen (siehe dazu auch Abbildung 6.2). Dabei zeigte sich, dass das Verfahren 1 (Edge-Type *subject*) bzw. das Verfahren 2 (Out-Degree) die besten Ergebnisse liefert. Aufgrund dieser Reduktion konnte die Berechnungszeit des Force-Directed Layouts (Abbildung 6.3) um 62% bzw. die Berechnungszeit des Organic Layouts um 50% gegenüber der ursprünglichen Layout-Zeit verkürzt werden. Abbildung 6.8 zeigt den Graphen vor bzw. nach Durchführung der semantischen Aggregierung (Organic Layout).

Die Elementanzahl des vereinfachten Graphen im Force-Directed Layout wurde abschließend mithilfe von K-Means (2 Iterationen) bzw. Hierarchical Clustering (Pairwise Average Linking) nochmals entscheidend verringert. Die Konfiguration mit einer fixen Anzahl von 29 Clustern führte bei beiden Algorithmen zu einer Reduktion der Gesamtelementanzahl um 98% gegenüber dem ursprünglichen Graphen. Die zweite Variante des Hierarchical Clustering (keine fixe Cluster-Anzahl, Pairwise Average Linking, Detail-Level 2) resultierte in einer Elementreduktion um 81% (Abbildung 6.4). Erwartungsgemäß benötige die K-Means Variante die kürzeste Berechnungszeit (ca. die Hälfte der beiden Hierarchical Varianten).

Eine Übersicht über die gesammelten Ergebnisse zu Testfall I findet sich in den Tabellen 6.2 und 6.3.

6.4.3 Testfall II

Der zweite Testfall setzt sich aus 28 mittleren bis größeren DBpedia-Datensätzen der Kategorie *Country* zusammen. Die Anzahl der RDF-Triple pro Datensatz bewegt sich dabei im Bereich von 230 bis maximal 6.700. Der gesamte Testdatensatz besteht somit aus 27.428 Statements. Dies resultiert in einem Graphen mit 16.857 Nodes und 27.428 Edges. Das File weist eine Größe von 6.166KB auf. Weitere Details zu diesem Testfall finden sich in Tabelle 6.1.



(a) Originalgraph **(b)** Semantische Aggregierung

Abbildung 6.8: Testfall I: Aggregation By Semantics (a) Ursprünglicher Graph (Organic Layout, Default Visual Style) (b) Vereinfachter Graph (Organic Layout, VIOSEG Visual Style); Reduktion der Gesamtelementanzahl um ca. 50%

Testfall I	**# Nodes** ursprünglich	**# Edges** ursprünglich	**gesamt**
Elementanzahl	1.624	2.028	3.652

	# Nodes	**# Edges**	**gesamt**	**Reduktion Nodes**	**Reduktion Edges**	**Reduktion gesamt**	
Semantic Aggregation 1 (Edge-Type: "*subject*")	830	995	1.825	49%	51%	50,0%	***Aggregation By Semantics***
Semantic Aggregation 2 (Out-Degree)	829	994	1.823	49%	51%	50,1%	
Semantic Aggregation 3 (In-Degree)	1.624	1.874	3.498	0%	8%	4,2%	
Semantic Aggregation 4 (Degree)	938	1.026	1.964	42%	49%	46,2%	
Hierarchical Clustering (pairwise average linking; Detail-level 2)	305	366	671	81%	82%	81,6%	***Aggregation By Clustering*** (Ausgangslage: Semantic Aggregation 1)
Hierarchical Clustering (pairwise average linking; fixe Clusteranzahl: 29)	29	61	90	98%	97%	97,5%	
K-Means Clustering (2 Iterationen; fixe Clusteranzahl: 29)	29	68	97	98%	97%	97,3%	

Tabelle 6.2: Testfall I: Reduktion der Elementanzahl

Testfall I

Berechnungszeiten (Sekunden)

Elemente gesamt ursprünglich	Force-Directed Layout Berechnungszeit ursprünglich	Organic Layout Berechnungszeit
3.652	13	4

	Elemente gesamt nach Aggregation	Aggregation Zeit	Force-Directed Layout Zeit	Organic Layout Zeit	Zeitersparnis Force-Directed Layout	Zeitersparnis Organic Layout
Semantic Aggregation 1 (Edge-Type: "*subject*")	1.825	3	5	2	62%	50%
Semantic Aggregation 2 (Out-Degree)	1.823	3	5	2	62%	50%
Semantic Aggregation 3 (In-Degree)	3.498	4	13	3	0%	25%
Semantic Aggregation 4 (Degree)	1.964	3	6	2	54%	50%
Hierarchical Clustering (pairwise average linking; Detail-level 2)	671	4	1	1	92%	75%
Hierarchical Clustering (pairwise average linking; fixe Clusteranzahl: 29)	90	4	0	0	100%	100%
K-Means Clustering (2 Iterationen; fixe Clusteranzahl: 29)	97	2	0	0	100%	100%

Aggregation By Semantics

Aggregation By Clustering (Ausgangslage: Semantic Aggregation 1)

Tabelle 6.3: Testfall I: Berechnungszeiten (Zeitangaben in Sekunden)

Da es sich ausschließlich um Daten der Klasse *Country* (einer Sub-Klasse von *Place*) handelt, sind die meisten Kanten vom Typ *birthplace*. Die semantische Aggregierung mit diesem Kantentyp als Node-Priority Criterion führte zu einer Reduktion der Kantenanzahl um 38% bzw. der Knotenanzahl um lediglich 5% (Gesamtreduktion somit ca. 26%). Das beste Ergebnis konnte mithilfe der zweiten Variante der semantischen Aggregierung (Prioritätskriterium Out-Degree) erzielt werden. Dabei wurde die Gesamtelementanzahl gegenüber dem originalen Graphen um ca. 32% reduziert. Die Berechnung des Force-Directed Layouts des auf diese Weise vereinfachten Graphen nahm somit in etwa 21% weniger Zeit in Anspruch. Die Berechnungszeit des Organic Layouts verkürzte sich nur minimal. Abbildung 6.9 zeigt eine Gegenüberstellung des originalen Graphen und des mittels *Aggregation By Semantics* vereinfachten Graphen (beide im Force-Directed Layout). Die meisten Meta-Nodes (türkise Punkte) finden sich dabei im Zentrum des Graphen (Bild **b**). Die dichten Bereiche am Rand, welche die größeren Datensätze des Testfalls repräsentieren, blieben im Zuge der Aggregierung nahezu unverändert.

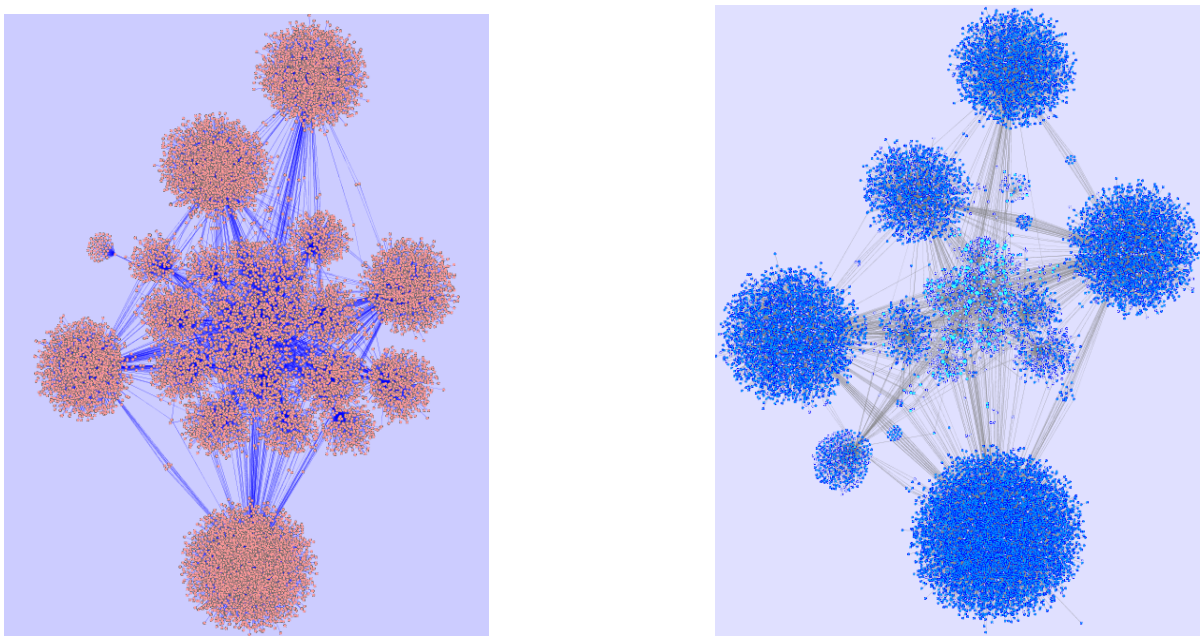
Aufgrund der höheren Elementanzahl in Verbindung mit der eingangs beschriebenen Konfiguration der Testumgebung (Speicherzuweisung, Stack Size) konnten die beiden Hierarchical Clustering Varianten nicht durchgeführt werden. Das effizientere K-Means Verfahren lieferte jedoch ohne Probleme die gewünschte Anzahl von 28 Clustern (Meta-Nodes). Abbildung 6.10 zeigt den zuvor mittels *Aggregation By Semantics* vereinfachten Graphen (Force-Directed Layout) nach Anwendung der *Aggregation By K-Means Clustering* Methode.

Eine Übersicht über die gesammelten Ergebnisse zu Testfall II findet sich in den Tabellen 6.4 und 6.5.

6.4.4 Testfall III

Der Testfall ist eine Kombination aus 180 DBpedia-Datensätzen unterschiedlicher Größen und Kategorien. In Summe umfasst der Testdatensatz in etwa 6.200 RDF-Triple. Der visualisierte Graph setzt sich aus ca. 38.000 Nodes und 55.000 Edges mit jeweils 20 Node- bzw. 11 Edge-Attributen zusammen. Die RDF-Datei besitzt eine Größe von 13.450KB (siehe auch Tabelle 6.1).

Tabelle 6.6 zeigt die Ergebnisse der einzelnen Aggregierungen. Die semantischen Aggregierungs-Verfahren 1 (Edge-Type *subject*) bzw. Verfahren 2 (Out-Degree) brachten bei annähernd gleicher Ag-



(a) Originalgraph **(b)** Semantische Aggregierung

Abbildung 6.9: Testfall II: Aggregation By Semantics (a) Ursprünglicher Graph (Force-Directed Layout (unweighted), Default Visual Style) (b) Vereinfachter Graph (Force-Directed Layout (weighted), VIOSEG Visual Style); Reduktion der Gesamtelementanzahl um ca. 32%

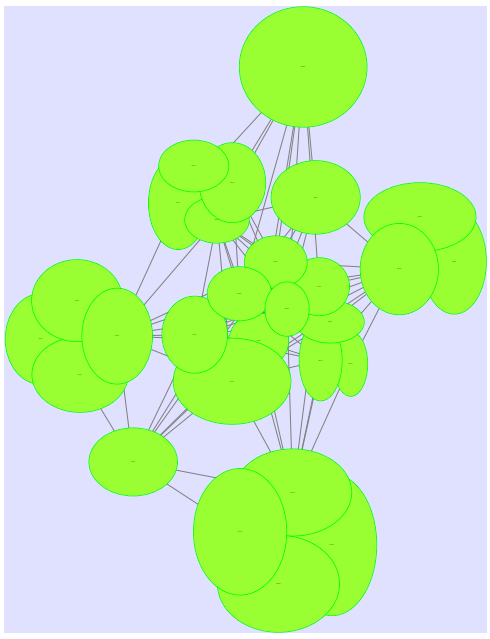


Abbildung 6.10: Testfall II: Graph aus Abbildung 6.9 (Bild **b**) nach Durchführung der *Aggregation By K-Means Clustering* Methode (2 Iterationen, 28 fixe Cluster)

Testfall II	**# Nodes** ursprünglich	**# Edges** ursprünglich	**gesamt**
Elementanzahl	16.857	27.428	44.285

	# Nodes	**# Edges**	**gesamt**	**Reduktion Nodes**	**Reduktion Edges**	**Reduktion gesamt**
Semantic Aggregation 1 (Edge-Type: "birthplace")	16.033	16.886	32.919	5%	38%	25,7%
Semantic Aggregation 2 (Out-Degree)	14.655	15.364	30.019	13%	44%	32,2%
Semantic Aggregation 3 (In-Degree)	16.879	18.333	35.212	0%	33%	20,5%
Semantic Aggregation 4 (Degree)	16.038	17.142	33.180	5%	38%	25,1%
Hierarchical Clustering (pairwise average linking; Detail-level 2)	-	-	-	-	-	-
Hierarchical Clustering (pairwise average linking; fixe Clusteranzahl: 28)	-	-	-	-	-	-
K-Means Clustering (2 Iterationen; fixe Clusteranzahl: 28)	28	100	128	100%	100%	99,7%

Aggregation By Semantics

Aggregation By Clustering (Ausgangslage: Semantic Aggregation 2)

Tabelle 6.4: Testfall II: Reduktion der Elementanzahl

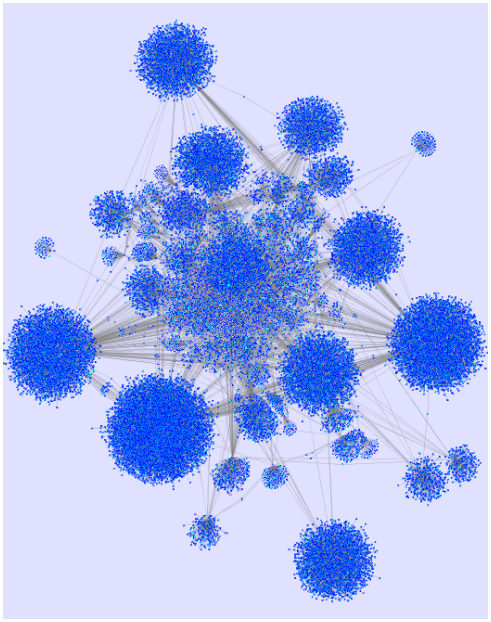
Testfall II	**Elemente gesamt** ursprünglich	**Force-Directed Layout** Berechnungszeit ursprünglich	**Organic Layout** Berechnungszeit
Berechnungszeiten (Sekunden)	44.285	843	36

	Elemente gesamt nach Aggregation	**Aggregation** Zeit	**Force-Directed Layout** Zeit	**Organic Layout** Zeit	**Zeitersparnis Force-Directed Layout**	**Zeitersparnis Organic Layout**
Semantic Aggregation 1 (Edge-Type: "*birthplace*")	32.919	78	760	34	10%	6%
Semantic Aggregation 2 (Out-Degree)	30.019	157	665	34	21%	6%
Semantic Aggregation 3 (In-Degree)	35.212	79	810	35	4%	3%
Semantic Aggregation 4 (Degree)	33.180	106	800	34	5%	6%
Hierarchical Clustering (pairwise average linking; Detail-level 2)	-	-	-	-	-	-
Hierarchical Clustering (pairwise average linking; fixe Clusteranzahl: 28)	-	-	-	-	-	-
K-Means Clustering (2 Iterationen; fixe Clusteranzahl: 28)	128	24	0	0	100%	100%

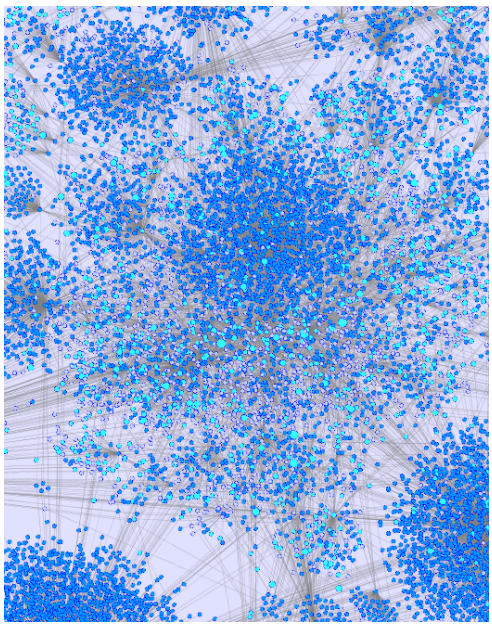
Aggregation By Semantics

Aggregation By Clustering (Ausgangslage: Semantic Aggregation 2)

Tabelle 6.5: Testfall II: Berechnungszeiten (Zeitangaben in Sekunden)



(a) Semantische Aggregierung



(b) Zoom auf Zentrum

Abbildung 6.11: Testfall III: Aggregation By Semantics (a) Vereinfachter Graph nach semantischer Aggregierung (Force-Directed Layout, Default Visual Style) (b) Zoom auf auf das Zentrum des Graphen. Dort findet sich die höchste Konzentration an Meta-Nodes.

gregierungszeit jeweils eine Reduktion der Gesamtelementanzahl um ca. 29% gegenüber der Gesamtelementanzahl des ursprünglichen Graphen (von ungefähr 93.000 auf ca. 66.000 Nodes und Edges). Die unveränderte Node-Anzahl vor und nach der Aggregierung bei Verfahren 3 (In-Degree) resultiert aus dem Umstand, dass hauptsächlich Meta-Nodes mit nur einem darin enthaltenen Knoten erzeugt wurden (siehe dazu die Anmerkung in Abschnitt 6.4.1). Verfahren 3 (In-Degree), welches lediglich eine Elementreduktion von ca. 15% erzielte, benötigte zur Berechnung weniger als die Hälfte der Zeit von Verfahren 1 (in Zahlen: 360 Sekunden zu 840 Sekunden). Die ist darauf zurückzuführen, dass in Summe weniger Meta-Nodes bzw. Meta-Edges erzeugt und Elemente darin zusammengefasst werden mussten.

Die Berechnungszeit des Circular Layouts verkürzte sich aufgrund der Elementreduktion im Vergleich zur Ausgangssituation um etwa 15%. Der Force-Directed Layout Algorithmus lieferte hingegen erst nach Durchführung der semantischen Aggregierung (Verfahren 1, 2 und 3) ein Ergebnis (siehe Tabelle 6.7). Aufgrund der hohen Anzahl an Nodes, Edges bzw. Attributen des Originalgraphen kam es im Zuge des Layouts zu einem Stack Overflow. Abbildung 6.11 zeigt den Graphen nach Durchführung des semantischen Aggregierungs-Verfahrens 1 im Force-Directed Layout. Die höchste Konzentration an Meta-Nodes findet sich dabei im Zentrum des aggregierten Graphen.

Aufgrund des limitierten Speichers bzw. der Konfiguration der Testumgebung konnten die beiden *Aggregation By Hierarchical Clustering* Varianten nicht vollständig durchgeführt werden. Das *Aggregation By K-Means Clustering* Verfahren (2 Iterationen, 180 zu identifizierende Cluster) erzielte, aufbauend auf den mittels semantischer Aggregierung modifizierten Graphen, eine Reduktion der Elementanzahl um 98%. Abbildung 6.12 zeigt den deutlich vereinfachten Graphen aus 6.11. Der auf die hohe Komplexität bzw. die hohe Node- und Edge-Anzahl des Originalgraphen zurückzuführenden merklichen Verzögerung bei der Interaktion mit der Darstellung konnte auf diese Weise entscheidend entgegengewirkt werden. Abbildung 6.13 zeigt einen vergrößerten Ausschnitt eines Teilbereichs dieses Graphen. Aufgrund des dynamischen Aggregierungs-Features wurde der Meta-Node innerhalb des Viewing-Rectangles aufgefächert und die darin enthaltenen Elemente dargestellt.

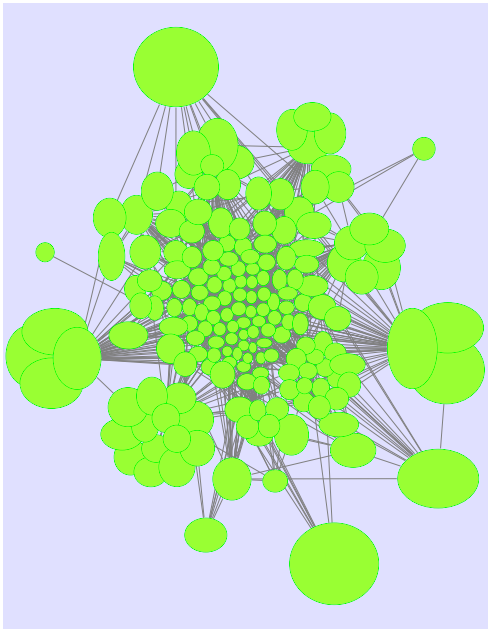


Abbildung 6.12: Testfall III: Graph aus Abbildung 6.11 (Bild **a**) nach Durchführung der *Aggregation By K-Means Clustering* Methode (2 Iterationen, 180 fixe Cluster)

Eine Übersicht über die gesammelten Ergebnisse zu Testfall III findet sich in den Tabellen 6.6 und 6.7.

6.4.5 Weitere Testfälle

Tabelle 6.8 zeigt einen Überblick über die Ergebnisse weiterer Testfälle. Der Fokus liegt dabei auf der Reduktion der Gesamtelementanzahl der jeweils effektivsten *Aggregation By Semantics* Methode sowie auf den Berechnungszeiten für Aggregierung und das Force-Directed Layout.

6.5 Fazit

Die einzelnen Testfälle zeigten, dass mithilfe der *Aggregation By Semantics* und *Aggregation By Clustering* Methoden die visuelle Komplexität eines Graphen entscheidend reduziert werden kann. Die besten Ergebnisse lassen sich dann erzielen, wenn zuerst aufgrund semantischer Informationen aggregiert, im Anschluss ein Layout unter Berücksichtigung des Edge-Weights berechnet, und abschließend das K-Means Verfahren zur Aggregierung der verbleibenden Nodes/Edges durchgeführt wird. Das Resultat ist ein Graph, welcher aus einigen wenigen Meta-Nodes und Meta-Edges besteht. Das dynamische Aggregierungs-Feature ermöglicht bei Bedarf die Darstellung von Details (= Einblendung aggregierter Nodes und Edges).

6.5.1 Beobachtungen

Die Ergebnisse der untersuchten semantischen Verfahren zur Aggregierung variieren je nach getestetem Datensatz. Verfahren 1 (Edge-Type) bzw. Verfahren 2 (Out-Degree) konnten in jenen Fällen, in denen eine Kombination mehrerer Einzeldatensätze vorlag, die Elementanzahl im Schnitt um ca. 41% reduzieren (Spitzenwert 55%, niedrigster Wert 29%). Wurde jedoch kein kombinierter sondern nur ein einzelner Datensatz getestet (Testfälle IV und V), lieferten Verfahren 3 bzw. Verfahren 4 die besseren Ergebnisse (im Schnitt jedoch lediglich eine 22%ige Gesamtreduktion). Der Vorteil der Elementreduktion offenbarte sich vor allem in der anschließenden Berechnung der Layouts, im Speziellen des Force-Directed

Testfall III

	# Nodes ursprünglich	# Edges ursprünglich	gesamt
Elementanzahl	37.521	55.346	92.867

	# Nodes	# Edges	gesamt	Reduktion Nodes	Reduktion Edges	Reduktion gesamt
Semantic Aggregation 1 (Edge-Type: "*subject*")	31.573	34.535	66.108	16%	38%	28,8%
Semantic Aggregation 2 (Out-Degree)	31.695	34.522	66.217	16%	38%	28,7%
Semantic Aggregation 3 (In-Degree)	37.520	40.808	78.328	0%	26%	15,7%
Semantic Aggregation 4 (Degree)	34.729	37.062	71.791	7%	33%	22,7%
Hierarchical Clustering (pairwise average linking; Detail-level 2)	-	-	-	-	-	-
Hierarchical Clustering (pairwise average linking; fixe Clusteranzahl: 180)	-	-	-	-	-	-
K-Means Clustering (2 Iterationen; fixe Clusteranzahl: 180)	180	1.487	1667	100%	97%	98,2%

Aggregation By Semantics

Aggregation By Clustering (Ausgangslage: Semantic Aggregation 1)

Tabelle 6.6: Testfall III: Reduktion der Elementanzahl

Testfall III

Berechnungszeiten (Sekunden)

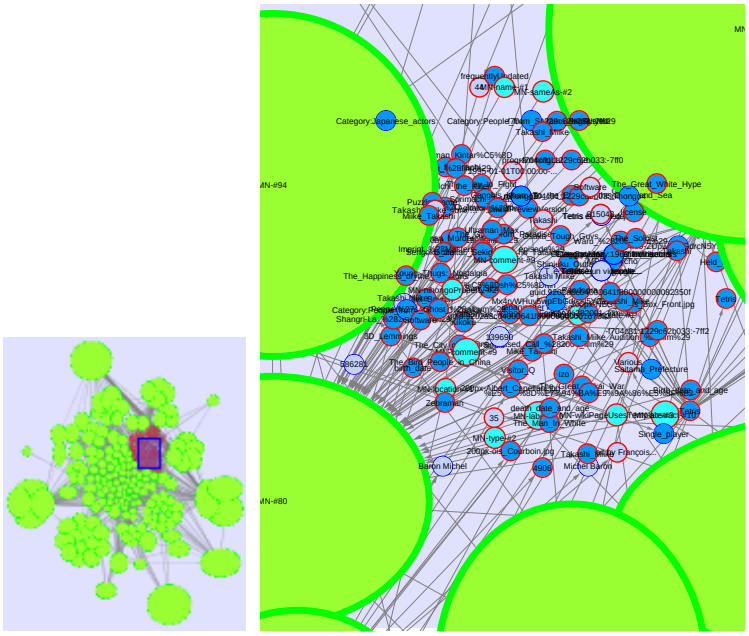
Elemente gesamt ursprünglich	Force-Directed Layout Berechnungszeit ursprünglich	Circular Layout Berechnungszeit
92.867	-	54

	Elemente gesamt nach Aggregation	Aggregation Zeit	Force-Directed Layout Zeit	Circular Layout Zeit	Zeitersparnis Force-Directed Layout	Zeitersparnis Circular Layout
Semantic Aggregation 1 (Edge-Type: "*subject*")	66.108	840	2.940	46	100%	15%
Semantic Aggregation 2 (Out-Degree)	66.217	815	2.990	46	100%	15%
Semantic Aggregation 3 (In-Degree)	78.328	360	-	53	-	2%
Semantic Aggregation 4 (Degree)	71.791	516	3.540	50	100%	7%
Hierarchical Clustering (pairwise average linking; Detail-level 2)	-	-	-	-	-	-
Hierarchical Clustering (pairwise average linking; fixe Clusteranzahl: 180)	-	-	-	-	-	-
K-Means Clustering (2 Iterationen; fixe Clusteranzahl: 180)	1.667	457	2	1	100%	98%

Aggregation By Semantics

Aggregation By Clustering (Ausgangslage: Semantic Aggregation 1)

Tabelle 6.7: Testfall III: Berechnungszeiten (Zeitangaben in Sekunden)



(a) Overview **(b)** Zoom

Abbildung 6.13: Testfall III: Dynamische Aggregierung (a) Außerhalb des Viewing-Rectangles bleiben Meta-Nodes zugeklappt. (b) Nach dem Expand der Meta-Node werden Details des Graphen sichtbar.

Testfälle IV - VII

Elementanzahl / Berechnungszeiten (Angaben in Sekunden)

Testfall	Nodes+Edges ursprünglich	Force-Directed Layout Zeit ursprünglich	Aggregierungs-Verfahren	Aggregierung Zeit	# Nodes neu	# Edges neu	Nodes+ Edges neu	Force-Directed Layout Zeit neu	Element-Reduktion gesamt	Zeitersparnis Force-Directed Layout
IV	29.219	355	**Semantic Aggregation 4** (Degree)	26	10.306	10.305	20.611	344	29,5%	3%
V	21.651	280	**Semantic Aggregation 4** (Degree)	17	9.071	9.070	18.141	260	16,2%	7%
VI	1.951	5	**Semantic Aggregation 2** (Out-Degree)	1	382	492	874	2	55,2%	60%
VII	11.309	75	**Semantic Aggregation 2** (Out-Degree)	16	3.421	3.822	7.243	45	36,0%	40%

Tabelle 6.8: Weitere Testfälle im Überblick: Reduktion der Gesamtelementanzahl der jeweils effektivsten *Aggregation By Semantics* Methode sowie Berechnungszeiten (Zeitangaben in Sekunden)

Layouts. Die Berechnungszeit wurde aufgrund der geringeren Anzahl an Nodes und Edges merklich verkürzt. Das Force-Directed Layout des Graphen aus Testfall III konnte überhaupt erst nach semantischer Aggregierung ermittelt werden.

Die Zeit zur Berechnung der semantischen Aggregierung spiegelt prinzipiell den Reduktionsumfang wider (d.h. je größer die Reduktion, desto länger die Aggregierungszeit da u.a. mehr Meta-Nodes bzw. Meta-Edges erstellt werden müssen). Dies wird vor allem bei Testfall III deutlich.

Die abschließende Aggregierung mittels Clustering-Techniken resultierte schließlich in einem stark vereinfachten Graphen, sodass beispielsweise eine weitestgehend verzögerungsfreie Interaktion mit der Darstellung ermöglicht wurde. Dabei zeigte sich, dass die beiden Hierarchical Clustering Methoden nur bei kleineren Graphen anwendbar sind. Eine höhere Anzahl an Nodes und Edges hatte entweder eine im Vergleich zum K-Means Verfahren lange Berechnungszeit oder im schlimmsten Fall einen Speicherüberlauf zur Folge. Der K-Means Algorithmus lieferte hingegen in allen Fällen in akzeptabler Zeit das gewünschte Ergebnis.

6.5.2 Verbesserungspotenzial

Im Zuge der Evaluierung offenbarten sich einige Schwächen bzw. Probleme des VIOSEG Tools. Diese sollten im Falle einer Weiterentwicklung bzw. Verbesserung der Software beseitigt werden. Einen Ausblick auf mögliche zukünftige Entwicklungen bietet auch das nächste Kapitel. An dieser Stelle sollen einige der erkannten Probleme kurz geschildert werden.

So zeigte die Evaluierung u.a, dass *Aggregation By Semantics* Methoden bei Graphen mit nur einem zentralen High-Degree Node (d.h. bei Einzeldatensätzen) hinsichtlich der Gesamtelementreduktion schlechtere Ergebnisse als bei kombinierten Datensätzen erzielen. Dies wird u.a. auch bei den Testfällen II und III deutlich: Die höchste Konzentration an Meta-Nodes findet sich dabei zwischen den kleineren dichten Regionen im Zentrum des aggregierten Graphen (siehe Abbildungen 6.9 bzw. 6.11). Die großen dichten Regionen an den Rändern, welche umfangreichere Einzeldatensätze repräsentieren, weisen nur wenige Meta-Nodes auf. Die implementierten Algorithmen sollten in neueren Versionen von VIOSEG dahingehend überarbeitet bzw. erweitert werden.

Testfall III, bestehend aus annähernd 100.000 Nodes und Edges mit jeweils 20 Node- bzw. 11 Edge-Attributen, offenbarte die noch nicht ausreichend effiziente Implementierung des dynamischen Aggregierungs-Features. Bei einer größeren Elementanzahl kam es aufgrund dessen mitunter zu Verzoegerungen beim Expand bzw. Collapse von Meta-Nodes. In diesem Fall empfiehlt es sich, die Sensitivität des Features auf einen hohen Zoomfaktor einzustellen bzw. ausgewählte Meta-Nodes manuell aufzufächern bzw. zuzuklappen. Eine Überarbeitung des Features sollte in nachfolgenden VIOSEG Versionen eine bessere Performance liefern.

Eine Auflistung weiterer Verbesserungs- und Erweiterungsmaßnahmen sowie die Diskussion moeglicher zukünftiger Entwicklungen findet sich im nächsten Kapitel.

Kapitel 7

Fazit und Ausblick

Das Schlusskapitel fasst die grundlegenden Inhalte sowie die erzielten Ergebnisse und gewonnenen Erkenntnisse dieser Masterarbeit nochmals zusammen. Zudem erfolgt die Skizzierung der prinzipiellen Vorgehensweise bei der Umsetzung der Ziele. Das Kapitel schließt mit einer Diskussion hinsichtlich erwägenswerter Weiterentwicklungen und Verbesserungen der implementierten Software bzw. mit einem Ausblick auf mögliche zukünftige Entwicklungen auf diesem Gebiet.

7.1 Vorgehensweise

Im Rahmen dieser Masterarbeit wurden verschiedene, im Kontext der Visualisierung von semantischen Graphstrukturen besonders relevante Konzepte, Techniken und Mechanismen vorgestellt. Zu diesem Zweck stand, ausgehend von der im ersten Kapitel definierten Problemstellung, am Beginn der Arbeit eine umfassende Literaturrecherche. Besonderes Augenmerk galt dabei den Bereichen Graph- bzw. Information-Visualisation, Semantic Web, Clustering, Aggregation, Graph Theory sowie Graph Drawing. Die Ergebnisse der Recherche finden sich in exzerpierter Form im Theorieteil dieses Dokuments.

Im nächsten Schritt folgte die Suche nach relevanten, bereits existierenden Softwarelösungen zur Visualisierung von Graphen. Auf Basis zuvor festgelegter Kriterien wurden relevante Softwarepakete ausgewählt und anschließend einer genaueren Untersuchung unterzogen. Im Zuge dieser Evaluierung wurden sowohl allgemeine als auch spezifische Features, sowie einzelne Vor- und Nachteile näher betrachtet. Im Hinblick auf die anschließende Implementierung erfolgte zudem eine Einschätzung des Erweiterungs- bzw. Modifikations-Potenzials jedes einzelnen Visualisierungssystems.

Auf Basis der Evaluierung wurden die Requirements für die zu implementierende Visualisierungssoftware definiert. Die Entwicklung erfolgte im Rahmen des praktischen Teils dieser Masterarbeit. Abschließend wurden die verschiedenen Features der Visualisierungssoftware mithilfe von Beispieldaten getestet und bewertet.

7.2 Erkenntnisse

Die Literaturrecherche zeigte, dass eine Vielzahl von Abhandlungen, wissenschaftlichen Artikeln, sowie Diplom- und Doktorarbeiten zu den im Rahmen dieser Masterarbeit behandelten Thematiken existiert. Sowohl aus dem Bereich der Graph-Visualisierung im Allgemeinen als auch aus dem Bereich der Visualisierung von semantischen bzw. RDF-Graphen im Speziellen findet sich ein umfangreiches Repertoire an aktuellen Publikationen.

Eine mit der Recherche einhergehende Suche nach bereits vorhandenen Softwarelösungen offenbarte eine große Anzahl an unterschiedlichen Systemen bzw. Tools zur Visualisierung von Graphen und

Netzwerken. Viele dieser Softwarepakete sind Open Source bzw. für nicht kommerzielle Nutzung frei verfügbar. Relevante Unterschiede ergeben sich u.a. hinsichtlich Funktionsumfang, Verwendungszweck, Lizenzierung und Aktualität. Aufgrund der großen Menge an verschiedenen Softwarelösungen ist es schwierig, den Überblick zu bewahren bzw. jene Tools herauszufiltern, welche hinsichtlich der eingangs erörterten Problemstellung am geeignetsten erscheinen.

Die im Rahmen der Evaluierung näher betrachteten Visualisierungspakete können prinzipiell zwei Gruppen zugeordnet werden: Die erste Gruppe umfasst Tools und Frameworks zur Visualisierung bzw. Analyse von generellen Graphen und Netzwerken. Der Fokus liegt dabei vor allem auf allgemeinen Aspekten des Graph Drawings, wie etwa Layout-Algorithmen oder graphtheoretischen Analysen. Diese Pakete sind auch zur Darstellung größerer Graphen und komplexerer Netzwerke geeignet. Der Nachteil ist die fehlende Unterstützung semantischer Features.

Zur zweiten Gruppe zählen spezielle Tools zur Verarbeitung von RDF-Graphen. Diese Pakete bieten besondere Features zur Darstellung, Navigation und Analyse von semantischen Graphen bzw. RDF-Modellen. Ein Schwachpunkt ist jedoch deren schlechte Skalierbarkeit (z.B. unübersichtliches Layout, schlechte Performance, etc.). Zudem werden kaum Funktionalitäten zur Reduktion der visuellen Komplexität, wie etwa Clustering oder Aggregation bereitgestellt.

Im Zuge der Evaluierung stellte sich *Cytoscape* als das umfangreichste und im Hinblick auf die Problemstellung am besten geeignete Softwarepaket heraus. Die auf Basis von Cytoscape im Rahmen dieser Masterarbeit entwickelte Visualisierungssoftware *VIOSEG* vereint somit Funktionalitäten des Graph Drawings der ersten Gruppe mit Semantik-Features der zweiten Gruppe. Zudem werden Mechanismen zur Komplexitätsreduktion bzw. zur dynamischen, zoomabhängigen Aggregierung bereitgestellt.

Die abschließende Evaluierung der VIOSEG Software zeigte, dass mithilfe der verschiedenen implementierten Aggregations- bzw. Clustering-Methoden die visuelle Komplexität eines Graphen entscheidend reduziert werden kann. Die dynamische, zoomabhängige Aggregierung ermöglicht zudem das automatische Ein- bzw. Ausblenden von Details. Dies bedeutet, dass in speziellen *Meta-Nodes* zusammengefasste Knoten und Kanten ab einer gewissen Zoomstufe sichtbar werden. Auf diese Weise lassen sich selbst größere und komplexere semantische Graphstrukturen visualisieren und navigieren. Zudem wird eine weitestgehend verzögerungsfreie Interaktion mit der Darstellung ermöglicht.

7.3 Weiterführende Arbeiten

Die im Zuge dieser Masterarbeit entwickelte Visualisierungssoftware VIOSEG unterstützt eine Reihe von Funktionalitäten um die im Einführungskapitel erörterte Problemstellung zu adressieren. Die als Resultat der Evaluierung bereits verfügbarer Visualisierungspakete definierten Requirements bzw. Main-Features wurden im Zuge der Implementierung weitestgehend realisiert. Darüber hinaus finden sich jedoch noch weitere Funktionalitäten und Features, welche im Rahmen dieser Arbeit nicht berücksichtigt werden konnten. Die Evaluierung der Software in Kapitel 6 offenbarte zudem einige Schwächen bzw. Probleme. Nachfolgend sollen die wichtigsten offenen Fragestellungen diskutiert sowie Verbesserungs- und Erweiterungsmöglichkeiten der Software vorgestellt werden.

Beim Test des VIOSEG Tools mit verschiedenen Beispieldaten stellte sich heraus, dass die Implementierung des dynamischen Aggregierungs-Features noch einiges Verbesserungspotenzial in sich birgt. Bei einer größeren Elementanzahl kann es mitunter zu Verzögerungen beim Expand bzw. Collapse von Meta-Nodes kommen. Eine Überarbeitung des Intersection-Test Algorithmus, welcher für jede Meta-Node über Expand oder Collapse entscheidet, sollte in nachfolgenden Versionen eine bessere Performance liefern.

Weiters zeigte die Evaluierung, dass Methoden zur semantisch basierten Aggregierung bei Graphen mit nur einem zentralen High-Degree Node (d.h. bei Einzeldatensätzen) hinsichtlich der Gesamtelementreduktion schlechtere Ergebnisse als bei kombinierten Datensätzen erzielen. Dahingehend wäre eine Modifikation bzw. eine Erweiterung der implementierten Algorithmen wünschenswert.

Aufgrund des limitierten Zeitrahmens konnten die in Kapitel 5 definierten optionalen Requirements in VIOSEG Version 1.0 nicht umgesetzt werden. Dazu zählt u.a. die Implementierung einer View Coordination Funktion. Diese sollte sowohl die Koordination der einzelnen Cytoscape Graph-Darstellungen als auch eine Koordination mit externen Visualisierungen hinsichtlich User-Navigation (Zoomstufe, aktuell dargestellter Ausschnitt, etc.) und Zustand der dargestellten Objekte (Farbe, Selektion, etc.) ermöglichen.

Zur Behandlung von Graphen, deren Struktur sich im Zeitablauf häufig ändert, wäre zudem die Implementierung von *Dynamic Graph Drawing* Algorithmen empfehlenswert. Diese stellen spezielle Interaktions- und Layout-Techniken zur Verfügung. Auf diese Weise könnte beispielsweise eine effektive Navigation großer semantischer Graphen und Netzwerke realisiert werden.

VIOSEG selbst bietet nur begrenzt Funktionen zur Analyse von semantischen Aspekten des dargestellten Graphen bzw. des RDF-Modells. Eine sinnvolle Erweiterung wäre somit etwa ein Query-Feature, welches die Abfrage von RDF-Daten mittels *SPARQL (SPARQL Protocol and RDF Query Language)* ermöglicht.

Darüber hinaus könnten verschiedene weitere Features und Algorithmen zur effizienten Visualisierung und Navigation von großen Graphen mit mehr als einer Million Elementen implementiert werden.

Um weitere Hinweise und Vorschläge für Verbesserungen zu erhalten wäre die Durchführung einer Usability Evaluation denkbar. Mittels Heuristic Evaluation und/oder formalen Experimenten unter Einbindung mehrerer Test-User könnten Usability-Probleme identifiziert bzw. die Nützlichkeit des Ansatzes besser beurteilt werden.

7.4 Ausblick

Computerunterstützte Visualisierung bzw. speziell die Visualisierung von semantischen Graphstrukturen wird im Hinblick auf permanent wachsende Datenmengen stetig an Bedeutung gewinnen. Die derzeitige bzw. prognostizierte Entwicklung auf dem Gebiet des *Semantic Web* unterstreicht zudem die Aktualität und besondere Relevanz der einzelnen Schwerpunkte dieser Masterarbeit. Wie die Schilderung der möglichen Erweiterungen und Verbesserungen der VIOSEG Implementierung im vorangegangen Abschnitt zeigt, bietet die Thematik einiges Potential für weitere Projekte und fortführende Arbeiten. Das Design sowie die Dokumentation des Source Code, in Verbindung mit den in dieser Arbeit erläuterten technischen Details, sollte zukünftige Weiterentwicklungen der Software erleichtern.

Anhang A

User Guide

VIOSEG (**VI**sualisation **O**f **SE**mantic **G**raphs) ist eine Open Source Software zur interaktiven Visualisierung von semantischen Graphen im RDF-Format. Nodes und Edges können mithilfe verschiedener Aggregierungs- bzw. Clustering-Verfahren zu Meta-Nodes und Meta-Edges zusammengefasst und auf diese Weise die visuelle Komplexität eines Graphen entscheidend reduziert werden. Weiters unterstützt die Software Möglichkeiten zur dynamischen Aggregierung, d.h. zuvor erstellte Meta-Nodes werden in Abhängigkeit des aktuellen Bildausschnitts bzw. der Zoomstufe aufgefächert oder wieder zugeklappt. Die Software basiert auf *Cytoscape*, einer Open Source Plattform für die Analyse und Visualisierung von komplexen Netzwerken und Graphen.

Dieser User Guide erläutert essentielle Features sowie die grundlegende Bedienung des VIOSEG Tools. Weiters finden sich Informationen hinsichtlich Requirements, Installation und Start der Visualisierungssoftware.

A.1 Start

VIOSEG Version 1.0 wurde vollständig in Java 6 implementiert und ist somit plattformunabhängig. Zum korrekten Betrieb wird Cytoscape in der Version 2.6.2 (oder 2.6.1) sowie Jena (Version 2.5.4) bzw. Pellet (Version 2.0.0) benötigt.

Grundsätzlich gibt es zwei Möglichkeiten zum Start der Visualisierungssoftware: Einerseits kann das Programm direkt als Cytoscape-Plugin verwendet werden. Dazu wird das bereits kompilierte VIOSEG jar-File in den Plugin-Folder der Cytoscape-Installation kopiert und Cytoscape gestartet. Anschließend kann VIOSEG über das 'Plugins'-Menü von Cytoscape ausgewählt und gestartet werden. Die zweite Möglichkeit besteht darin, VIOSEG als eigenständige Applikation zu verwenden. Dazu wird der VIOSEG-Source in das Root-Directory der Cytoscape Distribution kopiert, kompiliert und mithilfe der Klasse `VisualizerMain` gestartet. Zur Handhabung größerer Graphen empfiehlt sich eine dezidierte Speicherzuweisung von 512MB bzw. eine Stack Size von mindestens 5MB (Startparameter: `-Dswing.aatext=true -Xss5M -Xmx512M`)

A.2 Features

Die wichtigsten unterstützten Features und implementierten Funktionalitäten der VIOSEG Software im Überblick:

- **RDF-Import**: Einlesen von Files im RDF-Format
- **Visualisierung**: Darstellung von RDF-Modellen als Graph; grundlegende Visualisierungs-Features und Mechanismen

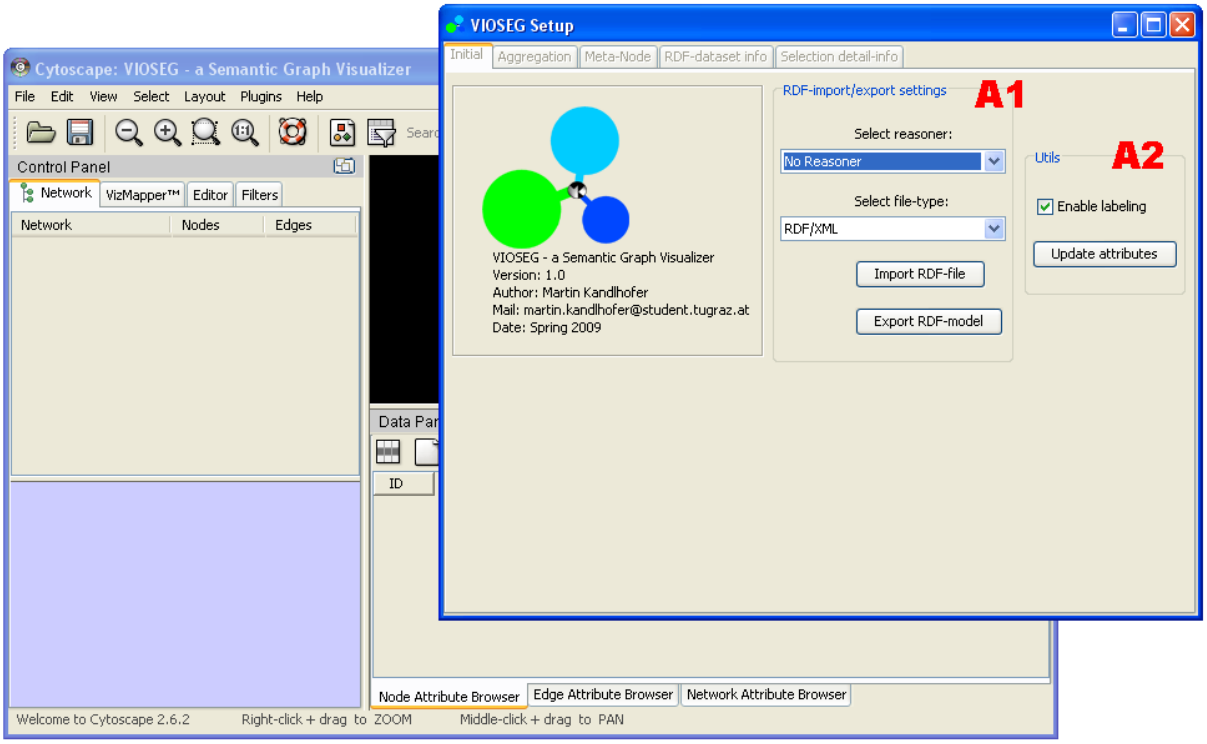


Abbildung A.1: VIOSEG Initial-Tab mit Cytoscape GUI im Hintergrund

- **Interaktivität**: Features zur Interaktion mit der Darstellung (Navigation, Zoom, Pan, Edit, etc.)
- **Reduktion der visuellen Komplexität**: Funktionalitäten zur Identifikation und Erstellung von Clustern/Meta-Nodes; dynamische Aggregierung; Level-Of-Detail Feature; Filter- und Such- Mechanismen
- **Berücksichtigung semantischer Informationen**: Einbindung bzw. Berücksichtigung semantischer Informationen aus dem RDF-Modell bei Aggregierung und visueller Repräsentation; Darstellung näherer Informationen des RDF-Datensatzes
- **Grafische Benutzeroberfläche**: Einfache, intuitive Bedienung der Software

A.3 Bedienung

Der Abschnitt erläutert u.a. anhand von Screenshots die Nutzung der Main-Features bzw. die grundlegende Bedienung der Visualisierungssoftware. Abbildung A.1 zeigt das VIOSEG User-Interface sowie im Hintergrund die GUI von Cytoscape. Die weiteren Registerkarten (Tabs) werden erst nach dem Import einer RDF-Datei aktiviert. Im Anschluss folgt eine Erklärung der einzelnen Registerkarten und deren Aufgaben.

A.3.1 Initial

Mittels Combo-Boxen können auf der *Initial*-Registerkarte (Abbildung A.1, Markierung **A1**) ein Reasoner sowie das gewünschte Dateiformat ausgewählt werden. Optionen sind dabei *No Reasoner, RDFS-Reasoner, OWL-Reasoner, OWL-Jena-Reasoner* bzw. *RDF/XML, N-Triple, N3*. Nach Betätigen des *Import RDF-File* Buttons kann die zu importierende Datei via Open-File-Dialog ausgewählt werden. Ein Progress-Bar zeigt anschließend den Fortschritt des Importvorgangs an. Nach dem erfolgreichen Import

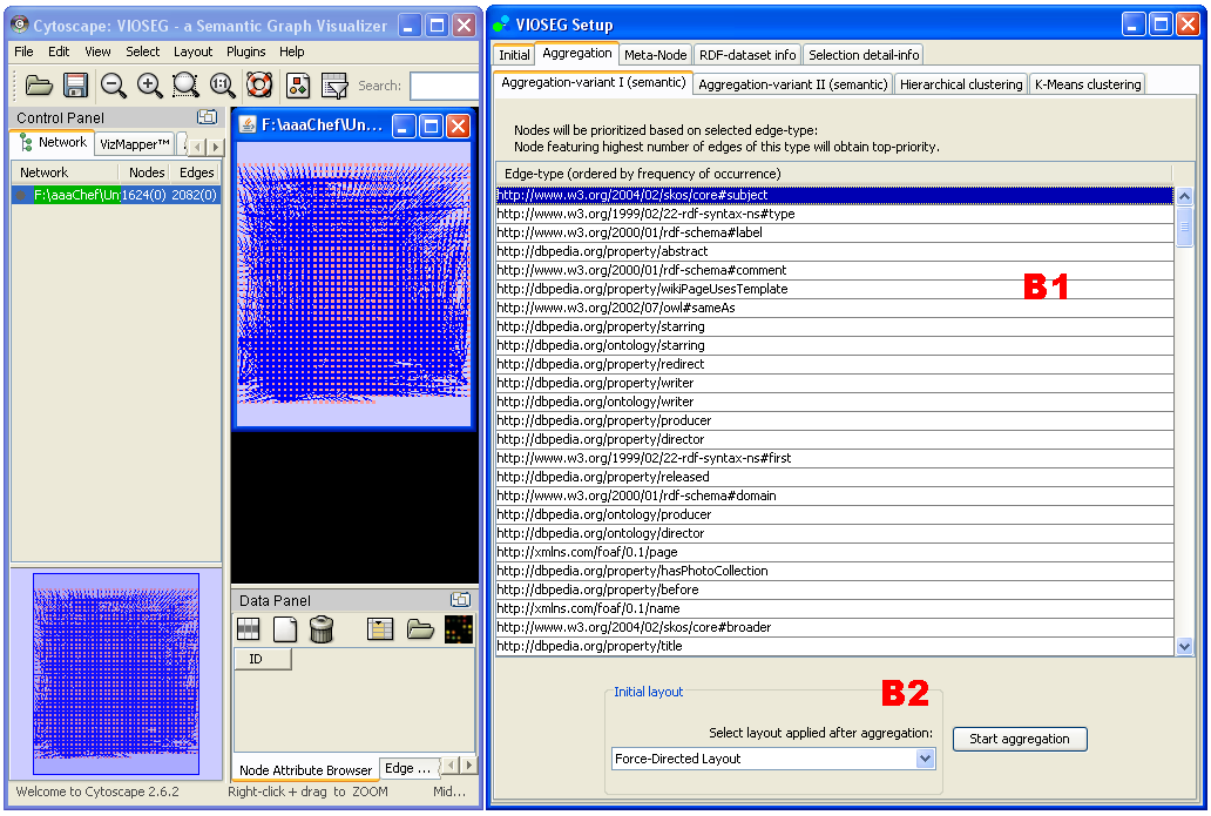


Abbildung A.2: Rechts: VIOSEG Aggregation-Tab, erster Sub-Tab (Aggregierung auf Basis semantischer Informationen); Links: Cytoscape GUI mit Graph-Darstellung

werden die weiteren Tabs der VIOSEG GUI aktiviert. Weiters kann das geladene RDF-Modell im aktuell ausgewählten Dateiformat auch exportiert werden.

Die initiale Registerkarte bietet via *Update attributes* Button auch die Möglichkeit, eine Aktualisierung spezieller Node- und Edge-Attribute (z.B. Edge-Length, Node-Size, Degree, etc.) zu erzwingen. Zudem können mithilfe der Check-Box Labels von Nodes und Edges ein- oder ausgeblendet werden (siehe Markierung **A2** der Abbildung A.1).

A.3.2 Aggregation

Abbildung A.2 zeigt die Cytoscape GUI mit dem visualisierten Graphen (Grid-Layout) sowie die *Aggregation*-Registerkarte nach erfolgreichem Import einer RDF-Datei. Diese Registerkarte gliedert sich in vier Sub-Tabs. Der erste Sub-Tab (*Aggregation-variant I*) beinhaltet die Konfigurationsmöglichkeiten für das semantische Aggregierungs-Verfahren 1. Die Tabelle (Abbildung A.2, Markierung **B1**) enthält alle Edge-Types des RDF-Modells, absteigend sortiert nach den häufigst auftretenden Kantentypen. Der gewählte Edge-Type dient in Folge als Node-Priority Criterion. D.h. jene Nodes, welche die größte Zahl an ausgehenden Kanten dieses Typs aufweisen, erhalten eine hohe Priorität während des Aggregierungsprozesses. Zusätzlich bietet der Sub-Tab die Möglichkeit ein Layout, welches nach Abschluss der Aggregierung auf den Graphen angewandt wird, auszuwählen (Markierung **B2**).

Abbildung A.3 zeigt den nächsten Sub-Tab (*Aggregation-variant II*), welcher der Konfiguration bzw. dem Start des zweiten semantischen Aggregierungs-Verfahrens dient. Als Node-Priority Criterion wird in diesem Fall der Node-Degree herangezogen. Mittels Combo-Box kann zwischen *outgoing edges* (Out-Degree), *incoming edges* (In-Degree) und *outgoing and incoming edges* (Degree) gewählt werden.

Der nächste Sub-Tab (*Hierarchical clustering*; Abbildung A.4) ermöglicht die Konfiguration bzw.

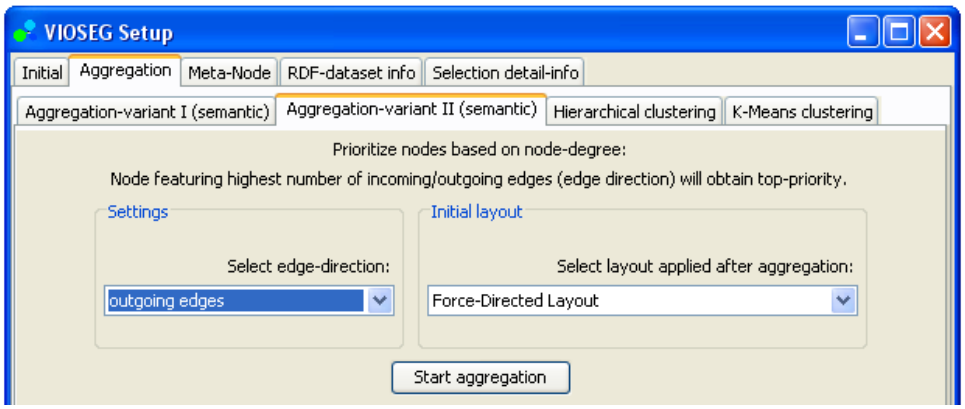


Abbildung A.3: VIOSEG Aggregation-Tab, zweiter Sub-Tab (Aggregierung auf Basis semantischer Informationen)

den Start der beiden auf hierarchischem Clustering basierenden Aggregierungs-Verfahren. Bei Variante 1 kann eine fixe Cluster-Anzahl angegeben werden (Markierung **C1**). Als Folge der Aggregierung wird exakt diese Anzahl an Meta-Nodes erstellt. Variante 2 unterscheidet sich dadurch, dass die genaue Cluster-Anzahl nicht bekannt ist. Diese ist dabei vom gewählten Detailgrad (*level*) abhängig (siehe Markierung **C2**). Dies ist ein Integer-Wert beginnend bei 1. Je höher dieser Wert, desto größere Meta-Nodes (hinsichtlich der Anzahl der darin aggregierten Nodes) werden erzeugt. Ein kleinerer Wert hat hingegen kleinere Cluster zur Folge. Für beide hierarchischen Clustering-Varianten kann zudem noch die zu verwendende Metrik bestimmt werden (Markierung **C3**; Optionen: *pairwise single-linkage, pairwise average-linkage, pairwise maximum-linkage, pairwise centroid-linkage*).

Der *K-Means clustering* Sub-Tab erlaubt die Konfiguration und den Start des Aggregierungsvorgangs auf Basis von K-Means Clustering. Justierbare Parameter sind die Anzahl an Iterationen des K-Means Algorithmus sowie die Cluster- bzw. Meta-Node Anzahl (siehe Abbildung A.5).

Die beiden Graph-Darstellungen in Abbildung A.6 sind das Ergebnis des semantischen Aggregierungsprozesses (Bild **a**) bzw. der anschließenden Durchführung des K-Means Clusterings (Bild **b**).

A.3.3 Meta-Node

Die *Meta-Node* Registerkarte (Abbildung A.7) ermöglicht die Konfiguration des dynamischen Aggregierungs -Features (automatisches Expand/Collapse von Meta-Nodes je nach Zoomstufe und aktuellem Betrachtungsausschnitt). Einerseits kann das Feature gänzlich aktiviert oder deaktiviert werden. Anderseits besteht die Möglichkeit via Slider die Sensitivität zu beeinflussen (Markierung **D1**). Ein niedriger Wert bedeutet, dass erst bei einem größeren Zoomfaktor ein Expand relevanter Meta-Nodes durchgeführt wird. Je höher der Wert bzw. je weiter nach rechts der Regler geschoben wird, desto schneller werden relevante Meta-Nodes aufgefächert (d.h. es genügt bereits ein kleinerer Zoomfaktor um ein Expand der Meta-Nodes auszulösen).

Weiters bietet die Registerkarte noch verschiedene Möglichkeiten zum manuellen Expand- bzw. Collapse (Abbildung A.7, Markierungen **D2, D3, D4**) sowie zum Entfernen von Meta-Nodes (**D5**).

Abbildung A.7 zeigt die Auswirkung der aktuellen Konfiguration des dynamischen Aggregierungs-Features auf die Graph-Visualisierung.

A.3.4 RDF-Dataset Info

Diese Registerkarte stellt nähere Informationen zum aktuell geladenen RDF-Modell in tabellarischer Form dar (Abbildung A.8). Sie gliedert sich in folgende Sub-Tabs:

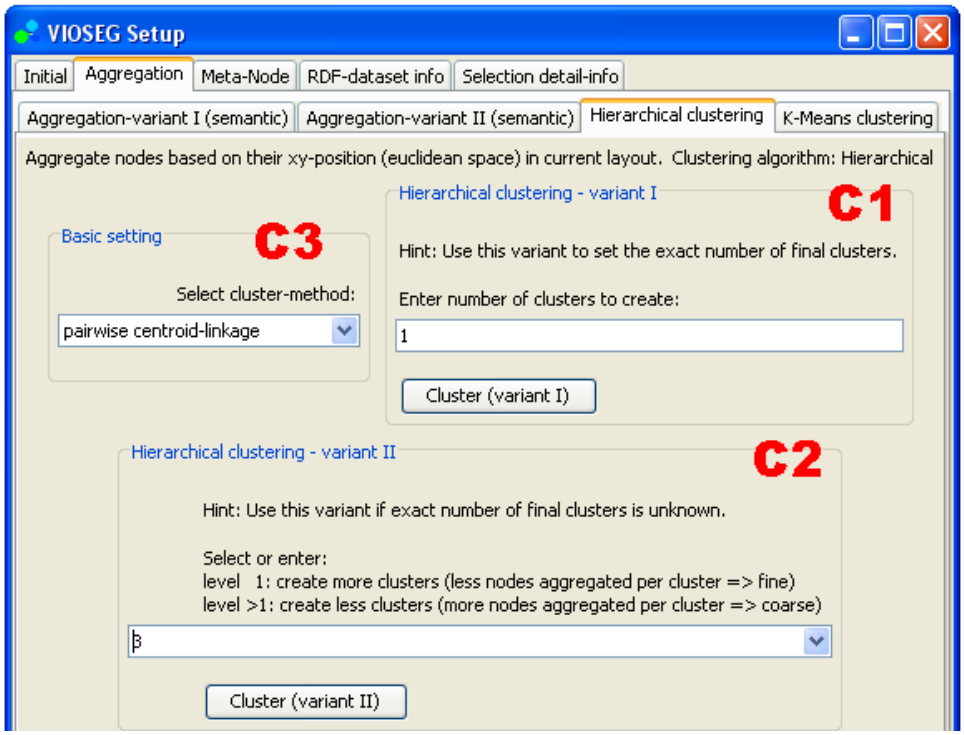


Abbildung A.4: VIOSEG Aggregation-Tab, dritter Sub-Tab (Aggregierung aufgrund Hierarchical Clustering)

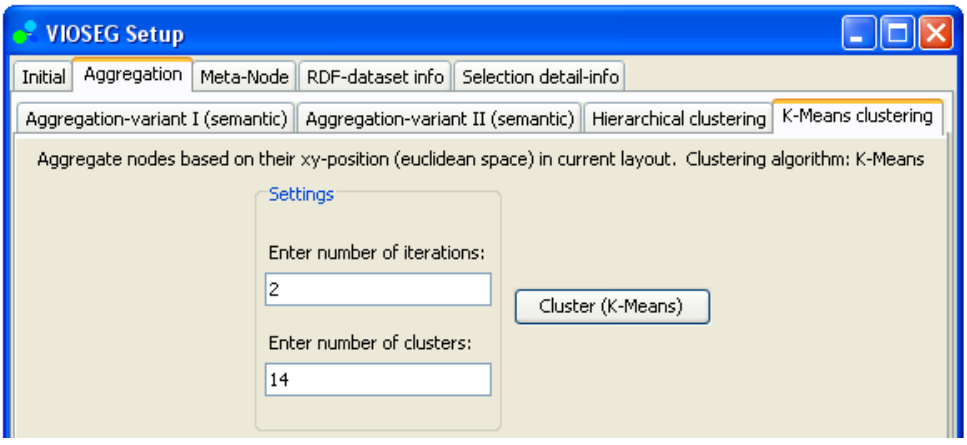
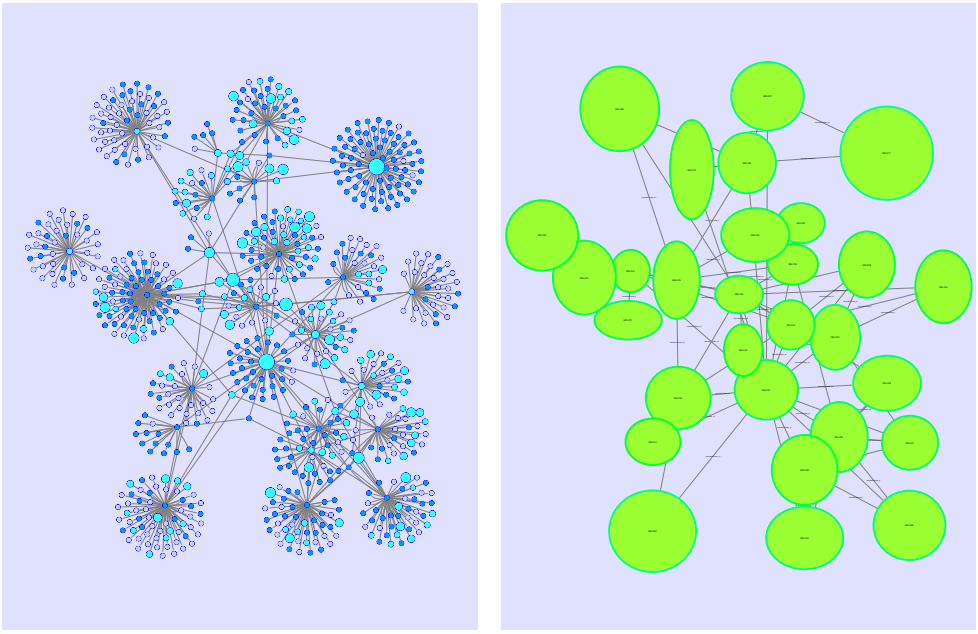


Abbildung A.5: VIOSEG Aggregation-Tab, vierter Sub-Tab (Aggregierung aufgrund K-Means Clustering)



(a) Semantic Aggregation **(b)** K-Means Clustering

Abbildung A.6: Aggregierungen: (a) Graph-Darstellung nach semantischer Aggregierung im Force-Directed Layout (türkise Meta-Nodes). (b) Graph nach Durchführung des K-Means Clusterings (grüne Meta-Nodes)

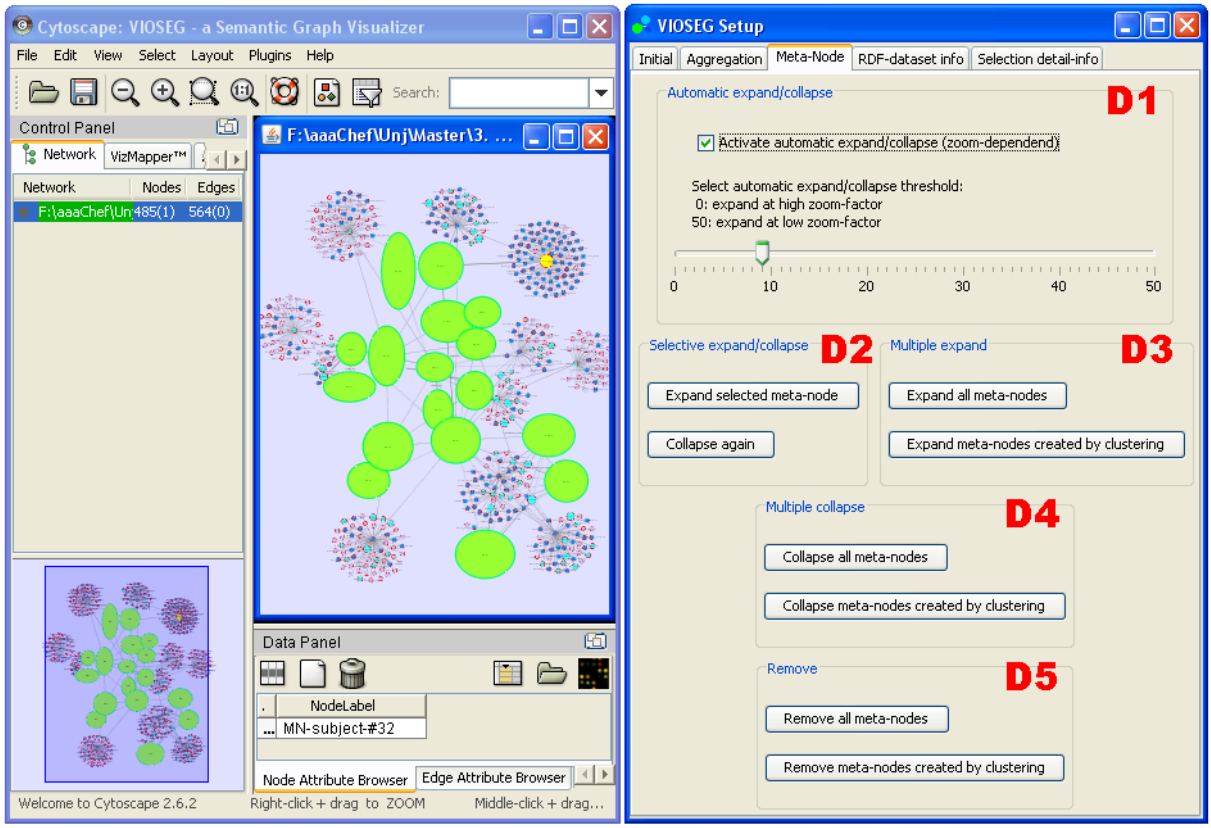
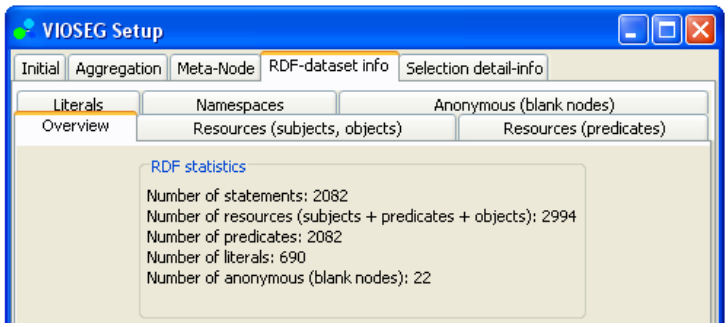
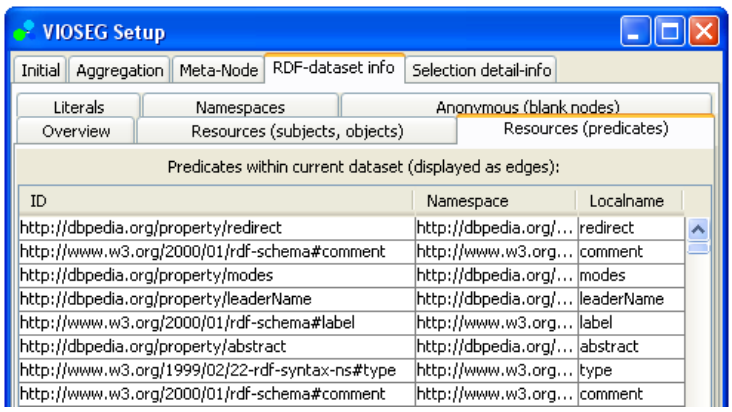


Abbildung A.7: Rechts: VIOSEG Meta-Node-Tab mit verschiedenen Einstellungen; Links: Cytoscape GUI; Aufgrund der Konfiguration des dynamischen Aggregierungs-Features wurden bereits einige der größeren Meta-Nodes aufgefächert.



(a) Sub-Tab Overview



(b) Sub-Tab Predicates

Abbildung A.8: RDF-Dataset Info-Tab: (a) Überblick, Statistik (b) Auflistung aller Predicates des RDF-Modells

- *Overview*: Überblick bzw. statistische Informationen (Anzahl an Statements, Resources,...).
- *Resources (Subjects, Objects)*: Auflistung aller Subjects und Objects des RDF-Modells. Für jedes Subject bzw. Object werden URI-Referenz (ID), Namespace und Localname angegeben.
- *Resources (Predicates)*: Auflistung aller Predicates des RDF-Modells. Für jedes Predicate werden wiederum URI-Referenz (ID), Namespace und Localname angegeben.
- *Literals*: Auflistung aller Literals des Modells, unterteilt in ID und Literal Value.
- *Namespaces*: Auflistung aller im Modell enthaltenen Namespaces.
- *Anonymous (Blank Nodes)*: Auflistung etwaiger Blank Nodes des RDF-Modells.

A.3.5 Selection Detail-Info

Die Registerkarte zeigt Details zum aktuell markierten Meta-Node (Label, ID, Anzahl an aggregierten Nodes). Zusätzlich werden in einer Tabelle nähere Informationen zu den einzelnen enthaltenen Nodes bzw. Meta-Nodes aufgelistet (ID, Label, Node-Type). Abbildung A.9 zeigt einen Screenshot dieses Tabs.

A.4 Tipps zur Verwendung

Nachfolgend einige Hinweise und Tipps zur bestmöglichen Verwendung der VIOSEG Visualisierungs-software.

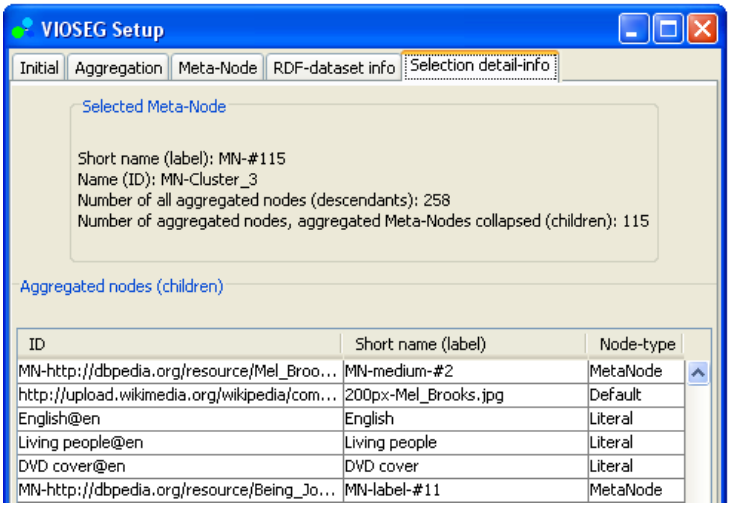


Abbildung A.9: VIOSEG Selection Detail Info-Tab

- Die einzelnen Aggregierungs-Verfahren können beliebig oft und in willkürlicher Reihenfolge angewendet werden. Hinsichtlich einer optimalen Komplexitätsreduktion empfiehlt es sich aber, Elemente zuerst mittels semantischer Verfahren zu aggregieren und die verbleibenden Nodes und Edges mittels K-Means oder Hierarchical Clustering zusammenzufassen.
- Bei dichten Graphen bzw. großen Graphen sollten zur besseren Übersicht bzw. aus Performancegründen die Node- und Edge-Labels deaktiviert werden.
- K-Means Clustering liefert auch bei größeren Graphen in akzeptabler Zeit zufriedenstellende Ergebnisse. Die Verwendung von Hierarchical Clustering sollte hingegen auf kleinere Graphen beschränkt bleiben.
- Um die Performance des dynamischen Aggregierungs-Features bei einer größeren Elementanzahl zu verbessern, sollte die Sensitivität auf einen höheren Zoomfaktor eingestellt werden. Ein weiterer Schritt wäre, das Feature zu deaktivieren und bei Bedarf Meta-Nodes manuell aufzufächern.

Literaturverzeichnis

Abello, J., van Ham, F., and Krishnan, N. (2006). Ask-graphview: A large scale graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12(5):669–676. online: http://www.research.ibm.com/visual/papers/ASK-Graphview3.pdf (12.05.2009). (Cited on pages 5, 28, 48, 49 and 50.)

Aduna, B. (2008). User guide for sesame 2.2. Website openRDF.org. online: http://www.openrdf.org/doc/sesame2/2.3-pr1/users/index.html (09.06.2009). (Cited on page 59.)

Aduna-Software (2008). Autofocus fact sheet. Website Aduna-Software. online: http://www.aduna-software.com/products/autofocus/factsheet.view (20.12.2008). (Cited on pages v and 40.)

Aigner, W., Miksch, S., Mueller, W., Schumann, H., and Tominski, C. (2007). Visualizing time-oriented data - a systematic view. *Computer and Graphics*, Vol. 31(3):401–409. online: http://www.donau-uni.ac.at/imperia/md/content/department/ike/ike_publications/2007/refereedjournalarticles/aigner_2007_cg_visualizing-time-oriented-data.pdf (17.04.2009). (Cited on page 13.)

Andrews, K. (2002). Visualising information structures: Aspects of information visualisation. Professorial Thesis; Graz University of Technology, Institute for Information Systems and Computer Media (IICM),. online: http://www.iicm.tugraz.at/iicm_papers/kandrews_habil.pdf (01.04.2009). (Cited on pages 3, 5, 8, 9, 10, 11, 12, 15, 18, 19, 23, 25, 28, 48, 49 and 52.)

Andrews, K. (2008). Writing a thesis: Guidelines for writing a master's thesis in computer science. online: http://ftp.iicm.tugraz.at/pub/keith/thesis/thesis.zip (02.04.2009). (Cited on page XIII.)

Andrews, K. (2009). Information visualisation: Course notes. online: http://courses.iicm.tugraz.at/ivis/ivis.pdf (18.04.2009). (Cited on page 8.)

Andrews, K., Kienreich, W., Sabol, V., Becker, J., Droschl, G., Kappe, F., Granitzer, M., Auer, P., and Tochtermann, K. (2002). The infosky visual explorer: exploiting hierarchical structure and document similarities. *Information Visualization*, Vol. 1(3/4):166–181. online: http://www.palgrave-journals.com/ivs/journal/v1/n3/abs/9500023a.html (17.04.2009). (Cited on page 15.)

Assenov, Y., Ramirez, F., Schelhorn, S., Lengauer, T., and Albrecht, M. (2009). Networkanalyzer. Website Max-Planck-Institut Informatik. online: http://med.bioinf.mpi-inf.mpg.de/netanalyzer/index.php (16.06.2009). (Cited on page 90.)

Bader, G. (2009a). Mcode. Website Bader Lab. online: http://baderlab.org/Software/MCODE (16.06.2009). (Cited on page 90.)

Bader, G. (2009b). Network analysis links. Cytoscape Wiki. online: http://www.cytoscape.org/cgi-bin/moin.cgi/Network_analysis_links (06.06.2009). (Cited on page 53.)

Bader, G. D. and Hogue, C. W. (2003). An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics 2003*. online: http://www.biomedcentral.com/content/pdf/1471-2105-4-2.pdf (30.12.2008). Note: (c) 2003 Bader and Hogue; licensee BioMed Central Ltd. (Cited on pages 35, 36 and 90.)

Barthelemy, M., Chow, E., and Eliassi-Rad, T. (2005). Knowledge representation issues in semantic graphs for relationship detection. *AAAI Spring Symposium 2005*, pages 91–98. online: http://arxiv.org/PS_cache/cs/pdf/0504/0504072v1.pdf (28.05.2009). (Cited on pages v, 41 and 42.)

Batagelj, V. and Mrvar, A. (2009a). Pajek - program for analysis and visualization of large networks. *Reference Manual*. online: http://vlado.fmf.uni-lj.si/pub/networks/pajek/doc/pajekman.pdf (17.06.2009). (Cited on pages 69, 70 and 71.)

Batagelj, V. and Mrvar, A. (2009b). Pajek - program for large network analysis. Pajek Wiki. online: http://pajek.imfm.si/doku.php?id=pajek (16.06.2009). (Cited on pages v, 69, 71 and 72.)

Battista, G. D., Eades, P., and Tamassia, R. (1999). *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall. ISBN: 0133016153. (Cited on pages 17 and 23.)

Beckett, D. (2005). Dave beckett's resource description framework (rdf) resource guide. Website Dave Beckett. online: http://planetrdf.com/guide/ (06.06.2009). (Cited on page 53.)

Bergman, M. K. (2008). Sweet tools (sem web). Website Michael K. Bergman. online: http://www.mkbergman.com/?page_id=325 (06.06.2009). (Cited on page 53.)

Bergman, M. K. (2009a). Cytoscape: Hands-down winner for large-scale graph visualization. Website Michael K. Bergman. online: http://www.mkbergman.com/?p=415 (20.06.2009). (Cited on pages 72 and 73.)

Bergman, M. K. (2009b). Large-scale rdf graph visualization tools. Website Michael K. Bergman. online: http://www.mkbergman.com/?p=414 (06.06.2009). (Cited on pages 53 and 60.)

Berkhin, P. (2002). Survey of clustering data mining technique. Technical Report. online: http://www.ee.ucr.edu/~barth/EE242/clustering_survey.pdf (20.01.2010). (Cited on page 30.)

Berners-Lee, T. (2006). Notation 3. Website W3C. online: http://www.w3.org/DesignIssues/Notation3.html (01.06.2009). (Cited on page 48.)

Brandes, U., Gaertler, M., and Wagner, D. (2003). Experiments on graph clustering algorithms. Vol. 2832:568–579. Proceedings of the 11th Annual European Symposium on Algorithms; Lecture Notes in Computer Science; Springer; online: http://i11www.iti.uni-karlsruhe.de/algo/people/dwagner/papers/bgw-egc-03.pdf (20.12.2008). (Cited on pages 30 and 37.)

Brickley, D. and Guha, R. (2004). Rdf vocabulary description language 1.0: Rdf schema. Website W3C. online: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ (01.06.2009). (Cited on page 45.)

Buchheim, C., Juenger, M., and Leipert, S. (2002). Improving walker's algorithm to run in linear time. In *Graph Drawing 2002*, volume 2528, pages 344–353. online: http://www.zaik.uni-koeln.de/~paper/index.html?show=zaik2002-431&preprint_session=4c17aae917ae4128a616b320d96814ab (17.04.2009). (Cited on pages 15 and 23.)

Burkhard, R. A. (2004). Learning from architects: The difference between knowledge visualization and information visualization. *Eighth International Conference on Information Visualisation, 2004*, pages 519–524. online: http://www.alexandria.unisg.ch/EXPORT/DL/Remo_Burkhard/34507.pdf (14.04.2009). (Cited on page 11.)

Card, S. K., Mackinlay, J. D., and Shneiderman, B. (1999). *Readings in Information Visualization : Using Vision to Think*. Morgan Kaufman. ISBN: 1558605339. (Cited on pages 3 and 9.)

Chuah, M. C. (1998). Dynamic aggregation with circular visual designs. *In Proceedings of the Symposium on Information Visualization,IEEE, North Carolina*, pages 35–43. online: http://www.shibashake.com/Tech/Publications/CircularVD.pdf (17.05.2009). (Cited on page 29.)

Clark and Parsia (2009). Pellet: The open source owl dl reasoner. Website Clark and Parsia. online: http://clarkparsia.com/pellet/ (29.06.2009). (Cited on pages 102 and 104.)

Collins, C. (2008). Docuburst: Visualizing document content using language structure. DocuBurst Project Website; Department of Computer Science, University of Toronto. online: http://www.cs.utoronto.ca/~ccollins/research/docuburst/index.html (18.04.2009). (Cited on pages v and 14.)

Cui, W. (2007). *A Survey on Graph Visualization*. PhD thesis, Computer Science Department, Hong Kong University of Science and Technology. online: http://www.cse.ust.hk/~weiwei/PQE/WeiweiPQE.pdf (20.1.2010). (Cited on page 17.)

Cytoscape (2009a). Community development process. Cytoscape Developer Wiki. online: http://www.cytoscape.org/cgi-bin/moin.cgi/Community_development_process (27.06.2009). (Cited on page 78.)

Cytoscape (2009b). Concepts document. Cytoscape Developer Wiki. online: http://www.cytoscape.org/cgi-bin/moin.cgi/Concepts_Document (27.06.2009). (Cited on pages v and 79.)

Cytoscape (2009c). Cytoscape 2.x plugins. Cytoscape Plugin-Website. online: http://chianti.ucsd.edu/cyto_web/plugins/index.php (20.06.2009). (Cited on pages 78, 89 and 90.)

Cytoscape (2009d). Cytoscape: Analyzing and visualizing network data. Website Cytoscape. online: http://www.cytoscape.org/ (20.06.2009). (Cited on pages v, 15, 72, 77, 78, 80, 82, 84, 85, 88 and 95.)

Cytoscape (2009e). Cytoscape api. Cytoscape API Javadoc. online: http://chianti.ucsd.edu/Cyto-2_6_3/javadoc/ (03.07.2009). (Cited on page 106.)

Cytoscape (2009f). Cytoscape architecture. Cytoscape Developer Wiki. online: http://www.cytoscape.org/cgi-bin/moin.cgi/Cytoscape_architecture (27.06.2009). (Cited on page 79.)

Cytoscape (2009g). Cytoscape user manual. Cytoscape Manual. online: http://cytoscape.org/manual/Cytoscape2_6Manual.pdf (20.06.2009). (Cited on pages v, 72, 73, 78, 80, 81, 82, 83, 84, 85, 86, 87 and 89.)

Cytoscape (2009h). Developer homepage. Cytoscape Developer Wiki. online: http://www.cytoscape.org/cgi-bin/moin.cgi/Developer_Homepage (27.06.2009). (Cited on pages 79 and 106.)

Cytoscape (2009i). Web service client manager. Cytoscape Wiki. online: http://cytoscape.org/cgi-bin/moin.cgi/Cytoscape_User_Manual/ImportingNetworksFromWebServices (21.06.2009). (Cited on page 81.)

DBpedia (2009). The dbpedia knowledge base. Wiki DBpedia. online: http://dbpedia.org/ (09.07.2009). (Cited on pages 61, 67, 68, 69, 70, 82 and 121.)

Deligiannidis, L., Kochut, K. J., and Sheth, A. P. (2007). Rdf data exploration and visualization. *ACM; CIMS 07*, pages 39–46. online: http://delivery.acm.org/10.1145/1320000/1317362/p39-deligiannidis.pdf?key1=1317362&key2=7090336221&coll=GUIDE&dl=GUIDE&CFID=10229393&CFTOKEN=17737221 (20.11.2008). (Cited on pages 41 and 49.)

Dokulil, J. and Katreniakova, J. (2008). Visual exploration of rdf data. *SOFSEM 2008: Theory and Practice of Computer Science; LNCS 4910; Springer*, pages 672–683. online: http://www.springerlink.com/content/apk706231057q82v/fulltext.pdf (28.05.2009). (Cited on page 49.)

Dynagraph (2009). dynagraph.org. Website Dynagraph. online: http://www.dynagraph.org/ (05.06.2009). (Cited on page 56.)

Eades, P. and Huang, M. L. (2000). Navigating clustered graphs using force-directed methods. *Journal of Graph Algorithms and Applications*, Vol. 4(3):157–181. online: http://www.emis.ams.org/journals/JGAA/accepted/00/EadesHuang00.4.3.ps.gz (12.05.2009). (Cited on pages 28, 39 and 49.)

Eclipse (2009). Eclipse.org. Website Eclipse Foundation. online: http://www.eclipse.org/ (29.06.2009). (Cited on page 95.)

Eklund, P., Roberts, N., and Green, S. (2002). Ontorama: Browsing rdf ontologies using a hyperbolic-style browser. *First International Symposium on Cyber Worlds*. online: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=63C44F7185554F71CE9A7EC1B775EA89?doi=10.1.1.20.6809&rep=rep1&type=pdf (07.05.2009). (Cited on page 23.)

Ellson, J., Gansner, E. R., Koutsofios, E., North, S. C., and Woodhull, G. (2003). Graphviz and dynagraph - static and dynamic graph drawing tools. *Graphviz documentation*. online: http://www.graphviz.org/Documentation/EGKNW03.pdf (20.11.2008). (Cited on pages 50, 51, 55, 56 and 57.)

Foell, H. (2008). Random walk: Prinzip und grundformel. Website Universitaet Kiel. online: http://www.tf.uni-kiel.de/matwis/amat/mw1_ge/kap_6/backbone/r6_3_1.html (29.12.2008). (Cited on page 36.)

Frasincar, F., Telea, A., and Houben, G.-J. (2005). Adapting graph visualization techniques for the visualization of rdf data. *Visualizing the Semantic Web, chapter 9, 2nd edition; Springer*. online: http://www.win.tue.nl/~alext/ALEX/PAPERS/SEMANTIC_WEB/chapter.pdf (20.11.2008). (Cited on pages 50 and 59.)

Furnas, G. W. (1986). Generalized fisheye views. *Human Factors in Computing Systems CHI 86 Conference Proceedings*, pages 16–23. online: http://www.si.umich.edu/~furnas/Papers/FisheyeCHI86.pdf (20.12.2008). (Cited on page 32.)

Gaertler, M. and Wagner, D. (2005). A hybrid model for drawing dynamic and evolving graphs. *GD 2005, LNCS 3843; Springer-Verlag*, pages 189–200. online: http://www.emis.ams.org/journals/JGAA/accepted/00/EadesHuang00.4.3.ps.gz (12.05.2009). (Cited on page 51.)

Gansner, E., Koren, Y., and North, S. (2005). Topological fisheye views for visualizing large graphs. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 11(4):457–468. online: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1432691 (09.05.2009). (Cited on page 28.)

Gansner, E. R. (2007). Drawing graphs with graphviz. *Graphviz Drawing Library Manual*. online: http://www.graphviz.org/pdf/libguide.pdf (06.06.2009). (Cited on pages 55 and 56.)

Gansner, E. R. and North, S. C. (1999). An open graph visualization system and its applications to software engineering. *Software - Practice and Experience*. online: http://www.graphviz.org/Documentation/GN99.pdf (06.06.2009). (Cited on pages 55, 56 and 57.)

GINY (2009). Graph interface library a.k.a. giny. Sourceforge Project-Website. online: http://csbi.sourceforge.net/index.html (27.06.2009). (Cited on page 78.)

Goyal, S. and Westenthaler, R. (2009a). Rdf gravity (rdf graph visualization tool). Website Salzburg Research, Austria. online: http://semweb.salzburgresearch.at/apps/rdf-gravity/user_doc.html (16.06.2009). (Cited on pages v, 66, 67, 68 and 69.)

Goyal, S. and Westenthaler, R. (2009b). Rdf gravity (rdf graph visualization tool). Website Salzburg Research, Austria. online: http://semweb.salzburgresearch.at/apps/rdf-gravity/download.html (16.06.2009). (Cited on page 68.)

Grant, J. and Beckett, D. (2004). Rdf test cases. Website W3C. online: http://www.w3.org/TR/rdf-testcases/ (01.06.2009). (Cited on page 48.)

Graphviz (2009a). Graphviz - graph visualization software. Website Graphviz. online: http://www.graphviz.org (05.06.2009). (Cited on pages v, 55, 56 and 58.)

Graphviz (2009b). Graphviz - graph visualization software. Website Graphviz. online: http://www.graphviz.org/Resources.php (05.06.2009). (Cited on pages 56 and 57.)

Graphviz (2009c). Graphviz - graph visualization software. Website Graphviz. online: http://www.graphviz.org/Download.php (05.06.2009). (Cited on page 56.)

Grinstead, C. M. and Snell, J. L. (1997). *Introduction to Probability*. Number 1. American Mathematical Society. online: http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/amsbook.mac.pdf (29.12.2008). (Cited on page 36.)

Haenelt, K. (2004). Mathematische grundlagen - graphen und operationen auf graphen. Course Slides. online: http://kontext.fraunhofer.de/haenelt/kurs/folien/MathBas-Graphen-6.pdf (29.04.2009). (Cited on page 26.)

Harel, D. and Koren, Y. (2002). A fast multi-scale method for drawing large graphs. *Journal of Graph Algorithms and Applications*, Vol. 6(3):179–202. online: http://www.ii.uj.edu.pl/EMIS/journals/JGAA/accepted/02/HarelKoren02.6.3.pdf (09.05.2009). (Cited on page 25.)

Heer, J. (2008a). Namevoyager. Website Prefuse Information Visualization Toolkit - Demo Gallery. online: http://www.prefuse.org/gallery/namevoyager/ (18.04.2009). (Cited on pages v and 13.)

Heer, J. (2008b). The prefuse visualization toolkit. Website Prefuse. online: http://www.prefuse.org/ (18.04.2009). (Cited on pages v, 14 and 86.)

Herman, I., Delest, M., and Melancon, G. (1998). Tree visualization and navigation clues for information visualization. *COMPUTER GRAPHICS forum*, Vol. 17(2):153–165. online: http://www.lirmm.fr/~melancon/InfoVisu/Publications/TreeVis_CGF_1998.pdf (20.12.2008). (Cited on page 32.)

Herman, I., Marshall, M., Melancon, G., Duke, D., Delest, M., and Domenger, J.-P. (1999). Skeletal images as visual cues in graph visualization. *Data Visualization '99*, pages 13–22. online: http://www.labri.u-bordeaux.fr/~maylis/ARTICLES.ps/latour2.ps.gz (20.12.2008). (Cited on pages 38 and 39.)

Herman, I., Melancon, G., and Marshall, M. S. (2000). Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 6(1):24–43. online: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=841119&isnumber=18188 (20.11.2008). (Cited on pages v, 3, 4, 9, 11, 15, 17, 18, 19, 20, 22, 23, 25, 28, 30, 31, 32, 38, 39, 49 and 50.)

Huang, M. L. and Eades, P. (1998). A fully animated interactive system for clustering and navigating huge graphs. *GD 98, Springer*, pages 374–383. online: http://www.springerlink.com/content/74a403xh3wxxtpqw/fulltext.pdf (19.12.2008). (Cited on pages 30 and 39.)

Jena (2009a). Jena - a semantic web framework for java. Sourceforge Project-Website. online: http://jena.sourceforge.net/ (16.06.2009). (Cited on pages 60, 66, 102 and 104.)

Jena (2009b). Jena 2 inference support. Sourceforge Project-Website. online: http://jena.sourceforge.net/inference/index.html (04.07.2009). (Cited on pages 45, 46 and 104.)

JGraph (2009). Java graph visualization and layout. Website JGraph. online: http://www.jgraph.com/ (23.06.2009). (Cited on page 86.)

JUNG (2009). Jung - java universal network/graph framework. Sourceforge Project-Website. online: http://jung.sourceforge.net/ (16.06.2009). (Cited on page 68.)

Jungnickel, D. (2008). *Graphs, Networks and Algorithms*. Springer; online: http://www.springerlink.com/content/978-3-540-72779-8 (07.05.2009). ISBN (online): 978-3-540-72780-4. (Cited on page 23.)

Kandlhofer, M. (2008). Extendedscatterplotview. Implementierung einer Scatter-Plot Visualisierungskomponente im Rahmen des Master-Praktikums; Technische Universitaet Graz. (Cited on page 12.)

Kap, I. (2009). Kap lab - graph layout. Website Kap IT. online: http://labs.kapit.fr/display/visualizer/Graph+Layout (03.05.2009). (Cited on page 24.)

Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C., and Giannopoulou, E. (2007). Ontology visualization methods - a survey. *ACM Computing Surveys (CSUR)*, Vol. 39(4). (Cited on page 17.)

Kaufmann, M. and Wagner, D. (2001). *Drawing Graphs: Methods and Models*. Springer LNCS Tutorial 2025; online: http://www.springerlink.com/content/xkru1gvnyh5p/ (18.04.2009). ISBN: 3540420622. (Cited on pages v, 3, 17, 18, 19, 20, 21, 23, 24, 25, 30, 32, 40, 49, 50 and 51.)

Kern, A. (2005). Layoutalgorithmen fure hierarchische graphen. Master's thesis, Universitaet Stuttgart, Institut fuer Softwaretechnologie. online: http://elib.uni-stuttgart.de/opus/volltexte/2006/2581/pdf/TR_2005_2324.pdf (03.04.2009). (Cited on pages 21, 23, 25 and 30.)

Klyne, G. and Carroll, J. J. (2004). Resource description framework (rdf): Concepts and abstract syntax. Website W3C. online: http://www.w3.org/TR/rdf-concepts/ (01.06.2009). (Cited on page 43.)

Koutsoos, E. and North, S. C. (1996). Editing graphs with dotty. *Graphviz Documenation*. online: http://www.graphviz.org/pdf/dottyguide.pdf (06.06.2009). (Cited on page 56.)

Lamping, J., Rao, R., and Pirolli, P. (1995). A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. CHI 95 Proceedings Papers. online: http://www.sigchi.org/chi95/Electronic/documnts/papers/jl_bdy.htm (03.05.2009). (Cited on page 24.)

Lee, B., Parr, C. S., Plaisant, C., Bederson, B. B., Veksler, V. D., Gray, W. D., and Kotfila, C. (2006). Treeplus: Interactive exploration of networks with enhanced tree layouts. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12(6):1414–1426. online: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1703363&isnumber=35944 (02.02.2009). (Cited on pages 23 and 49.)

Linz, U. (2002). Graphen. Course Slide; Software Engineering - Institut für Wirtschaftsinformatik. online: http://www.swe.uni-linz.ac.at/teaching/lva/ws01-02/algo2_uebung/Uebung2/graphen1.pdf (17.04.2009). (Cited on pages 18, 19, 23, 26 and 37.)

Liu, S., Pan, Y., Yang, L., and Liu, W. (2005). An lod model for graph visualization and its application in web navigation. *Springer-Verlag Berlin Heidelberg 2005*, pages 441–452. online: http://www.springerlink.com/content/ggn187xwk815kqly/fulltext.pdf (10.12.2008). (Cited on pages 28 and 37.)

Lux, M. (2004). Magick - ein werkzeug fuer cross-media-clustering und visualisierung. Master's thesis, Graz University of Technology, Institute for Information Systems and Computer Media (IICM),. online: http://www.know-center.tugraz.at/content/download/767/4413/file/ (03.01.2009). (Cited on pages 8, 31 and 35.)

Manning, C. D., Raghavan, P., and Schuetze, H. (2008a). Introduction to information retrieval; chapter 16. online: http://nlp.stanford.edu/IR-book/pdf/16flat.pdf (20.12.2008). (Cited on page 31.)

Manning, C. D., Raghavan, P., and Schuetze, H. (2008b). Introduction to information retrieval; chapter 17. online: http://nlp.stanford.edu/IR-book/pdf/17hier.pdf (20.12.2008). (Cited on page 31.)

Manola, F. and Miller, E. (2004). Rdf primer. Website W3C. online: http://www.w3.org/TR/rdf-primer/ (01.06.2009). (Cited on pages v, 43, 45, 46 and 47.)

Matteucci, M. (2009a). Hierarchical clustering algorithms. Tutorial; Politecnico di Milano. online: http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html (10.05.2009). (Cited on pages v and 35.)

Matteucci, M. (2009b). K-means - interactive demo. Java Applet; Politecnico di Milano. online: http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html (13.05.2009). (Cited on pages v, 33 and 34.)

Matteucci, M. (2009c). K-means clustering. Tutorial; Politecnico di Milano. online: http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html (10.05.2009). (Cited on page 33.)

Mazzocchi, S. and Ciccarese, P. (2008a). Simile/welkin. Website Massachusetts Institute of Technology. online: http://simile.mit.edu/welkin/ (14.06.2009). (Cited on pages v, 63, 65, 66, 67 and 68.)

Mazzocchi, S. and Ciccarese, P. (2008b). Welkin user guide. Website Massachusetts Institute of Technology. online: http://simile.mit.edu/welkin/guide.html (14.06.2009). (Cited on pages 63, 65 and 66.)

Mazzocchi, S. and Ciccarese, P. (2008c). Welkin webstart. Website Massachusetts Institute of Technology. online: http://simile.mit.edu/repository/welkin/latest/docs/welkin.jnlp (14.06.2009). (Cited on page 66.)

McGuinness, D. L. and van Harmelen, F. (2004). Owl web ontology language. Website W3C. online: http://www.w3.org/TR/owl-features/ (01.06.2009). (Cited on page 45.)

Moody, J., McFarland, D., and Bender-deMoll, S. (2005). Dynamic network visualization. *American Journal of Sociology*, pages 1206–1241. online: http://www.soc.duke.edu/~jmoody77/ajs_online.pdf (22.05.2009). (Cited on page 24.)

Morris, J. (2009). Metanodeplugin2. Meta-Node Plugin Source Code (SVN). online: http://chianti.ucsd.edu/svn/csplugins/trunk/ucsf/scooter/metaNodePlugin2/ (17.06.2009). (Cited on page 90.)

Morris, J. and Avila-Campillo, I. (2009). Cytoscape metanodes plugin and library. Website University of California, San Francisco. online: http://www.rbvi.ucsf.edu/Research/cytoscape/metanodes/metanodes.html (17.06.2009). (Cited on pages 39 and 90.)

Morris, J., Baumbach, J., and Wittkop, T. (2009). clustermaker. clusterMaker Plugin Source Code (SVN). online: http://chianti.ucsd.edu/svn/csplugins/trunk/ucsf/scooter/clusterMaker (17.06.2009). (Cited on page 90.)

Moscher, W., Sattler, M., and Zwetti, G. (2005). Praxisorientierte aspekte des wissensmanagements; kapitel 2: Begriffsdefinitionen und grundlagen. *Journal of Applied Knowledge Management*, Vol. 1:17–40. online: http://www.moedritscher.com/papers/journal_moedritscher_jakm_vol1.pdf (14.04.2009). (Cited on page 3.)

Mueller, F. (2005). *Granularity based multiple coordinated views to improve the information seeking process*. PhD thesis, Universitaet Konstanz. online: http://deposit.d-nb.de/cgi-bin/dokserv?idn=977933482&dok_var=d1&dok_ext=pdf&filename=977933482.pdf (17.04.2009). (Cited on pages 3, 8, 13, 16 and 49.)

Munzner, T. (2000). *Interactive Visualization of Large Graphs and Networks*. PhD thesis, Stanford University, Department of Computer Science. online: http://graphics.stanford.edu/papers/munzner_thesis/all.print.pdf (31.03.2009). (Cited on pages 8, 11, 15, 16, 19, 23 and 52.)

Mutton, P. and Golbeck, J. (2003). Visualization of semantic metadata and ontologies. *Proceedings of the Seventh International Conference on Information Visualization (IV 03); IEEE Computer Society*, pages 300–305. online: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1217994&isnumber=27378 (28.05.2009). (Cited on page 43.)

Neidhart, T. (2005). Semiautomatische erstellung von wissenslandkarten mittels knowledge mining techniken. Master's thesis, Graz University of Technology, Institute for Knowledgemanagement. online: http://www.know-center.tugraz.at/content/download/881/4940/file/ (03.05.2009). (Cited on pages 31, 32, 33 and 35.)

North, S. C. and Woodhull, G. (2001). On-line hierarchical graph drawing. *Lecture Notes in Computer Science - Graph Drawing*, pages 77–81. online: http://www.graphviz.org/Documentation/NW01.pdf (19.05.2009). (Cited on pages 29 and 51.)

Pfeffer, J. (2009). txt2pajek: Converting text file data sets into pajek format. Pajek Wiki. online: http://pajek.imfm.si/doku.php?id=faq:text2pajek (16.06.2009). (Cited on page 69.)

Pietriga, E. (2004a). Graph stylesheets (gss) in isaviz. Website W3C - RDF Developer. online: http://www.w3.org/2001/11/IsaViz/gss/gssmanual.html (09.06.2009). (Cited on page 59.)

Pietriga, E. (2004b). Isaviz: Installation instructions. Website W3C - RDF Developer. online: http://www.w3.org/2001/11/IsaViz/install.html (09.06.2009). (Cited on page 60.)

Pietriga, E. (2005a). Isaviz user manual. Website W3C - RDF Developer. online: http://www.w3.org/2001/11/IsaViz/usermanual.html (09.06.2009). (Cited on page 59.)

Pietriga, E. (2005b). A toolkit for addressing hci issues in visual language environments. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 00:145–152. online: http://doi.ieeecomputersociety.org/10.1109/VLHCC.2005.11 (12.06.2009). (Cited on page 71.)

Pietriga, E. (2007). Isaviz: A visual authoring tool for rdf. Website W3C - RDF Developer. online: http://www.w3.org/2001/11/IsaViz/ (09.06.2009). (Cited on pages v, 57, 59 and 61.)

Pietriga, E. (2009). Zgrviewer, a graphviz/dot viewer. Sourceforge Project-Website. online: http://zvtm.sourceforge.net/zgrviewer.html (15.06.2009). (Cited on page 71.)

Prinz, W. (2006). The graph visualization system (gvs). Master's thesis, Graz University of Technology, Institute for Information Systems and Computer Media (IICM),. online: http://www.iicm.tu-graz.ac.at/x-coll22100 (20.10.2008). (Cited on pages 3, 4, 11, 14, 17, 18, 19, 20, 21, 26, 28, 31, 55, 57, 60 and 63.)

Purchase, H. (1997). Which aesthetic has the greatest effect on human understanding? *Lecture Notes in Computer Science - Graph Drawing*, Vol. 1353:248–261. online: http://www.springerlink.com/content/39m598l765882401/ (05.05.2009). (Cited on page 22.)

Reingold, E. and Tilford, J. (1981). Tidier drawings of trees. *IEEE Transactions on Software Engineering*, Vol. SE-7(2):223–228. (Cited on pages 15 and 23.)

Sabol, V. (2001). Visualisation islands - interactive visualisation and clustering of search result sets. Master's thesis, Graz University of Technology, Institute for Information Systems and Computer Media (IICM),. online: http://www.know-center.tugraz.at/content/download/577/3506/file/2004_Dip_VSabol.pdf (03.03.2009). (Cited on pages v, 8, 16, 30, 31, 32, 33, 35, 38, 48, 49 and 50.)

Sabol, V. (2008). Informationsvisualisierung. Course Slide. online: http://kmi.tugraz.at/staff/markus/courses/SS2008/707.010_applications-of-KM/slides/week-information-visualization.pdf (17.01.2009). (Cited on pages v, 8, 10 and 11.)

Sayers, C. (2004). Node-centric rdf graph visualization. *Mobile and Media Systems Laboratory, HP Laboratories Palo Alto*. online: http://www.hpl.americas.hp.net/techreports/2004/HPL-2004-60.pdf (10.06.2009). (Cited on page 59.)

Schaffer, D., Zuo, Z., Greenberg, S., Bartram, L., Dill, J., Dubs, S., and Roseman, M. (1998). Navigating hierarchically clustered networks through fisheye and full-zoom methods. online: http://www.markroseman.com/pubs/fisheyetochi96.pdf (19.12.2008). (Cited on pages 39 and 40.)

Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. University of Maryland; Department of Computer Science. online: http://www.cs.umd.edu/Library/TRs/CS-TR-3665/CS-TR-3665.ps.zip (09.04.2009). (Cited on pages 3, 8, 9 and 12.)

Shneiderman, B. and Aris, A. (2006). Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12(5). online: http://hcil.cs.umd.edu/trs/2006-19/2006-19.pdf (20.04.2009). (Cited on page 18.)

Siek, J. (2001). Graph algorithms - breadth-first search. Website Indiana University. online: http://www.boost.org/doc/libs/1_37_0/libs/graph/doc/graph_theory_review.html (22.12.2008). (Cited on page 37.)

Silva, S. F. and Catarci, T. (2000). Visualization of linear time-oriented data: a survey. In *In Proceedings of the First International Conference on Web Information Systems Engineering; IEEE*, pages 310–319. online: http://www.ifs.tuwien.ac.at/~silvia/wien/vu-infovis/articles/WISE2000_silva.PDF (17.04.2009). (Cited on page 13.)

Sourceforge (2009). Tulip. Sourceforge Download. online: http://sourceforge.net/project/showfiles.php?group_id=61223 (13.06.2009). (Cited on pages 60 and 62.)

Spivack, N. (2007). Defining the semantic graph - what is it really? Website Nova Spivack. online: http://novaspivack.typepad.com/nova_spivacks_weblog/2007/11/defining-the-se.html (27.05.2009). (Cited on pages 4 and 42.)

Splendiani, A. (2008). Rdfscape. Wiki RDFScape. online: http://www.bioinformatics.org/rdfscape/wiki/Main/HomePage (16.06.2009). (Cited on page 91.)

Sprenger, T. C., Gross, M. H., Bielser, D., and Strasser, T. (1998). Ivory - an object-oriented framework for physics-based information visualization in java. *Proceedings of IEEE Information Visualization '98*, pages 79–86. online: http://graphics.ethz.ch/Downloads/Publications/Papers/1998/p_Spr98.pdf (20.12.2008). (Cited on page 39.)

Sugiyama, K. (2002). *Graph Drawing and Applications for Software and Knowledge Engineers*. World Scientific. ISBN: 9810248792. (Cited on pages 17, 18, 19, 21, 22, 23, 24 and 51.)

Sugiyama, K., Tagawa, S., and Toda, M. (1981). Methods for visual understanding of hierachical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 11(2):109–125. (Cited on page 23.)

Sun (2009). Sun developer network. Website Sun Microsystems. online: http://java.sun.com/ (29.06.2009). (Cited on page 95.)

Symposium, G. (2009). Graphdrawing. Website Graph Drawing Symposium. online: http://graphdrawing.org/index.html (18.04.2009). (Cited on page 17.)

Tatzmann, B. (2004). Dynamic exploration of large graphs. Master's thesis, Graz University of Technology, Institute for Information Systems and Computer Media (IICM). online: http://www.iicm.tugraz.at/thesis/btatzmann.pdf (22.05.2009). (Cited on pages 51 and 52.)

Tulip (2007a). Tulip developer handbook. *Website Tulip Software: Developer Handbook*. online: http://www.labri.fr/perso/auber/projects/tulip/developerHandbook/devHandbook.pdf (13.06.2009). (Cited on page 63.)

Tulip (2007b). Tulip-software.org. Website Tulip-Software. online: http://www.labri.fr/perso/auber/projects/tulip/ (13.06.2009). (Cited on pages v, 60, 62 and 64.)

Tulip (2007c). Tulip-software.org - samples. Website Tulip-Software. online: http://www.labri.fr/perso/auber/projects/tulip/samples.php (13.06.2009). (Cited on page 60.)

Tulip (2007d). Tulip user manual. *Website Tulip Software: User Manual*. online: http://www.labri.fr/perso/auber/projects/tulip/userHandbook/userManual.pdf (13.06.2009). (Cited on pages 60, 62 and 63.)

Tuomi, I. (1999). Data is more than knowledge. *Journal of Management Information Systems*, Vol. 16(3):107–121. online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.3389&rep=rep1&type=pdf (10.04.2009). (Cited on page 3.)

Utah, U. (2003). Sci image gallery. Website Scientific Computing and Imaging Institute at the University of Utah. online: http://www.sci.utah.edu/sci_gallery.php (10.04.2009). (Cited on pages v and 11.)

van Dongen, S. (2000). *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht. online: http://www.micans.org/mcl/lit/svdthesis.pdf.gz (28.12.2008). (Cited on pages v, 30, 36 and 37.)

Wagner, F., Wolff, A., Kapoor, V., and Strijk, T. (2000). Three rules suffice for good label placement. *Algorithmica Special Issue on GIS*, Vol. 2000. online: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.28.7332 (03.05.2009). (Cited on page 20.)

Wang, X. and Miyamoto, I. (1996). Generating customized layouts. *Lecture Notes in Computer Science; Springer*, pages 504–515. online: http://www.springerlink.com/content/p4231581gk8549wr/ (17.05.2009). (Cited on pages v and 40.)

Weitlaner, E. (1999). Metadata visualisation - visual exploration of file systems and search result sets based on metadata attributes. Master's thesis, Graz University of Technology, Institute for Information Systems and Computer Media (IICM),. online: http://www.iicm.tugraz.at/thesis_95/eweitlaner.pdf (17.04.2009). (Cited on page 13.)

Wikipedia (2009). Wikipedia, the free encyclopedia. Website. online: http://www.wikipedia.org/ (09.07.2009). (Cited on page 121.)

Wills, G. J. (1999). Nicheworks - interactive visualization of very large graphs. *Journal of Computational and Graphical Statistics*, Vol. 8(2):190–212. online: http://www.amstat.org/PUBLICATIONS/jcgs/pdf99/wills.pdf (19.12.2008). (Cited on pages 30 and 39.)

Wong, P. C., Jr., G. C., Foote, H., Mackey, P., and Thomas, J. (2006). Have green - a visual analytics framework for large semantic graphs. *IEEE Symposium on Visual Analytics Science and Technology*, pages 67–74. online: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4035749&isnumber=4035730 (20.04.2009). (Cited on pages 4, 17, 20 and 42.)

Xu, R. and Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, Vol. 16(3):645–678. (Cited on page 30.)

yWorks (2009a). Automatic label placement. yFiles for Java - Developer's Guide. online: http://www.yworks.de/products/yfiles/doc/developers-guide/labeling.html (03.05.2009). (Cited on page 20.)

yWorks (2009b). Edge routing algorithms. yFiles for Java - Developer's Guide. online: http://www.yworks.de/products/yfiles/doc/developers-guide/major_routers.html (03.05.2009). (Cited on page 20.)

yWorks (2009c). yworks - the diagramming company. Website yWorks. online: http://www.yworks.com/en/index.html (22.06.2009). (Cited on page 86.)