Visuelles Data Mining komplexer Strukturen

Diplomarbeit
Universität Rostock, Institut für Informatik

vorgelegt von: Hans-Jörg Schulz
geboren am 26. Februar 1979 in Rostock
Matrikelnummer: 099201929

1. Gutachter: Prof. Dr. Heidrun Schumann
2. Gutachter: Prof. Dr. Andreas Brandstädt
Betreuer: Dipl. Inf. Thomas Nocke

Abgabedatum: 30.09.2004

Danksagung

An dieser Stelle möchte ich die Gelegenheit nutzen und mich bei den Menschen bedanken, die zu der Entstehung dieser Arbeit beigetragen haben. Zuvorderst gilt mein Dank Frau Prof. Schumann für die unkomplizierte Betreuung und das spannende Thema sowie Herrn Prof. Brandstädt für seine geduldige Art und Weise, auch und gerade wenn einmal etwas nicht so geklappt hat. Ebenso möchte ich Thomas Nocke, meinem Betreuer, für die unzähligen Ideen und Anregungen zu dieser Arbeit danken. Ferner haben mich die Gebrüder Pohl bei Fragen zu C++ und OpenGL stets mit Tips und Tricks versorgt, die mir viel Zeit beim Implementieren ersparten. Schlußendlich gilt dem OnlineMaus-Projekt der Universitätskinderklinik Rostock und insbesondere dem ehemaligen Zivildienstleistenden Marco Döbel mein Dank für die kurzfristige Bereitstellung von Rechentechnik.

Zusammenfassung

Um die heutzutage in der Praxis anfallenden großen Datenmengen zu handhaben, wurde mittlerweile eine Vielzahl an Verfahren zur rechnerischen Vorverarbeitung, Visualisierung und interaktiven Exploration entwickelt.
In der vorliegenden Arbeit werden solche Verfahren, die sich zur Behandlung großer struktureller Datensätze eignen, kategorisiert. Insbesondere wird ein Ähnlichkeitskriterium eingeführt, welche die praxisnahe Systematisierung vorhandener Strukturvisualisierungstechniken ermöglicht. Die beschriebenen Verfahren werden in einem darauf aufbauenden Framework integriert, welches erstmals alle Verarbeitungsschritte des visuellen Data Minings in komplexen Strukturen vereinheitlicht und ihre wechselseitigen Abhängigkeiten berücksichtigt.
Um die Funktionalität der entwickelten Software exemplarisch zu belegen, wurde sie an zwei Beispieldatensätzen getestet.

Abstract

Meanwhile an enormous number of techniques for numerical preprocessing, visualization and interactive exploration has been developed, in order to handle the vast amount of ever accumulating data.
This paper categorizes such techniques which are feasible for handling large structural datasets. Furthermore a practical similiarity-based criterion is introduced that enables us to systemize existing methods for structure visualization. The presented techniques are assembled into a framework that unifies all processing stages of visual data mining in complex structures and their mutual dependencies for the first time.
Additionally the developed software is tested on two examples in order to prove its functionality.

CR-Klassifikation

E, G.2.2, G.2.3, G.4, H.1.2, H.3.3, H.5.0, I.3, I.3.6, I.5.3, J.0

Key Words

Visual Data Mining, Visual Data Exploration, Graph Clustering, Information Visualization, Information Structure, Structure Visualization, Treelikeness, Pattern Matching

Inhaltsverzeichnis

1 Einleitung **3**

2 Begriffe und Problemstellung **7**
2.1 Begriffe 7
2.2 Problembeschreibung des Data Minings in Strukturen 10
2.2.1 Spezielle Probleme großer Strukturen 11
2.2.2 Spezielle Probleme der Visualisierung 13

3 Ausgewählte Lösungsansätze **15**
3.1 Automatische Methoden 16
3.1.1 Strukturelle Maßzahlen 16
3.1.2 Hierarchische Clusterverfahren 20
3.1.3 Pattern Matching 23
3.2 Visualisierungstechniken 24
3.2.1 Implizite Baumdarstellungen 26
3.2.2 Explizite Baumdarstellungen 28
3.2.3 Implizite Netzwerkdarstellungen 31
3.2.4 Explizite Netzwerkdarstellungen 33
3.2.5 Baumähnlichkeitsklassifikation der Verfahren 36
3.2.6 Klassifikationsmöglichkeiten für Layouts 38
3.2.7 Vermeiden des Ausgabe-Engpasses 39
3.3 Allgemeine Interaktionstechniken 42

4 Konzeption eines Frameworks zum Struktur-Mining **45**
4.1 Nutzerinteraktion im Vorfeld 50
4.2 Berechnung von Strukturdeskriptoren 51
4.3 Berechnung von Strukturmaßen 51
4.4 Dekomposition und Clustering 52
4.5 Visualisierung 53
4.6 Nutzerinteraktion und Post-Processing im Darstellungsraum . . . 54

5 Realisierung des Frameworks **57**
5.1 Software zur Visualisierung und Analyse großer Graphen 57
5.2 Anforderungen an eine eigene Umsetzung 60

5.3 Designentscheidungen und Architektur 60
5.3.1 Die Klasse `cList` 61
5.3.2 Die Klasse `cTable` 62
5.3.3 Die Klasse `cGraph` 65
5.4 Die Auswahl konkreter Methoden 67
5.5 Die Realisierung der Methoden im Detail 70
5.5.1 Die automatischen Methoden 70
5.5.2 Die Darstellungstechniken und ihre Interaktionsmöglichkeiten 74

6 Visuelle Analyse zweier Beispieldatensätze 79
6.1 Edinburgh Associative Thesaurus 79
6.2 Linkstruktur von Webseiten 82

7 Zusammenfassung und weitere Ideen 85

Anhang A: Details zum Web-Datensatz i

Anhang B: Literaturverzeichnis vii

Anhang C: Abbildungsverzeichnis xiv

Anhang D: Glossar xvi

Kapitel 1

Einleitung

Registrierkassen im Supermarkt, Mautbrücken über der Autobahn, elektronisches Rezept mit der Gesundheitskarte — es gibt kaum einen Lebensbereich unserer heutigen Gesellschaft, der inzwischen nicht von moderner IT durchdrungen ist. Keine Flugbuchung, keine Telefonverbindung, kein Möbelkauf per Kreditkarte, der nicht in großen Datenbanken verzeichnet wird. Während nach einer Schätzung der University of California Berkeley [40] die weltweit im Jahr 1999 erzeugte Informationsmenge bei ca. 2 Exabyte lag (das sind 2 Millionen Terabyte), waren es 2002 schon 5 Exabyte. Das entspricht in etwa dem 500.000-fachen aller Printmedien der amerikanischen „Library of Congress“.

Um dieser Datenflut dennoch Herr zu werden, reicht auch die schnellste Hardware inzwischen kaum noch aus, und es werden ausgeklügelte Algorithmen und Mechanismen benötigt, um aus den archivierten Daten sinnvolle Informationen zu extrahieren. Diese Aufbereitung der Rohdaten durch geschickte Verknüpfung unterschiedlicher Datenquellen oder durch statistische Methoden wird üblicherweise als „Data Mining“ oder „Knowledge Discovery“ bezeichnet. Die Aktualität und Relevanz dieses Forschungsgegenstands wird u.a. daran deutlich, daß einige Universitäten bereits eigene Institute dafür eingerichtet haben, z.B.:

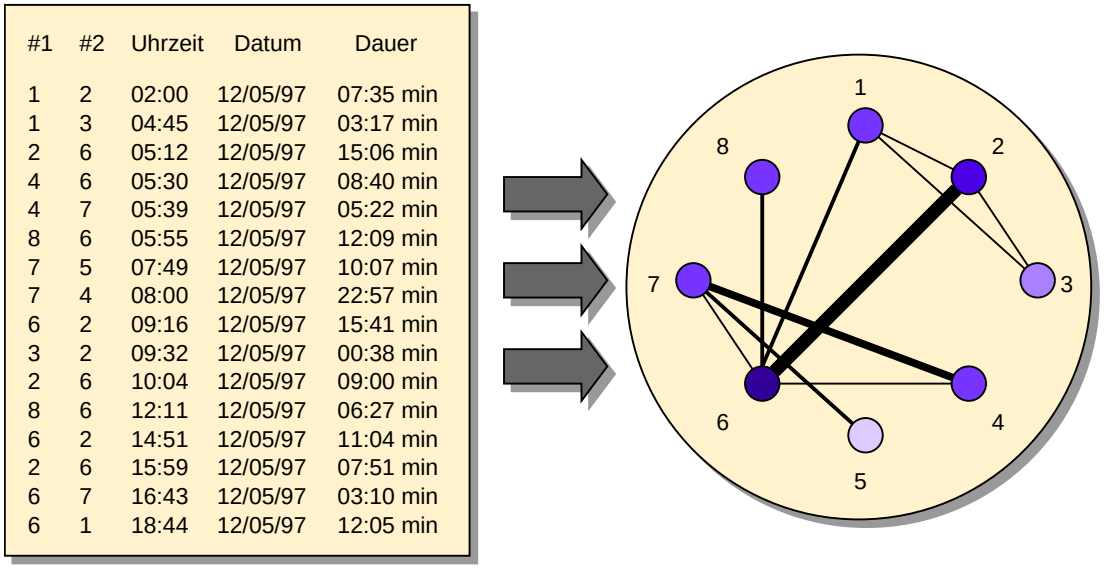
- Laboratory f. Knowledge Discovery in Databases, Kansas State University[1]
- Knowledge Discovery Laboratory, University of Massachusetts Amherst[2]
- Knowledge Discovery and Management Laboratory, Flinders University[3]

Wichtige Anwendungsgebiete des Data Minings finden sich u.a. im sogenannten „Web Mining“, welches von Suchmaschinen automatisch betrieben wird. Ergebnis des Web Minings ist meist ein numerisches Maß relativer Wichtigkeit

[1] http://www.kddresearch.org/
[2] http://kdl.cs.umass.edu/
[3] http://kdm.first.flinders.edu.au/

eines zu bewertenden Hypertextdokuments. Als Rohdaten stehen hierfür der textuelle Inhalt des Dokuments (*Web Content Mining*), dessen Linkstruktur (*Web Structure Mining*) und die bisherige Häufigkeit von Klicks in den Ergebnislisten der Suchmaschine (*Web Usage Mining*) zur Verfügung. Wie wichtig allein dieses Teilgebiet geworden ist, läßt sich an dem rasanten Aufstieg des Suchmaschinenunternehmens GOOGLE zu einer der gegenwärtig bestnotierten Aktiengesellschaften nachvollziehen. Eine gänzlich andere Anwendung des Data Minings ist wiederum das „Crime Data Mining“, welches u.a. für die Analyse kriminieller und terroristischer Netzwerke und finanzieller Transaktionen zur Aufklärung von Geldwäsche und Kreditkartenbetrug eingesetzt wird [17, 18].

Zur algorithmischen Realisierung des Data Minings stehen mittlerweile je nach Anwendungsgebiet und Beschaffenheit der Rohdaten eine Vielzahl unterschiedlich leistungsfähiger Verfahren zur Verfügung. Eines dieser Verfahren ist das **visuelle Data Mining**, welches automatische Mining-Methoden mit Methoden der Informationsvisualisierung verbindet. Dabei macht es sich die Tatsache zunutze, daß der Mensch in der Lage ist, wichtige Zusammenhänge sehr rasch aus einer geeigneten grafischen Repräsentation der Daten zu erfassen und zu erlernen. So sind die Daten und ihre Zusammenhänge in Abbildung 1.1 im Gegensatz zur textuellen Darstellung aus der grafischen Repräsentation wesentlich schneller zu extrahieren:



#1	#2	Uhrzeit	Datum	Dauer
1	2	02:00	12/05/97	07:35 min
1	3	04:45	12/05/97	03:17 min
2	6	05:12	12/05/97	15:06 min
4	6	05:30	12/05/97	08:40 min
4	7	05:39	12/05/97	05:22 min
8	6	05:55	12/05/97	12:09 min
7	5	07:49	12/05/97	10:07 min
7	4	08:00	12/05/97	22:57 min
6	2	09:16	12/05/97	15:41 min
3	2	09:32	12/05/97	00:38 min
2	6	10:04	12/05/97	09:00 min
8	6	12:11	12/05/97	06:27 min
6	2	14:51	12/05/97	11:04 min
2	6	15:59	12/05/97	07:51 min
6	7	16:43	12/05/97	03:10 min
6	1	18:44	12/05/97	12:05 min

Abbildung 1.1: Darstellung von Telefonverbindungen (nach [65]).

Wie man in Abbildung 1.1 sieht, können Rohdaten nicht immer nur in Form einzelner, voneinander gänzlich unabhängiger Meßpunkte gegeben sein. Das würde z.B. für den obigen Datensatz bedeuten, daß man nur wüßte, welcher

Telefonteilnehmer wie oft und wie lange zum Hörer gegriffen hat. Stattdessen sind die gegebenen Daten weitaus reichhaltiger, denn man hat ebenfalls Informationen darüber, zwischen welchen Teilnehmern Verbindungen aufgebaut wurden. Nur damit konnte die nebenstehende grafische Darstellung der Informationsstruktur überhaupt erstellt werden, wobei die Stärke der Kanten der akkumulierten Dauer entspricht und der Helligkeitswert der Knoten mit der Anzahl der geführten Gespräche korrespondiert.

Bisher wurde visuelles Data Mining in erster Linie auf Datenwerte angewandt, deren Struktur dabei jedoch vernachlässigt. Ausnahmen bilden die in [3] vorgestellten Techniken und die Verfahren der visuellen Analyse sozialer Netzwerke, die jedoch entweder eine starke Fokussierung auf ihr jeweiliges Anwendungsgebiet und daher oft einen reduzierten Funktionsumfang haben oder aber nur für Strukturen konzipiert wurden, die gewissen Constraints genügen (sehr kleine Strukturen, gerichtete Strukturen, usw.). Ziel dieser Arbeit ist es, Methoden des visuellen Data Minings und der Netzwerkanalyse auf ihre Verwendungsmöglichkeiten für die Extraktion relevanter Informationen aus beliebigen Strukturdaten zu untersuchen und darauf aufbauend erstmals ein einheitliches Konzept für deren Integration in ein flexibles Rahmenwerk zu erarbeiten. Dieses soll in seinem modularen Aufbau auch zusätzlichen Herausforderungen, wie sehr großen Strukturen oder geringer Analysezeit gewachsen sein. Im folgenden Kapitel werden die dazu nötigen Begriffe formal definiert und häufig auftretende Probleme beim Data Mining in Strukturen beschrieben. Kapitel 3 stellt ausgewählte Lösungsansätze für einen Teil der formulierten Probleme vor, und im 4. Kapitel werden diese einzelnen Techniken und Verfahren in einem flexiblen Framework zusammengefügt. Das 5. Kapitel widmet sich dann der durchgeführten Implementierung des Frameworks, und in Kapitel 6 wird die Leistungsfähigkeit des entwickelten Systems an zwei Fallbeispielen bewiesen. Schlußendlich werden im 7. und letzten Kapitel dieser Arbeit noch einmal alle Ergebnisse zusammengefaßt und weiterführende Ideen skizziert.

Kapitel 2

Begriffe und Problemstellung

2.1 Begriffe

Jede Struktur, wie sie z.B. im vorangegangenen Kapitel 1 in Form von Telefonverbindungsdaten gegeben war, kann als eine auf einer Trägermenge M definierte **Relation** $\Re = (M, R)$ aufgefaßt werden, wobei $R \subseteq M \times M$ gilt. Solch eine Relation läßt sich als **Graph** $G(V, E)$ interpretieren. Dabei entspricht die Trägermenge der Relation M nun der Menge der Knoten V und die Teilmenge R des kartesischen Produkts $M \times M$ der Kantenmenge E. Es werden folgende Spezialfälle unterschieden:

- Die **Nullrelation** $R = \emptyset$ entspricht der Graphklasse der unabhängigen Mengen.
- Die **Allrelation** (oder Universalrelation) $R = M \times M$ entspricht der Klasse der Cliquen.
- Die Relation $\Re$ muß aber nicht zwingend binär sein. In diesem Fall spricht man von einem **Hypergraphen** $H(V, \mathcal{E})$, dessen Hyperkanten beliebig große Teilmengen der Knotenmenge V sein können.

Beim visuellen Data Mining struktureller Datenbestände setzt sich die Trägermenge (und damit auch die Knotenmenge des korrespondierenden Graphen) aus diskreten, eventuell abstrakten Objekten, den **Informationsobjekten**, zusammen. Dies können z.B. einfach Meßpunkte sein oder auch Schriftstücke oder Telefonanschlüsse wie im vorangegangenen Beispiel. Basierend auf dieser Sichtweise der Daten wurde in [35, 67] ein formales Informationsmodell entwickelt, welches in einer eigens für diese Arbeit angepaßten und erweiterten Version folgendermaßen definiert ist:

- Die **Informationsmenge** $IM = \{IO_1, \ldots, IO_n\}$ ist eine endliche Menge bestehend aus paarweise verschiedenen Informationsobjekten. Sie bildet damit die Knotenmenge V des zugehörigen Graphen $G(V, E)$.

- Dabei liefert die **Attributfunktion** $attr(IO_i, \ldots, IO_j)$ die charakteristischen Merkmale der angegebenen Informationsobjekte. Diese entsprechen den eigentlichen Datenwerten und können beliebigen numerischen oder nominalen Wertebereichen zugeordnet sein.

- Die Gesamtheit der Attribute aller Informationsobjekte $AM = attr(IM)$ wird als **Attributmenge** bezeichnet.

- Die Wertebereiche der Attribute bestimmen die Skalierung der durch die Attribute definierten Dimensionen eines durch sie aufgespannten **Informationsraumes** IR. Dessen Dimensionalität entspricht also der Kardinalität der Attributmenge: $|AM| = dim(IR)$. So können alle Informationsobjekte anhand ihrer Attribute jeweils einem konkreten Punkt innerhalb dieses Informationsraumes zugeordnet werden.

- Durch eine **Informationsstruktur** $IS \subseteq IM \times IM \times \mathbb{R}$ können nun schließlich gewichtete Abhängigkeiten zwischen je zwei Informationsobjekten definiert werden. Dies erlaubt die formale Beschreibung der bis jetzt als „strukturelle Daten“ oder „Strukturdaten“ bezeichneten Zusammenhänge zwischen den Informationsobjekten und bildet somit die Kantenmenge E des zugehörigen Graphen $G(V, E)$. Informationsstrukturen können entweder explizit gegeben und damit Teil der Rohdaten sein oder aber nachträglich aus den Attributen der Informationsobjekte generiert werden. Dazu wird häufig ein Maß definiert, welches die Ähnlichkeit zweier Informationsobjekte quantifiziert. Überschreitet dieses Maß einen gewissen Schwellwert, wird der Informationsstruktur eine Kante zwischen den beiden Informationsobjekten hinzugefügt. Ein Beispiel für solch eine implizite Informationsstruktur zeigt Abbildung 2.1:

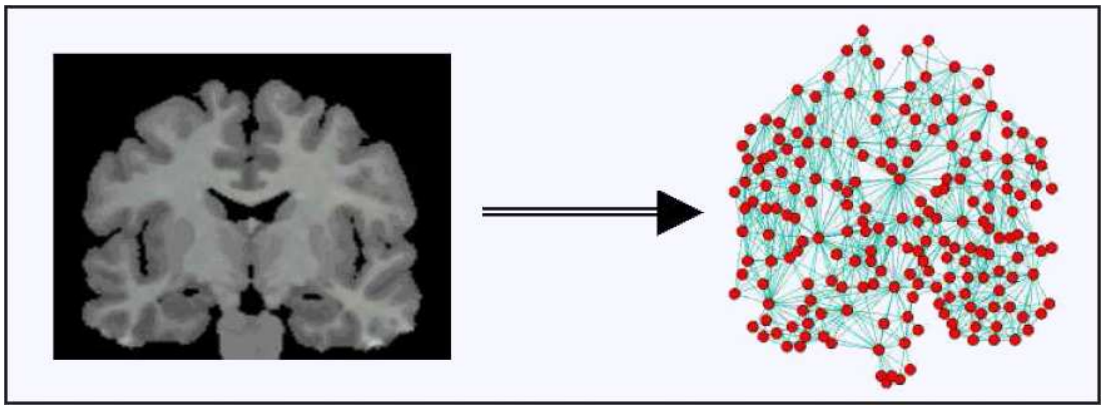


Abbildung 2.1: Ein Beispiel für eine implizit durch nebenstehendes Computertomogramm gegebene Informationsstruktur (aus [7]).

- Falls mehrere Informationsstrukturen für dieselbe Informationsmenge vorliegen (multirelationale Daten), werden diese in der **Strukturmenge** $SM = \{IS_1 \dots IS_k\}$ zusammengefaßt.

Auch wenn dieses Informationsmodell für die vorliegende Arbeit vollkommen ausreicht, liegt es auf der Hand, daß es sich weiter verallgemeinern läßt:

- Informationsstrukturen könnten nicht nur simple Gewichtungen $\in \mathbb{R}$, sondern wiederum komplette Attributmengen zugeordnet sein. Ein Beispiel für solch eine komplexe Attributmenge sind die Telefonverbindungen aus Abbildung 1.1: dort wird nicht nur die Dauer als einziges Gewicht genannt, sondern auch Uhrzeit und Datum. Eventuell wäre es sogar wünschenswert, den Namen des verwendeten Call-by-Call-Anbieters und dessen Minutentarif als zusätzliche Kantenattribute mit in die Attributmenge der Kanten aufzunehmen. Bei solch komplexen Gewichten ist es außerdem sinnvoll, einen eigenen Informationsraum für jede Informationsstruktur zu definieren.

- Es wäre möglich, die streng binäre Informationsstruktur derart zu erweitern, daß beliebig viele Objekte gleichzeitig miteinander in Relation stehen können [42]. Damit entstünden Hypergraphkanten, für die aber bis jetzt noch keine zufriedenstellenden Darstellungstechniken existieren, weswegen sie sich nicht für das visuelle Data Mining eignen. Die Abbildung 2.2 zeigt zwei am Tokyo Institute of Technology entstandene automatisch generierte Darstellungen einfacher Hypergraphen:

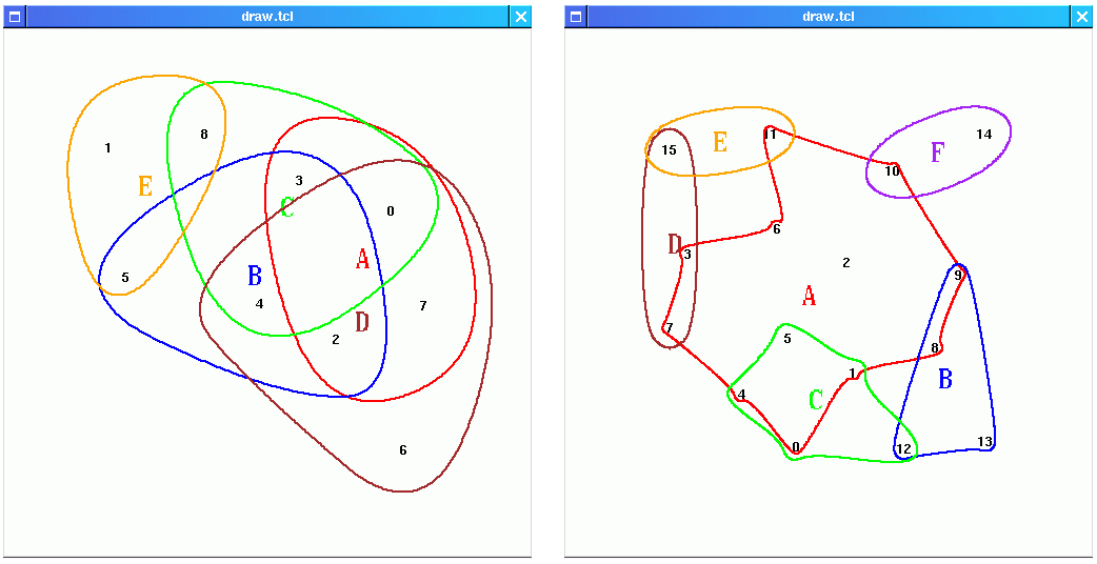


Abbildung 2.2: Autom. generierte Euler-Venn-Diagramme zur Visualisierung einfacher Hypergraphen (http://www.fz.dis.titech.ac.jp/~murofusi/venn_v.html).

Für den weiteren Verlauf dieser Arbeit spielt vor allem die auf der Informationsmenge erklärte Informationsstruktur eine bedeutende Rolle. Diese Einheit aus der Menge der Informationsobjekte und ihrer Struktur wird nachfolgend als Graph $G(V, E)$ bezeichnet und behandelt.

2.2 Problembeschreibung des Data Minings in Strukturen

Ein Anwender, der Data Mining in Graphen betreiben möchte, sucht entweder explorativ nach Auffälligkeiten in den Graphen oder hat ein konkretes Analyseziel auf der Anwendungsebene, welches in eine abstrakte Formulierung als Graphenproblem übertragen werden muß. Hier einige z.T. stark vereinfachte Beispiele:

- **gegeben:** Datenbank mit wissenschaftlichen Veröffentlichungen
  gesucht: Die meistzitierte Veröffentlichung innerhalb der Datenbank
  Modellierung: Knoten = Veröffentlichungen, gerichtete Kante von A nach B = A hat B zitiert
  Problem: Finde Knoten mit den meisten eingehenden Kanten.

- **gegeben:** Zwei Haltestellen in einem U-Bahn-Netz
  gesucht: Schnellste Verbindung zwischen den Haltestellen.
  Modellierung: Knoten = Haltestellen, Kanten = Verbindungen, Kantengewichte = Fahrtdauer
  Problem: Finde den kürzesten Weg zwischen zwei Knoten in diesem Graphen.

- **gegeben:** Finanzielle Transaktionen zwischen Firmen, die zu derselben Holding-Gesellschaft gehören
  gesucht: Indizien für Geldwäsche.
  Modellierung: Knoten = Firmenkonten, Kanten = Transaktionen, Kantengewichte = Summe der transferierten Gelder
  Problem: Finde gerichtete Zyklen innerhalb dieses Graphen.

- **gegeben:** Militärische Strukturen der Armee Polens, Ungarns, Tschechiens und der Ukraine
  gesucht: Zwei Paarungen, bestehend aus den Armeen je zweier Staaten, deren Militär am ähnlichsten strukturiert ist. So lassen sich die Truppen bei einer UN-Friedensmission möglichst gut integrieren.
  Modellierung: Knoten = Soldat, Kanten = Befehlsgewalt
  Problem: Bilde zwei disjunkte Paare mit den jeweils ähnlichsten Graphen.

Diese vier Fragestellungen zu Informationsstrukturen stehen stellvertretend für eine Vielzahl solcher Probleme. Sie lassen sich in die vier zugehörigen Problemkategorien einordnen:

- Das **Finden besonderer Knoten**. Oft sucht der Anwender nach Knoten, die sich in irgendeiner Weise gegenüber den anderen auszeichnen. Dies können besonders „zentrale“ Knoten oder z.B. auch Artikulationspunkte[4] sein.

- Das **Finden besonderer Kanten**. Es kann interessant sein, eine minimale Kantenmenge zu finden, deren Entfernen aus einem gerichteten zyklischen Graphen einen azyklischen macht. Auch spezielle Kantenzüge wie der kürzeste Weg aus dem zweiten Beispiel sind häufig von Interesse für den Anwender.

- Das **Finden spezieller Teilstrukturen**. Hierbei wird nach Unterstrukturen gesucht, die entweder vom Nutzer vorgegeben wurden (größte Clique, längster Kreis, Hierarchien, etc.) oder sich durch besondere Charakteristika hervorheben (häufiges Auftreten, hohe Dichte[5]).

- Das **Vergleichen von Graphen**. Dazu zählt der Vergleich untereinander (wie im vierten Beispiel), aber auch mit einer vorgegebenen Graphklasse (Baumähnlichkeit, Kreisähnlichkeit, etc.).

Wenn dann das konkrete Graphenproblem mit Hilfe des visuellen Data Minings gelöst wurde, muß das Ergebnis (die größte Clique, der dichteste Teilgraph, usw.) wieder zurück auf die Anwendungsebene übertragen werden.

Ferner ist zu beachten, daß die vorliegende Informationsstruktur sowohl statisch als auch dynamisch sein kann. Während eine statische Struktur fest vorgegeben ist, kann sich eine dynamische Struktur jederzeit ändern. Solche sich ständig ändernden Graphen können beispielsweise Peer-to-Peer-Netzwerke sein, bei denen sich fortwährend neue Teilnehmer an- oder abmelden und Verbindungen zu anderen P2P-Teilnehmern auf- oder abbauen [19, 68]. Ebenso können die Attribute der Informationsobjekte statisch oder dynamisch sein. Der dynamische Fall tritt z.B. beim Propagieren von Marken in einem Petri-Netz auf. Solche Änderungen der Struktur oder der Attribute sind je nach Anwendungsgebiet getaktet oder kontinuierlich. Während also ein Knoten eines P2P-Netzwerks jederzeit Verbindungen auf- und abbauen kann, schaltet ein klassisches Petri-Netz immer nur in diskreten Zeitschritten.

2.2.1 Spezielle Probleme großer Strukturen

Viele der im vorigen Abschnitt aufgeführten Probleme sind NP-vollständig. So ist z.B. das Finden einer minimalen Kantenmenge, deren Entfernen aus

[4]Dies sind Knoten, deren Entfernen den Graphen in mehrere Zusammenhangskomponenten zerfallen läßt.

[5]Das Verhältnis aus der Anzahl der vorhandenen Kanten zu der Zahl aller möglichen Kanten.

einem zyklischen Graphen einen DAG[6] liefert, unter dem Namen Minimum-Feedback-Arc-Set-Problem oder Maximum-Acyclic-Subgraph-Problem als NP-vollständig bekannt. Auch das Finden der größten Clique (Maximum-Clique-Problem) oder eines vorgegebenen Teilgraphen (Subgraph-Isomorphism-Problem) zählen zu dieser Klasse von Problemen, für die bis heute kein deterministischer polynomialer Lösungsalgorithmus existiert.

Viele andere Probleme lassen sich zwar in Polynomzeit lösen, aber auch das kann für große Graphen noch sehr lange dauern. Deshalb versucht man Algorithmen zu finden, die eine subquadratische Laufzeitkomplexität aufweisen. Dies ist häufig für Graphen mit bestimmten Struktureigenschaften möglich. So können viele der in den nachfolgenden Kapiteln angegebenen Probleme auf Graphen mit geringer Dichte[7] schneller gelöst werden. Und gerade solche Graphen liegen in der Praxis besonders häufig vor[8].

Bei sehr großen Graphen spielt neben der Laufzeitkomplexität der einzelnen Algorithmen aber auch deren Speicherkomplexität eine bedeutende Rolle. Dabei unterscheidet man nach [1] die folgenden Fälle:

- **Interner Fall:** Sowohl Knoten- als auch Kantenmenge passen gleichzeitig in den Arbeitsspeicher.
- **Semi-Externer Fall:** Die Knotenmenge findet im Arbeitsspeicher Platz, die Kantenmenge nur auf externen Speichermedien.
- **Externer Fall:** Sowohl Knoten- als auch Kantenmenge sind zu groß, um im Arbeitsspeicher Platz zu finden und müssen auf Datenträgern vorgehalten werden.

Während für den ersten der genannten Fälle die üblichen graphentheoretischen Verfahren aus der Literatur [14, 60] Anwendung finden, müssen für die beiden letztgenannten Fälle speziell angepaßte Algorithmen eingesetzt werden, die in erster Linie die Anzahl der durchzuführenden Lade- und Speichervorgänge (sogenannte *swaps*) auf die Datenträger minimieren. Denn das Lesen und Beschreiben eines externen Speichermediums ist ungleich langsamer, als das Lesen und Beschreiben des Arbeitsspeichers[9]. Diese Diskrepanz zwischen dem vergleichsweise

[6] gerichteter, azyklischer Graph (***d****irected* ***a****cyclic* ***g****raph*)

[7] *sparse graphs*; Gegenteil: *dense graphs*

[8] In einem Telefonnetz mit sehr vielen Teilnehmern wird jeder dennoch nur mit den wenigen ihm bekannten Personen innerhalb des Netzes Kontakt haben. Auch in der Bindungsstruktur eines großen Moleküls geht kein Atom mehr chemische Bindungen ein, als Elektronen in seine äußere Atomschale passen, wenn von einer geringen Zahl zusätzlicher Wasserstoffbrückenbindungen abgesehen wird.

[9] Während die Zugriffszeiten einer Festplatte im zweistelligen Millisekundenbereich liegen, betragen die des Hauptspeichers nur wenige Nanosekunden.

kleinen aber schnellen Arbeitsspeicher und dem heutzutage verfügbaren riesigen externen Speicherplatz mit langsameren Datenzugriff wird unter dem Begriff „*I/O-Bottleneck*“ zusammengefaßt [3].

2.2.2 Spezielle Probleme der Visualisierung

Die große Stärke des visuellen Data Minings, nämlich die grafische Darstellung der Daten, ist auch gleichzeitig die größte Schwäche. Denn gerade bei sehr großen Graphen kommt eine Visualisierung schnell an ihre Grenzen — die Ausgabe ist unübersichtlich und eventuell werden Knoten durch andere verdeckt. Selbst bei einer noch so geschickten Anordnung des Graphen läßt sich spätestens dann nichts mehr ausrichten, wenn die Kardinalität der Knotenmenge größer als die Anzahl verfügbarer Pixel auf dem Ausgabemedium ist. Sollen dann zusätzlich noch Strukturinformationen (Relationen) in Form von verbindenden Linien zwischen den Meßpunkten übersichtlich dargestellt werden, verschärfen sich die Anforderungen an die Visualisierung. Dieses Problem ist unter dem Namen „*Screen Bottleneck*“ (Ausgabe-Engpaß) bekannt [3] und kann nur mit Hilfe einer Reihe spezieller Techniken zur Reduktion der visuellen Komplexität umgangen werden. Dabei können z.B. Teilgraphen ausgeblendet oder in einem Stellvertreterknoten zusammengefaßt werden. Dennoch müssen auch die letzten Details einer Informationsstruktur durch den Nutzer erschlossen werden können, so daß leistungsstarke Interaktionstechniken nötig sind, um solch eine interaktive Exploration der Struktur zu ermöglichen.

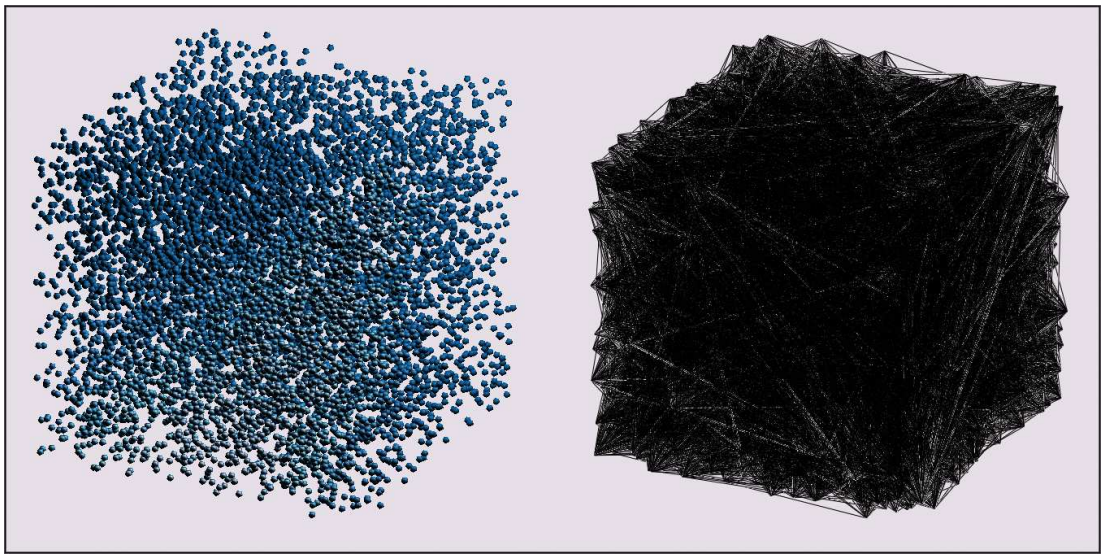


Abbildung 2.3: Illustration des Ausgabe-Engpasses bei der Darstellung aller 8.210 Datenpunkte (links) und 261.453 Relationen (rechts) des EAT-Datensatzes (siehe Abschnitt 6.1) in einem Würfel.

Kapitel 3

Ausgewählte Lösungsansätze

Für die unterschiedlichen im vorhergehenden Kapitel aufgeworfenen Fragestellungen und Probleme existieren eine Vielzahl von Lösungsansätzen, von denen hier einige näher erläutert und in den Ablauf des Data Mining Prozesses eingeordnet werden. Dieser Ablauf wird durch das 1996 von Ben Shneiderman formulierte **„Mantra der visuellen Informationssuche“** (*Visual Information-Seeking Mantra*) [56] folgendermaßen gegliedert: beginnend bei einer Übersichtsdarstellung sämtlicher Datensätze (*Overview*) vergrößert man zuerst den interessierenden Bereich (*Zoom*), blendet dort Unwichtiges aus (*Filter*) und läßt sich dann je nach Bedarf zusätzliche Informationen zu den verbliebenen Datensätzen ausgeben (*Details on Demand*) [56]. Idealerweise liegt am Ende des visuellen Mining-Prozesses nicht nur ein auf den Rohdaten basierender Erkenntnisgewinn, sondern gleichzeitig auch eine expressive, effektive und angemessene grafische Darstellung zur Kommunikation des Sachverhalts an verschiedene Nutzergruppen. Durch die Dualität von Graphen und Informationsstrukturen stehen dabei neben den allgemein üblichen Verfahren des visuellen Data Minings in Informationsmengen auch die Methoden der Graphentheorie zur Analyse und die des „*Graph Drawing*“ zur Darstellung der Strukturen zur Verfügung. Die nachfolgend vorgestellten Lösungsansätze werden in Abhängigkeit von ihrer Funktion in Anlehnung an [52, 64] in die folgenden Kategorien unterteilt:

- **Automatische Methoden** umfassen alle Verfahren, die auf den abstrakten (Roh-)Daten arbeiten. Hierzu gehören zum Einen Vorverarbeitungsverfahren, welche die Rohdaten für die graphische Darstellung und anschließende Exploration aufbereiten, wie z.B. Clusterverfahren oder Verfahren, die gewisse Metadaten extrahieren. Zum Anderen zählen hierzu auch alle automatischen Techniken, die in der Lage sind, Knoten, Kanten oder gar komplette Teilstrukturen nach vorgeschriebenen Kriterien aus den Datenbeständen zu extrahieren.

- Als **Visualisierungsverfahren** bezeichnet man alle Methoden, die den ab-

strakten Daten konkrete Positionen, Farb- und Helligkeitswerte, Ikonen[10] usw. in einem beliebigen Bezugssystem (Darstellungsraum, Präsentationsraum) zuordnen.

- Unter der Bezeichnung **Interaktionsverfahren** werden nachfolgend sämtliche Navigationsmöglichkeiten, Manipulationen des Darstellungsraumes, interaktive Selektionsmöglichkeiten und der Zugriff auf Detailinformationen zusammengefaßt, die nicht bereits Bestandteil des verwendeten Visualisierungsverfahrens sind.

3.1 Automatische Methoden

3.1.1 Strukturelle Maßzahlen

Beim Data Mining auf Attributwerten kommen sehr häufig statistische Maße zum Einsatz, um charakteristische Eigenschaften des Datensatzes zu beschreiben. Für das Data Mining auf Strukturen eignen sich zu diesem Zweck besonders strukturelle Maße der Graphentheorie, die als abgeleitete Metadaten des jeweiligen Datensatzes interpretiert werden können. Mit ihrer Hilfe werden die gegebenen Rohdaten durch zusätzliche Informationen angereichert. Dieser wichtige Schritt der Vorverarbeitung kann dazu dienen, bestimmte Aspekte einer Informationsstruktur numerisch zu erfassen. Solche Maße lassen sich sowohl für die Knoten als auch für die Kanten des mit solch einer Informationsstruktur korrespondierenden Graphen $G(V, E)$ berechnen. Eine kleine Auswahl wichtiger struktureller Maße wird nachfolgend gegeben[11]:

- Der **Grad eines Knotens** (*degree*) $deg(v)$ mit $v \in V$ gibt die Anzahl der zu v inzidenten Kanten an.

- Die **Größe der k-Nachbarschaft** (*k-neighborhood size*) $|N_k(v)|$ mit $v \in V$ entspricht der Anzahl aller Knoten, die nicht weiter als k Kantenzüge von v entfernt sind: $N_k(v) = \{u : u \in V \setminus \{v\} \wedge dist(u, v) \leq k\}$

- Die **Nähe eines Knotens** $v \in V$ (*closeness*) ist definiert als die Summe der Längen der kürzesten Wege zu jedem anderen Knoten $\sum_{\forall u \in V \setminus \{v\}} dist(v, u)$.

- Die **Exzentrizität eines Knotens** $v \in V$ (*eccentricity*) gibt hingegen nur die Länge des längsten aller von diesem Knoten $v \in V$ ausgehenden kürzesten Wege an: $\max_{\forall u \in V \setminus \{v\}} dist(v, u)$. Der Maximalwert der Exzentrizitäten

[10] Nach *Charles S. Peirce* unterscheidet man drei Zeichentypen: Indizes, Ikonen und Symbole. Im Gegensatz zu Symbolen oder Indizes weisen Ikonen immer eine Ähnlichkeit zu dem durch sie repräsentierten Gegenstand, Sachverhalt oder Zustand auf (Piktogramme, Hieroglyphen, Emoticons). In diesem Sinne werden Ikonen auch in der Informationsvisualisierung benutzt, um Datenwerte oder Kategorien bildhaft darzustellen.

[11] Diese Liste basiert auf einer Tabelle in [13].

aller Knoten, also die Länge des insgesamt längsten aller kürzesten Wege, wird als **Durchmesser des Graphen** (*diameter*) $diam(G)$ bezeichnet. Der kleinste aller berechneten Exzentrizitätswerte wird auch **Radius des Graphen** genannt. Die Menge der Knoten mit minimaler Exzentrizität, deren Exzentrizitätswert also gleich dem Radius ist, nennt man das **Zentrum des Graphen** (*center*).

- Die **Zwischenzahl eines Knotens** (*node betweenness*) ist definiert als die Anzahl aller kürzesten Wege, die ihn durchlaufen. Für Kanten ist dieses Maß in analoger Weise erklärt (*edge betweenness*).

- In gewichteten und gerichteten Graphen läßt sich die **Fluß-Zwischenzahl einer Kante** (*flow betweenness*) bestimmen, die der Summe aller Maximalflüsse[12] entspricht, die diese gerichtete Kante passieren.

Häufig werden die oben aufgeführten strukturellen Maße auch als **Zentralitätsmaße** bezeichnet und dabei als Kenngrößen für die Wichtigkeit eines Knotens innerhalb des gegebenen Netzwerkes verstanden. Um sie auch mit Zentralitätsmaßen anderer Graphen vergleichbar zu machen, muß allerdings meist noch eine geeignete Normierung durchgeführt werden.

Weiterhin gibt es **strukturelle Ähnlichkeitsmaße** für je zwei Knoten $v \epsilon V, u \epsilon V$ mit $u \neq v$, wie z.B.:

- die **Konnektivität zweier Knoten** (*connectivity*), welche die minimale Zahl zu entfernender Kanten angibt, so daß die Knoten im Ergebnis nicht mehr in einer Zusammenhangskomponente liegen, also kein Weg zwischen den beiden Knoten mehr existiert.

- die **Abhängigkeitszahl** des Knotens v vom Knoten u (*dependency*), die als die Anzahl aller kürzesten Wege von v, die u passieren, definiert ist. Diese Abhängigkeitszahl fällt z.B. als Zwischenergebnis bei der Berechnung der Knoten-Zwischenzahlen an (siehe Abschnitt 5.5.1).

All‘ die angegebenen Maßzahlen stammen aus dem Gebiet der Analyse sozialer Netzwerke, welches sich schon früh mit strukturellen Daten beschäftigt hat. Dabei geht es darum, z.B. durch Analyse innerbetrieblicher eMail-Kommunikation die Meinungsmacher und Multiplikatoren in einer Firma ausfindig zu machen.

[12]Dabei betrachtet man den Graphen $G(V, E)$ als **Netzwerk** mit zwei ausgezeichneten Knoten, der **Quelle** q und der **Senke** $s \neq q$, und einer Gewichtsfunktion $c : E \rightarrow \mathbb{R}^+$, die **Kapazitäten** für jede Kante des Graphen definiert. Als Fluß bezeichnet man jede Funktion $f : E \rightarrow \mathbb{R}^+$, so daß für alle Kanten $e \epsilon E : 0 \leq f(e) \leq c(e)$ gilt und die **Flußerhaltung** gewährleistet bleibt. Das bedeutet, daß die Summe der eingehenden Flüsse eines jeden Knoten $v \epsilon V \setminus \{q, s\}$ der Summe seiner ausgehenden Flüsse entspricht. Der **Gesamtfluß** eines Netzwerks ist gleich dem von der Senke s ausgehenden Fluß abzüglich dem eventuell eingehenden. Das Maximum des Gesamtflusses wird auch als **Maximalfluß** des Netzwerks bezeichnet.

Informationsstrukturen treten aber auch in anderen Bereichen auf: als Transportnetzwerke, als Ontologien, als Stammbäume oder als Moleküldatensatz mit komplexen chemischen Bindungsstrukturen, Bindungswinkeln und Bindungsenergien. Somit erheben die aufgelisteten Maße keinen Anspruch auf Vollständigkeit; vielmehr ist darauf zu achten, daß es für den jeweiligen Anwendungskontext eines Datensatzes vielfältige andere Strukturmaße geben kann.

Von Interesse können auch **globale Maßzahlen** des Graphen sein, die als Attribute der gesamten Struktur deren Eigenheiten beschreiben:

- Zu den einfachsten Maßen dieser Kategorie zählen wohl die **Durchschnittsmaße**. Sie werden durch das arithmetische Mittel eines Maßes aller Knoten/Kanten gebildet. Der **durchschnittliche Knotengrad** läßt sich daher wie folgt berechnen: $\sum_{v \in V} \frac{deg(v)}{|V|}$.

- Weiterhin kann die sogenannte **Dichte des Graphen** (*density*) berechnet werden, für die in einigen Veröffentlichungen auch der Name **Kompaktheit** (*compactness*) benutzt wird. Sie ist das Verhältnis der Zahl der vorhandenen Kanten in einem Graphen zu der Zahl aller möglichen Kanten, also $\frac{|E|}{|\wp_2(V)|}$ mit $|\wp_2(V)| = \frac{|V| \cdot (|V|-1)}{2}$.

Besonders wichtig sind nachfolgend die **Ähnlichkeitsmaße**, welche die strukturelle Nähe eines Graphen zu einer Graphklasse auf verschiedene Art und Weise zu quantifizieren versuchen. So spiegelt die **Baumähnlichkeit** eines Graphen dessen Nähe zur Klasse der Bäume, also den zusammenhängenden, kreisfreien Graphen, wider. Die Baumähnlichkeitsmaße dienen in dieser Arbeit dazu, Visualisierungstechniken für Informationsstrukturen nach ihrem jeweiligen Anwendungsgebiet zu klassifizieren. Dabei werden die verwendeten Baumähnlichkeitsmaße in **globale und lokale Maße** unterteilt. Dabei ist für das Verständnis der lokalen Baumähnlichkeitsmaße vor allem der Begriff der **k-Bäume** von Bedeutung, welcher nach [14] rekursiv definiert wird:

1. Jede k-Clique, also jeder vollständige Graph mit k Knoten, ist ein k-Baum.

2. Das Hinzufügen eines Knotens v zu einem k-Baum resultiert wiederum genau dann in einem neuen k-Baum, wenn v mit allen k Knoten einer beliebigen k-Clique des k-Baumes verbunden wird.

3. Es existieren keine weiteren k-Bäume.

Ein Beispiel zur Veranschaulichung der k-Bäume wird in Abbildung 3.1 gegeben. Während bei einem 1-Baum, also dem klassischen Baum, immer Punkte aneinandergereiht werden, sind es beim 2-Baum Dreiecke, die immer eine Seite gemeinsam haben, beim 3-Baum Tetraeder, die je eine Fläche miteinander teilen,

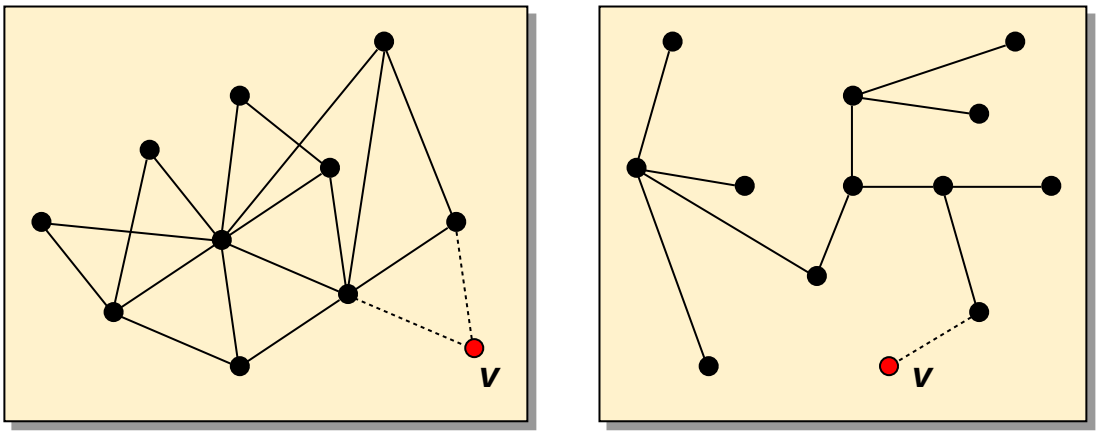


Abbildung 3.1: Das Anfügen eines Knotens v an einen 2-Baum (links) und an einen „klassischen“ 1-Baum (rechts).

usw...[13] Ein weit entfernter Betrachter der grafischen Darstellung eines 2- oder 3-Baumes würde wohl ohne Schwierigkeiten davon zu überzeugen sein, daß er es in der Tat mit einem Baum zu tun hat — die kleinen lokalen Ungereimtheiten (Dreiecke, Tetraeder, etc.) ändern nichts an der baumähnlichen Gesamtstruktur und fallen ab einer gewissen Entfernung auch nicht mehr auf.

Zwei gebräuchliche lokale Baumähnlichkeitsmaße sind:

- die **Baumweite eines Graphen** (*treewidth*) $tw(G)$, die das kleinste k liefert, für das $G(V, E')$ ein **partieller k-Baum** ist, d.h. $G(V, E)$ ist ein k-Baum mit $E' \subseteq E$.

- die **Länge des längsten sehnenlosen Kreises** innerhalb des Graphen, da diese lokale Maßzahl lediglich Auskunft über eine Stelle im Graphen gibt, nämlich die, wo der längste Kreis auftritt. Wieviele solcher längsten Kreise es gibt, und ob und wieviele kleinere Kreise noch existieren, läßt sich mit diesem Maß nicht ermitteln. Wenn diese Maßzahl sehr klein ist (im Falle von k-Bäumen wäre sie höchstens 3), ist dies ebenfalls ein Indiz für eine hohe Baumähnlichkeit.

Als globale Baumähnlichkeitsmaße sollen die folgenden selbstentwickelten, an die Visualisierungspraxis angelehnten Maße dienen. Mit ihrer Hilfe lassen sich im Abschnitt 3.2.5 einzelne Visualisierungsverfahren bezüglich ihrer Verwendbarkeit flexibel systematisieren. Dabei bezeichne E^- nachfolgend eine Teilmenge der Kantenmenge E, durch deren Entfernen der Graph $G(V, E \setminus E^-)$ ein Baum wird[14].

[13] An dieser Stelle sei der interessierte Leser auf die offensichtlichen Verbindungen der k-Bäume zu den Simplizialkomplexen der Topologie hingewiesen.

[14] Dabei vernachlässigt man im Falle von gerichteten Graphen deren Orientierung, da sie bei einer Visualisierungsentscheidung, wenn überhaupt, nur eine untergeordnete Rolle spielt.

Die Kantenmenge des Baumes $E \setminus E^-$ wird desweiteren auch als E^+ bezeichnet, so daß $E^+ \cup E^- = E$ gilt.

- Eine **n-Baumähnlichkeit** liegt vor, wenn die Kardinalzahl der zu entfernenden Kantenmenge E^- einen gegebenen Schwellwert n nicht überschreitet. Da für Bäume $|E| = |V| - 1$ gilt [14], läßt sich die n-Baumähnlichkeit eines schlichten, zusammenhängenden Graphen leicht berechnen: $n \geq |E| - (|V| - 1)$.

- Ein Graph ist **p-baumähnlich**, wenn der Anteil der Baumkanten E^+ einen gegebenen Prozentsatz p nicht unterschreitet. Die Berechnung der p-Baumähnlichkeit ist wiederum trivial: $0 < p \leq |E^+| \div |E|$.

- Die **(p,n)-Baumähnlichkeit** kombiniert diese beiden Maße derart, daß ein Graph genau dann (p, n)-baumähnlich ist, wenn er p-baumähnlich und n-baumähnlich ist.

In Abhängigkeit von den Analyse- und Visualisierungzielen können auch andere Ähnlichkeitsmaße zum Einsatz kommen: Kreisähnlichkeit, Pfadähnlichkeit, usw.

3.1.2 Hierarchische Clusterverfahren

Ein wichtiger Vorverarbeitungsschritt zur Reduktion der Komplexität der Daten oder zum Aufspüren verdeckter struktureller Abhängigkeiten ist das Clustering. Dabei ist das hierarchische Clustering von besonderem Interesse, da es die spätere Navigation innerhalb der Daten in unterschiedlichen Detailstufen ermöglicht und z.T. Grundvoraussetzung für die im nachfolgenden Abschnitt 3.2 beschriebenen Visualisierungsmethoden ist. Das Ergebnis solch eines hierarchischen Clusterings läßt sich in Form eines Dendrogramms[15] ausgeben, in dem sich sämtliche Inklusionen nachvollziehen lassen:

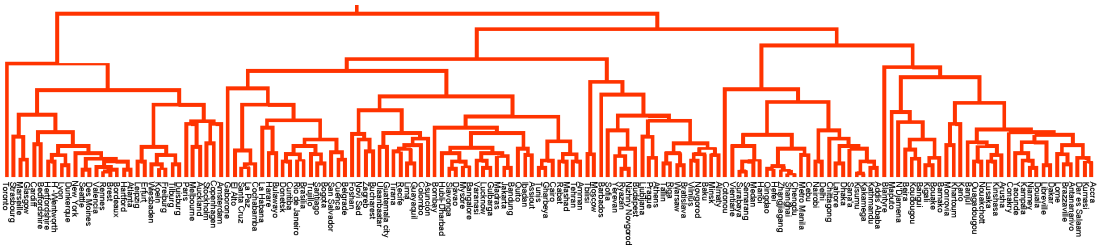


Abbildung 3.2: Beispiel für ein Dendrogramm (Quelle: http://www.chforum.org)

Hierarchische Clustertechniken werden dabei grundsätzlich in die folgenden zwei Klassen unterteilt:

[15]Binärbaum, dessen Blätter den Informationsobjekten und dessen inneren Knoten den gebildeten Cluster entsprechen.

Top-Down-Clustering

Beim **Top-Down-Clustering** geht man anfänglich von einem Supercluster aus, der alle Knoten enthält und dann mit Hilfe eines Distanzmaßes in immer kleinere Cluster zerlegt wird, bis man schließlich die Ebene der einzelnen Knoten erreicht hat. Dabei werden also durch die Unterclusterbildung unähnliche Objekte voneinander separiert. Wichtige Methoden dieser Art sind:

- das **Edge-Betweenness-Centrality-Clustering**[16] [46], bei dem nach und nach die Kanten mit der größten Zwischenzahl (siehe Abschnitt 3.1.1) entfernt werden, bis der Graph in eine vorher festgelegte Anzahl von Zusammenhangskomponenten zerfallen ist. Diese bilden dann die Cluster und das Verfahren wird rekursiv auf sie angewandt, bis alle Kanten entfernt wurden.

- die **Normalized-Cut-Methode** [54], bei der man den Graphen durch minimale Schnitte in Zusammenhangskomponenten zerlegt. Damit dabei immer ungefähr gleich große neue Cluster entstehen, wird ein normalisierter minimaler Schnitt durchgeführt:

  $$ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(B,A)}{assoc(B,V)}$$

  mit $cut(X,Y) = assoc(X,Y) = \sum_{x \in X, y \in Y} w_{xy}$.

- das Clustering nach **distance-k Cliques** [21]. Eine Knotenmenge V' heißt distance-k Clique, wenn der Durchmesser des durch sie induzierten Teilgraphen $diam(G(V'))$ höchstens k beträgt. Man beginnt mit $k = diam(G(V,E))$, also dem gesamten Graphen, und verringert k Schritt um Schritt, wobei man jeweils versucht, maximale distance-k Cliquen zu bilden. Dadurch zerfällt der Graph in Teilcluster, bis man schließlich für $k = 1$ bei den klassischen Cliquen und für $k = 0$ bei den einzelnen Knoten angelangt ist.

- die **k-Core Dekomposition** [5] nutzt eine ähnliche Verallgemeinerung von Cliquen: ein induzierter Teilgraph $G'(V', E')$ heißt k-Core des Graphen $G(V,E)$, wenn der Grad $deg(v')$ jedes Knotens $v' \epsilon V'$ größer oder gleich k ist. Dadurch ergibt sich die in Abbildung 3.3 am Beispiel skizzierte hierarchische Zerlegung des Graphen.

Manche der angegebenen Clusterverfahren, wie z.B. die beiden letztgenannten, liefern allerdings kein Dendrogramm, da diese den Graphen nicht immer binär zerlegen, so daß in einem Clusterschritt auch mehr als zwei Untercluster entstehen können.

[16] Online Demonstration unter: http://jung.sourceforge.net/applet/clusteringdemo.html

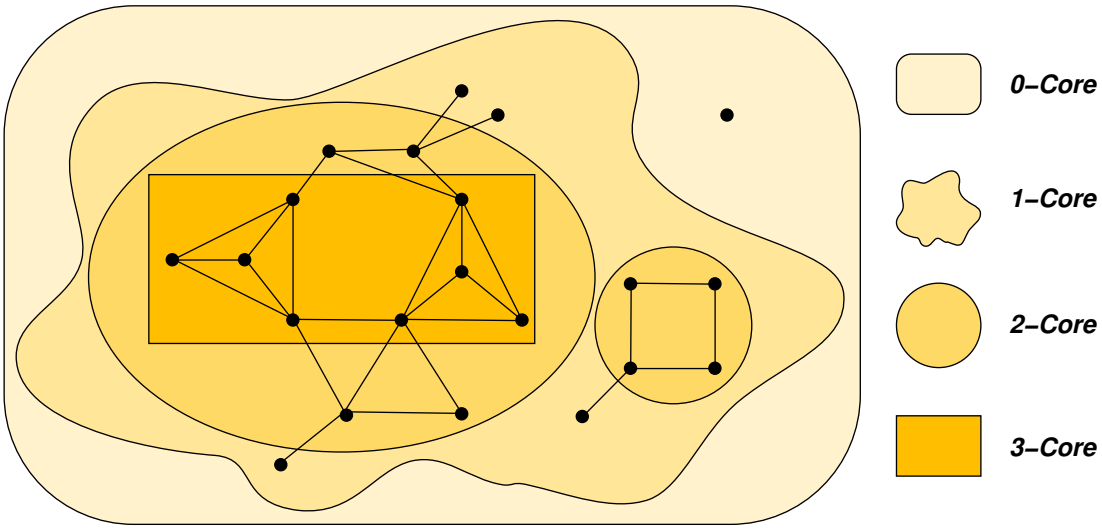


Abbildung 3.3: Beispiel einer k-Core Dekomposition (nach [5]).

Bottom-Up-Clustering

Das **Bottom-Up-Clustering** funktioniert nun in genau der entgegengesetzten Richtung: man beginnt mit den einzelnen Knoten, auf denen ein Ähnlichkeitsmaß definiert sein muß, und aggregiert diese, bis alle Knoten innerhalb eines Superclusters liegen. Dabei unterscheiden sich die Verfahren oft nur in ihrer Definition ähnlicher Cluster:

- Beim **Average-Linkage-Verfahren** sind diejenigen Cluster einander am ähnlichsten und werden agglomeriert, deren über alle Objekte gemittelter Abstand am geringsten ist.

- Das **Single-Linkage-Verfahren** funktioniert ähnlich, nur wird dabei nicht die mittlere Distanz zwischen allen Informationsobjekten zweier Cluster betrachtet, sondern nur die minimale Distanz zwischen je zwei Repräsentanten der Cluster.

- Dagegen werden bei der **Normalized-Association-Methode** [54] immer diejenigen Cluster fusioniert, die am stärksten miteinander verbunden sind, also am meisten verbindende Kanten oder das größte akkumulierte Gewicht solcher Kanten besitzen. Die Normalisierung ist wiederum nötig, um möglichst gleichgroße Cluster zu bilden, also das Verfahren auszubalancieren und Kettenbildung zu vermeiden:

$$nassoc(A,B) = \frac{assoc(A,A)}{assoc(A,V)} + \frac{assoc(B,B)}{assoc(B,V)}$$

ebenfalls mit $assoc(X,Y) = \sum_{x \in X, y \in Y} w_{xy}$.

3.1.3 Pattern Matching

Das Extrahieren von Teilknotenmengen und Teilmengen der Kantenmenge eines Graphen bereitet wenig Probleme. Sequentiell werden alle in Frage kommenden Knoten bzw. Kanten durchgemustert, ob sie einer gegebenen Bedingung genügen; z.B. beim Finden aller Knoten, deren Grad eine gewisse Konstante k überschreitet. Oft sind es jedoch nicht einzelne Datenwerte, die für einen Anwender interessant sind, sondern besondere Muster in den gegenseitigen Abhängigkeiten mehrerer Informationsobjekte. Dazu zählen häufig große Cliquen und ihre vielfältigen Verallgemeinerungen wie die bereits genannten k-Cores und Distance-k Cliques oder andere durch den Benutzer vorgegebene Strukturen. Die algorithmische Komplexität für das Auffinden solcher Strukturen liegt dabei häufig gar im Bereich der NP-Vollständigkeit, so z.B. beim Maximum-Clique-Problem. Um dennoch in akzeptabler Zeit solche Unterstrukturen in großen Graphen aufzuspüren, bedient man sich häufig schneller approximativer Verfahren oder Heuristiken, da deren Genauigkeit für das visuelle Data Mining oftmals bereits vollkommen ausreicht. In [4] wird ein solches randomisiertes Verfahren für große Graphen zur Suche maximaler **Quasicliquen** angegeben. Quasicliquen sind dabei induzierte Teilgraphen, die eine gegebene minimale Dichte (siehe Abschnitt 3.1.1) nicht unterschreiten. So ist eine 1-Quasiclique identisch mit dem klassischen Cliquenbegriff, während bei einer 0.5-Quasiclique durchaus die Hälfte der Kanten fehlen darf. Für das Finden beliebiger durch den Nutzer gegebener Teilgraphen werden die angebotenen Verfahren in zwei Klassen geteilt:

- Beim **exakten Graph Matching** möchte man wirklich die vorgegebene Struktur ohne Abstriche in Form von einigen fehlenden Kanten oder Knoten im Graphen suchen. Dieses Problem ist unter dem Namen Subgraph Isomorphie Problem als NP-vollständig bekannt und benötigt im schlimmsten Fall $O(n^m)$ Vergleichsoperationen, wobei m die Anzahl der Knoten im zu durchsuchenden Graphen und n die Größe der Knotenmenge des zu suchenden Patterns angibt [61]. Lediglich für einige spezielle Graphklassen konnten bisher Polynomialzeitalgorithmen gefunden werden, so z.B. für das Pattern Matching in Bäumen [43, 53].

- Das **inexakte Graph Matching** sucht hingegen einen Teilgraphen, der möglichst ähnlich zu dem vorgegebenen Pattern ist. Es können dabei also durchaus ein paar der induzierten Kanten vernachlässigt werden. Dazu werden meist moderne Ansätze wie die Theorie der neuronalen Netze oder Evolutionsalgorithmen genutzt [7].

Ferner kann es von Interesse sein, ganz generell häufig auftretende Teilgraphen zu finden, ohne diese bereits explizit vorzugeben [37].

3.2 Visualisierungstechniken

Im Visualisierungsumfeld kategorisiert man Graphen üblicherweise in Netzwerke und hierarchische Strukturen, also Bäume. Und für jede dieser beiden Kategorien existieren diverse Visualisierungstechniken, die ein zugrundeliegendes Darstellungsprinzip auf einer abstrakten, globalen Ebene beschreiben. Lokale Entscheidungen, also ob z.B. die Kanten eines Graphen geschwungen sind oder an einem orthogonalen Gitternetz ausgerichtet werden, sind daher zumeist unabhängig von dem verwendeten Visualisierungsprinzip. So geben die Visualisierungsverfahren im Grunde lediglich einen Rahmen vor, in den sich je nach Anwendungskontext und persönlichem Ästhetikempfinden verschiedene konkrete Layout-Algorithmen integrieren lassen. Ein konkretes Layout für Knoten und Kanten soll nach [6] dabei meist gewissen Bedingungen genügen, so z.B.:

(1) Knoten müssen einen gewissen Mindestabstand haben, um unterscheidbar zu bleiben.

(2) Die Anzahl der sich schneidenden Kanten soll minimiert werden, da solche Schnittpunkte häufig als Knoten mißverstanden werden.

(3) Zwei Knoten, die über eine Kante miteinander verbunden sind, sollen nah beieinander platziert werden.

(4) Häufig auftretende Teilgraphen sollen zur einfachen Identifikation immer auf die gleiche Art und Weise layoutet werden.

(5) Die Darstellung soll den gegebenen Platz so gut wie möglich ausnutzen.

Da einige dieser Forderungen kaum zu erfüllen sind oder anderen widersprechen, gibt es viele verschiedene Layout-Algorithmen, die diese und andere Nebenbedingungen unterschiedlich wichten und daher unterschiedliche Resultate liefern. Je nach Bedarf ist also im Einzelfall zu entscheiden, wie wichtig welche Kriterien sind und dann ein geeigneter Algorithmus zu wählen, der dies realisiert. So wurde in Abbildung 3.4 (a) die Planarität des Graphen, also das Vermeiden von kreuzenden Kanten (Bedingung 2), stärker gewichtet, als die Nähe der beiden grauen Knoten (Bedingung 3). Wird diese Priorisierung umgekehrt, erhält man ein anderes Layout (b). In Abbildung 3.5 (a) wurde hingegen die raumfüllenden Anordnung der Knoten (Bedingung 5) höher als die Forderung nach gleichem Layout für glei-

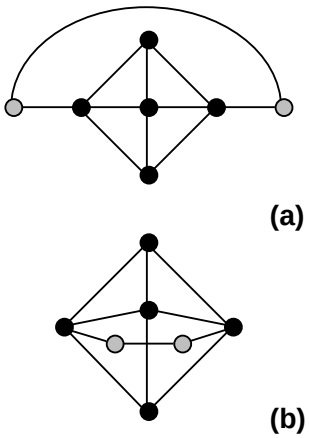


Abbildung 3.4: Layout-Beispiel 1.

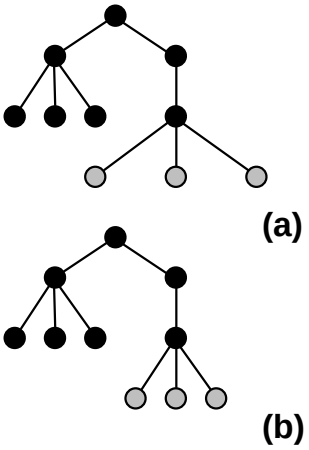


Abbildung 3.5: Layout-Beispiel 2.

che Teilgraphen (Bedingung 4) bewertet. Daher sind die drei grauen Blätter im Gegensatz zu den drei schwarzen so weit wie möglich auseinandergezogen. Wie in (b) zu sehen ist, ändert sich dies jedoch bei einer Umkehr der Gewichtung.
Derartige Einzelheiten werden in der Regel nicht durch die jeweilige Visualisierungstechnik vorbestimmt und können daher in ihrer Umsetzung vom Nutzer frei angepaßt werden. Solche Abänderungsmöglichkeiten werden z.T. an geeigneter Stelle bei der nachfolgend vorgestellten Auswahl von Visualisierungstechniken für Informationsstrukturen mit angegeben. Die vorgenommene Auswahl an Visualisierungsverfahren ist dabei insofern repräsentativ, als daß alle der vier Hauptkategorien für Visualisierungsverfahren durch sie abgedeckt werden:

- **explizite Baumdarstellungen / explizite Netzwerkdarstellungen**: Sie stellen den Graphen in seiner üblichen Repräsentation mit Punkten, Kreisen oder Kugeln als Knoten und verbindenden Linien als Kanten dar.

- **implizite Baumdarstellungen / implizite Netzwerkdarstellungen**: Diese nutzen eine davon abweichende Metapher zur Darstellung von Knoten und Kanten. So können Knoten durch beliebige Objekte (Rechtecke, Kreissegmente,...) repräsentiert werden. Kanten, die im Falle von Hierarchien Vater-Sohn-Beziehungen modellieren, werden dabei z.B. durch Enthaltenseins- oder Nachbarschaftsbeziehungen dargestellt.

Es gilt zu beachten, daß alle Techniken für hierarchische Strukturen immer von einem gegebenen Wurzelknoten ausgehen. Ist dieser nicht vorhanden, kann ein geeigneter Wurzelknoten vom Anwender bestimmt oder über ein Zentralitätsmaß berechnet werden. So sind z.B. die Knoten des im Abschnitt 3.1.1 definierten Zentrums des Graphen gute Kandidaten für die Wahl des Wurzelknotens. Ferner sind hybride Ansätze denkbar, die Elemente von Baum- und Netzwerkdarstellung oder auch von expliziten und impliziten Repräsentationen vereinen. Auf den ersten der beiden Ansätze wird in Abschnitt 3.2.5 näher eingegangen.

Genauso wie die vorgestellten automatischen Verfahren haben die nachfolgenden Visualisierungstechniken Stellvertretercharakter und können nach den Prinzipien der Modularisierung im späteren Framework durch Techniken mit gleichem Funktionsumfang ausgetauscht werden. Dies kann insbesondere dann empfehlenswert sein, wenn die Methoden des Frameworks an ein bestimmtes Anwendungsgebiet angepaßt werden sollen.

3.2.1 Implizite Baumdarstellungen

Tree-Maps

Eine weitverbreitete Visualisierungstechnik für Bäume ist sicher die **Tree-Map** [55]. Solch eine Tree-Map nutzt rekursiv verschachtelte Rechtecke, um Unterbäume darzustellen. Zusätzliche Attribute können z.B. durch Farbe oder Textur der Rechtecke kodiert werden. Durch ihr rechteckiges Layout ist eine Tree-Map in der Lage, den gesamten Bildschirm bzw. die gesamte Druckseite auszufüllen und so effektiv zu nutzen. Sie wurde in den letzten Jahren massiv weiterentwickelt und an die verschiedenen Bedürfnisse einzelner Anwendungen angepaßt. So wurde mit dem Visualisierungswerkzeug für Dateisysteme „StepTree“ [17] der schwedischen Luleå tekniska universitet[18] eine 3-dimensionale Umsetzung dieser Technik vorgenommen [9]. Abbildung 3.6 zeigt beispielsweise das Dateisystem einer LaTeX-Installation mit der Software „StepTree“, wobei die diversen Dateitypen farblich unterschiedlich kodiert sind:

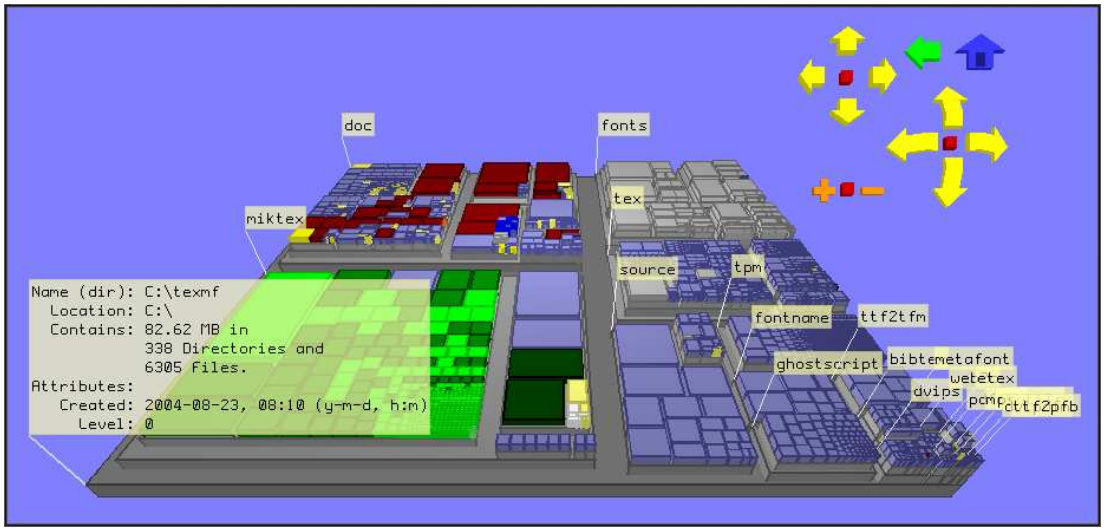


Abbildung 3.6: Beispiel einer Visualisierung mit StepTree.

Eine andere moderne Variante sind die **Cushion Tree-Maps** [66]. Mit ihnen versucht man, die einzelnen Hierarchieebenen durch Schattierung hervorzuheben. Am leichtesten ist das Prinzip an der umseitigen Skizze nachzuvollziehen (aus [66]). Die Cushion Tree-Maps wurden von ihren Entwicklern in das Visualisierungstool „SequoiaView“ [19] umgesetzt, welches ebenfalls Verzeichnisbäume darstellt.

[17] Download unter http://www.sm.luth.se/csee/csn/visualization/files/StepTree_1.6.1.526.zip
[18] http://www.luth.se/
[19] Download unter http://www.win.tue.nl/sequoiaview/

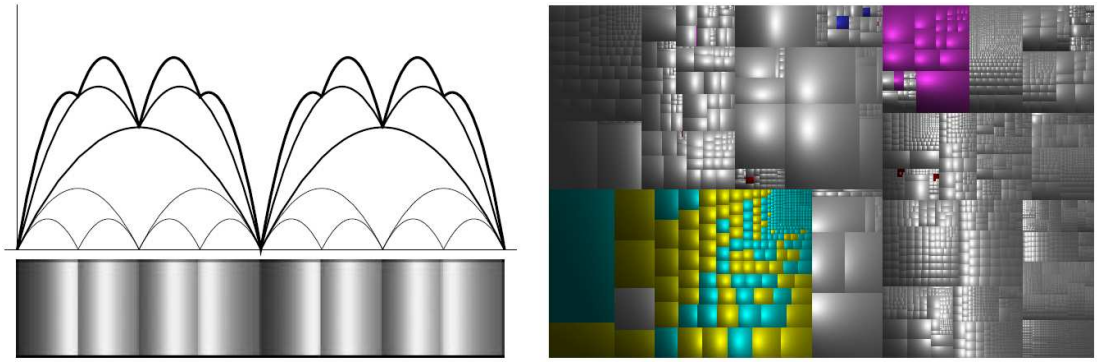


Abbildung 3.7: Schematische Darstellung des Cushion-Prinzips anhand einer binären Intervallzerlegung (links). Visualisierung desselben LaTeX-Dateisystems mit SequoiaView (rechts).

Sunburst

Eine andere Technik, die statt des rechteckigen ein radiales Layout wählt, ist die **Sunburst-Technik** [58]. Sie hat den Vorteil, daß im Gegensatz zu den Tree-Maps die einzelnen Hierarchiestufen nicht übereinandergeschichtet werden und einander dabei verdecken, sondern in konzentrischen Kreisen umeinandergelegt werden. Dabei liegt der Wurzelknoten in der Mitte der Darstellung und die Blätter außen. So läßt sich an den Kreissegmenten die Zuordnung von Vater- und Sohnknoten des zugrundeliegenden Baumes auf einen Blick erfassen.

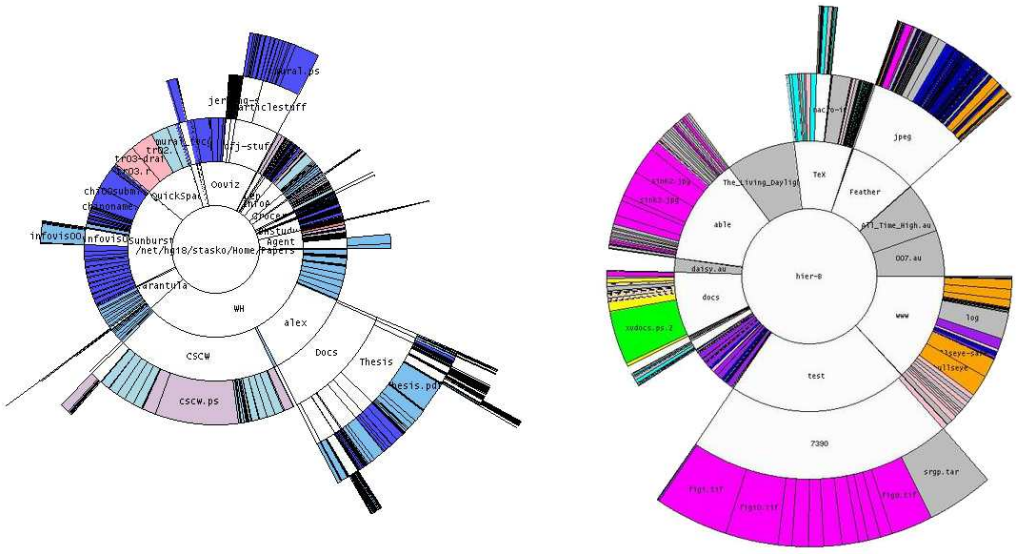


Abbildung 3.8: Sunburst-Darstellung zweier Verzeichnisstrukturen, wobei die Farbe auf der linken Seite das Alter einer Datei und auf der rechten Seite den Dateityp kodiert (Quelle: http://www.cc.gatech.edu/gvu/ii/sunburst/).

Beamtrees

Während beim Tree-Mapping Verschachtelungen genutzt werden, um Vater-Sohn-Beziehungen darzustellen, sind es beim Sunburst gemeinsame Abschnitte des Umfangs eines jeden der konzentrisch angeordneten Kreise. Die **Beamtrees** [28] wiederum machen von gegenseitigen Überlappungen Gebrauch, um solche Beziehungen zu visualisieren. Dabei werden Balken oder Zylinder abwechselnd horizontal und vertikal übereinandergelegt, wobei jede Schicht eine neue Hierarchiestufe repräsentiert. Zusätzlich lassen sich Attribute durch Einfärbung und Variation der Breite der verwendeten Balken bzw. Zylinder in die Darstellung integrieren. Als Beispiel stellen die Entwickler ebenfalls ein Visualisierungstool für Verzeichnisse bereit[20], welches bei Tests mit der LaTeX-Dateistruktur aus den Abbildungen 3.6 und 3.7 allerdings nicht fehlerfrei funktionierte. Die umseitigen Beispiele wurden daher der Originalpublikation [28] entnommen:

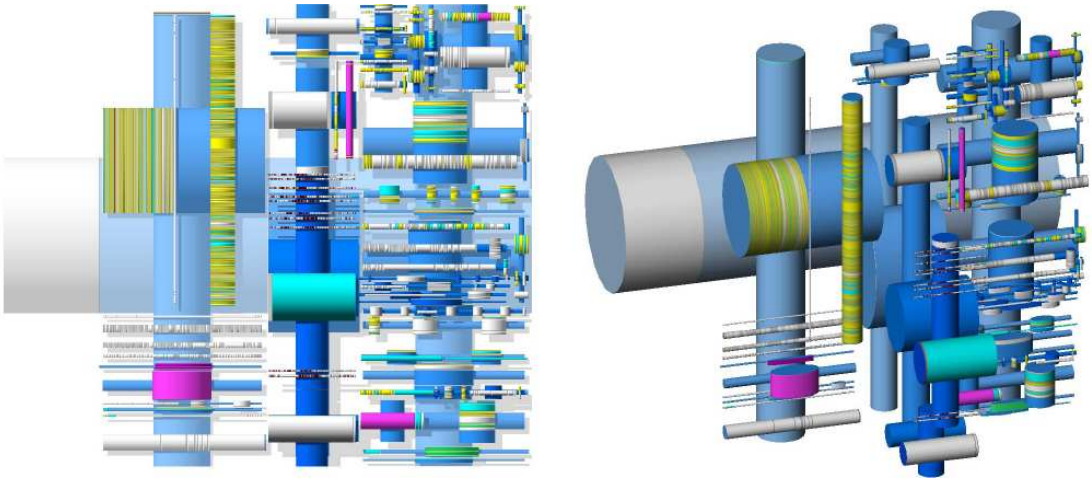


Abbildung 3.9: 2-dimensionale (links) und 3-dimensionale Darstellung (rechts) von ein und derselben Verzeichnisstruktur mit der Beamtree-Technik.

3.2.2 Explizite Baumdarstellungen

Magic Eye View

Im Jahr 1999 ursprünglich für die Visualisierung ausgedehnter genealogischer Daten konzipiert [16], wurde der **Magic Eye View** in den letzten Jahren für andere Anwendungsgebiete und Ausgabemedien angepaßt und weiterentwickelt (z.B. in [62]). Bei dieser Darstellungsmethode wird der Graph auf die Oberfläche einer Halbkugel projiziert, so daß mit zunehmender Entfernung vom Wurzelknoten mehr Platz für die Darstellung der Knoten bereitsteht. Daß dieser trotzdem oft

[20] Download unter http://www.win.tue.nl/~fvham/beamtrees/Downloads/BeamTrees.exe

nicht ausreicht, zeigt das rechte Beispiel in Abbildung 3.10, bei dem sich aufgrund der Datenfülle Baumkanten schneiden. Der Magic Eye View verwendet als konkreten Layout-Algorithmus eine radiale Variante des Walker-Algorithmus‘ [15, 63]. Dieser bietet sich aufgrund seiner linearen Laufzeitkomplexität für eine schnelle Berechnung auch von großen Layouts an, kann aber genauso durch jeden anderen radialen Layout-Algorithmus für Bäume ersetzt werden. Weiterhin integriert die MagicEye-Technik Möglichkeiten zur Rotation der Halbkugel und zur Verlagerung des Fokus‘. Dadurch erhalten die Knoten der fokussierten Baumabschnitte noch mehr Darstellungsplatz auf der Halbkugel und können somit besser voneinander unterschieden werden:

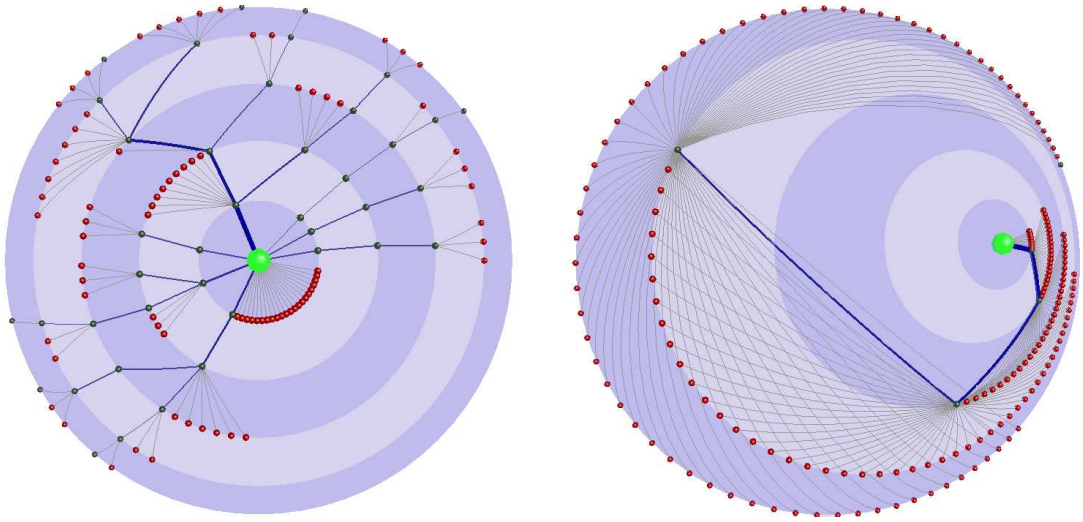


Abbildung 3.10: Darstellung von Clusterhierarchien des EAT-Datensatzes (siehe Abschnitt 6.1) mit Hilfe des Magic Eye Views. Im Vergleich zur Standarddarstellung, bei welcher der Fokus auf dem Wurzelknoten liegt (links), wurde der Fokus in der rechten Abbildung verschoben.

Cone Trees

Die sogenannten **Cone Trees** [50] wurden bereits 1991 bei Xerox PARC entwickelt und ordnen die Sohn-Knoten im 3-dimensionalen Raum jeweils auf dem Umkreis der Grundfläche eines Kegels an, dessen Spitze durch den Vater-Knoten gebildet wird. So hat die jeweils nachfolgende Hierarchiestufe des anzuzeigenden Baumes mehr Platz zur Darstellung ihrer Knoten als die vorhergehende. Aus dieser Konzeption ergibt sich die besondere Eignung des Verfahrens für massiv verzweigte Bäume, also Bäumen, in denen die Vaterknoten eine hohe Zahl an Kindknoten besitzen. Ein Binärbaum wird hingegen den Platz um die Kegelgrundfläche kaum ausnutzen können.

In Abbildung 3.11 wird z.B. eine Klassenhierarchie der 3D- und Multimedia-Klassenbibliothek „Jun4Java“ als Cone Tree mit Hilfe der Visualisierungssoftware FAHAM[21] dargestellt.

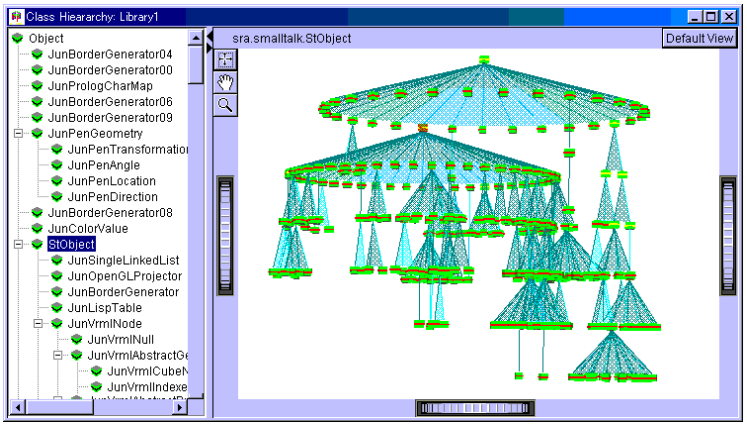


Abbildung 3.11: Darstellung eines ConeTrees mit der Visualisierungssoftware FAHAM.

H3

Ebenfalls bei Xerox PARC wurde 1995 die **Hyperbolic Browser-Technik** [38] entwickelt, die Hierarchien und Bäume auf einer hyperbolischen Fläche anordnet und dann auf eine euklidische Kreisfläche projiziert. Die Übertragung dieser Darstellungsmethode auf den hyperbolischen Raum wurde mit **H3** in [45] vorgestellt.

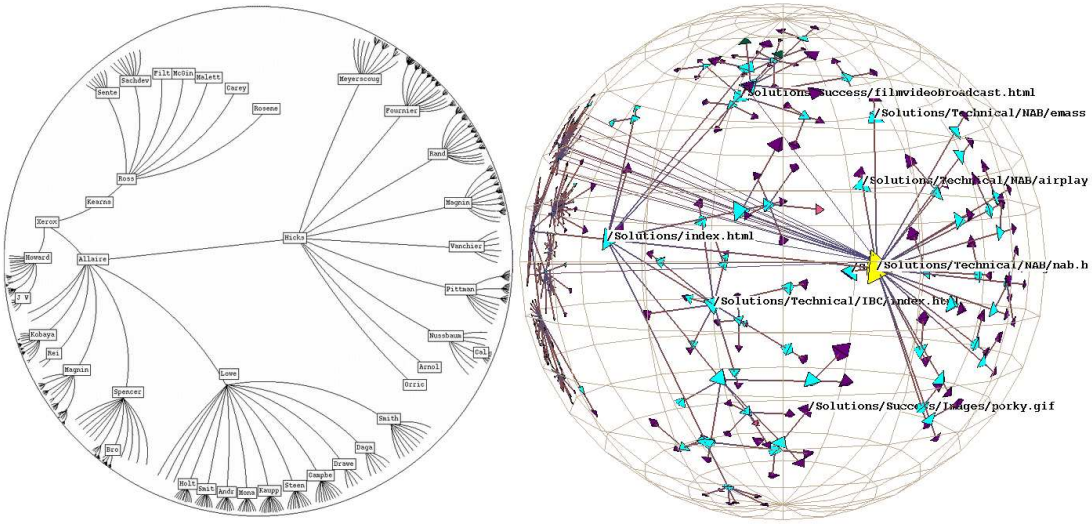


Abbildung 3.12: Beispiele für Hyperbolic Browser [38] (links) und H3 [45] (rechts).

[21] http://www.srainc.com/people/jauhar/Faham/index.html

Die Visualisierungstechnik H3 benutzt im Original das Poincaré-Modell als Grundlage der hyperbolischen Geometrie, legt dieses aber nicht fest. Stattdessen können ebenso das Minkowski- oder das Klein-Beltrami-Modell verwendet werden, was zu jeweils unterschiedlichen Layouts führt.

3.2.3 Implizite Netzwerkdarstellungen

Graph Sketches

Die **Graph Sketches** [2, 3] nutzen eine tabellenartige Abstraktion zur Darstellung von großen Netzwerken. Ihr Anwendungsgebiet sind sehr große Graphen, deren Knotenzahl $k \log(k)$ übersteigt, wobei k der Anzahl der auf dem Ausgabemedium zur Verfügung stehenden Bildpunkte entspricht. Die Graph-Sketches setzen ein hierarchisches Clustering der Struktur voraus, wie es z.B. mit den Methoden aus Abschnitt 3.1.2 erzeugt werden kann.

Die einzelnen Clusterhierarchien stellt man sich übereinanderliegend vor, wobei der den gesamten Graphen umfassende Supercluster zuoberst liegt und die Hierarchiestufen nach unten in den Darstellungsdetails zunehmen. Ganz zu unterst befindet sich dann schließlich der vollständige Graph. Dieses Prinzip wird in Abbildung 3.13 verdeutlicht (nach [10]). Werden also Cluster oder Knoten in der darüberliegenden Hierarchiestufe zu einem neuen Cluster agglomeriert, werden gegebenenfalls auch die Kanten zu anderen Knoten oder Clustern zusammengefaßt. Dies ist im Beispiel mit den Kanten von a nach c und von b nach c geschehen; beide wurden in der nächsthöheren Hierarchieebene zu einer Kante zwischen den Clustern d und e vereinigt. Dieses Prinzip machen sich die Graph Sketches zunutze, indem sie den Zeilen und Spalten einer quadratischen Matrix genau je einen Cluster der aktuell betrachteten Hierarchieebene zuordnen und in jedes Feld der Matrix die Anzahl der agglomerierten Kanten zwischen den Clustern eintragen. Durch eine geeignete farbliche Kodierung läßt sich diese Matrix dann grafisch darstellen. Eine Exploration des Graphen ist durch Auswahl eines Bereiches der Matrix möglich. Die Cluster, deren Kantenbeziehungen in diesem Bereich dargestellt werden, können dann in einer höheren Detailstufe in eine neue Graph-Sketch-Darstellung eingetragen werden. Hat man einen kleineren Teilgraphen erreicht, der sich wieder explizit darstellen läßt (Abbildung 3.14 rechts), kann die Ansicht zu einer anderen Visualisierungstechnik wechseln. Ferner können natürlich auch abweichende strukturelle Maße wie Ähnlichkeits- oder Zentralitätsmaße in die Farbkodierung der auszugebenden Matrix mit einfließen,

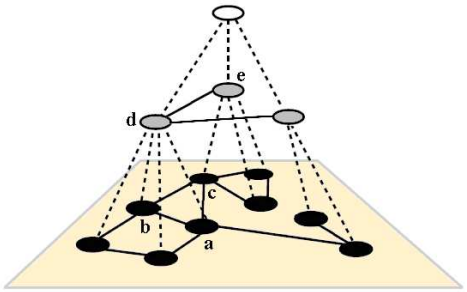


Abbildung 3.13: Clusterhierarchien.

um auch diese mit einem Blick erfaßbar zu machen. Dabei sind neben Farbe und Helligkeit durchaus auch andere Kodierungen denkbar, wobei sich vor allem ikonenbasierte Techniken anbieten (z.B. **Needle Grid** [3]).

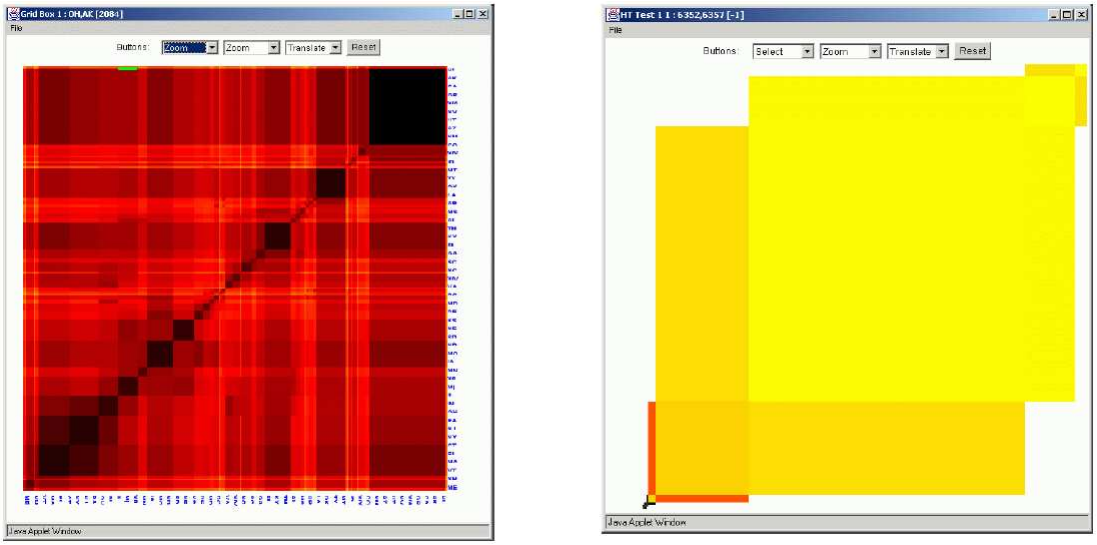


Abbildung 3.14: Graph Sketch mit Überblicksansicht (links) und Detailansicht (rechts) aus [2].

Intervalldarstellungen

Für gewisse Graphen können auch Intervalldarstellungen auf der reellen Zahlenachse gefunden werden. Solche Graphen werden aufgrund dieser Eigenschaft auch als **Intervallgraphen** bezeichnet. Formal wird ein Intervallgraph als Graph $G(V, E)$ definiert, für den es eine Menge reeller Intervalle $\{I_v \mid v \in V\}$ gibt, die einander genau dann überlappen ($I_u \cap I_v \neq \emptyset$), wenn die Kante $uv \in E$ ist. Dabei ist die Intervallrepräsentation nicht eindeutig bestimmt und kann durch den jeweiligen Layout-Algorithmus individuell gestaltet werden. Für verallgemeinerte Intervallgraphen (probe interval graphs, interval filaments, etc.) können angepaßte implizite Darstellungstechniken genutzt werden. Ein Beispiel für einen einfachen Intervallgraphen zeigt die Abbildung 3.15 (nach [60]):

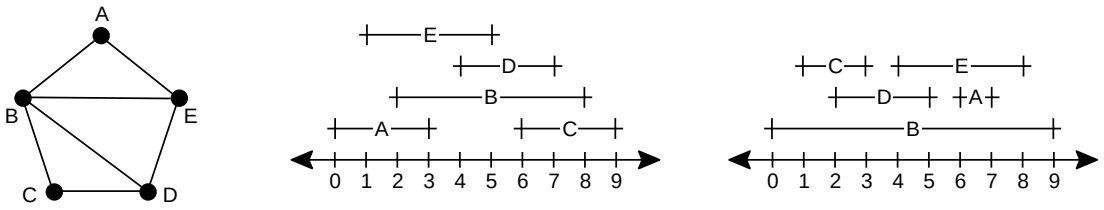


Abbildung 3.15: Zwei Intervalldarstellungen ein und desselben Graphen.

Permutationsdiagramme

Eine andere Klasse von Graphen läßt sich durch sogenannte Permutationsdiagramme darstellen. Solche Diagramme sind spezielle bipartite Graphen, für die gilt, daß jeder Knoten einer Partition mit genau einem Knoten der anderen gleichgroßen Partition durch eine gerade Kante verbunden ist. Werden die beiden Partitionen in der Ebene parallel zueinander angeordnet, schneiden sich die Kanten entsprechend der Abfolge der Knoten. Wie im Beispiel in Abbildung 3.16 nachvollzogen werden kann, entspricht solch ein Schnitt einer gemeinsamen Kante im Ausgangsgraphen, so daß die komplette Struktur in der Permutation der Knotenmenge enthalten ist. Graphen, die solch eine Repräsentation besitzen, werden **Permutationsgraphen** genannt. Wie sich leicht prüfen läßt, gehören z.B. unabhängige Mengen und Cliquen zur Klasse der Permutationsgraphen.

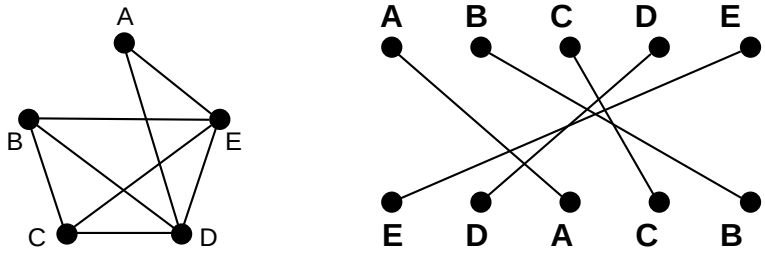


Abbildung 3.16: Beispiel eines Permutationsgraphen und seines Permutationsdiagramms (nach [60]).

3.2.4 Explizite Netzwerkdarstellungen

In diesem Abschnitt werden zwei Verfahren vorgestellt, denen ein sogenanntes Federkraftmodell zugrunde liegt. Bei dieser aus dem Bereich des „graph drawing“ stammenden Technik werden die Kanten zwischen den Knoten als Federn mit gewissen Anziehungskräften interpretiert, so daß benachbarte Knoten nahe beieinander liegen, wie es die eingangs aufgestellte Bedingung 3 fordert. Damit der Graph aber nicht in einem Punkt zusammenfällt, existieren außerdem zwischen allen Knoten Abstoßungskräfte. Nach einer zufällig gewählten initialen Anordnung wird dieses System aus Attraktoren (den Federn mit ihren Anziehungskräften) und Separatoren (den Abstoßungskräften zwischen je zwei Knoten) mehrfach iteriert, bis es einen stabilen Zustand erreicht hat. Solche Modelle bieten sich deshalb an, weil sie im Visualisierungsumfeld sehr gebräuchlich sind. Weitere Techniken können [6] entnommen werden.

Der Fruchterman-Reingold-Algorithmus

Der Fruchterman-Reingold-Algorithmus [25] nutzt ein klassisches Federkraftmodell zur Platzierung der Knoten im Darstellungsraum. Um ein zyklisches Hin- und

Herschwingen des Systems zu vermeiden, wird mit jedem Iterationsschritt nach dem Prinzip des *simulated annealing* eine Temperaturvariable dekrementiert, welche die Schwingbewegungen zunehmend räumlich begrenzt. Nachfolgend wird der Algorithmus für den 2-dimensionalen Präsentationsraum angegeben (nach [25]):

Algorithmus 1 Fruchterman-Reingold-Algorithmus

1: $area \leftarrow W * H$;
▷ W und H entsprechen Breite und Höhe des Anzeigefensters
2: $G(V, E)$;
▷ allen Knoten v wurden zufällige Positionsvektoren $v.pos$ zugewiesen
3: $k \leftarrow \sqrt{area/|V(G)|}$;
4: **function** $f_a(x) \leftarrow$ **begin return** x^2/k **end**;
5: **function** $f_r(x) \leftarrow$ **begin return** k^2/x **end**;

6: **for** $i \leftarrow 1, iterations$ **do**
7: **for all** $v \in V(G)$ **do** ▷ Berechne Abstoßungskräfte
8: $v.disp \leftarrow 0$; ▷ Vektor der Positionsänderung
9: **for all** $u \in V(G)$ **do**
10: **if** $u \neq v$ **then**
11: $\Delta \leftarrow v.pos - u.pos$;
▷ Δ ist der Differenzvektor der Positionen beider Knoten
12: $v.disp \leftarrow v.disp + (\Delta/|\Delta|) * f_r(|\Delta|)$;
13: **end if**
14: **end for**
15: **end for**
16: **for all** $e \in E(G)$ **do** ▷ Berechne Anziehungskräfte
17: $\Delta \leftarrow e.v.pos - e.u.pos$;
▷ Jede Kante e verweist auf ein geordnetes Knotenpaar v und u
18: $e.v.disp \leftarrow e.v.disp - (\Delta/|\Delta|) * f_a(|\Delta|)$;
19: $e.u.disp \leftarrow e.u.disp - (\Delta/|\Delta|) * f_a(|\Delta|)$;
20: **end for**
21: **for all** $v \in V(G)$ **do**
22: $v.pos \leftarrow v.pos + (v.disp/|v.disp|) * min(v.disp, t)$
▷ Beschränke Positionsänderungen auf die Temperatur t
23: $v.pos.x \leftarrow \min(W/2, \max(-W/2, v.pos.x))$;
24: $v.pos.y \leftarrow \min(H/2, \max(-H/2, v.pos.y))$;
▷ Verhindere Positionsänderung außerhalb des Anzeigefensters
25: **end for**
26: $t \leftarrow cool(t)$; ▷ Reduziere die Temperatur t
27: **end for**

Dieser klassische Graph-Drawing-Algorithmus läßt sich leicht für höherdimensionale Darstellungsräume verallgemeinern. Dabei werden sogar häufig Darstellungen im 4- oder 5-dimensionalen Präsentationsraum erzeugt, die dann je nach Bedarf in den darstellbaren 2- oder 3-dimensionalen Raum projiziert werden. Es ist leicht ersichtlich, daß der Algorithmus durch die Berechnung der abstoßenden Kräfte zwischen je zwei Knoten in Zeile 7–15 quadratische Laufzeitkomplexität hat. Auch die Berechnung der Anziehungskräfte kann im schlechtesten Fall ($G(V, E)$ ist eine Clique) quadratische Komplexität haben. Eine Beschleunigung des Algorithmus‘ kann man z.B. dadurch erreichen, daß die initiale Anordnung der Knoten durch eine geeignete Heuristik bereits möglichst nah an dem letztendlichen Layoutergebnis liegt und der Fruchterman-Reingold-Algorithmus nur noch lokale Positionsanpassungen vornimmt. Oder man beschleunigt den Schritt der Berechnung der Anziehungskräfte, indem man ebenfalls per Heuristik eine maximale unabhängige Menge[22] bestimmt und nur noch die Kräfte zwischen Knoten dieser unabhängigen Menge und deren Nachbarn berechnet [26].

Ein hierarchisches Federkraftmodell

Eine Weiterentwicklung klassischer Federkraftmodelle, ist das hierarchische Federkraftmodell, welches für das **Framework DA-TU** entwickelt wurde [20]. Basierend auf einem hierarchischen Clustering wird dabei auf jeder Hierarchie-

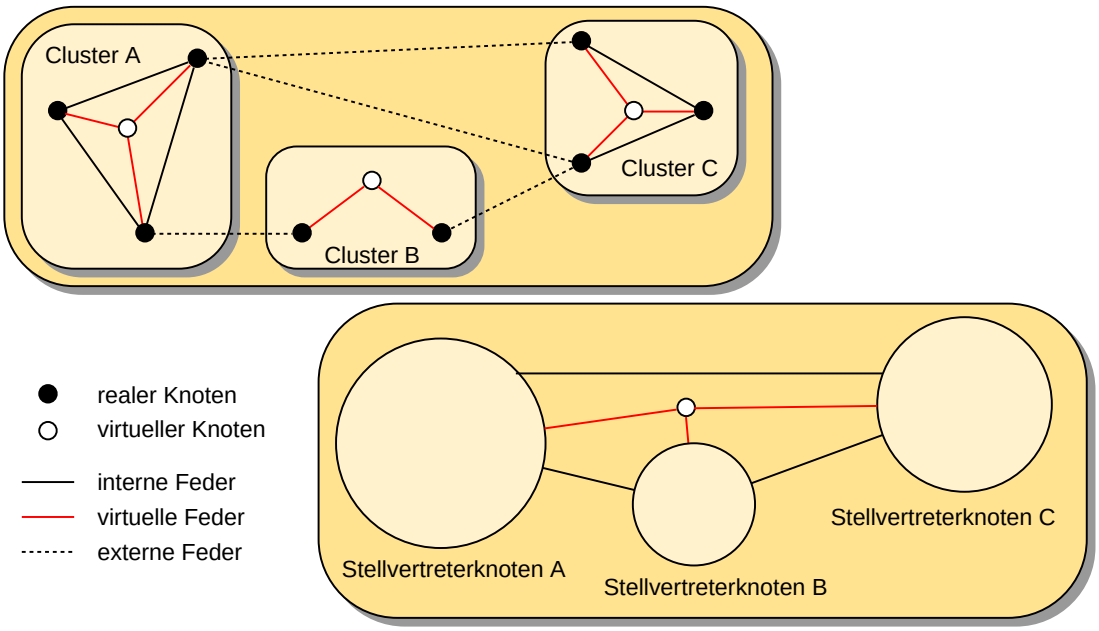


Abbildung 3.17: Schematische Darstellung des hierarchischen Federkraftmodells.

[22] Eine Heuristik oder ein Approximationsalgorithmus ist deshalb nötig, da das **Maximum-Stable-Set-Problem** als NP-vollständig bekannt ist.

stufe ein neuer virtueller Knoten pro vorhandenem Cluster hinzugefügt und mit virtuellen Kanten/Federn mit allen darin befindlichen Knoten verbunden. Die Federkonstanten der internen und der virtuellen Federn werden dabei größer gewählt als die der externen Federn, so daß die Knoten eines Clusters zueinander gezogen werden. In der darüber liegenden Ebene wird ebenso verfahren, nur daß jetzt die realen Knoten dieser Ebene Stellvertreter für die Cluster der darunter liegenden sind. Jeder der drei hellen Cluster aus Abbildung 3.17 schrumpft also zu einem Stellvertreterknoten. Die vormals externen werden jetzt zu internen Federn zwischen diesen Stellvertreterknoten des großen, dunklen Clusters zusammengefaßt, und ein neuer virtueller Knoten und virtuelle Federn werden eingefügt. Die Effektivität dieses Verfahrens hängt dabei jedoch stark von der Güte des hierarchischen Clusterings ab.

3.2.5 Baumähnlichkeitsklassifikation der Verfahren

Die in den letzten Abschnitten nachvollzogene klassische Trennung von Darstellungsverfahren für Hierarchien und Netzwerke ist in der Praxis bereits häufig durch einen flexibleren Umgang mit den Verfahren abgelöst worden. Ein Beispiel dafür zeigt die nachfolgende Abbildung (aus [62]):

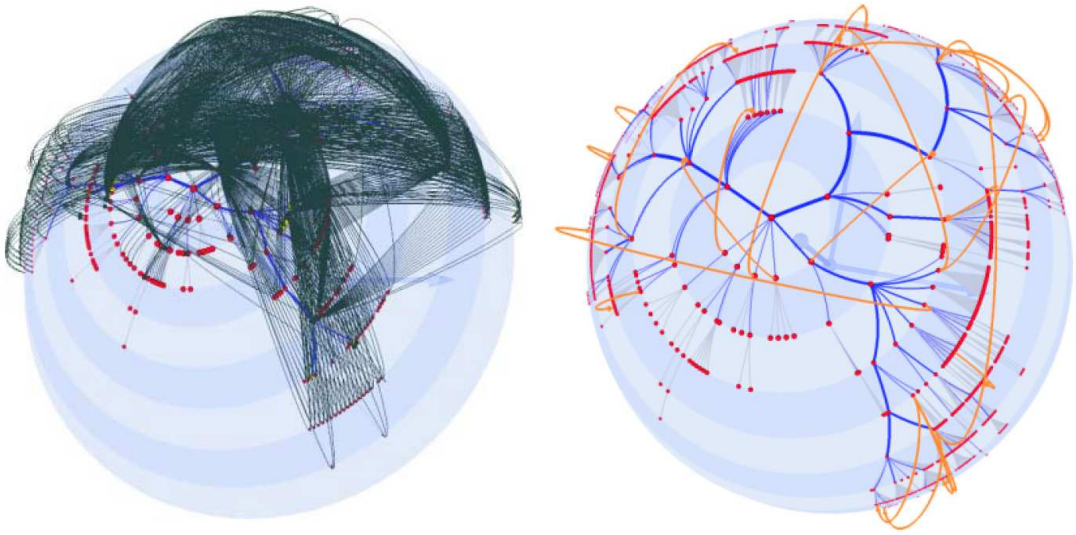


Abbildung 3.18: Darstellung von Netzwerken mit der Baumvisualisierungstechnik Magic Eye View.

In dem abgebildeten Fall wurde die Technik derart erweitert, daß zusätzliche Nicht-Baum-Kanten (*cross edges*) als Bögen über der als Projektionsfläche dienenden Halbkugel eingefügt werden. Solche oder ähnliche Anpassungen lassen sich an nahezu allen Visualisierungstechniken für hierarchische Strukturen vornehmen,

so daß diese eingeschränkt in der Lage sind, auch Netzwerke darzustellen. Daß sich die Darstellungstechniken für Hierarchien dabei nicht uneingeschränkt für Netzwerke eignen, kann in Abbildung 3.18 nachvollzogen werden. So führt das Einfügen vieler Nicht-Baum-Kanten in der linken Darstellung zu einem verwirrenden Layout, bei dem ein Großteil der Strukturinformationen durch die zusätzlichen Bögen verdeckt wird. Wenige, zur besseren Unterscheidung eventuell sogar anders eingefärbte Nicht-Baum-Kanten können jedoch ohne Probleme die zugrundeliegende Baumstruktur ergänzen, wie es die rechte Darstellung verdeutlicht. Je nach Visualisierungsmethode können mehr oder weniger Nicht-Baum-Kanten in solch eine erweiterte Darstellung integriert werden, ohne diese unübersichtlich oder gar unkenntlich zu machen. Mit Hilfe des in Abschnitt 3.1.1 eingeführten (p, n)-Baumähnlichkeitsbegriffs läßt sich die maximale Zahl von Nicht-Baum-Kanten, die eine Visualisierungstechnik in der Lage ist, übersichtlich darzustellen, erstmals parametrisieren. So kann jedem Strukturvisualisierungsverfahren ein Tupel (p, n) zugeordnet werden. Dieses gibt die minimale (p, n)-Baumähnlichkeit an, die ein Graph aufweisen muß, um mit dieser Technik dargestellt werden zu können. Hat eine Technik beispielsweise das (p, n)-Tupel $(100\%, 0)$, eignet sie sich ausschließlich für Bäume, und bereits bei einer zusätzlichen Nicht-Baum-Kante kann sie nicht mehr angewendet werden. Die Parameter p und n stellen also Schwellwerte dar, deren Unterschreiten (im Falle von p) und Überschreiten (im Falle von n) das „Nicht-Geeignet-Sein“ einer Visualisierungstechnik anzeigt. Das linke Beispiel in Abbildung 3.18 verdeutlicht, wie wichtig die Vorgabe sinnvoller p- und n-Werte für die Auswahl geeigneter Visualisierungsverfahren sein kann. Die (p, n)-Tupel der Darstellungsmethoden sind jedoch Erfahrungswerte, die nur empirisch zu ermitteln sind und dabei wiederum von dem jeweiligen Anwendungskontext und Visualisierungsziel [48] abhängen können. Auch die Größe des Ausgabemediums (z.B. Workstation mit Multi-Monitor-System, Tablet-PC, PDA oder Handy) kann die (p, n)-Parametrisierung eines Visualisierungsverfahrens beeinflussen. Bis solche empirischen Werte ermittelt sind, lassen sich die Verfahren informal wie folgt einteilen:

- p sehr groß, n sehr klein: Dies beschreibt die meisten impliziten Darstellungstechniken für Hierarchien. So kann z.B. bereits eine geringe Anzahl an Bögen über einer TreeMap zu einer unübersichtlichen Darstellung führen, die den Mining-Prozeß eher behindert als ihn unterstützt.

- p groß, n klein: In diese Kategorie gehört wohl die Mehrzahl der expliziten Baumdarstellungstechniken. Da sie in ihrer grafischen Repräsentation bereits von Knoten und Kanten Gebrauch machen, läßt sich relativ einfach eine geringe Anzahl an Nicht-Baum-Kanten hinzufügen.

- $p = 0\%, n = \infty$: Dies sind die expliziten und impliziten Visualisierungstechniken für Netzwerke, bei denen der Grad der Baumähnlichkeit keine Rolle spielt.

Weiterhin kann eine zusätzliche Layout-Bedingung zu den fünf anfänglich genannten hinzugefügt werden:

(6) Finde eine Knotenanordnung in der Baumdarstellung, welche die Länge von Nicht-Baum-Kanten minimiert.

Je nach Gewichtung dieser Bedingung kann sich die Zahl der darstellbaren Nicht-Baum-Kanten erhöhen. Denn sollte es gelingen, die Nicht-Baum-Kanten in der Darstellung jeweils auf einen lokalen Teilgraphen zu beschränken und lange *cross edges*, die quer über die gesamte Darstellung laufen, zu vermeiden, können durchaus mehr Nicht-Baum-Kanten dargestellt werden, ohne daß die Übersichtlichkeit darunter leidet.

3.2.6 Klassifikationsmöglichkeiten für Layouts

Da die Visualisierungsverfahren lediglich allgemeine Prinzipien darstellen, die meist noch nichts über das konkrete Layout beinhalten, sollten neben den sechs bereits genannten Bedingungen auch ganz praxisnahe Kriterien in das Layout mit einfließen. So soll die Anordnung nebeneinander liegender Äste eines Baumes oder die Reihenfolge der Cluster in der Matrix eines Graph Sketches nicht nur durch eine optimale Platzausnutzung bestimmt werden. Denn eventuell müssen inhaltlich benachbarte Daten oder gar Daten mit geographischem Bezug auch in der Ausgabe nachbarschaftlich anordnet werden, um deren Zusammenhang zu verdeutlichen und eine Selektion dieser zusammengehörenden Daten zu erleichtern. So sind auch die in den Darstellungsverfahren jeweils verwendeten Layout-Algorithmen je nach Beschaffenheit des darzustellenden Graphen und Analyseziel mal besser und mal schlechter für das Mapping vom Daten- in den Repräsentationsraum geeignet. Beispielsweise sollte für Kreise und kreisähnliche Strukturen eher ein radiales Layout gewählt werden, während hierarchische Organigramme, die z.B. die Struktur einer Firma schematisch darstellen, am besten horizontal repräsentiert werden. Für besonders reguläre Strukturen (Mobilfunknetze, chemische Strukturen, triangulierte Flächen...) eignen sich wiederum Eigenvektorbasierte Layout-Verfahren [33]. Als Basis für solch eine Klassifikation können die sogenannten „*data relationships*“ nach [8] dienen. Diese wurden bereits 1981 von Jacques Bertin aufgestellt und unterteilen Graphen in die folgenden fünf Kategorien:

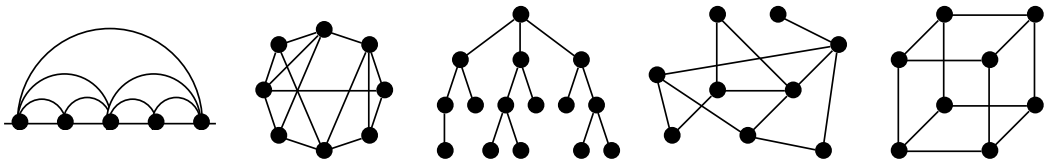


Abbildung 3.19: Die fünf Abhängigkeitstypen der Daten nach [8].

Da Bertin keine formale Definition dieser Kategorien vornimmt, läßt diese Einteilung viel Spielraum für Interpretationen und Ergänzungen. Das Finden geeigneter Ähnlichkeitskriterien, die analog zu den Baumähnlichkeitswerten eine „Verbandsähnlichkeit“, eine „Kreisähnlichkeit“ usw. definieren, stellt einen Gegenstand zukünftiger Forschung dar. Auch die Kategorisierung der Layout-Techniken nach solchen Ähnlichkeitskriterien ist keineswegs trivial und nicht Gegenstand dieser Arbeit. Dennoch sind die *data relationships* in Voraussicht einer praktikablen Lösung bereits in das Konzept des im nachfolgenden Kapitel 4 vorgestellten Data-Mining-Frameworks für große Strukturen mit eingegangen.

3.2.7 Vermeiden des Ausgabe-Engpasses

Viele Visualisierungsverfahren eignen sich lediglich für überschaubare, kleine Graphen, so daß zusätzliche Techniken angewendet werden müssen, um den *screen bottleneck* zu umgehen. Eine Auswahl gängiger Techniken, die sich z.T. leicht in eine bestehende Visualisierungstechnik integrieren lassen, sind:

- **Filterung:** Dabei werden irrelevante Daten komplett ausgeblendet (*information hiding*). Dadurch gehen jedoch Kontextinformationen verloren, so daß die Zusammenhänge mit anderen Teilen des Graphen nicht mehr ersichtlich sind. Um dieses Problem zu lösen, werden die unerwünschten Daten häufig nicht komplett ausgeblendet, sondern in ihrer Darstellung abgeschwächt, so daß sie für den Nutzer nur angedeutet wahrnehmbar sind.

- **Hervorhebung:** Dieses Prinzip hebt den relevanten Teil der Daten hervor (*brushing*). Dies kann z.B. durch Einfärbung in einer ansonsten monochromen Darstellung oder abweichenden Helligkeitswerten (besonders hell in dunklen und besonders dunkel in hellen Darstellungen) geschehen [23].

- **Fokus&Kontext-Techniken:** Sind die relevanten Daten auf einen lokalen Bereich des Darstellungsraums begrenzt, läßt sich dieser wie mit einer Lupe vergrößert darstellen, während der restliche Graph zum Ausgleich des größeren Platzbedarfs in mehreren Stufen verkleinert wird. Ein Beispiel für solch eine Technik zeigt die Abbildung 3.20.

- **Klappen von Unterbäumen:** Diese Spezialform der Filterung blendet gezielt alle mittelbaren und unmittelbaren Sohnknoten eines Baumknotens und die durch sie induzierten Kanten aus.

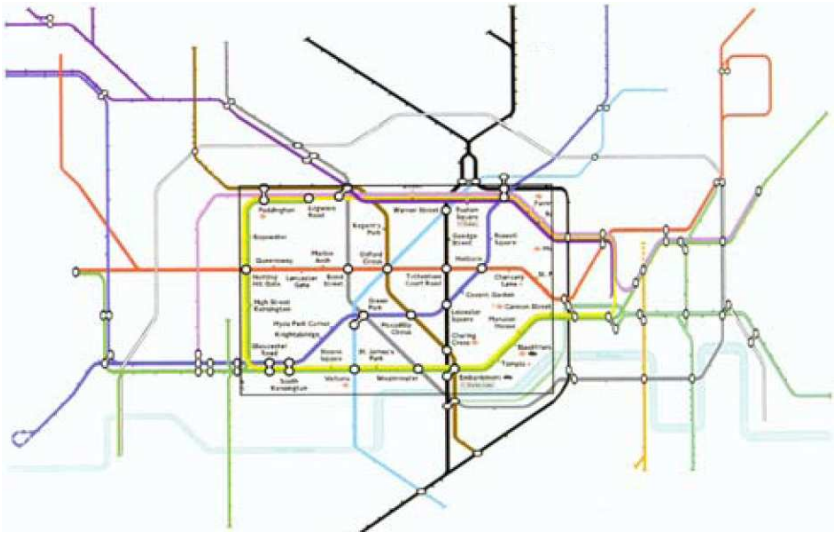


Abbildung 3.20: Londons U-Bahnplan mit dem Stadtzentrum im Fokus (aus [57]).

- **Zusammenfassen durch Clusterung:** Wurden bereits ähnliche Informationsobjekte rechnerisch zu Clustern zusammengefaßt, kann diese Zusammenfassung zu einem Stellvertreterknoten auch in die grafischen Darstellung übernommen werden. Das beschriebene hierarchische Federkraftmodell und die Graph Sketches nutzen diese Technik als Grundlage ihrer Darstellung.
- **Kindsenken-Prinzip:** Hierbei werden die irrelevanten Sohnknoten eines Baumknotens in einen speziell dafür eingerichteten grafischen „Container“ verschoben [10, 11]. Je nach Bedarf kann es auch mehrere solcher Container geben, wodurch eine Gruppierung der versenkten Knoten realisiert werden kann. Dies erleichtert vor allem das gezielte Entfernen von Knoten aus den Senken und deren erneutes Einfügen in die ursprüngliche Baumdarstellung.

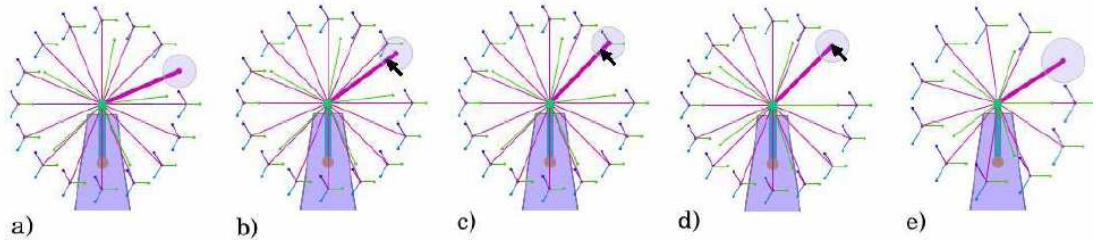


Abbildung 3.21: Beispiel der Kindsenke mit der Visualisierungstechnik RINGS [59], bei dem nach und nach einige Teilbäume versenkt werden (aus [10]).

- **Überblick und Detail:** Dieses Prinzip bietet dem Nutzer gleich zwei Sichten auf die Daten. Eine ist die Überblicksansicht mit niedriger Detailstufe, in der einzelne Teilbereiche selektiert werden können. Dabei verwendet die Übersichtsdarstellung meist eine andere grafische Repräsentation der Struktur als die Detailansicht, welche Teilbereiche mit hoher Detailstufe dargestellt. Ein Beispiel für eine Implementation dieser Technik zeigt die Abbildung 3.22.

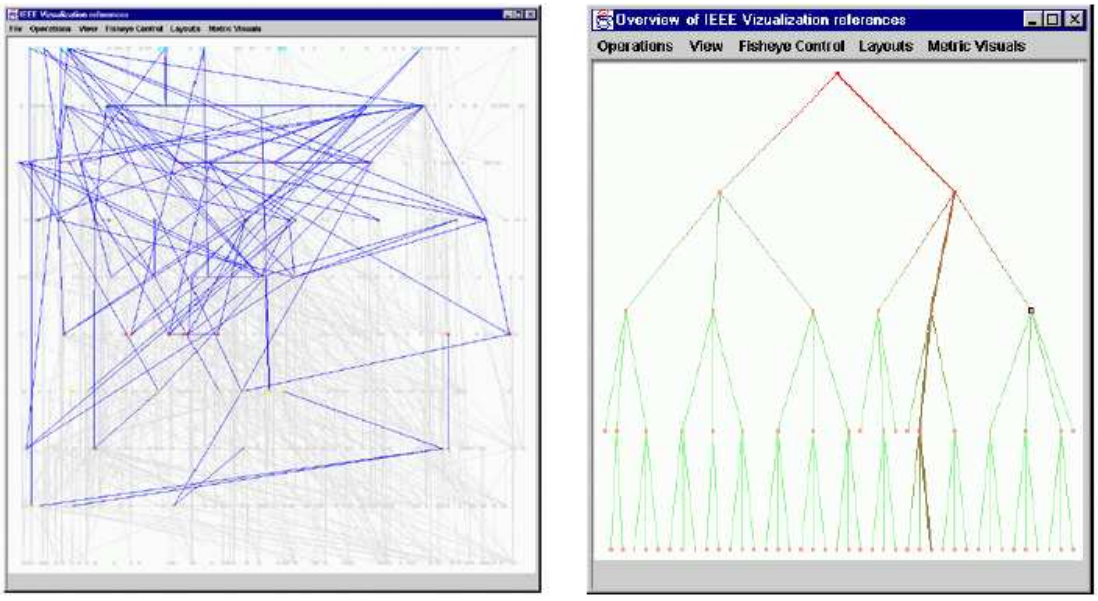


Abbildung 3.22: Interactive Overview Diagram — Detailansicht (links) und Überblicksansicht (rechts) — aus [30].

Die aufgezählten Techniken reduzieren die visuelle Komplexität einer Strukturdarstellung mit Hilfe einfacher Funktionsprinzipien. Durch Kombination einzelner dieser Methoden lassen sich anspruchsvolle Reduktionstechniken realisieren, ohne dafür ausgesprochen komplexe Verfahren benutzen zu müssen. Die Verfahren können dabei sowohl interaktiv durchgeführt werden als auch automatisch ablaufen, wobei dann Kriterien definiert sein müssen, nach denen die Knoten entweder als relevant oder irrelevant eingestuft werden. Gute Relevanzkriterien sind Zentralitätsmaße oder globale Strukturmaße wie die Dichte eines Teilgraphen. Speziell für Bäume eignen sich die in [24] benutzten ***D****egrees* ***o****f* ***I****nterest* (DOI).

3.3 Allgemeine Interaktionstechniken

Ein hohes Maß an Interaktion ist für das visuelle Data Mining von entscheidender Bedeutung. So dürfen gewisse Grundprinzipien einer modernen Nutzerinteraktion nicht fehlen. Beispielhaft seien hier genannt:

- **Navigation** zur explorativen Analyse im Präsentationsraum. Navigation faßt alle zur Erkundung des Darstellungsraumes nötigen Interaktionstechniken zusammen. Diese beeinflussen also den dargestellten Ausschnitt des Datensatzes, indem sie ihn verschieben, mit Hilfe einer gekoppelten Übersichtsdarstellung direkt in einen anderen Ausschnitt wechseln oder einfach wieder zum ursprünglichen Ausschnitt zurückwechseln. Ferner erleichtert eine Übersichtsdarstellung die Orientierung in großen Datenmengen.

- **Annotation**, die nicht nur zur besseren Orientierung bei der Navigation dient, sondern die Darstellung mit zusätzlichen Informationen versieht, um diese verständlicher zu gestalten.

- **Rearrangement** zur manuellen Anpassung des Layouts der automatisch generierten Darstellung an die eigenen Bedürfnisse.

- **Undo-/Redo-Funktionalität**, um Arbeitsschritte bis zu einer beliebigen Stelle rückgängig machen zu können. Diese kann über ein komplexes History-Konzept [36] realisiert werden, welches dann auch als Grundlage der Definition von **Makros** (abgespeicherte Sequenzen von Arbeitsschritten) genutzt werden kann. Diese sind dann jederzeit bei einem ähnlichen Datensatz wieder abrufbar und beschleunigen dadurch den Mining-Prozess.

- **Selektion im Präsentationsraum** zur Markierung von einzelnen Objekten oder sogar Mengen von Objekten.

- **Selektion im Datenraum** zur automatischen Markierung von Objekten aufgrund gewisser Eigenschaften. Dies können zum einen strukturelle Kriterien sein, bei denen in erster Linie Graph-Matching-Verfahren verwendet werden. Zum Anderen kann die Selektion von Informationsobjekten anhand ihrer Attribute vorgenommen werden. Hierfür kommen häufig Operatorgraphen visueller Anfragesprachen zum Einsatz. Diese reichen in ihrer Mächtigkeit von einfachen Vergleichen ($<, \leq, =, \geq, >$) über Boolesche Verknüpfungen solcher atomarer Vergleiche bis hin zu SQL-ähnlichen Dialekten. Dabei können in Abhängigkeit vom Attributtyp zusätzlich komplexe Vergleichsoperatoren eingeführt werden, wie z.B. reguläre Ausdrücke (*regular expressions*) für Zeichenketten. Ein Beispiel für solch eine visuelle Anfrage wird in Abbildung 3.23 gegeben.

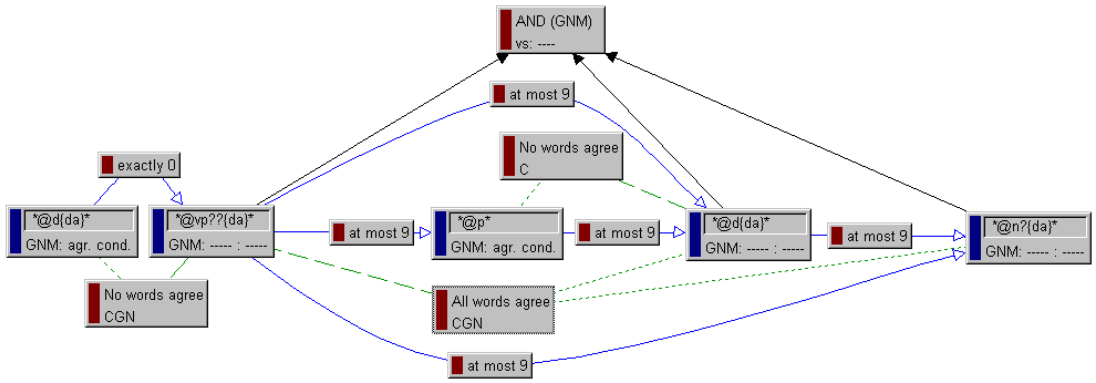


Abbildung 3.23: Beispiel einer visuellen Anfrage mit der Software BibleWorks 4.0.

Die selektierten Informationsobjekte stehen danach für die weitere Verarbeitung zur Verfügung. Zwei wichtige Aktionen, die sich häufig an die Selektion anschließen, sind:

- **Ausschneiden von selektierten Teildatensätzen**, um zu einem späteren Zeitpunkt nur mit diesen weiterzuarbeiten. Dies macht gerade bei einem großen Graphen Sinn, von dem nur ein klar erkennbarer Teilgraph für die weiteren Berechnungen interessant ist. Solch eine Beschränkung des Data Minings auf einen Teilgraphen kann zu enormen Laufzeitverbesserungen z.B. beim Graph Matching führen und damit den gesamten Mining-Prozeß erheblich beschleunigen. Außerdem werden dadurch Visualisierungsmethoden erschlossen, die vorher für den Gesamtgraphen inadäquat gewesen wären.

- **gezieltes Löschen** bestimmter Informationsobjekte zwecks Datenaufbereitung und -bereinigung (*data scrubbing*). Hierbei werden offensichtliche Meßfehler, die leicht als Ausreißer in der grafischen Darstellung zu erkennen und zu selektieren sind, aus der Datenbasis entfernt.

Diese Aktionen sind deshalb von besonderem Interesse, weil sie den iterativen Prozeß des visuellen Data Minings belegen: ein Datensatz wird analysiert und visualisiert, dann basierend auf den Ergebnissen abgeändert und aufs Neue analysiert und visualisiert...

Zur Unterstützung dieses iterativen Prozesses wird im nun folgenden Kapitel ein Framework konzipiert, welches die hier vorgestellten automatischen Methoden und Visualisierungstechniken in den Data Mining-Prozess einordnet und sie über einige der genannten interaktive Techniken zugänglich und konfigurierbar macht.

Kapitel 4

Konzeption eines Frameworks zum Struktur-Mining

Data Mining wird von den Anwendern in den folgenden zwei Szenarien eingesetzt:

- Der Anwender hat noch kein Analyseziel und eventuell sogar keinerlei Vorstellung von den zu analysierenden und darzustellenden Daten. Er nutzt das visuelle Data Mining, um sich einen Überblick über die Daten zu verschaffen und sie explorativ zu erkunden. Eine grafische Darstellung ist gerade beim Mining in Informationsstrukturen eine enorme Hilfe, da sich diese in ihrer Komplexität nur schlecht aus Zahlenkolonnen und Balkendiagrammen erschließen lassen. Aus dieser ungerichteten Suche nach Auffälligkeiten in den Daten (Ausreißern, Clustern, häufig auftretende Teilstrukturen) kann sich durch das visuelle Feedback schnell eine gerichtete Suche entwickeln.

- Weiß der Anwender hingegen schon, wonach er sucht, besteht seine Hauptaufgabe darin, seine Analyseziele so abstrakt wie möglich zu formulieren (Modellbildung), geeignete Analyseverfahren auszuwählen, zu parametrisieren und deren Ergebnisse wieder in den Anwendungskontext zu übertragen. Je besser die Modellbildung gelingt, umso effizienter können die nachfolgenden Analyseschritte durchgeführt werden. Ein zeitraubendes Ausprobieren einzelner Verfahren kann damit zwar nicht vermieden, aber zumindest eingeschränkt werden.

In beiden Fällen benötigt der Anwender Unterstützung durch das verwendete Data Mining-System: eine Auswahl geeigneter automatischer Methoden und Visualisierungstechniken ist dem Anwender vorzuschlagen, passende Interaktionsverfahren sind bereitzustellen. Gerade die Auswahl geeigneter Verfahren hängt dabei jedoch von so vielfältigen Kriterien wie der Beschaffenheit und dem Umfang der Daten, von ihrem Anwendungskontext, vom Visualisierungsziel und nicht zuletzt wiederum vom ganz persönlichen Ästhetikempfinden des Anwenders ab. Durch diese Vielzahl kann es vorkommen, daß der Anwender

durch Fehleinschätzung der Daten oder Unkenntnis der verwendeten Verfahren Fehlentscheidungen bei der Wahl der zu verwendenden Methoden trifft. So ist ein häufiges Wechseln der verwendeten Techniken und ein ständiges Nachjustieren ihrer Parametrisierungen für das visuelle Data Mining charakteristisch. Erst wenn ein aussagekräftiges Analyseergebnis vorliegt, bricht der Mining-Prozeß ab.

Hilfestellung erhält der Anwender z.B. durch Frameworks, die ihn bei der Wahl geeigneter Verfahren unterstützen und durch den gesamten Mining-Prozess leiten. Solche Frameworks enthalten entweder anwendungsspezifische Erfahrungswerte, um bei der Formulierung eines Analyseziels behilflich zu sein [48], oder sie richten sich nach der Beschaffenheit der Daten, um in diesem Fall sogar gleich passende automatische Methoden vorzuschlagen [47].

Ziel dieser Arbeit ist es, ein an die speziellen Bedürfnisse des visuellen Data Minings in komplexen Strukturen angepaßtes Framework zu entwickeln, welches Verfahren aller Verarbeitungsschritte so flexibel verknüpft, daß der Anwender bei den zu bewältigenden Aufgaben (*Overview, Zoom, Filter, Details on Demand,...*) auf beliebigen Strukturdatensätzen ein Höchstmaß an Unterstützung erfährt. Zwar existieren bereits Frameworks für Informationsvisualisierung und gewisse Mining-Prozesse in den unterschiedlichsten Abstraktionsstufen [27, 31, 34, 39], diese sind aber alle auf das klassische Data Mining in Attributmengen und deren Darstellung ausgerichtet. Abbildung 4.1 zeigt einen Entwurf für solch ein Framework zum visuellen Data Mining auf Attributwerten. Dabei wurden bereits einige in Frage kommende konkrete Methoden den einzelnen Funktionsblöcken exemplarisch zugeordnet:

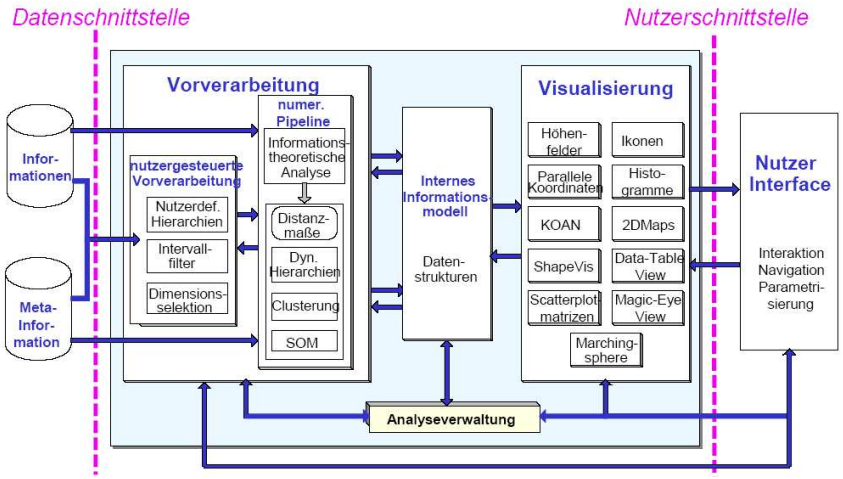


Abbildung 4.1: Architekturvorschlag für ein Framework zum visuellen Data Mining (Kreuseler, 2000) — aus: http://wwwicg.informatik.uni-rostock.de/~schumann/Manuskripte2000/VisDM/VisDM_kap2.pdf.

Da sich die beschriebenen vielfältigen Wechselbeziehungen beim Struktur-Mining jedoch nur mühsam in solche bestehenden Frameworks integrieren lassen, wurde ein speziell an das visuelle Data Mining großer Informationsstrukturen angepaßtes Modell entwickelt, welches in Abbildung 4.2 schematisch dargestellt ist. Die genannte Forderung nach einem Maximum an Hilfestellung für den Nutzer spiegelt sich in dem entwickelten Modell in einer umfangreichen Integration automatischer Methoden wider. Denn nur eine solide automatische Vorverarbeitung kann im Verlauf des Mining-Prozesses als Basis für eine geeignete Methodenauswahl dienen. So ist es z.B. nicht möglich, eine automatische Vorauswahl geeigneter Visualisierungstechniken nach dem in Abschnitt 3.2.5 vorgestellten (p, n)-Baumähnlichkeitskriterium vorzunehmen, ohne den konkreten (p, n)-Wert für einen Graphen je berechnet zu haben. Genausowenig können Darstellungs- und Filtertechniken, die auf einem hierarchischen Clustering beruhen, verwendet werden, wenn ein solches Clustering nicht vorliegt. Die Visualisierung ordnet sich dabei den Ergebnissen der Vorverarbeitung unter.

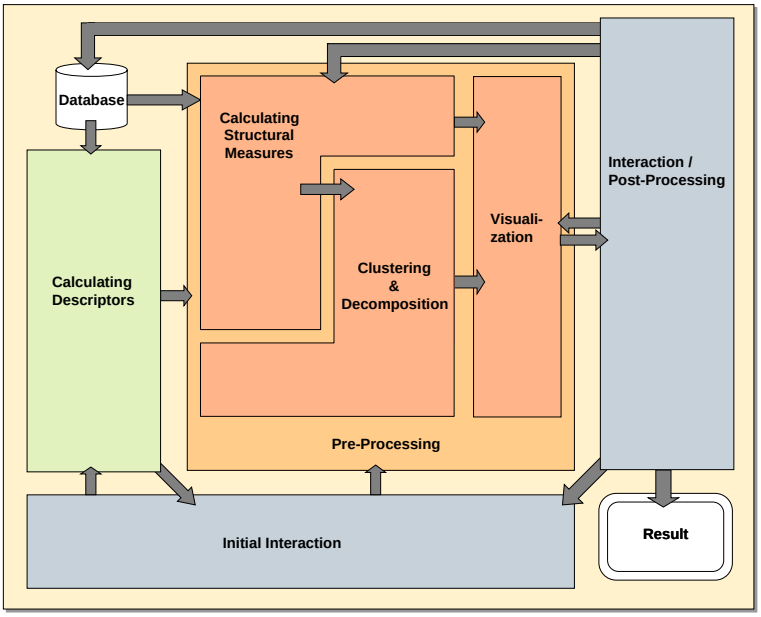


Abbildung 4.2: Modell des visuellen Data Mining Prozesses auf großen Strukturen.

Daß das Struktur-Mining dennoch keine vollautomatische BlackBox-Operation ist, sondern der Vorgaben des Nutzers und zusätzlicher Strukturinformationen bedarf, wird durch die explizite Dreiteilung des Prozesses deutlich:

- **Nutzerinteraktion / VDM Control** (blau unterlegt). Im Vorfeld oder im Verlauf des Minings müssen alle notwendigen Informationen über den Datensatz und die Intentionen des Nutzers erfragt werden. Dies geschieht meist über eine Schnittstelle der graphischen Benutzungsoberfläche und hat

entscheidenden Einfluß auf die Auswahl adäquater Vorverarbeitungs- und Visualisierungsmethoden. Spezielle Frameworks zur Kapselung dieses Prozesses und zur Unterstützung des Anwenders sind hierfür besonders geeignet [47, 48]. Auch die im Anschluss an die Visualisierung möglichen Interaktionstechniken, wie sie in Kapitel 3.3 vorgestellt wurden, zählen hierzu.

- **Automatische Berechnung von Strukturdeskriptoren** (grün hinterlegt). Als **Strukturdeskriptoren** werden nachfolgend alle Metadaten bezeichnet, welche die globale Struktur des Graphen beschreiben. Solch ein globales Metadatum eines Graphen kann z.B. die Information sein, ob dieser gerichtet ist oder nicht. Weitere Beispiele für globale Metadaten und somit für Strukturdeskriptoren sind die Dichte eines Graphen und sein durchschnittlicher Knotengrad. Die in der Vorverarbeitung lokal für jeden einzelnen Knoten und jede einzelne Kante gewonnenen Metadaten wie z.B. gewisse Zentralitätsmaße sind jedoch keine Strukturdeskriptoren. Diese Deskriptoren haben neben den interaktiv vom Nutzer festgelegten Vorgaben ebenfalls maßgeblichen Einfluß auf den Vorverarbeitungs- und Visualisierungsprozeß, da z.B. viele Vorverarbeitungsverfahren, wie in Abschnitt 2.2.1 erwähnt, nur auf Graphen mit geringer Dichte effizient sind.

- **Vorverarbeitung und Visualisierung der Daten** (rot hinterlegt). Dieser dritte Teil des entwickelten Frameworks bildet sozusagen den algorithmischen Kern des visuellen Data Minings und wird wiederum in drei Unterbereiche gegliedert, in denen die Strukturmaße berechnet, Clustering bzw. Dekomposition durchgeführt und schließlich ein geeignetes Mapping der Daten in den Präsentationsraum bestimmt werden.

Der nachfolgende Entwurf eines allgemeinen Frameworks zum visuellen Data Mining komplexer Strukturen (Abbildung 4.3) greift viele der im vorigen Kapitel 3 vorgestellten Verfahren und Konzepte auf und fügt diese, ähnlich der Abbildung 4.1, in die Übersichtsdarstellung aus Abbildung 4.2 ein. Dies verdeutlicht an konkreten Beispielen nochmal die Intentionen, die hinter den einzelnen Funktionsblöcken der Abbildung 4.2 stehen, und gibt zudem einen Eindruck von der Leistungsfähigkeit und Komplexität des vorgestellten Modells. Da das entwickelte Framework einen modularen Aufbau besitzt, können die gezeigten konkreten Verfahren jederzeit entfernt, durch andere Verfahren ersetzt oder durch sie erweitert werden. Dabei sollte man sich aber der Abhängigkeiten zwischen den einzelnen Methoden bewußt sein: ohne eine Methode zur Berechnung der Kantenzwischenzahlen ist beispielsweise auch kein *Edge-Betweenness-Centrality-Clustering* möglich. Auch können bestimmte Vorverarbeitungsschritte (Clustering, Berechnung von Baumähnlichkeitsmaßen) Voraussetzung für nachfolgende Visualisierungs- und Navigationstechniken bilden. Deshalb wurde besonderer Wert auf die modellhafte Abbildung inhaltlicher Zusammenhänge zwischen den Verarbeitungsschritten und innerhalb einzelner Verarbeitungsstufen gelegt.

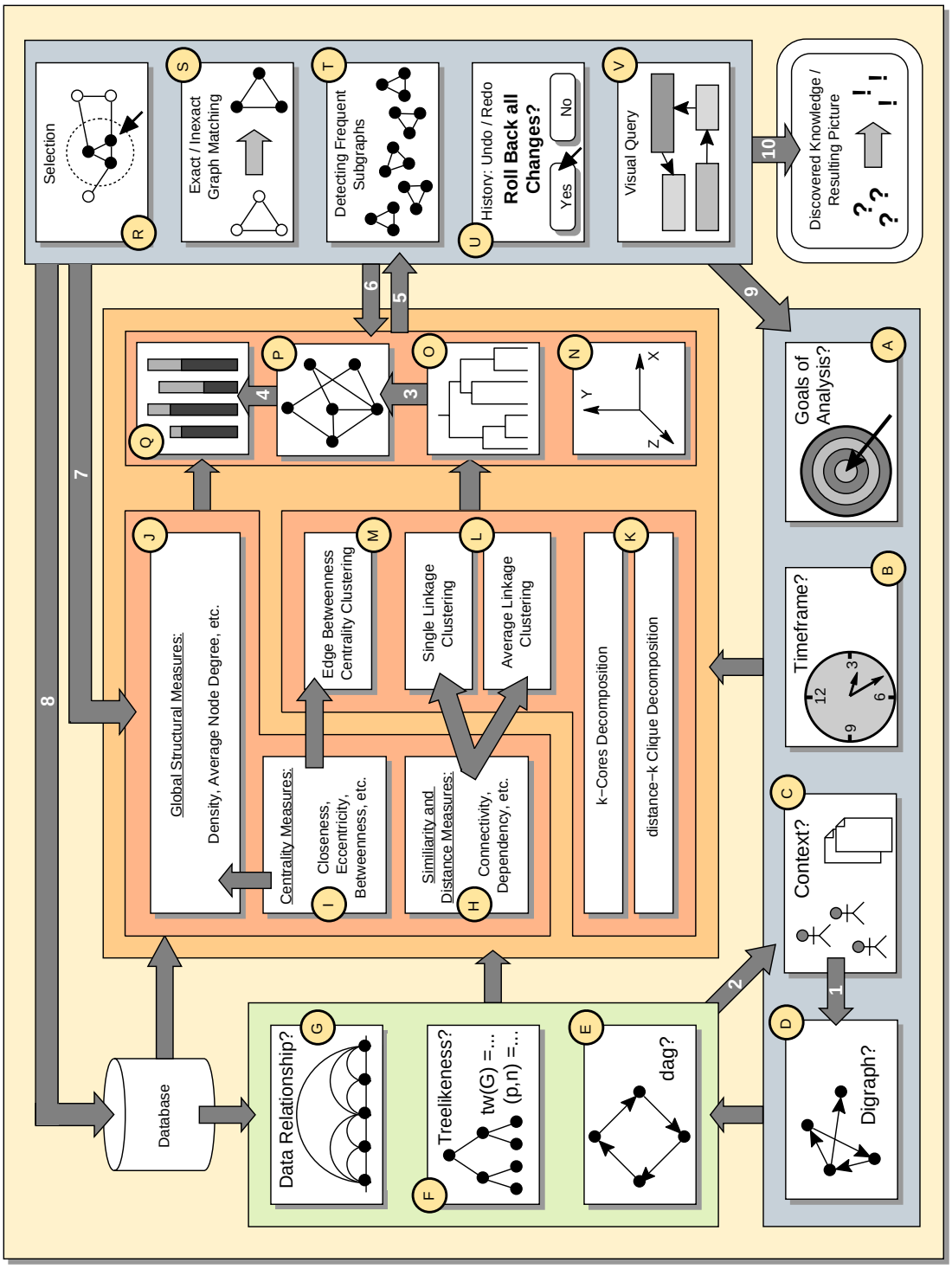


Abbildung 4.3: Framework zum visuellen Data Mining in komplexen Strukturen.

In den nachfolgenden Abschnitten werden die Teile des Frameworks detailliert mit ihren Abhängigkeiten und ausgewählten zugehörigen Verfahren vorgestellt. Die umrandeten Buchstaben auf gelbem Grund beziehen sich dabei auf die jeweiligen Markierungen in Abbildung 4.3. Gleiches gilt für die Angabe von Pfeilnummern.

4.1 Nutzerinteraktion im Vorfeld

Hierbei steuert der Nutzer die Vorverarbeitungsprozesse: entweder durch direkte Auswahl der zu verwendenden Verfahren und Einstellen ihrer Parameter und/oder indirekt durch die Vorgabe gewisser Randbedingungen und Sichtweisen auf die Datenbasis. Im einzelnen können dies sein:

(A) **Die Vorgabe von Nutzerzielen**. Zielt das Data Mining z.B. darauf ab, große Cliquen innerhalb der Informationsstruktur zu finden, wäre es sinnvoll, bereits bei der Vorverarbeitung die distance-k Clique Dekomposition als Clusteralgorithmus auszuwählen. Denn wie in Abschnitt 3.1.2 beschrieben bildet dabei für $k = 1$ jede maximale Clique einen eigenen Cluster und ist somit rasch nach Größe zu filtern und zu identifizieren.

(B) **Die Beschränkung der Rechenzeit**. Viele der vorgestellten Algorithmen haben eine hohe Laufzeitkomplexität. Im Zusammenspiel mit riesigen Datenmengen kann sich dies rasch in inakzeptablen Rechenzeiten niederschlagen. Deshalb ist es wichtig, die zu verwendenden Algorithmen sorgfältig auszuwählen und einen eventuell vorhandenen Genauigkeitsparameter behutsam zu erhöhen.

(C) **Zusätzliche Informationen über den Anwendungskontext**. Diese sind besonders wichtig, da z.T. spezielle Verfahren für bestimmte Anwendungsgebiete existieren. So erfordert das Data Mining in Kommunikationsstrukturen andere Methoden als in chemischen Strukturen.

(D) **Orientierung der Datenstrukturen**. Ist die Struktur der Daten nicht durch eine Adjazenzliste oder Adjazenzmatrix gegeben, sondern z.B. als einfache sequentielle Kantenliste, muß zusätzlich ermittelt werden, ob diese Kanten gerichtet sind. Dann wiederum können Verfahren und Maße wie z.B. die in Abschnitt 3.1.1 genannte Fluß-Zwischenzahl einer Kante, die ausschließlich für gerichtete Graphen definiert sind, angewendet werden. Diese Information kann unter Umständen auch aus dem Anwendungskontext geschlossen werden (Pfeil #1). So sind beispielsweise Linkstrukturen in Hypertexten immer gerichtet, chemische Bindungen hingegen nicht.

Es ist ersichtlich, daß die manuellen Vorgaben des Anwenders für die zielgerichtete Auswahl geeigneter Verarbeitungsmethoden trotz aller Automatisierung von Bedeutung bleiben. Natürlich hat der Anwender die Möglichkeit für häufig wiederkehrende Durchläufe mit ähnlichen Daten (z.B. monatliche Auswertung von

Telekommunikationsverbindungen) die beschriebenen Abläufe per Makro zu automatisieren. Für den Fall, daß die Daten einmal eventuell doch vom spezifizierten Normalfall abweichen, könnte solch einem Makro dann konkrete Schwellwerte enthalten, die angeben, ab welcher Größenordnung der Abweichung eine Nutzerinteraktion erforderlich sein soll.

4.2 Berechnung von Strukturdeskriptoren

Mithilfe der Strukturdeskriptoren werden die Vorgaben des Nutzers durch passende Metadaten ergänzt und erweitert:

(E) **Test auf Kreisfreiheit**. Wurde durch den Nutzer eine Orientierung des Graphen festgelegt, kann der Graph durch den Versuch einer topologischen Sortierung [14] auf Azyklizität überprüft werden.

(F) **Quantifizierung der Baumähnlichkeit**. Bei der Auswahl geeigneter Visualisierungsverfahren sind lokale und globale Baumähnlichkeiten von besonderer Bedeutung (siehe Abschnitt 3.2.5). Mit ihrer Hilfe können dem Nutzer geeignete Darstellungstechniken vorgeschlagen werden, aus denen er dann nach eigenen praktischen oder ästhetischen Gesichtspunkten auswählen kann.

(G) **Bestimmung des Abhängigkeitstyps der Daten** (*Data Relationship*). Dabei wird die Informationsstruktur einer der folgenden fünf relationalen Strukturklassen zugeordnet: lineare Strukturen, kreisförmige Strukturen, Bäume, Netzwerke und Verbände (siehe Abbildung 3.19). Diese wurden bereits erfolgreich in Frameworks zur Informationsvisualisierung integriert [31, 42]. Sie können ebenfalls als Grundlage für eine Visualisierungsentscheidung dienen (siehe Abschnitt 3.2.6).

Die Berechnungsergebnisse dieses Abschnitts beeinflussen nicht nur Vorverarbeitung und Visualisierung, sondern können durch ihr Feedback durchaus auch dem Nutzer dazu dienen, seine Analyseziele zu verfeinern (Pfeil #2). So ist es z.B. denkbar, daß der Nutzer vor der Berechnung des Azyklizitäts-Deskriptors nichts über die Kreisfreiheit der vorliegenden Informationsstruktur wußte. Nun, da sich diese aber herausgestellt hat, könnte er sein Augenmerk zuvorderst auf diejenigen Knoten richten, die z.B. ausschließlich eingehende oder ausgehende Kanten besitzen. Das schlägt sich wiederum in einer geänderten Auswahl der zu verwendenden Verfahren nieder — eine Anpassung des Analyseziels hat stattgefunden.

4.3 Berechnung von Strukturmaßen

In diesem Block, der bereits zum algorithmischen Kern des visuellen Data Minings zählt, wird die Grundlage für ein eventuell nachfolgendes Clustering und für die

spätere Visualisierung gelegt:

H **Berechnung von Ähnlichkeits- bzw. Distanzmaßen**. Diese sind in erster Linie fürs spätere Clustering von Bedeutung, können aber auch in die Bestimmung eines geeigneten grafischen Layouts mit einfließen. So ist es z.B. möglich, ein Federkraftmodell zur Darstellung des Graphen so anzupassen, daß ein Ähnlichkeitsmaß mit in die Federkonstanten einfließt und einander ähnliche Knoten in der Darstellung auf diese Weise näher beieinander liegen.
Diese Maße werden jedoch eher selten verwandt, weil sie quadratischen Speicherplatzbedarf haben. Denn schließlich wird dabei jedem möglichen Knotenpaar solch ein Maß zugewiesen.

I **Berechnung von Zentralitätsmaßen**. Solche können wiederum Voraussetzung für ein Clustering sein oder ins Layout der grafischen Ausgabe mit eingehen. Dabei eignen sich Knoten mit hoher Zentralität gut als Ausgangsknoten für eine Breitensuche zur Konstruktion eines Spannbaumes mit minimaler Höhe. Solch ein Baum wird benötigt, um z.B. einen baumähnlichen Graphen darzustellen: die Kanten des spannenden Baumes verbinden die Knoten in einer Baumdarstellung, die verbleibenden Nicht-Baum-Kanten werden als *cross edges* nachträglich eingefügt.

J **Globale Strukturmaße**. Sie dienen in erster Linie dazu, berechnete Strukturmaße einzelner Knoten und Kanten in Relation zu setzen (siehe **Q**). Ferner können sie ebenfalls als Strukturdeskriptoren verstanden werden und die Auswahl von Visualisierungstechniken unterstützen. So kann bei der Darstellung eines großen Graphen mit geringer Dichte (also wenigen Kanten) anders verfahren werden, als bei einem Graphen mit hoher Dichte.

4.4 Dekomposition und Clustering

Zur Gewinnung einer Hierarchie innerhalb der Daten, die vor allem für spätere Navigationstechniken von Bedeutung ist, wird nun eine Graphzerlegung bestimmt. Dazu eignet sich jedes der in Abschnitt 3.1.2 aufgeführten Verfahren:

K **Graphdekomposition**. Dies faßt alle Methoden zusammen, die eine hierarchische Ordnung auf der Informationsstruktur direkt aus den Eingangsdaten bestimmen. Die ansonsten nötige Rechenzeit für die Vorausberechnung struktureller Maße kann so u.U. eingespart werden.

L **Clustering nach (Un-)Ähnlichkeit**. Basierend auf Ähnlichkeits- und Distanzmaßen wird hierbei eine mögliche Hierarchie innerhalb der Daten bestimmt. Wegen des genannten quadratischen Speicherbedarfs der zugrundeliegenden Maße eignet sich dieses Verfahren nur selten für die Behandlung

großer Graphen. Sinnvoll sind hier Ähnlichkeitsmaße, die sich rasch immer wieder neu berechnen lassen und daher nicht gespeichert werden müssen. So wäre z.B. bei gleichlangen Zeichenketten ihr Hamming-Abstand ein geeignetes Maß, da sich dieser sehr schnell in Linearzeit bestimmen läßt.

(M) **Clustering nach Kantenzentralität**. In einem sehr dichten Graphen, in dem also viele der möglichen Kanten auch vorhanden sind, ergibt sich hierbei ebenfalls ein Speicherproblem. Da die meisten Graphen in der Praxis aber nur eine geringe Dichte aufweisen (*sparse graphs*), läßt sich ein Zentralitätsmaß häufig leicht im Speicher zur Verfügung stellen und ein entsprechendes Clustering daraus errechnen.

4.5 Visualisierung

Bei der Visualisierung können mehrere unterschiedliche funktionale Sichten auf die Daten generiert werden [51]. Die vier in Abbildung 4.3 schematisch dargestellten Ansichten sind:

(N) **Die Überblicksansicht**. Sie ist zur Anzeige des Kontextes einer Teilansicht der Daten gedacht und hilft dabei, diese global einzuordnen. Ferner kann sie die Orientierung des Nutzers durch Einblendung eines Koordinatensystems oder Gitterrasters erleichtern, da diese durch die heute üblichen Vergrößerungs- und Rotationsfunktionen nur allzuschnell verlorengeht.

(O) **Die Navigationsansicht**. Sie ist ein interaktives Medium zum Filtern der unüberschaubaren Datenmenge. So können etwa Teilcluster in einer Dendrogrammdarstellung oder in einem *Graph Sketch* ausgewählt werden. Dabei beeinflußt die aktuelle Auswahl die von ihr abhängige Inhaltsansicht, was im Schema durch Pfeil #3 dargestellt wird. Bemerkenswert ist an dieser Stelle, daß ein automatisches Clustering der Daten keine aussagekräftige Benennung der Cluster vornehmen kann. Zur besseren Orientierung innerhalb des Dendrogramms sollte der Nutzer deshalb die Möglichkeit haben, die Cluster manuell zu benennen.

(P) **Die Inhaltsansicht**. Sie bildet die Hauptansicht und gibt damit das wichtigste visuelle Feedback für den Anwender. In ihr wird der aktuell ausgewählte Bereich der Informationsstruktur angezeigt. Außerdem findet hier ein Teil der Nutzerinteraktion wie Vergrößern und Drehen statt.

(Q) **Die Detailansicht**. Wünscht der Nutzer detaillierte Informationen über ein Informationsobjekt, kann er es in der Inhaltsansicht markieren und damit eine Anzeige in der Detailansicht auslösen (Pfeil #4). Hier werden einzelne Attribute, die entweder Teil der Ausgangsdaten oder nachberechnete

strukturelle Maße sind, dargestellt. Dafür reicht oft ein einfaches Balkendiagramm aus, das sowohl den individuellen Wert und den Mittelwert über alle Informationsobjekte darstellen kann. So sieht der Anwender auf einen Blick, ob ein Wert über- oder unterdurchschnittlich ausfällt. Auch komplexere Detailansichten sind denkbar.

4.6 Nutzerinteraktion und Post-Processing im Darstellungsraum

Die nachfolgenden Verfahren sind eng an die Visualisierung gekoppelt. So bemerkt der Nutzer beispielsweise an einer Darstellung im Inhaltsfenster, daß ein ganz bestimmtes Pattern von entscheidender Bedeutung ist. Also ruft er das Graph-Matching-Modul auf und läßt nach dem vermuteten Pattern suchen. Wird eins gefunden, ändert sich wiederum die Inhaltsansicht, um das Pattern in seiner Nachbarschaft darzustellen. Dieses Zusammenspiel wird im grafischen Schema des Frameworks durch Doppelpfeile dargestellt (Pfeil #5 & Pfeil #6). Auch das nachträgliche Berechnen eines interessierenden Strukturparameters ist hierbei möglich (Pfeil #7). Unter anderem können die folgenden Interaktionstechniken zur Verfügung stehen:

(R) **Selektion**. Der Nutzer kann dabei Teile der Informationsstruktur markieren und sie beispielsweise in einer eigenen Datenbasis ablegen, um diese bei Bedarf später getrennt analysieren zu können (Pfeil #8). Solch eine Selektion eines Teildatensatzes ist auch direkt auf der Datenbasis möglich, jedoch ist oft im Vorhinein gar nicht genau spezifizierbar, welche Daten selektiert werden sollen. Dafür muß der Nutzer erst einmal einen grafischen Überblick erhalten und kann dann dort einen Teildatensatz von Interesse interaktiv zusammenstellen. Dabei sind verschiedenste Möglichkeiten denkbar, die sich in der Anzahl der selektierbaren Objekte und der Dauer einer Selektion unterscheiden. So kann es sinnvoll sein, z.B. zwei Knotenmengen gleichzeitig selektieren zu können (möglicherweise eine mit der rechten und eine mit der linken Maustaste), um anschließend Ähnlichkeitsmaße für diese beiden Selektionen berechnen zu lassen.

(S) **Graph Matching**. Wie im vorhergehenden Beispiel bereits erwähnt, hat der Nutzer hiermit die Möglichkeit, Strukturen zu suchen, die bei der Exploration durch ihre Häufigkeit, ihre zentrale Lage oder andere Charakteristika aufgefallen sind.

(T) **Suche von häufig auftretenden Teilgraphen**. Ist dem Nutzer hingegen kein Teilgraph durch häufiges Auftreten besonders ins Auge gefallen, hat er außerdem die Möglichkeit nach solchen zu suchen.

(U) **History-Funktionen**. Soll im Zuge der Analyse ein früherer Zustand der Daten oder der Ansichten wiederhergestellt werden können, ist solch eine History-Funktionalität zu gewährleisten. Ähnlich einem Operatorgraphen für die visuelle Anfrage ist die History-Funktion häufig in Form eines visuellen Schemas zugänglich.

(V) **Visuelle Anfrage**. Filterungsmöglichkeiten müssen auch außerhalb der Visualisierung bereitgestellt werden. Denn nicht immer befinden sich die interessierenden Informationsobjekte in genau einem Cluster und können daher nicht direkt über die Navigationsansicht ausgewählt werden. Ein Beispiel für einen Operatorgraphen solch einer visuellen Anfrage wurde in Abbildung 3.23 gegeben.

Kommt dieser Kreislauf des ständigen Nachjustierens und Anpassens der grafischen Ausgabe schließlich zu einem Ende, hat der Nutzer die Analyse der Daten nach dem vorgegebenen Analyseziel abgeschlossen. Dann kann das Ergebnis in Form eines exportierten Bildes oder einer Animation aus dem System extrahiert werden (Pfeil #10). Dabei ist anzumerken, daß auch ein negatives Ergebnis als Wissenszuwachs über den analysierten Datensatz betrachtet wird. Hat man z.B. bei der Suche nach großen Cliquen feststellen müssen, daß gar keine existieren, läßt sich dies zwar schlecht in eine grafische Darstellung fassen, ist aber dennoch ein vollwertiges Resultat des visuellen Data Minings. Eventuell sollte der Nutzer dann über die Abänderung seiner Analyseziele nachdenken (Pfeil #9). In der Regel wird eine Relaxation der Analysebedingungen vorgenommen. Dies trifft auch auf den Fall zu, daß die Datenqualität schlechter ist als angenommen. So können Datensätze fehlen oder Duplikate auftreten. Ferner können sogenannte *dangling ends* vorhanden sein, also Kanten, die zwei Knoten verbinden, von denen einer nicht existiert. Im Web-Datensatz (siehe Kapitel 6.2 und Anhang A) traten solche *dangling ends* in Form von *Deadlinks* auf, also Hyperlinks, deren Zieldokument nicht erreichbar war.

Ferner ist eine leistungsstarke grafische Benutzeroberfläche für solch ein komplexes Framework von großer Bedeutung. Immerhin müssen bis zu vier Ansichten gleichzeitig auf dem Bildschirm Platz finden. Dazu kommen dann noch die Darstellungen für die History und Schaltelemente für die Navigation. Solch eine Oberfläche muß einfach bedienbar sein und dabei trotzdem das gesamte Leistungsspektrum der zugrundeliegenden Mining-Algorithmen unterstützen. Dabei ist zu bedenken, daß das Data Mining auf Strukturen sehr viel Platz benötigt, um diese übersichtlich und unterscheidbar darzustellen. Im Besonderen sind hierfür Multi-Monitor-Systeme geeignet, so daß beispielsweise die Inhaltsansicht als Hauptansicht auf einem Bildschirm dargestellt wird, die anderen Ansichten und Schaltflächen auf einem anderen.

Kapitel 5

Realisierung des Frameworks

Um die Eignung einzelner Vorverarbeitungs- und Visualisierungstechniken für unterschiedlichste Datenbestände evaluieren zu können und ihr Zusammenspiel im Rahmen des vorgestellten Frameworks zu beurteilen, ist eine softwaretechnische Umsetzung des Frameworks nötig. Neben dem praktischen Einsatz einer solchen Software können nur mit ihrer Hilfe empirische Maße wie die (p, n)-Baumähnlichkeitsklassifikation einer Technik bestimmt werden.

5.1 Software zur Visualisierung und Analyse großer Graphen

Es existiert bereits eine ganze Reihe an Programmpaketen, die zur grafischen Darstellung und rechnerischen Analyse von Netzwerken entwickelt wurden. Viele von ihnen sind sehr flexibel in der Art der darzustellenden Daten, verarbeiten aber häufig nicht mehr als wenige tausend Knoten. Auch sind viele der benötigten Navigations- und Interaktionstechniken, wenn überhaupt, nur rudimentär vorhanden. Somit eignen sich diese Programme weniger für das visuelle Data Mining, dienten beim Entwurf des Frameworks und seiner Implementierung aber durchaus in manchen Bereichen als Inspiration.

Cyram NetMiner 2.5[23]:
Diese Software, die laut Herstellerangaben der explorativen Datenanalyse dient, hat keinerlei Einschränkungen in Hinblick auf die Knoten- oder Kantenzahl und bietet eine wahre Fülle an automatischen Analysetechniken. Diese reichen von statistischen Verfahren über die auch in dieser Arbeit vorgestellten graphentheoretischen Methoden (Abschnitt 3.1.1) bis hin zu Transformationsoperationen auf der Datenbasis. Letztere ermöglichen z.B. das Normieren von Gewichten, die Erzeugung des Komplementgraphen[24] oder die Berechnung der transitiven Hülle

[23] http://www.netminer.com

[24] Für den **Komplementgraphen** $\overline{G}(V, \overline{E})$ eines Graphen $G(V, E)$ gilt $\overline{E} = \wp_2(V) \setminus E$ [14].

eines DAGs. Jedoch sind Darstellung und Vorverarbeitung nur lose gekoppelt, und die Nutzerinteraktion erfolgt lediglich in einem einzigen Inhaltsfenster. Navigations- und Übersichtsdarstellungen fehlen der Software vollständig. Die Software geht in jedem Fall von einem zu analysierenden Netzwerk aus. Daher fehlen ebenfalls angepaßte Verfahren für Bäume und baumähnliche Strukturen. Dies hätte z.T. gerade bei der grafischen Darstellung zu mehr Übersichtlichkeit führen können. Der Benutzer wird in seiner Wahl der Methoden lediglich durch ein Onlinehilfesystem unterstützt, welches sich aber auf die Erläuterung von Sinn und Zweck der Verfahren, einer Angabe zur Laufzeitkomplexität und einen Verweis auf die Originalpublikation des Verfahrens beschränkt. Ein anleitendes Framework zur Festlegung von Analysezielen oder zeitlichen Grenzen der Berechnung fehlt ebenfalls.

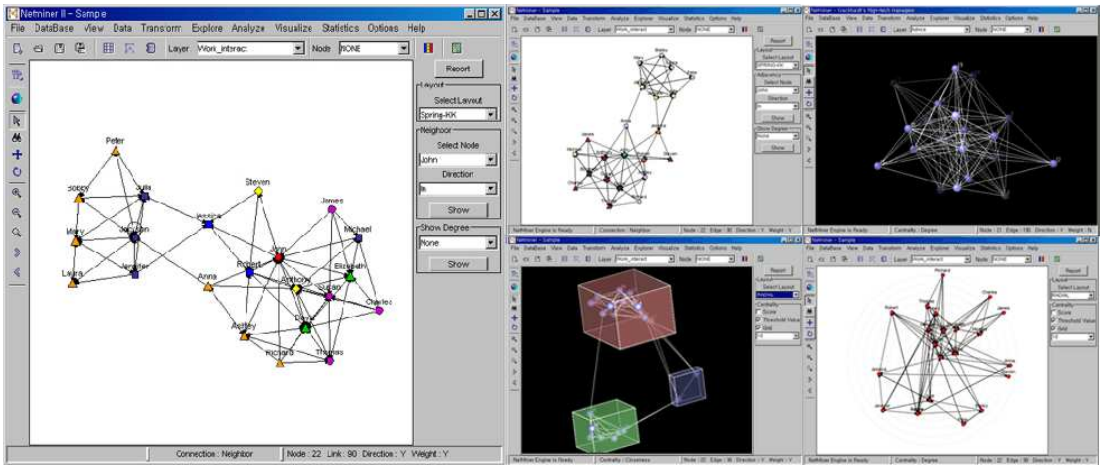


Abbildung 5.1: Beispiele für die grafischen Ausgabemöglichkeiten von Cyram NetMiner 2.5. (Quelle: http://www.netminer.com).

Ucinet 6.0[25]:
Ucinet ist eine lose zusammengefügte Softwaresammlung, bestehend aus dem Analyse-Tool Pajek, der 2D-Graphvisualisierung NetDraw, der 3D-Graphvisualisierung Mage3D und der Ucinet-Oberfläche, die all‘ diese Tools zusammenfaßt und um einige zusätzliche Berechnungsmethoden erweitert. Dieses Softwarepaket unterstützt Graphen mit bis zu 32.767 Knoten, wobei der Hersteller auf seiner Homepage selbst davon abrät, es für mehr als 5.000 Knoten zu verwenden. In seinem Funktionsumfang gleicht es sehr dem vorgestellten NetMiner 2.5 mit allen dort genannten Schwachstellen. Zusätzlich fehlt bei Ucinet eine detaillierte Onlinehilfe.

[25]http://www.analytictech.com/ucinet.htm

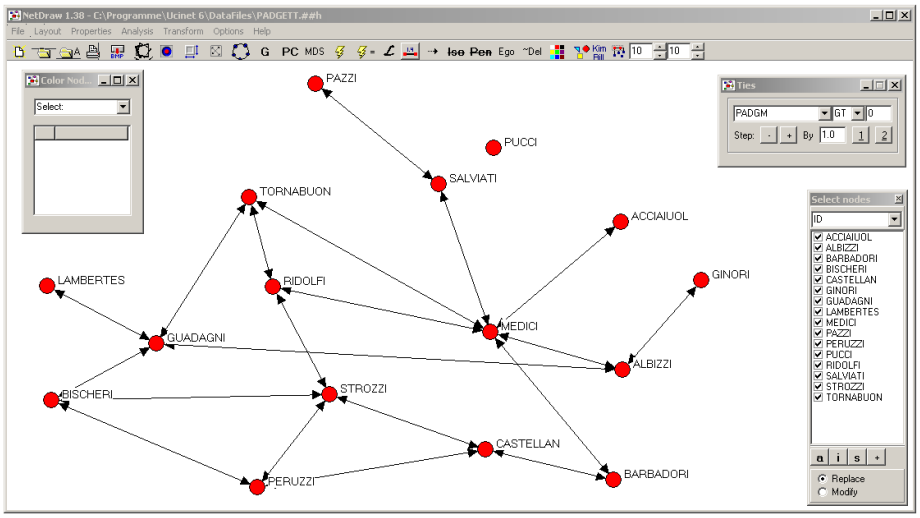


Abbildung 5.2: Beispiel für die grafische Ausgabe von Ucinet 6.0.

Pajek 1.0[26]**:**
Diese Software, die auch dem Ucinet-Paket beiliegt, unterstützt bis zu 9.999.997 Knoten, was in der Praxis normalerweise ausreicht. Ähnlich zu Ucinet bietet Pajek keinerlei weiterführende Hilfe zu den einzelnen Programmfunktionen an. Die Visualisierung ist ganz klar von der Vorverarbeitung getrennt und entspricht damit der sogenannten *Feature Visualization*, d.h. vorausberechnete Besonderheiten (Cluster, Hierarchien, besonders zentrale Knoten, etc.) werden in der grafischen Darstellung gesondert hervorgehoben. Eine interaktive grafische Suche solcher Features ist allerdings nicht möglich und damit auch kein visuelles Data Mining.

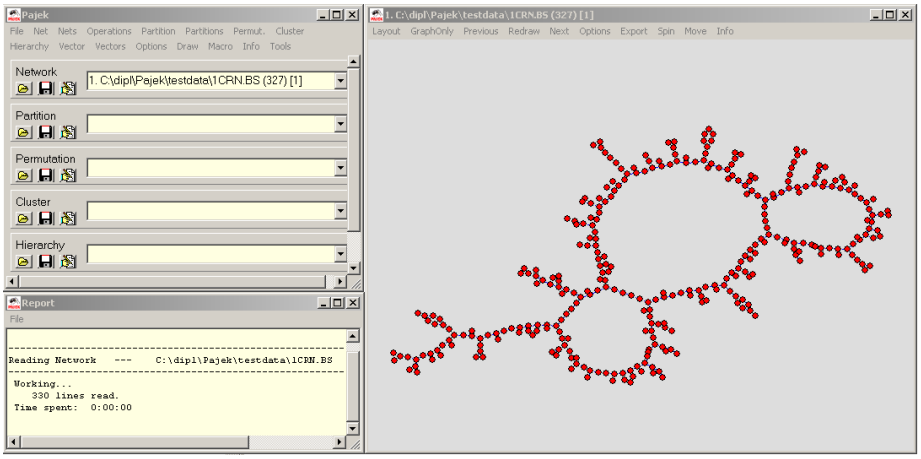


Abbildung 5.3: Beispiel für die grafische Ausgabe von Pajek 1.0.

[26]http://vlado.fmf.uni-lj.si/pub/networks/pajek/default.htm

5.2 Anforderungen an eine eigene Umsetzung

Für die Vorverarbeitung und anschließende Visualisierung großer Informationsstrukturen ist besonders auf Speicher- und Laufzeitkomplexitäten achtzugeben. So sollten die verwendeten Algorithmen zumindest für Graphen geringer Dichte, wie sie nach Abschnitt 2.2.1 in der Praxis zumeist vorkommen, eine subquadratische Laufzeitkomplexität aufweisen. Ebenfalls ist auf eine effiziente Datenhaltung im Arbeitsspeicher zu achten. Diese sollte Duplikate vermeiden sowie benötigte Attribute und zusätzliche Strukturen erst bei Bedarf erzeugen oder von externem Speicher nachladen. Trotzdem muß ein schneller Datenzugriff möglich sein, und aufwendige Berechnungsergebnisse sollten über eine Speicher- und Lademöglichkeit für nachfolgende Kalkulationen oder Darstellungen abrufbar sein. Die Daten sollten dabei je nach Verarbeitungsschritt getrennt abgelegt werden, so daß ein Datensatz aus den folgenden Komponenten besteht:

- der Knotenmenge (Informationsobjekte) und ihren Attributen
- beliebig vielen Kantenmengen (Informationsstrukturen)
- einer variablen Anzahl von Clusterdaten
- verschiedene Layouts

Außer der Knoten- und mindestens einer Kantenmenge müssen aber nicht alle Komponenten vorhanden sein. So können die einzelnen Komponenten ganz nach Wunsch miteinander kombiniert werden. Neben einer plattformunabhängigen Umsetzung ist ferner ein möglichst hohes Maß an Wiederverwendbarkeit der Module und Datenstrukturen beim Entwurf zu gewährleisten.

5.3 Designentscheidungen und Architektur

Für die Umsetzung wurde die imperative Programmiersprache C++ gewählt, da sie gerade im numerischen Bereich besonders performant ist und zusätzlich das Konzept der Objektorientierung integriert, welches eine flexible Wiederverwendbarkeit der Module ermöglicht. Zudem sind C++ -Compiler, die verwendete Bibliothek Qt[27] 3.2.1 und der Grafikstandard OpenGL auf allen wichtigen Plattformen verfügbar. Da die Software enorm große Graphen im Hauptspeicher unterbringen und einen raschen Datenzugriff gewährleisten muß, wurde eine einfache listenbasierte Datenstruktur gewählt. Deshalb konnten auch etablierte Modulbibliotheken für Graphen (LEDA[28], Boost[29]) nicht verwendet werden, da diese vollständig objektbasiert sind, also für jeden Knoten und jede Kante

[27] http://www.trolltech.com/
[28] http://www.algorithmic-solutions.com/leda.htm
[29] http://www.boost.org/

ein neues Objekt anlegen. Diese Art der Datenorganisation benötigt nicht nur mehr Speicher, sondern ist für multiple Kantenmengen, also multirelationale Datensätze, auch gänzlich ungeeignet.

Die Basisdatenstruktur, in welcher der Graph mit seinen Knoten- und Kantenattributen gespeichert wird, wurde in drei Schritten aus der Klasse `QValueVector` abgeleitet, welche von der Qt-Bibliothek zur Verfügung gestellt wird.

5.3.1 Die Klasse `cList`

Die Klasse `cList` erweitert die Qt-Klasse `QValueVector` u.a. um folgende Methoden:

- `quicksort`: Sortiert die Liste mit Hilfe des Quicksort-Algorithmus‘. Die Laufzeitkomplexität beträgt $O(n \log n)$.
- `idx_binsearch`: Sucht in einer sortierten Liste binär nach einem vorgegebenen Wert. Wird dieser gefunden, gibt die Funktion dessen Index zurück. Wurde der Wert nicht gefunden, wird eine negative Zahl zurückgegeben, welche die negierte Position zum Einfügen des Wertes darstellt. ($O(\log n)$)
- `idx_insert`/`idx_remove`: Dient zum Einfügen/Löschen eines Wertes an der durch den Index angegebenen Position in der Liste. ($O(1)$).
- `remove_dupl`: Entfernt in einer sortierten Liste vorhandene Duplikate. ($O(n)$)
- `plus`: Hängt eine zweite Liste an. ($O(m)$, wobei m der Größe der anzufügenden Liste entspricht)
- `s_plus`: Fügt eine sortierte Liste zu einer ebenfalls sortierten Liste hinzu. Das Ergebnis ist wiederum eine sortierte Liste.
- `sd_plus`: Fügt eine sortierte Liste zu einer sortierten Liste ohne Duplikate hinzu. Das Ergebnis ist ebenfalls eine sortierte Liste ohne doppelte Einträge.
- `minus`: Entfernt aus einer sortierten Liste alle Werte einer zweiten sortierten Liste.
- `cutset`: Bildet die Schnittmenge zweier sortierter Listen.

Alle der vier letztgenannten Operationen auf sortierten Listen haben dabei eine Laufzeitkomplexität von $O(\min(n, m \log n))$, wobei n der Elementzahl der größeren und m der der kleineren der beiden Listen entspricht. Sie kommt dadurch zustande, daß für jede dieser Operationen gleich zwei Algorithmen implementiert

wurden, von denen in Abhängigkeit vom Umfang der Listen der jeweils günstigere genutzt wird. Zum einen ist es möglich, beide Listen von Beginn an sequentiell durchzumustern, was zu höchstens $O(n)$ Vergleichsoperationen führt. Ist eine der beiden Listen jedoch so klein, daß $n > m \log n$ gilt, dann ist es effizienter, die Elemente der kleineren Liste per Binärsuche in der größeren zu suchen und je nach Operation dort zu löschen, einzufügen, etc. Dies erfordert $O(m \log n)$ Vergleichsoperationen. In dem unten dargestellten Beispiel würde die Binärsuche also für $m < 100$ gewählt werden, für $m \geq 100$ die sequentielle Verarbeitung.

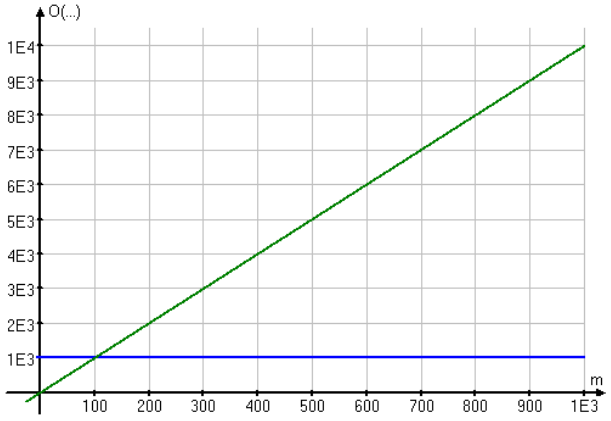


Abbildung 5.4: Funktionsplot der Laufzeitkomplexitäten für $n = 1000$; Breakeven bei $m \approx 100$. (blau – n, grün – $m \log n$).

5.3.2 Die Klasse `cTable`

Diese Klasse erzeugt eine Tabelle, deren Spalten durch die oben beschriebenen Listen gebildet werden:

ID	Flag	Name	Abteilung	Einkommen		Projektbeteiligung		
1	1	Stenzel		1850,-	3	DFG 132	SWING	
2	1	Meier		1300,-	6	ERMUR	G.R.I.P.	DFG 132
3	0	Ahrend		900,-	9	SWING		
4	1	Weise		2100,-	2	ERMUR		
5	0	Struck		1680,-	7	G.R.I.P.	e.LIAS	KO-AG
6	0	Krause		970,-	5	G.R.I.P.	ERMUR	
7	1	Bartsch		1450,-	1	KO-AG	SWING	DFG 132
8	1	v.Linden		2230,-	4	e.LIAS	StAUN	
9	1	Jedicke		1180,-	8	KO-AG		

Abbildung 5.5: Beispiel für eine Instanz der Klasse `cTable`.

Eine Tabellenspalte kann vier unterschiedliche Formen annehmen:

- **nicht presente Spalte**: Um Speicherplatz zu sparen, werden die Spalten zuerst nur als Platzhalter angelegt. Erst wenn die erste Zelle einer Spalte beschrieben wird, legt die Klasse `cTable` die komplette Struktur dieser Spalte an. Die Spalte „Abteilung“ ist solch eine Platzhalterspalte.

- **einfache Spalte**: Diese kann entweder vom Typ STRING, FLOAT oder INTEGER sein. In Abbildung 5.5 ist die Namens-Spalte solch eine einfache Spalte vom Typ STRING.

- **sortierte Spalte**: Jede einfache Spalte kann um eine Sortierung ergänzt werden. Diese wird in einer zweiten, der Spalte zugeordneten Liste abgelegt und enthält die Zeilen-IDs, beginnend bei der kleinsten Zelle bis hin zur größten. Im Beispiel wurde die Spalte „Einkommen“ sortiert, so daß die Zeilen-ID 3, die auf das kleinste Gehalt verweist, am Anfang steht und die Zeilen-ID 8 mit dem größten Gehalt am Ende.

- **Spalten mit Listen als Zelleninhalt**: Für viele Anwendungen reichen einfache Spalten nicht aus, da z.B. unterschiedlich viele Werte in einer Zeile abzulegen sind. Deshalb können Zellen wiederum Listen der Klasse `cList` enthalten, wie es in der Spalte „Projektbeteiligung“ zu sehen ist.

Die Zeilen-IDs sind positive ganze Zahlen, die entweder vom Nutzer oder fortlaufend von der Klasse `cTable` selbst vergeben werden können. Die Liste der Zeilen-IDs ist zu jedem Zeitpunkt sortiert, um einen schnellen Zugriff auf die zugehörige Zeile zu ermöglichen ($O(\log n)$ per Binärsuche). Soll mehrmals hintereinander auf ein und dieselbe Zeile zugegriffen werden, kann anstelle der Zeilen-ID auch der Bezeichner `THE_SAME` benutzt werden. Dies spart eine neuerliche Suche in der Liste der Zeilen-IDs, da deren Ergebnis gespeichert wird und durch die Verwendung von `THE_SAME` jederzeit wieder abrufbar ist. Ferner wurde eine Iterator-Klasse implementiert, die das zeilenweise Durchlaufen der Tabelle ermöglicht. Dies kann wahlweise in auf- oder absteigender Reihenfolge der IDs geschehen. Bei Angabe einer sortierten Spalte wird stattdessen in deren auf- oder absteigender Reihenfolge vorgegangen. Die wichtigsten Methoden der Klasse `cTable` sind:

- `appendCol`, `removeCol`, `appendRow`, `removeRow`: Dienen zum Hinzufügen und Entfernen von Spalten und Zeilen.

- `getItem`, `putItem`: Liest einen Datenwert aus einer Zelle oder beschreibt diese.

- `addItem`: Addiert einen Wert zu einer Zelle hinzu. Bei Zeichenketten wird eine Konkatenation durchgeführt.

- `addCol`: Addiert alle Einträge einer Spalte zu einer anderen vom gleichen Typ hinzu.

- `sortCol`: Wandelt eine einfache in eine sortierte Spalte um.

- `findRows`: Gibt eine Liste mit allen Zeilen-IDs zurück, die einer angegebenen Bedingung entsprechen.

- `findRowsWithin`: Sucht aus einer vorgegebenen Liste mit Zeilen-IDs diejenigen heraus, die einer gegebenen Bedingung genügen. Damit lassen sich die Suchergebnisse von `findRows` weiter verfeinern.

Mit Hilfe der Flag-Spalte können einzelne Zeilen selektiert und andere deselektiert werden. Damit läßt sich bequem konfigurieren, welche Zeilen in eine Berechnung mit eingehen sollen oder auch, welche Zeilen von einem Iterator übersprungen werden sollen. Hier zeigt sich die Überlegenheit der Tabellenstruktur gegenüber einer objektbasierten Graphmodellierung, da sich diese Funktionalität nur sehr mühsam in eine solche integrieren ließe. Zur Manipulation der Flags stehen die folgenden Methoden zur Verfügung:

- `SetFlag`, `SetAllFlags`: Setzt die Werte einzelner oder aller Flags.

- `IDs2Flags`: Setzt alle Flags der in einer übergebenen Liste enthaltenen Zeilen-IDs. Damit lassen sich Suchergebnisse der Methoden `findRows` und `findRowsWithin` leicht in eine entsprechende Selektion der Zeilen umsetzen.

- `Flags2IDs`: Liefert eine Liste der IDs aller selektierter Zeilen.

- `testFlag`: Prüft, ob eine Zeile selektiert ist oder nicht.

Ferner besitzt jede Tabellenspalte jeweils fünf Register der Typen INTEGER, FLOAT und STRING zur Speicherung zusätzlicher Informationen wie etwa Durchschnittswerten oder textuellen Beschreibungen des Inhalts. Auf diese kann über die Methoden `getReg` und `putReg` zugegriffen werden.

Die Geschwindigkeit des Zugriffs auf einzelne Tabelleneinträge ist von wesentlicher Bedeutung für die Performanz des Gesamtsystems. Besonders das Testen einzelner Selektionsflags, welches in allen Methoden durchgeführt wird, gilt es zu beschleunigen. Zu diesem Zweck wurde ein zusätzliches, tabellenweites Flag eingeführt, welches den Status der Flag-Spalte anzeigt. Nur wenn dieses globale Flag gesetzt ist, sind einzelne Zeilen selektiert worden und müssen mit `testFlag` daraufhin geprüft werden. Denn häufig wurde gar keine Selektion vorgenommen, so daß eine Berechnung alle Zeilen als relevant betrachten kann. Auch die Ausführungsgeschwindigkeiten von `putItem`, `getItem` und `addItem` sollten z.B. durch die häufige Verwendung des `THE_SAME`-Bezeichners optimiert werden. In einer späteren Ausbaustufe kann dieses Prinzip zur Minimierung der Zugriffszeiten zu einem frei konfigurierbaren Cache-Konzept weiterentwickelt werden. Dieses würde z.B. über eine Hash-Tabelle schnellen Zugriff auf die n zuletzt benutzten Zeilen ermöglichen. Alle anderen Methoden sind in dieser

Hinsicht weniger kritisch, da sie meist nur wenige Male bei der Initialisierung der Tabelle oder vereinzelt während der Programmausführung benötigt werden.

5.3.3 Die Klasse `cGraph`

Diese Klasse speichert den gesamten Graphen mit all seinen Knoten- und Kantenattributen in Instanzen der Klasse `cTable`. So enthält sie ein Array solcher Tabellen, von denen die erste der Knotenmenge vorbehalten ist, jede weitere enthält eine beliebig große Kantenmenge. Im Framework wurde zwar nur eine Kantenmenge implementiert, aber die Klasse ist somit bereits für die Verarbeitung multirelationaler Daten vorbereitet. Genauso einfach läßt sich die Klasse für die Speicherung von Hypergraphen anpassen. Denn während im implementierten Framework Start- und Zielknoten einer jeden Kante in zwei dafür reservierte Spalten eingetragen werden, könnten Hyperkanten leicht in einer Spalte mit Listen als Zelleninhalte abgelegt werden.

Die Klasse `cGraph` stellt die folgenden Methoden zur Verfügung:

- `addNode`, `removeNode`, `addEdge`, `removeEdge`: Dienen zum Hinzufügen und Löschen von Knoten und Kanten aus dem Graphen.
- `addAttr`, `removeAttr`: Mit ihrer Hilfe lassen sich Spalten zu einer Tabelle (Knotenmenge oder eine der Kantenmengen) zur Aufnahme zusätzlicher Attribute hinzufügen oder entfernen.
- `putAttr`, `getAttr`, `putAttrReg`, `getAttrReg`: Mit diesen Methoden können Attributwerte und Register solch einer Spalte manipuliert oder ausgelesen werden.
- `addAttr`, `addAll`: Addiert einen Attributwert zu einem bereits vorhandenen hinzu oder addiert gleich zwei komplette Attributspalten auf.
- `sortAttr`: Sortiert eine Attributspalte.
- `select`, `deselect`, `isSelected`: Durch diese Methoden können Knoten oder Kanten über die Flag-Spalte ihrer Tabelle de-/selektiert werden.

Neben diesen grundlegenden Funktionen und zusätzlichen Knoten- und Kanteniteratoren bietet die Klasse `cGraph` bereits einige vorgefertigte Methoden für häufig auftretende Graphprobleme:

- `buildAdjList`: Fügt eine neue Spalte zur Knotenmenge hinzu, welche die Adjazenzliste jedes Knotens enthält.
- `degree`: Gibt den Grad eines Knotens, also die Zahl der inzidenten Kanten, zurück.

- `nodesNeighbors`: Liefert eine Liste mit allen Nachbarn eines Knotens oder den Nachbarn einer Gruppe von Knoten.

- `nodesEdges`: Gibt eine Liste der zu einem gegebenen Knoten inzidenten Kanten zurück.

- `inducedEdges`: Gibt eine Liste der induzierten Kanten einer Gruppe von Knoten zurück.

- `findComponents`: Fügt eine Attributspalte zur Knotentabelle hinzu, in welcher die Zugehörigkeit eines Knotens zu einer Zusammenhangskomponente festgehalten ist.

- `shortestPath`: Sucht einen kürzesten Pfad zwischen zwei gegebenen Knoten per Breitensuche und gibt diesen zurück.

- `SSSP`: Berechnet die Lösung des **S**ingle-**S**ource-**S**hortest-**P**ath-Problems für einen gegebenen Knoten und gibt die Ergebnisse in einer neuen Attributspalte der Knotentabelle zurück.

Da diese Klasse auf `cTable` basiert, ist sie an deren Laufzeitverhalten gekoppelt. So ist die Ausführung der Methode `putAttr` höchstens so schnell wie die der Methode `putItem`. Gleiches gilt für die ebenfalls zeitkritischen Methoden `getAttr`, `addAttr` und `isSelected`, da diese ebenfalls ihre äquivalenten Methoden der Klasse `cTable` aufrufen. Von den graphentheoretischen Basismethoden der Klasse `cGraph` ist im Besonderen bei der Methode `inducedEdges` auf eine möglichst rasche und effiziente Abarbeitung zu achten. Denn im Zuge der visuellen Exploration des Graphen wird der Nutzer häufig eine gewisse Teilmenge der Knoten selektieren, um diese z.B. in einer neuen Ansicht darzustellen oder mit automatischen Analysemethoden zu untersuchen. In jedem Fall wird eine manuelle Filterung der Knotenmenge durchgeführt, die jedesmal eine automatische Filterung der Kantenmenge mit Hilfe der Methode `inducedEdges` nach sich zieht. Dies ist nötig, um *Dangling Ends* in der Darstellung zu vermeiden.

Die Laufzeitkomplexitäten können dabei durch die Verwendung unterschiedlicher Algorithmen z.T. stark variieren. So läßt sich der Grad eines Knotens $deg(v)$ in einem schlichten[30], ungerichteten Graphen $G(V, E)$ auf mehrere Arten bestimmen:

1. Ist für den Knoten v eine Adjazenzliste vorhanden, entspricht deren Größe genau seinem Knotengrad. Laufzeitkomplexität: $O(1)$

2. Steht jedoch keine Adjazenzliste zur Verfügung, muß die zu verwendende Kantenliste einmal komplett durchlaufen werden, um jedes Auftreten des

[30]Ein Graph ist **schlicht**, wenn er keine Schlingen oder Multikanten enthält.

Knotens v als Start- oder Endknoten einer Kante zu zählen. Laufzeitkomplexität: $O(m)$ mit $m = |E|$

3. Sind die Spalten der Start- und Endknoten allerdings sortiert, ist das Durchlaufen unnötig, da die entsprechenden Zeilen schnell per Binärsuche gefunden werden können. Laufzeitkomplexität: $O(\log(m))$ mit $m = |E|$

So kann die Klasse nach diesen einfachen Kriterien automatisch den jeweils geeignetsten der implementierten Algorithmen auswählen. Dabei ist zu beachten, daß eine Beschleunigung der Ausführung zumeist mit zusätzlichem Speicherbedarf für Adjazenzlisten oder Sortierungen einhergeht.

5.4 Die Auswahl konkreter Methoden

Basierend auf der beschriebenen Datenstruktur läßt sich nun das konzipierte Framework umsetzen. Es soll in dieser ersten Version mindestens ein Verfahren jeder Hauptkomponente (Vorverarbeitung, Visualisierung und Interaktion&Nachverarbeitung) enthalten. Für die Vorverarbeitung sind dies im Einzelnen:

- die Berechnung des **(p,n)-Baumähnlichkeitswertes** einer Struktur (Abschnitt 3.1.1), da diese einfach zu realisieren und in ihrer Laufzeitkomplexität unproblematisch ist, dafür aber vergleichsweise viel über die Gesamtstruktur des Graphen aussagt (Strukturdeskriptor).

- die automatische Ermittlung der **Abhängigkeitszahlen** für Knoten (Abschnitt 3.1.1), da diese nicht nur ein geeignetes Ähnlichkeitsmaß liefern, sondern gleichzeitig als Berechnungsgrundlage für die Knotenzwischenzahlen dienen können, wie es in Abschnitt 5.5.1 ausgeführt wird.

- die Bestimmung der **Größe der k-Nachbarschaft** eines Knotens (Abschnitt 3.1.1), da diese ein guter Indikator für den Grad der Konnektivität des Knotens, also seiner Anbindung an den Rest des Graphen, ist.

- die Durchführung einer **k-Core-Dekomposition** (Abschnitt 3.1.2), da sie ohne aufwendige Vorverarbeitung zur Berechnung und Speicherung von Ähnlichkeitsmaßen auskommt.

Für jede der drei Ansichten wird je ein Visualisierungsverfahren umgesetzt:

- die Berechnung eines 3-dimensionalen grafischen Layouts mit Hilfe des **Fruchterman-Reingold-Algorithmus‘** (Abschnitt 3.2.4), da dieser prinzipiell für die Darstellung sowohl von Netzwerken, als auch von Bäumen in der **Inhaltsansicht** anwendbar ist.

- die Erzeugung eines **MagicEye-Views** (Abschnitt 3.2.2) für die Baumdarstellung der Clusterhierarchien, da diese Technik bereits die nötigen Interaktionsmethoden einer **Navigationsansicht** integriert und schon erfolgreich zur Visualisierung von Clusterungen eingesetzt wird (siehe [36]).

- die Darstellung von Attributen und Maßen eines Knotens in der **Detailansicht** durch **Balkendiagramme**. Auch wenn für die meisten Maße eine textuelle Ausgabe reichen würde, sind die Zahlenwerte in ihrer grafischen Darstellung häufig leichter zu erfassen.

Aus der großen Fülle von Interaktions- und Nachverarbeitungsmethoden wurden die folgenden für diese erste Version des Frameworks ausgewählt:

- **Abfrage**, ob die Struktur gerichtet ist (Abschnitt 4.1) und Anpassung der grafischen Ausgabe (Ein-/Ausblenden von Pfeilspitzen an den Kanten), da diese Information nicht in den Datensätzen enthalten ist.

- **Zoom und Rotation** der Inhalts- und der Navigationsansicht, da diese bei 3-dimensionalen Darstellungen absolut notwendig sind, um gegebenenfalls weit entfernte oder verdeckte Strukturen zu untersuchen.

- **Annotation** zur manuellen oder, falls möglich, automatischen Beschriftung von Knoten sowohl in der Navigations- als auch in der Inhaltsansicht. Dies ist besonders wichtig, um Clustern in der Navigationsansicht zu benennen und damit identifizierbar zu machen.

- **Selektion** genau eines Knotens zur Anzeige zusätzlicher Informationen in der Detailansicht. Die Selektion ist notwendig, um darauf aufbauende Verfahren wie die Anzeige eines kürzesten Pfades zu realisieren.

- **Filterung** der Daten in der Inhaltsansicht über ihre Attribut- oder Strukturwerte und **Klappen von Unterbäumen** in der Navigationsansicht. Dies ist aufgrund des Ausgabe-Engpasses beim Data Mining in sehr großen Strukturen unerläßlich (Abschnitt 3.2.7).

- **Ermittlung interessierender Teilstrukturen**, in diesem Fall kürzester Pfade. Auch dieses im Grunde einfache Nachverarbeitungsmodul liefert bereits wertvolle Informationen zur Konnektivität zweier Knoten.

- lediglich rudimentäre Implementationen von **Rearrangement** und **Undo-Funktionalität** (Abschnitt 3.3), da diese beiden Techniken zwar sehr nützlich sind, in ihrer Komplexität jedoch eigene Frameworks als Grundlage benötigen. Sie sollen nicht Teil dieser Arbeit sein und wurden nur aufgrund ihrer Wichtigkeit für die Exploration in einer simplen Form integriert.

Wie bereits in Kapitel 4 erwähnt können diese einzelnen Techniken durch den modularen Aufbau des Frameworks jederzeit durch leistungsfähigere Methoden ausgetauscht oder ergänzt werden. Die getroffene Auswahl an Verfahren stellt ein absolutes Minimum an Funktionalität zur Verfügung, mit der sich aber bereits überprüfen läßt, ob das konzipierte Framework flexibel und leistungsfähig genug ist, um tatsächlich den Anforderungen eines Einsatzes in der Praxis zu genügen. In Abbildung 5.6 werden nochmal alle genannten Methoden übersichtlich in ihren Funktionsblöcken dargestellt. Die beiden genannten unvollständig umgesetzten Interaktionstechniken (History und Rearrangement) wurden der Vollständigkeit halber mit in die Darstellung aufgenommen, jedoch nur angedeutet dargestellt:

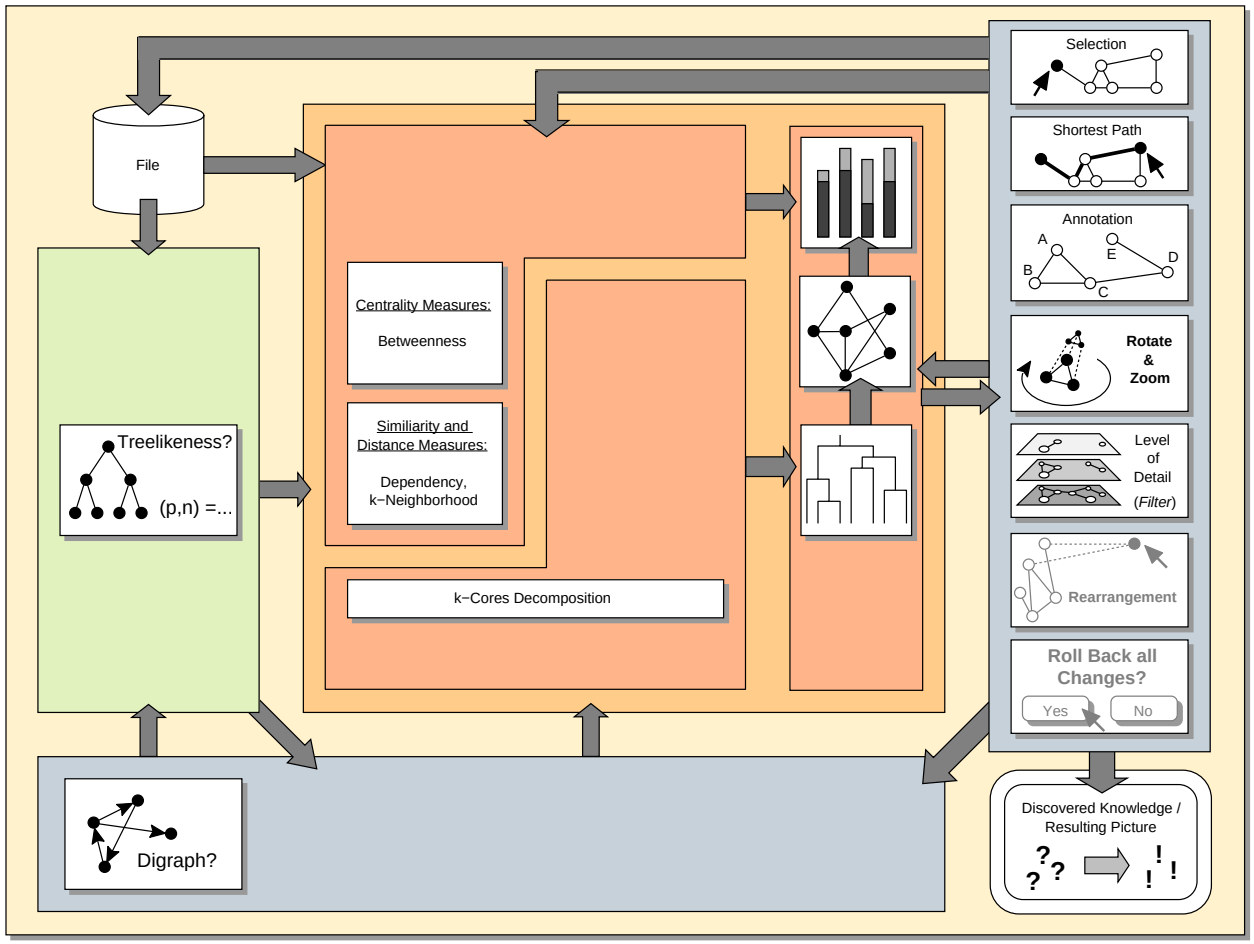


Abbildung 5.6: Die umgesetzten Module im Kontext des Frameworks.

5.5 Die Realisierung der Methoden im Detail

Einige der aufgezählten Methoden lassen sich auf unspektakuläre Art und Weise in ihrer „Lehrbuchimplementation“ umsetzen. Dazu gehören die Suche eines kürzesten Pfades und die Berechnung der Größe der k-Nachbarschaft eines Knotens per Breitensuche (BFS, ***b**readth **f**irst **s**earch*) [14, 60] oder auch das Ermitteln der p- und n-Baumähnlichkeitswerte nach den in Abschnitt 3.1.1 beschriebenen Formeln. Viele andere Verfahren mußten jedoch speziell für die Verwendung innerhalb des Frameworks angepaßt werden. Diese Anpassungen und einige wichtige Implementierungsdetails werden in diesem Abschnitt erläutert.

5.5.1 Die automatischen Methoden

Die k-Core-Dekomposition

Dieses Verfahren, welches zur Bildung eines hierarchischen Clusterings dient, wurde analog zu dem in [5] angegebenen Algorithmus implementiert. Dieser hat eine Laufzeitkomplexität von $O(m)$, wobei $m = |E|$, also der Anzahl der Kanten des Graphen $G(V, E)$ entspricht. Die grundlegende Funktionsweise des Algorithmus‘ wird nachfolgend angegeben:

Algorithmus 2 k-Core-Dekomposition nach [5]

```
1:  for all v ∈ V(G) do
2:      v.deg ← deg(v)                          ▷ Berechne Knotengrad
3:  end for
4:  Sortiere die Knoten aufsteigend nach ihrem Grad
5:  for all v ∈ V(G) in der Reihenfolge der Sortierung do
6:      v.core ← v.deg
7:      for all u ∈ V(G) mit uv ∈ E(G) do       ▷ für alle Nachbarn u von v
8:          if u.deg > v.deg then
9:              u.deg ← u.deg − 1
10:             Sortierung entsprechend anpassen
11:         end if
12:     end for
13: end for
```

Um die Sortierung der Knoten in Zeile 4 des Algorithmus‘ in Linearzeit durchzuführen, wird in [5] das **BinSort-Verfahren** vorgeschlagen, welches für jeden möglichen Knotengrad einen „Behälter“ (engl. *bin*) zur Verfügung stellt, in die jeder Knoten gemäß seines Grades einsortiert wird. Die in Zeile 10 angegebene Anpassung der Sortierung beschränkt sich dann auf das Verschieben eines Knotens in den Behälter mit nächstkleinerem Knotengrad. Als Behälter wurden im Framework Instanzen der Klasse `cList` gewählt.

Die Berechnung der Abhängigkeitszahlen

Hierbei kommt ein einfacher Algorithmus zum Einsatz, der auf eine in [12] diskutierte Technik basiert: um die gegebene geringe Dichte eines Graphen auszunutzen, werden Algorithmen zur Traversierung des Graphen angewandt, deren Laufzeitkomplexität in erster Linie von der Anzahl der Kanten abhängig ist. Der entwickelte Algorithmus zur Berechnung der Abhängigkeitszahlen basiert daher auf dem bereits erwähnten BFS-Verfahren, welches lineare Laufzeitkomplexität in Abhängigkeit von der Kantenzahl aufweist. In der angegebenen entrekursivierten Fassung benötigt der Algorithmus lediglich eine Warteschlange **Q** und einen Kellerspeicher **S** als Datenstrukturen:

Algorithmus 3 Berechnung der *Dependency* eines Knotens v^*

```
 1: v*.dep ← 1; v*.layer ← 0;
 2: Q.queue(v*);
 3: while ¬Q.isEmpty do                                  ▷ Führe BFS durch
 4:     v ← Q.dequeue;
 5:     S.push(v);
 6:     for all u ∈ V(G) mit vu ∈ E(G) do
 7:         if u ∉ Q ∧ u ∉ S then                        ▷ Knoten u zum 1. Mal erreicht
 8:             u.dep ← 0; u.layer ← v.layer + 1;
 9:             Q.queue(u);
10:         end if
11:         if u.layer = v.layer + 1 then               ▷ Alle kürzesten Wege nach v sind
12:             u.dep ← u.dep + v.dep;                  ▷ auch kürzeste Wege nach u
13:         end if
14:     end for
15: end while
16: while ¬S.isEmpty do                                 ▷ BFS rückwärts
17:     v ← S.pop;
18:     Σ ← 0;
19:     for all u ∈ V(G) mit uv ∈ E(G) do
20:         if u.layer = v.layer − 1 then
21:             Σ ← Σ + u.dep;                          ▷ Alle kürzesten Wege nach v
22:             Q.queue(u);
23:         end if
24:     end for
25:     while ¬Q.isEmpty do                             ▷ Alle kürzesten Wege nach u
26:         u ← Q.dequeue;          ▷ plus alle kürzesten Wege zu Knoten hinter u
27:         u.dep ← u.dep + u.dep ÷ Σ ∗ v.dep;
28:     end while
29: end while
```

Die Breitensuche zerlegt den Graphen $G(V, E)$ in **Schichten**, wobei die k-te Schicht genau diejenigen Knoten $u \in E$ enthält, deren Abstand $dist(v^*, u)$ zu v^* gleich k ist. Ein **Nachfolgerknoten** eines anderen Knotens muß zu diesem benachbart sein und in der nächsthöheren Schicht liegen. Ein **Vorgängerknoten** ist analog dazu ebenfalls ein benachbarter Knoten, der jedoch zu der darunterliegenden Schicht gehört. Ein Beispiel zur Funktionsweise des Algorithmus‘ zeigt die Abbildung 5.7, wobei die durch das BFS-Verfahren erzeugte Schichtung des Graphen durch gestrichelte Linien angedeutet wurde:

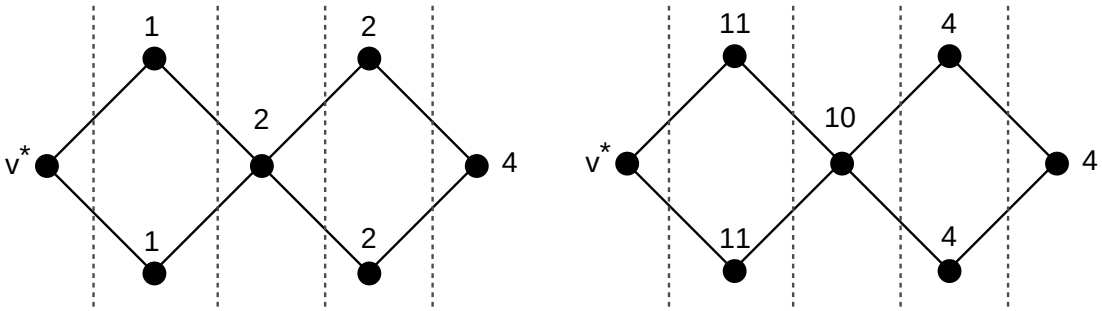


Abbildung 5.7: Beispiel zum Algorithmus zur Berechnung der Abhängigkeitszahlen.

Der Algorithmus 3 zerfällt in zwei funktionale Abschnitte. Im ersten (Zeilen 1–15) wird die Breitensuche ausgehend vom Knoten v^* durchgeführt. Dabei wird für jeden Knoten bestimmt, wieviele kürzeste Wege es von v^* zu jedem der Knoten gibt. Auf der linken Seite der Abbildung 5.7 ist die beschriebene Durchführung des BFS-Schrittes verdeutlicht, wobei die Nummern die Zahl kürzester Wege von v^* angeben: da es von v^* jeweils einen kürzesten Weg zu seinen Nachfolgern gibt, tragen diese die Nummer 1. Der eine Knoten in der darauffolgenden Schicht ist jeweils über einen der beiden vorhergehenden Knoten auf kürzestem Wege zu erreichen, daher gibt es 1+1=2 kürzeste Wege zu ihm. Dieser Berechnungsschritt wird in Zeile 12 durchgeführt, wo genauso wie im Beispiel jedem Nachfolgerknoten des jeweils aktuellen Knotens dessen Anzahl kürzester Wege aufaddiert wird. Dies ist logisch, da die Anzahl aller kürzesten Wege zu einem Knoten u gleichzeitig die Zahl aller über u laufenden, kürzesten Wege zu seinen Nachfolgerknoten ist.

Der zweite Teil des Algorithmus‘ (Zeilen 16–29) durchläuft die durch das BFS-Verfahren gewonnene Knotenreihenfolge rückwärts. Denn schließlich ist die Abhängigkeitszahl nicht als Anzahl aller kürzesten Wege von v^* zu einem anderen Knoten definiert, sondern als die Zahl aller kürzesten Wege von v^*, die zu diesem Knoten führen oder ihn passieren (siehe Abschnitt 3.1.1). Somit muß jeweils noch die Anzahl der kürzesten Wege, die einen Knoten passieren und in dahinterliegende Schichten führen, hinzugerechnet werden. Das Ergebnis

dieses Schrittes ist auf der rechten Seite der Beispiel-Abbildung 5.7 zu sehen. Schicht für Schicht werden jeweils die Anteile der Vorgängerknoten an dem Abhängigkeitswert jedes Knotens bestimmt und zu diesen hinzuaddiert (Zeile 27). Im Beispiel bedeutet das, daß der Abhängigkeitswert des letzten Knotens gleichmäßig auf seine Vorgänger aufgeteilt wird, da die vier kürzesten Wege zu ihm jeweils zur Hälfte über jeden der beiden Vorgänger laufen. Also haben die Vorgänger jetzt ebenfalls einen Abhängigkeitswert von 4, da zwei kürzeste Wege zu ihnen führen und zwei sie passieren.

Es gilt zu beachten, daß sich für den häufigen Fall mehrerer kürzester Wege zwischen zwei Knoten die Anzahl kürzester Wege exponentiell erhöht. Dies ist in Abbildung 5.8 verdeutlicht, wobei die obere Zeichnung die Werte nach dem BFS-Schritt des Algorithmus‘ zeigt und die untere dann das Endergebnis mit den konkreten Abhängigkeitszahlen. Das Problem bei solch großen Werten ist, daß es dabei rasch zu Überläufen der verwendeten Integer-Datentypen kommen kann. Dies gilt umso mehr für große Graphen, auf die das Framework abzielt.

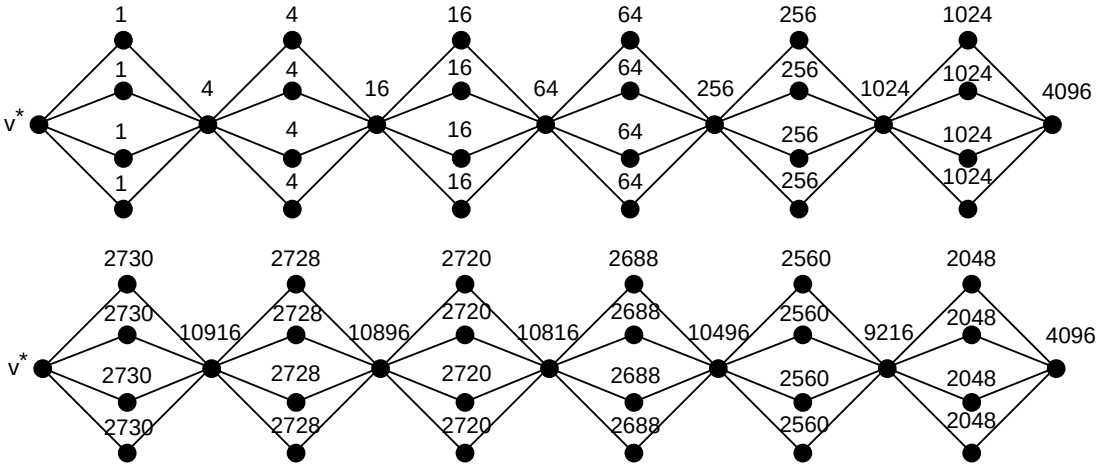


Abbildung 5.8: Beispiel für die exponentielle Zunahme der Zahl kürzester Wege.

Offensichtlich gibt es Ausnahmen, bei denen die Größe des Graphen keine solch entscheidende Rolle spielt. Dazu zählen z.B. Bäume oder Cliquen, da in diesen Graphklassen immer genau ein kürzester Weg zwischen zwei Knoten existiert.

Aufbauend auf die Abhängigkeitszahlen lassen sich auch die Knotenzwischenzahlen (siehe Abschnitt 3.1.1) berechnen. Dazu bestimmt man die Abhängigkeitszahlen nach und nach bezüglich jedes Knotens und addiert diese auf, um letztlich die Knotenzwischenzahlen zu erhalten. Für deren Bestimmung existieren aber auch speziell angepaßte Algorithmen [12] oder approximative Verfahren [22], die den Berechnungsprozess beschleunigen können und für Graphen mit geringer Dichte sogar subquadratische Laufzeitkomplexität haben.

5.5.2 Die Darstellungstechniken und ihre Interaktionsmöglichkeiten

Diese beiden eigentlich getrennten Funktionsblöcke wurden hier zusammengefaßt, da Visualisierung und Interaktion stark miteinander verknüpft sind und die Leistungsfähigkeit der Verfahren bei einer getrennten Erläuterung nur ansatzweise deutlich wird. Beide bilden einen integralen Bestandteil der grafischen Nutzeroberfläche des Frameworks (kurz: GUI für ***G**raphical **U**ser **I**nterface*) und sollen daher in diesem Abschnitt in erster Linie durch ausgewählte Bildschirmphotos dokumentiert werden.

Das Framework stellt in dieser ersten Version bereits drei Ansichten zur Verfügung:

- **Die Inhaltsansicht:** Sie ist die wichtigste Sicht auf die Daten, denn mit ihrer Hilfe kann der Graph in seiner ursprünglichen Struktur exploriert werden.

- **Die Detailansicht:** Für einen Knoten und sein Umfeld sind bei Bedarf zusätzliche Informationen abrufbar. Diese werden in einer separaten Detailansicht dargestellt, um den knappen Bildschirmplatz nicht ständig zu belegen.

- **Die Navigationssicht:** Sie dient der Darstellung automatischer Clusterergebnisse, die gleichzeitig als Navigationshilfe im Präsentationsraum dient.

Die Inhaltsansicht

Sie stellt die zu analysierende Struktur mit Hilfe des Fruchterman-Reingold-Algorithmus‘ im 3-dimensionalen Raum dar. Dieses Layout-Verfahren steht dabei nur stellvertretend für jedes andere implizite oder explizite Visualiserungsverfahren, das an dieser Stelle zum Einsatz kommen könnte. Der in Abschnitt 3.2.4 angegebene Algorithmus aus [25] läßt jedoch noch viele Fragen unbeantwortet, so daß er sich in dieser Form nicht direkt implementieren läßt. Dazu gehören:

- Wieviele Iterationen sind für ein hinreichend gutes Layout notwendig?

- Wie genau arbeitet die `cool`-Funktion? Welche der aus dem *simulated annealing* bekannten `cool`-Funktionen sind hier geeignet?

Um diese Fragen nicht erst durch langwieriges Ausprobieren zu beantworten, wurde die Implementierung an das frei verfügbare Programmpaket GUIDE[31] (***G**raph **U**tility with **I**nteractive **D**isplay and int**E**rface*) angelehnt.

[31] http://www.cs.arizona.edu/~kobourov/GRIP/guide.zip

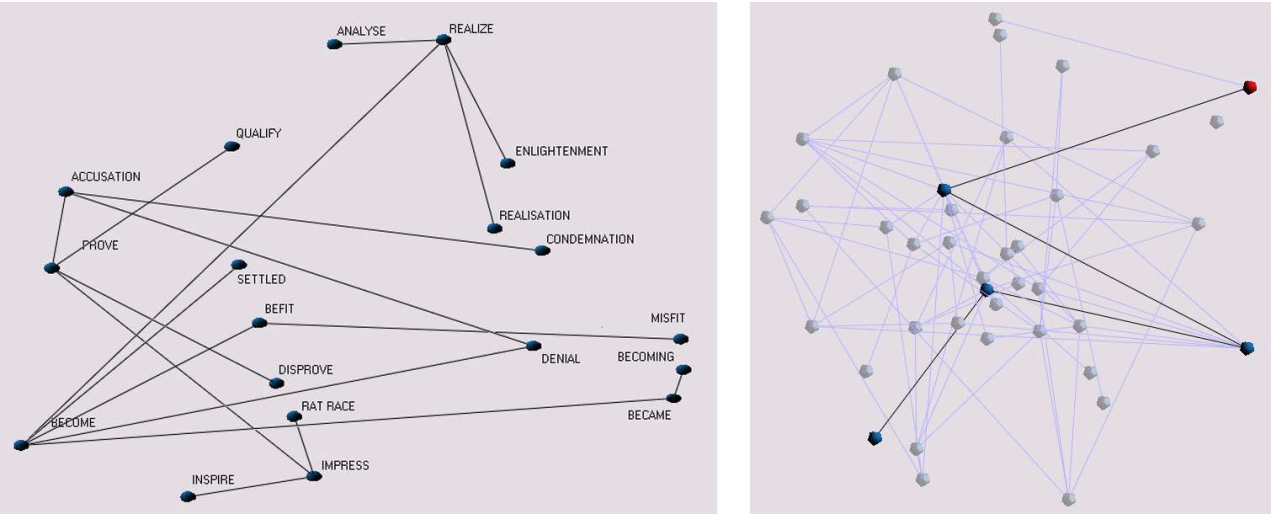


Abbildung 5.9: Beispiele für die Inhaltsansicht und Interaktionstechniken.

In Abbildung 5.9 ist auf der linken Seite die Darstellung eines kleinen Teilgraphen des EAT-Datensatzes (siehe Abschnitt 6.1) mit Beschriftungen (automatischer **Annotation**) an den Knoten zu sehen. Rechts ist ebenfalls ein Teilgraph mit einem berechneten **kürzesten Weg** abgebildet, bei dem ein Knoten **selektiert** ist (roter Knoten oben rechts). Alle Knoten und Kanten, die nicht Teil des Weges sind, wurden abgeschwächt dargestellt. So ist der Weg deutlich zu erkennen, aber trotzdem geht keine Kontextinformation verloren: die Umgebung des Weges ist immer noch ersichtlich, und auch zu abgeschwächt dargestellten Knoten können Detailinformationen angezeigt werden. Die 3-dimensionale Darstellung läßt sich bequem mit der Maus **drehen** und **zoomen**; einzelne Knoten können durch ein einfaches Ziehen mit der Maus in gewissen Grenzen repositioniert werden (**Rearrangement**). Im Falle von gerichteten Graphen können zusätzliche Pfeilspitzen die Orientierung der Kanten verdeutlichen. Filterung im Datenraum und hierarchisches Clustering sind über die beiden folgenden Dialoge zugänglich:

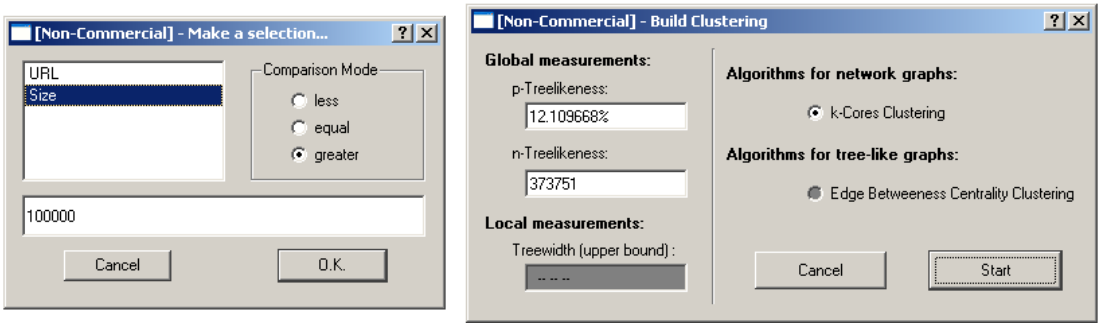


Abbildung 5.10: Steuerung komplexer Operationen über Dialoge.

Über den linken Dialog in Abbildung 5.10 lassen sich große Datenbestände nach ihren Attributen filtern. Im abgebildeten Beispiel, welches sich auf den Web-Datensatz (siehe Abschnitt 6.2 und Anhang A) bezieht, werden die Knoten (Web-Seiten) nach ihrem Attribut „Size“ gefiltert, indem alle Knoten mit „Size $\leq$ 100.000“ und alle Kanten, deren Start- oder Zielknoten sie sind, in der Inhaltsansicht ausgeblendet werden. Ist die Anzahl immer noch zu groß, kann innerhalb der aktuellen Filterung sukzessive weiter gefiltert werden. Attribute des Typs STRING werden über reguläre Ausdrücke gefiltert. War eine Filterung zu restriktiv, so daß zuviele oder gar alle Knoten ausgeblendet wurden, kann die Filterung über eine **Undo-Funktion** rückgängig gemacht werden.

Im rechten Dialog der Abbildung 5.10 sind zusätzlich zur Anzeige der p- und n-Baumähnlichkeitswerte des Graphen bereits Platzhalter für die einfache Integration weiterer modularer Funktionsbausteine vorhanden. Durch sie soll in einer späteren Ausbaustufe die Parametrisierung des Clusterings möglich werden. Die Idee ist hierbei, die Auswahl eines geeigneten Clusterverfahrens u.a. von globalen und lokalen Baumähnlichkeitsmaßen abhängig zu machen. Denn in Abschnitt 5.5.1 wurde bereits erwähnt, daß die Berechnung von Abhängigkeitszahlen und damit auch der auf diesen basierenden Knoten- und Kantenzwischenzahlen trotz der erwähnten Probleme für Bäume gut funktioniert. Die Frage ist also, wie groß der Anteil zusätzlicher Nicht-Baum-Kanten sein darf, bevor ein Überlauf bei dem verwendeten Integer-Datentyp auftritt. Da ein Edge-Betweenness-Centrality-Clustering nur möglich ist, wenn vorher die Kantenzwischenzahlen berechnet wurden, ist diese Frage also indirekt auch für die Auswahl des Clusterverfahrens von Bedeutung, und so wurden die Baumähnlichkeitsmaße vorausschauend bereits mit in diesen Dialog aufgenommen. Daß sie nicht alleiniges Kriterium sein können, zeigt das Beispiel der Clique, die zwar kaum als baumähnlich bezeichnet werden kann, aber bei der Berechnung der Abhägigkeitszahlen dennoch überschaubare Werte liefert, nämlich jeweils 1.

Zusätzlich ist zu erwähnen, daß das Framework häufigen Gebrauch von den nicht presenten Spalten der in Abschnitt 5.3.2 beschriebenen Klasse `cTable` macht. Da gerade der Umgang mit Speicherplatz bei der Verarbeitung großer Graphen ein sehr sensibler Bereich ist, werden beim Öffnen eines Graphen nur die allernotwendigsten Attribute geladen. Dazu gehört z.B., falls vorhanden, das erste Attribut vom Typ STRING, da dies in den allermeisten Fällen die Bezeichnungen der Knoten enthält. Werden im Zuge des Mining-Prozesses andere, nicht presente Attribute benötigt, werden diese erst dann vom Framework nachgeladen. Ein automatisches Freigeben von Attributen, die z.B. längere Zeit nicht benutzt wurden, ist in einer späteren Ausbaustufe jederzeit ergänzbar.

Die Detailansicht

Diese gibt die Größe der k-Nachbarschaft eines Knotens $v \in V(G)$ für $1 \leq k \leq 5$ über ein Balkendiagramm aus. Wie man in Abbildung 5.11 sieht, erfolgt die Ausgabe des Diagramms innerhalb eines Dialogs, dem bei Erweiterung durch zusätzliche Detailansichten einfach neue Reiter hinzugefügt werden können (modularer Aufbau). Die k-Nachbarschaft ist die Anzahl aller Knoten, die in einer Entfernung $\leq \ k$ zu v liegen (siehe Abschnitt 3.1.1). Die Begrenzung auf $k \leq 5$ wurde vorgenommen, da die Knoten jenseits davon nur noch wenig über die strukturellen Eigenschaften des Knotens v aussagen, sondern schon eher globale Maße sind. Einfache Beispiele für strukturelle Schlußfolgerungen aus den k-Nachbarschaften sind:

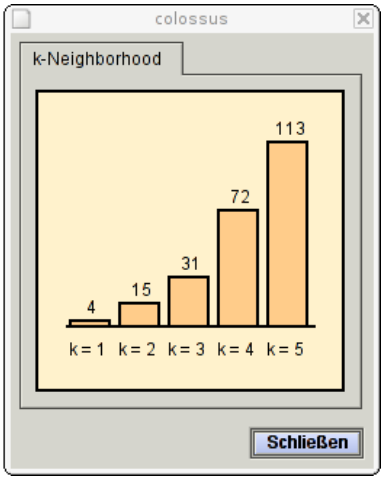


Abbildung 5.11: Die Detailansicht unter LINUX.

- $|\mathbf{N_1(v)}| = \mathbf{1}, |\mathbf{N_2(v)}| = \mathbf{2}, |\mathbf{N_3(v)}| = \mathbf{3}, |\mathbf{N_4(v)}| = \mathbf{4}, |\mathbf{N_5(v)}| = \mathbf{5}$
  Dies bedeutet, daß der Knoten v am Ende eines mindestens fünf Kanten langen Pfades liegt.

- $|\mathbf{N_1(v)}| = \mathbf{2}, |\mathbf{N_2(v)}| = \mathbf{4}, |\mathbf{N_3(v)}| = \mathbf{6}, |\mathbf{N_4(v)}| = \mathbf{8}, |\mathbf{N_5(v)}| = \mathbf{10}$
  Hier liegt der Knoten mittig auf einem mindestens zehn Kanten langen Pfad. Dieser kann ein induzierter Teilgraph eines größeren Graphen sein, z.B. eines Kreises mit mindestens zwölf Knoten.

- $|\mathbf{N_1(v)}| = \mathbf{5}, |\mathbf{N_2(v)}| = \mathbf{5}, |\mathbf{N_3(v)}| = \mathbf{5}, |\mathbf{N_4(v)}| = \mathbf{5}, |\mathbf{N_5(v)}| = \mathbf{5}$
  Dieser Graph hat entweder nur sechs Knoten oder mehrere Zusammenhangskomponenten. Der Knoten v ist mit jedem der anderen fünf verbunden. Über deren Verbindungsstruktur ist hingegen nichts bekannt, so daß dieser Graph zwischen einem $K_{1,5}$, bei dem keiner der Nachbarn mit einem anderen verbunden ist, und einer Clique K_6 liegen kann.

Die Navigationsansicht

Diese basiert auf ein hierarchisches Clustering und zeigt dessen Clusterhierarchie mit Hilfe der am Forschungsbereich für Computergraphik der Universität Rostock entwickelten MagicEye-Komponente an. Sie beherrscht u.a. Rotation, Zoom, Selektion (*Picking*), Klappen von Teilbäumen und automatische Annotation. Die Komponente wurde derart angepaßt, daß Knoten, die einzelne Informationsobjekte repräsentieren, rot gefärbt werden. Cluster, die solche Informationsobjekte und andere Cluster zusammenfassen, werden in grüner Farbe dargestellt. Bei der Selektion eines Clusters (gekennzeichnet durch eine

gelbe, kreisförmige Umrandung) wird die Anzahl der durch ihn repräsentierten Informationsobjekte angezeigt. Selektierte Cluster können als Wurzelknoten in den Mittelpunkt einer neuen MagicEye-Ansicht gestellt werden. Dies erlaubt die Exploration der Hierarchiestufen des Clusterings, von denen aus Platzgründen höchstens fünf gleichzeitig dargestellt werden.

In Abbildung 5.12 wird solch ein Vorgang gezeigt: auf der linken Seite wird ein Cluster markiert, der dann als Wurzelknoten in der neuen Darstellung auf der rechten Seite dient. An diesem Beispiel-Clustering auf dem EAT-Datensatz (siehe Abschnitt 6.1) ist auch noch einmal die automatische Annotation und das Verschieben des Fokus‘ zu sehen.

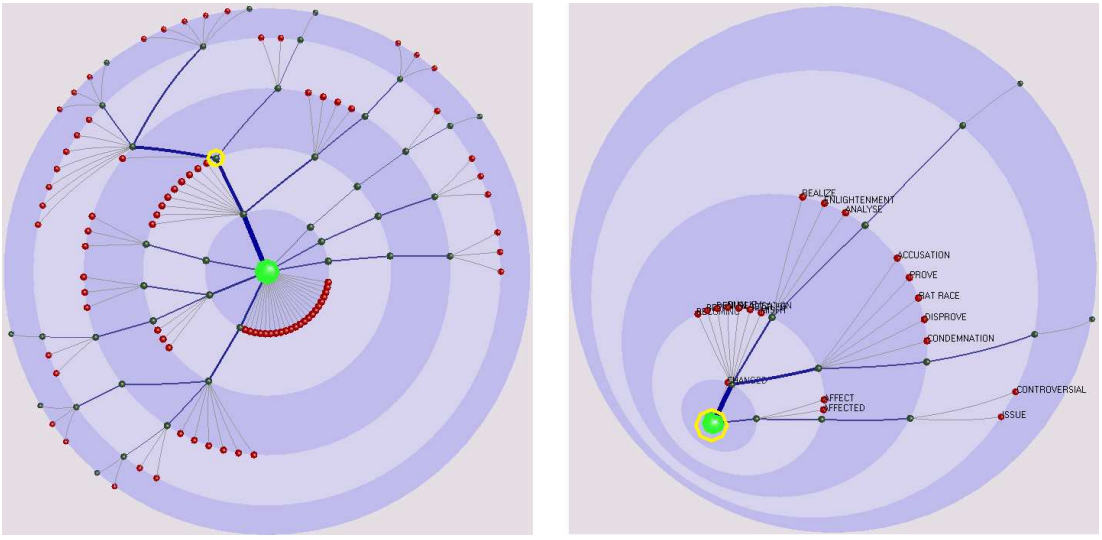


Abbildung 5.12: Selektion eines Clusters (links) und neue Ansicht mit dem selektierten Cluster als Wurzelknoten (rechts).

Ferner erlaubt die grafische Benutzungsoberfläche des Frameworks, einen selektierten Cluster in die Inhaltsansicht zu übertragen. So läßt sich auch ein großer Datensatz über das hierarchische Clustering erschließen: der Nutzer traversiert den Hierarchiebaum Stufe für Stufe in einer gewünschten Richtung, bis ein Cluster klein genug ist, damit er in die Inhaltsansicht übertragen werden und dort in seiner expliziten Graphrepräsentation exploriert werden kann.

Kapitel 6

Visuelle Analyse zweier Beispieldatensätze

Um die vielfältigen Möglichkeiten aufzuzeigen, die das Framework bereits in der umgesetzten Basisversion bietet, wurden zwei Beispieldatensätze für eine erste visuelle Analyse mit der entstandenen Software ausgewählt. Dabei bestand die Vorverarbeitung in beiden Fällen aus Clustering und Layout-Berechnung. Die Grenzen dieser beiden realisierten Funktionsmodule wurden im Zuge der Berechnungen rasch deutlich. So ist unabhängig vom jeweiligen Datensatz zu bemerken, daß die Layout-Berechnung für einen großen Datensatz trotz aller Optimierungen der Datenstrukturen zuweilen zwölf Stunden und länger benötigt. Dies ist jedoch für die verwendeten statischen Strukturen noch akzeptabel, da diese Berechnung nur einmal durchzuführen und ihr Ergebnis abzuspeichern ist. Danach steht es jederzeit binnen Sekunden wieder zur Verfügung. Gleiches gilt für die k-Core Dekomposition, die als Clusterverfahren zum Einsatz kommt. Nach der Vorverarbeitung wurden die Daten und ihre Clusterhierarchien in unterschiedlicher Reihenfolge exploriert und analysiert.

6.1 Edinburgh Associative Thesaurus

Der **Edinburgh Associative Thesaurus**[32] (nachfolgend **EAT** abgekürzt) wurde 1973 von Linguisten aus einer Liste der eintausend häufigsten Wörter der englischen Sprache erstellt. Probanden wurde eine zufällige Auswahl von jeweils einhundert Wörtern dieser Liste vorgelegt und diese sollten dann in möglichst kurzer Zeit jeweils das erste Wort, welches ihnen zu jedem der vorgegebenen Wörter (*stimuli*) einfiel, dazuschreiben (*response*). Die häufigsten der Antworten wurden wiederum der Liste hinzugefügt, bis diese insgesamt über 8.000 Wörter enthielt. Aus dieser resultierenden Liste wurden einhundert Teilmengen mit je einhundert Wörtern gebildet und abermals Probanden als *stimuli* vorgelegt. De-

[32] http://www.eat.rl.ac.uk/

ren Assoziationen bilden den EAT-Datensatz, der nun zu jedem der insgesamt 8.210 *stimuli* eine Liste von Assoziationen sowie deren Häufigkeit enthält. Dieser Datensatz ist in Form einer einfachen Textdatei aus dem Internet zu beziehen und diente als Grundlage für die nachfolgenden Untersuchungen. Wie aus Abbildung 6.1 deutlich wird, ist die manuelle Exploration dieser textuellen Daten ein fast unmögliches Unterfangen — Zeile um Zeile reihen sich die Datenkolonnen aus *responses* und Häufigkeiten aneinander:

```
ILL|7|FOR|7|OF|3|ACCOUNTANT|2|CHEQUE|2|COST|2|FIGURES|2|NUMBER|2|BANKS|1|BOOK|1|BUY|1|CARD|1|CASH
OOKS|4|CHARTERED|4|DEPARTMENT|4|FIGURES|4|BILLS|3|CLERK|2|FIDDLE|2|FOR|2|PAPER|2|TELLS|2|ACCOUNTA
E|13|EXACT|8|RIGHT|7|GOOD|6|CLOSE|3|FINE|2|INACCURATE|2|MATHS|2|PRECISION|2|TIMING|2|TRUE|2|AIM|1
|FINGER|4|ACCUSE|3|CHARGE|3|FALSE|3|GUILT|3|JUDGE|3|THREAT|3|AGAINST|2|DENY|2|GUILTY|2|LAW|2|LIAR
|BACK|5|TOOTH|3|HEART|2|CHEST|1|DULL|1|ELBOW|1|FEET|1|GRUMBLE|1|LEGS|1|MUSCLE|1|OUCH|1|OUT|1|SOOT
PHURIC|10|BASE|5|TEST|5|BATH|4|BITTER|4|BOTTLE|3|CHEMISTRY|3|ALKALINE|2|DROP|2|HCL|2|HYDROCHLORIC
BASES|8|BURN|8|SCIENCE|3|AMINO|2|BITTER|2|CORROSION|2|DROPS|2|HYDROCHLORIC|2|NASTY|2|NITRIC|2|SOU
```

Abbildung 6.1: Ein Ausschnitt der Textdatei mit den Daten des EAT.

Doch diese Daten können als Graph aufgefaßt werden, wobei die Knoten den *stimuli* entsprechen und die Kanten einer Assoziation zwischen zwei *stimuli*. Der Datensatz bildet damit nichts anderes als eine Adjazenzliste für die *stimuli*, bei der die Häufigkeiten der Assoziationen als Kantengewichte interpretiert werden. *Dangling Ends* in Form von Assoziationen mit Wörtern, die ihrerseits wiederum nicht in der Liste vertreten sind, wurden aus dem Datensatz entfernt, wonach noch 261.453 Kanten übrigblieben. Auch wenn der resultierende Graph damit nur eine Teilmenge des Datensatzes modelliert, und die Ergebnisse der auf ihm durchgeführten Analysen daher nicht repräsentativ für den gesamten EAT stehen, eignet sich die visuelle Exploration dennoch, um ein Grundverständnis der Daten zu erlangen. So wird beispielsweise auf den ersten Blick deutlich, daß der Graph eine hohe Konnektivität aufweist, da die Knoten durch vergleichsweise viele Kanten verbunden sind. Dies vermutet man bereits nach einem Blick auf seine Volldarstellung in Abbildung 2.3 (rechts) und verifiziert es im Clusterdialog durch dem niedrigen p-Baumähnlichkeitswert von gut 3%. Das bedeutet, daß fast 97% aller Kanten entfernt werden müßten, damit aus dem Netzwerk ein Baum wird. Diese Eigenschaft liegt zuvorderst in der Tatsache begründet, daß eben gerade diejenigen Wörter als *stimuli* ausgewählt wurden, die besonders häufig als Antwort gegeben wurden. Was dieser hohe Grad an Konnektivität für das einzelne Wort bedeutet, soll nachfolgend am Beispiel des zufällig ausgewählten Wortes PEANUTS verdeutlicht werden. Dieses wird mit insgesamt 45 anderen Wörtern direkt assoziiert, die zusammen seine 1-Nachbarschaft bilden:

- einmal in seiner Bedeutung „Erdnuß“ z.B. mit BUTTER (Butter), BEERMUG (Bierkrug) oder SALTY (salzig)
- in seiner umgangssprachlichen Bedeutung „Geld“ z.B. mit MONEY (Geld) oder EARN (verdienen)

- mit einigen z.T. sehr „obskuren“ Assoziationen wie MIRROR (Spiegel) oder GOD (Gott)

Die große Zahl an Assoziationen ist deshalb erstaunlich, weil das Wort PEANUTS im Gegensatz zu BE (sein) oder NOT (nicht) nicht zu den 5.000 häufigsten Wörtern der englischen Sprache zählt[33]. Wie aus dem k-Neighborhood-Diagramm für das Wort PEANUTS in Abbildung 6.2 ersichtlich ist, bilden bereits für $k = 3$ alle anderen Knoten des Graphen die k-Nachbarschaft des Wortes. Daher bleibt für $k = 4$ und $k = 5$ die Größe der k-Nachbarschaft konstant, weil es keine weiteren Knoten gibt, die noch hinzugefügt werden können. Das bedeutet, daß alle anderen Wörter über maximal drei Assoziationsschritte von PEANUTS aus erreichbar sind. Zur Bestimmung der kürzesten Assoziationskette zwischen zwei Begriffen kann der kürzeste Weg zwischen zwei Knoten berechnet werden. Als Beispiel wurde die kürzeste Kette sukzessiver Assoziationen zwischen den Begriffen LOVE AFFAIR (Liebesaffäre) und PRIESTHOOD (Priesterschaft) gesucht. Die überraschend simple Lösung ist in Abbildung 6.3 (rechts) zu sehen:

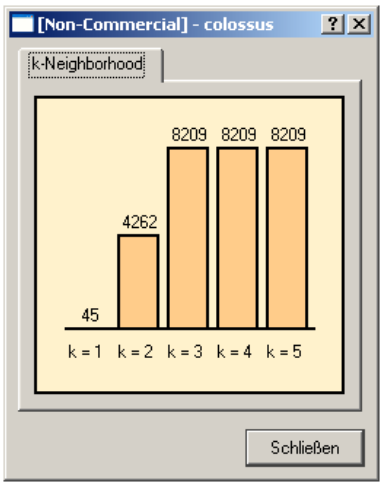


Abbildung 6.2: k-Neighborhood-Diagramm für das Wort PEANUTS.

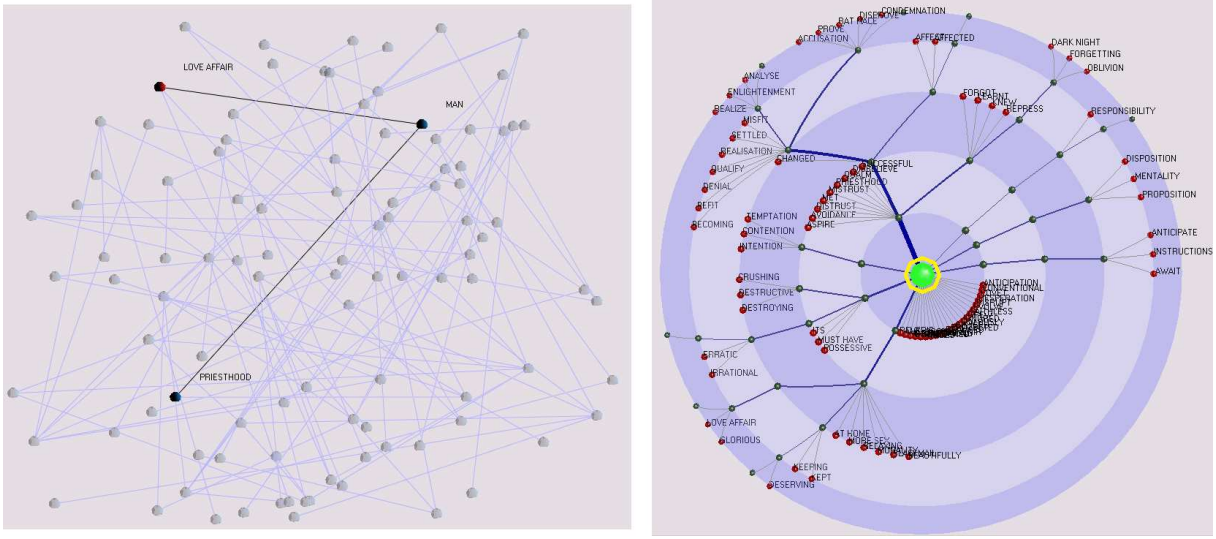


Abbildung 6.3: Explizite Darstellung (links) des markierten Clusters (rechts).

[33] http://www.edict.com.hk/lexiconindex/frequencylists/words2000.htm
http://www.edict.com.hk/lexiconindex/frequencylists/words2-5k.htm

Die k-Core Dekomposition liefert für diesen Datensatz insgesamt 370 Cluster in 57 Hierarchieebenen. Dabei werden in den meisten Fällen durchaus Wörter in einem Cluster zusammengefaßt, die in ihrer Bedeutung verwandt sind. Im Beispiel in der Abbildung 6.3 (rechts) sind dies unter anderem:

- ITS (sein/ihr), MUST HAVE (haben müssen), POSSESSIVE (besitzanzeigend)
- CRUSHING (zerquetschen/zermalmen), DESTRUCTIVE (zerstörerisch), DESTROYING (zerstörend)
- ANTICIPATE (voraussehen/erahnen), INSTRUCTIONS (Anweisungen), AWAIT (erwarten)

Abweichungen kommen in erster Linie durch die bereits erwähnten „obskuren“ Assoziationen zustande. Da diese jedoch sehr selten gehäuft auftreten, ließe sich mit einem anderen Dekompositions- oder Clusterverfahren, welches im Gegensatz zur k-Core-Dekomposition auch die Kantengewichte berücksichtigt, deren Einfluß auf ein Minimum beschränken und ein noch besseres Clusterergebnis erzielen.

6.2 Linkstruktur von Webseiten

Die Datenbasis setzt sich in diesem Fall aus allen statischen Internetseiten des Instituts für Informatik der Universität Rostock zusammen, die am 30.März 2004 von der Hauptseite `http://www.informatik.uni-rostock.de` aus erreichbar waren. Auf eine detaillierte Beschreibung, wie die Daten gesammelt und aufbereitet wurden, wird hier verzichtet, da sie Anhang A zu entnehmen ist. Zu den nachfolgenden Analysen ist zu bemerken, daß sie kein vollständiges Bild aller Webseiten des Instituts liefern, da viele der Webdokumente dynamischen Inhalt besitzen und daher nicht mit in den Datensatz aufgenommen wurden. Dies hätte zu Inkonsistenzen in den Daten führen können, denn die URL[34] einer dynamischen Seite (`.../show.cgi?page=4`) kann nicht als eindeutige Identifikation gelten. Daher fehlt im Datensatz beispielsweise ein großer Teil der Webpräsenz der technischen Informatik.

Die Daten bestehen aus den Linkinformationen und den URLs der 51.497 gefundenen Webseiten inklusive deren Dateigröße als zusätzliches Attribut. Die insgesamt 425.247 Links zwischen den Webseiten bilden somit die gerichtete Kantenmenge des zu analysierenden Graphen und die Webdokumente dessen Knotenmenge. Aus dem k-Neighborhood-Diagramm der Startseite in Abbildung 6.4 ist ersichtlich, daß 3.633 Webseiten über 5 Klicks von ihr aus erreichbar sind. Dies ist ein Anteil von gut sieben Prozent der Gesamtmenge der Webseiten. Bei

[34] Adresse eine Webseite (***U**niform **R**esource **L**ocator*)

der Exploration der insgesamt 4.824 Cluster der k-Core-Dekomposition dieses Graphen stößt man relativ schnell auf zwei besonders stark verlinkte Unterstrukturen: den Dokumentationen der Java Development Kits 1.3[35] (auf den Webseiten der Softewaretechnik) und 1.2.1[36] (auf den Webseiten der Abteilung für Rechentechnik). Diese machen zusammen fast ein Fünftel der gefundenen Webseiten aus.

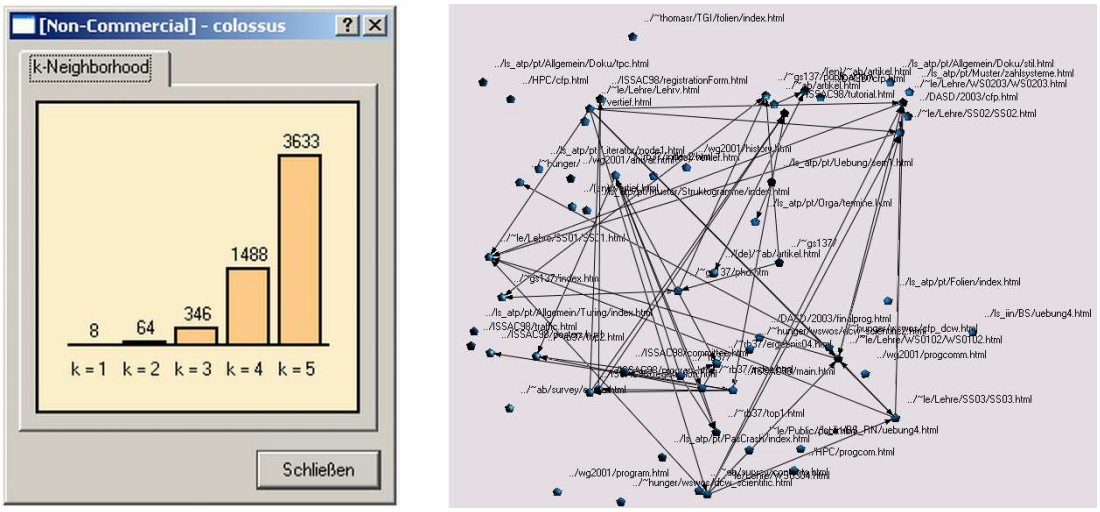


Abbildung 6.4: k-Neighborhood-Größen der Startseite (links) und gefilterte Inhaltsansicht (rechts).

Sowohl URL als auch die Seitengröße eignen sich sehr gut, um diesen relativ großen Datensatz nach interessanten Bereichen zu filtern: sollen beispielsweise nur die Webseiten der Wissenschaftsbereiche „Theoretische Informatik“ und „Computergraphik & Kommunikation“ angezeigt werden, filtert man die Knoten nach ihrer URL mit dem regulären Ausdruck `://wwwicg\.|://wwwteo\.` (QT-Syntax). Sollen nun noch alle Webseiten, die kleiner als 5 kByte sind, ausgeblendet werden, so läßt sich dies analog zu dem linken Beispiel in Abbildung 5.10 durchführen. Das Ergebnis einer solchen Filterung auf dem Gesamtgraphen ist in Abbildung 6.4 zu sehen, wobei hier lediglich alle Seiten der theoretischen Informatik angezeigt werden, die größer als 5000 Byte sind. Dabei kann es von Bedeutung sein, welcher Filterschritt zuerst angewandt wird, denn ein einfacher Vergleich mit einem Schwellwert ist nicht so aufwendig, wie ein Vergleich über einen regulären Ausdruck. So kann die gefilterte Inhaltsansicht in Abbildung 6.4 im Schnitt 25% schneller bestimmt werden, wenn statt der oben angegebenen Filterreihenfolge zuerst mit Hilfe der Dateigröße alle Webdokumente, die kleiner als 5000 Byte sind, gefiltert werden und dann in den übriggebliebenen nach

[35] http://wwwswt.informatik.uni-rostock.de/deutsch/Links/jdk1.3/docs/index.html
[36] http://www.informatik.uni-rostock.de/FB/art/software/jdk121/

solchen mit `://wwwteo\.` in der URL gesucht wird. Da in einem gefilterten Teilgraphen nur die induzierten Kanten angezeigt werden, sind in der Abbildung 6.4 relativ viele einzelne Knoten zu sehen. Die Struktur scheint dabei komplexer, als sie in Wirklichkeit ist. Abbildung 6.5 zeigt den gleichen Teilgraphen nach einem Rearrangement, wie es auch ein guter 2-dimensionaler Layout-Algorithmus vorgenommen hätte. Solch eine manuelle Repositionierung kann also nicht nur zur Nachbearbeitung des Graph-Layouts für Präsentationszwecke dienen, sondern auch das Verständnis für die Struktur in weit größerem Maße erhöhen, als es durch Rotation und Zoom der 3-dimensionalen Darstellung möglich ist. So erkennt man beispielsweise in der Abbildung unten rechts deutlich die Cliquenstruktur des Teilgraphen:

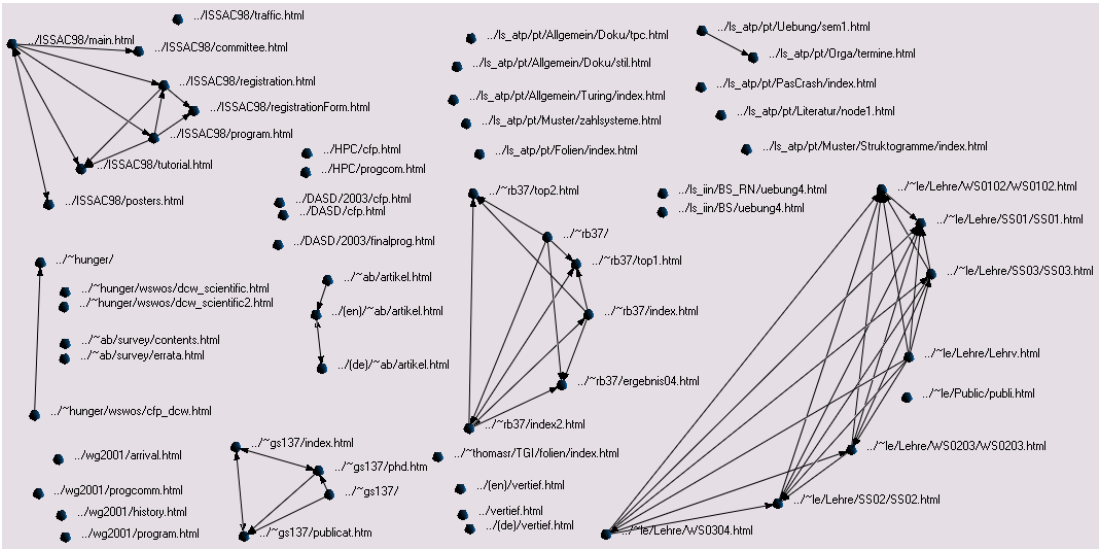


Abbildung 6.5: Rearrangement des Graphen aus Abbildung 6.4 (rechts).

Kapitel 7

Zusammenfassung und weitere Ideen

Ziel dieser Arbeit war es, das Konzept eines flexiblen Frameworks zum visuellen Data Mining in komplexen Strukturen zu entwickeln und dieses prototypisch zu implementieren. Ein vergleichbares Framework existiert zu diesem Zeitpunkt noch nicht, da das Gebiet des Struktur-Minings bisher nur am Rande des visuellen Data Minings von Attributwerten betrachtet wurde.

Da das visuelle Data Mining automatische Methoden und Visualisierungsverfahren miteinander kombiniert, mußten im ersten Teil der vorliegenden Arbeit Methoden gefunden werden, die sich für die automatische Vor- und Nachverarbeitung eignen. Strukturen lassen sich dabei in den meisten Fällen als Graphen modellieren, und so scheinen viele bekannte graphentheoretische Methoden (kürzester Weg, Graph Matching,...), Clusterverfahren (k-Core Dekomposition, Linkage Verfahren,...) und anwendungsspezifische Maße (Zentralitätsmaße, Ähnlichkeitsmaße,...) prinzipiell für deren Analyse in Frage zu kommen. Da der Fokus der Arbeit auf großen, komplexen Strukturen liegt, sind vor allem die hierarchischen Clusterverfahren von besonderer Bedeutung für das visuelle Data Mining in solch umfangreichen Datenmengen. Durch sie ist der Nutzer in der Lage, die Struktur mit Hilfe von detailärmeren, überschaubaren graphischen Darstellungen zu erschließen oder durch Traversierung der Clusterhierarchien zu explorieren.

Dafür werden jedoch spezielle Strukturvisualisierungstechniken benötigt, von denen ebenfalls eine repräsentative Auswahl vorgestellt und mit Hilfe von Baumähnlichkeitswerten neu parametrisiert wurde. Durch diese neuartige Technik läßt sich die übliche strikte Trennung von Strukturvisualisierungsverfahren in Hierarchie- und Netzwerkdarstellungen als ein allmählicher Übergang zwischen diesen beiden Klassen gestalten. Die Parametrisierung kann dabei auch als Grundlage für eine automatische Vorauswahl geeigneter Darstellungstechniken in

Abhängigkeit von der darzustellenden Struktur und dem Ausgabemedium dienen. Doch Visualisierungsverfahren lassen sich auch nach anderen Gesichtspunkten klassifizieren, von denen die Baumähnlichkeit nur eine ausgewählte Variante darstellt. Eine Idee für auf diesen Ansatz basierende Erweiterungen wurde bereits in Abschnitt 3.2.6 kurz erläutert.

Welche Visualisierungsverfahren und automatischen Methoden in einem konkreten Anwendungsfall praktikabel sind, läßt sich im Vorhinein nicht sagen. Deshalb muß die Konzeption des Frameworks flexibel genug sein, um gegebenenfalls durch zusätzliche Methoden erweitert werden zu können. Ein solches Framework, welches sich mit geeigneten Methoden an unterschiedliche Anwendungsgebiete anpassen läßt, wurde im Rahmen dieser Arbeit entwickelt (Abbildung 4.2). Eine erweiterte Fassung, die das Framework in einer fortgeschrittenen Ausbaustufe vorstellt, gibt bereits einen Einblick in die Leistungsfähigkeit dieses modularen Konzepts (Abbildung 4.3).

Für die prototypische Implementation wurde eine Auswahl grundlegender, vom jeweiligen Anwendungsgebiet unabhängiger Methoden getroffen, so daß erste Analysen von Beispieldatensätzen möglich waren.

Das entwickelte Framework geht von der Prämisse aus, daß der Anwender durch die Kopplung automatischer Analysemethoden mit Visualisierungstechniken zu einem besseren Verständnis der strukturellen Abhängigkeiten innerhalb eines Datensatzes gelangen will. Das bereits 1996 in [27] vorgestellte Framework für visuelles Data Mining geht hingegen davon aus, daß der Anwender durch die erwähnte Kopplung in erster Linie ein besseres Verständnis für die automatischen Methoden entwickeln will, um diese dann zu verbessern. Gerade für die Entwicklung, Implementierung und Parametrisierung effizienter Graphenalgorithmen kann solch eine visuelle Evaluierung, z.B. anhand von speziell konstruierten Referenzdatensätzen, eine entscheidende Hilfe sein. Daher gilt es zu untersuchen, inwieweit die beiden Frameworks kompatibel zueinander sind und ob eine Integration dieser gänzlich anderen Sichtweise auf das visuelle Data Mining in das entwickelte Framework möglich ist.

Neben vielen technischen Ergänzungen, wie sie stellenweise bereits in einigen der vorangegangenen Kapitel angemerkt wurden, wäre eine Anpassung des Frameworks zur Verarbeitung externer oder dynamischer Daten sinnvoll und wünschenswert. Denn es gibt einige Anwendungsgebiete, die sich erst dann mit dem entwickelten Framework erschließen lassen (z.B. riesige Genom-Datenbanken oder Peer-2-Peer-Netzwerke). Auch Verfahren zur vergleichenden Analyse von Informationsstrukturen müssen in diesem Zusammenhang noch untersucht werden, um das Framework zu komplettieren.

Anhänge

Details zum Web-Datensatz

Eine der interessantesten Anwendungen ist das visuelle Data Mining in großen Hypertextbeständen, deren Dokumente über Hyperlinks miteinander verknüpft sind. So stand schon zu einem frühen Zeitpunkt der Diplomarbeit fest, daß einer der zu untersuchenden Datensätze die Linkstruktur universitärer Webseiten sein soll. Da ein entsprechender Datensatz bis dahin nicht zur Verfügung stand, mußten die Daten im Rahmen der Diplomarbeit zusammengetragen werden. Ein eigens in PERL programmierter Web-Crawler hat in der Nacht des 30. März 2004 die Linkinformationen aller öffentlichen von der Start-URL `http://www.informatik.uni-rostock.de` erreichbaren, statischen Internetseiten in einer MySQL-Datenbank gesammelt.

Dem waren bereits einige Versuche vorausgegangen, die aber immer daran scheiterten, daß z.T. viele unterschiedliche URLs auf ein und dieselbe Internetseite verweisen. So sind die folgenden Möglichkeiten und ihre vielfältigen Kombinationen untereinander alle äquivalent:

- `http://wwwicg.informatik.uni-rostock.de`
  `http://wwwicg.informatik.uni-rostock.de/`
  `http://wwwicg.informatik.uni-rostock.de//`
  usw.

- `http://www.icg.informatik.uni-rostock.de`

- `http://wwwicg.informatik.uni-rostock.de/index.html`

- `http://wwwicg.informatik.uni-rostock.de:80`

Da nicht all diese Varianten bereits im Vorraus berücksichtigt wurden und die meisten Seiten nur relativ und nicht absolut verlinkt sind, haben sich diese kleinen Unterschiede in nur einer gefundenen URL sogleich fortgepflanzt und es wurde eine zusätzliche Kopie der Informatik-Homepages abgesucht — doppelte Datenbankeinträge und vor allem doppelte Laufzeit waren die Folge.

Das PERL-Skript wurde daher Lauf für Lauf angepaßt, bis keine offensichtlichen Dopplungen mehr in der Datenbank auftraten. Bei einem ersten Clustering mit dem erstellten Data Mining Framework konnten allerdings dennoch zwei identische Cluster gefunden werden, die vorher nicht aufgefallen sind. Dabei handelte es sich um ein paar tausend Seiten, die sowohl mit `http://wwwswt.informatik.uni-rostock.de/...` als auch mit `http://wwwswt.informatik.uni-rostock.de//...` in der Datenbank eingetragen waren. Grund dafür ist eine URL auf der Seite `http://www.informatik.uni-rostock.de/en/divisions/index.html`, welche die Softwaretechniker eben mit zwei Slashes, statt mit einem verlinkt — eine Variante synonymer URLs, die nicht

von dem Web-Crawler berücksichtigt wurde. Diese doppelten Einträge wurden daraufhin kurzerhand nachträglich gelöscht und der unkonventionelle Link von der besagten Seite auf den richtigen Eintrag in der Datenbank umgebogen. Nachfolgende Clusterings lieferten keine weiteren Auffälligkeiten mehr.

Zum Quellkode des Crawler-Skripts ist zu bemerken, daß der oben aufgetretene Fall mehrfacher Slashes noch nicht berücksichtigt wurde, da er ja erst nachträglich beim Clustern bemerkt wurde. Dies ließe sich aber leicht durch Löschen überzähliger Slashes (ausgenommen denen in `http://...` natürlich) realisieren. Weiterhin ist zu beachten, daß der Crawler entgegen allen Netiquetten die eventuell vorhandene Datei `robots.txt` nicht beachtet. Um einem möglichen Blocking zu entgehen, identifiziert sich das Skript ferner als Mozilla-Browser. Dies ist eventuell wichtig, damit der Crawler wirklich alle statischen Seiten erreicht, die auch ein menschlicher Nutzer besuchen kann und nicht dem Blocking eines Intrusion-Detection-Systems zum Opfer fällt.

Listing: crawler.pl

```
 1: #!/usr/bin/perl
 2: use LWP::UserAgent;
 3: use HTML::LinkExtor;
 4: use URI::URL;
 5: use DBI;
 6: $|++;
 7: my $url = "http://www.informatik.uni-rostock.de/";  # start URL
 8: my $ident = ".INFORMATIK.UNI-ROSTOCK.DE";
 9: my $ua = LWP::UserAgent->new;
10: $ua->agent('Mozilla/5.0');
11: $ua->timeout(10);
12: my $user = "user"; my $pass = "pass"; # MySQL login
13: my $countfast = 1; my $countslow = 1;
14: my $found = 0; my $errors = 0;
15: my @links = ();  # collection of newly found links
16: my @linklist = (); # collection of link-indices
17: # define Data Source Name
18: my @DSN= ("DBI:mysql:"."database=crawler;"."host=localhost",$user,$pass);
19: # connect to database
20: my $dbh = DBI->connect(@DSN,{PrintError => 0,AutoCommit => 1,});
21: die $DBI::errstr unless $dbh; # Catch connection error!
22: # prepare DB-operations once and for all
23: my $insert = $dbh->prepare("INSERT INTO sites (URL, count) VALUES (?,?)")
24:                 or die $dbh->errstr();
```

```
25: my $selnextURL = $dbh->prepare("SELECT URL FROM sites WHERE count = ?")
26:                  or die $dbh->errstr();
27: my $findURL = $dbh->prepare("SELECT count FROM sites WHERE URL = ?")
28:                  or die $dbh->errstr();
29: my $update = $dbh->prepare("UPDATE sites SET links = ?, size = ?
30:                  WHERE URL = ?") or die $dbh->errstr();
31: my $last1; my $last3; my $last4; my $last5;
32: my $pos; my $size; my $base; my $identpos;
33:
34: # Set up a callback that collects links and frames
35: sub callback {
36:   my($tag, %attr) = @_;
37:   # no images, javascripts or s.th. else
38:   return if (($tag ne 'a') and ($tag ne 'frame'));
39:   while (my ($key, $value) = each %attr) {
40:     if (index($value, '#') != -1) # Truncating ankers
41:       { $value = substr($value, 0, (index($value, '#'))); }
42:     # extract endings for comparison
43:     $last1 = substr($value, -1, 1);
44:     $last3 = uc(substr($value, -3, 3));
45:     $last4 = uc(substr($value, -4, 4));
46:     $last5 = uc(substr($value, -5, 5));
47:     # good extensions, directory only and no funny ?-pages
48:     if ((index($value, '?') == -1)          # no question mark!
49:       and (($last1 eq '/')                  # and ending is "..."
50:        or ($last3 eq '.DE')
51:        or ($last3 eq ':80')
52:        or ($last4 eq '.HTM')
53:        or ($last4 eq '.ASP')
54:        or ($last4 eq '.JSP')
55:        or ($last4 eq '.PHP')
56:        or ($last5 eq '.HTML')
57:        or (index($value,'.') == -1)))
58:       { push(@links, $value); }
59:   } # end while(38)
60: } # end sub(34)
61:
62: # initialize HTML::LinkExtor with callback routine
63: $p = HTML::LinkExtor->new(\&callback);
64: # put start-URL in the database
65: $insert->execute($url, $countfast) or die $dbh->errstr();
66: $countfast++;
67: do { # fetch new URL
```

```
 68:   $selnextURL->execute($countslow) or die $dbh->errstr();
 69:   my @result = $selnextURL->fetchrow_array;
 70:   $countslow++;
 71:   $url = $result[0];
 72:   # Request document
 73:   my $request = HTTP::Request->new(GET => $url);
 74:   my $response = $ua->request($request);
 75:   if ($response->is_success) { # Get its size and parse it
 76:     $size = length($response->content);
 77:     $p->parse($response->content);
 78:     # Expand all (partial) URLs to absolute ones
 79:     $base = $response->base;
 80:     @links = map { $_ = url($_, $base)->abs; } @links;
 81:     foreach $link (@links) { # enter new links in the database
 82:       $identpos = index(uc($link),$ident);
 83:       if ((uc(substr($link,0,5)) eq 'HTTP:')
 84:             and ($identpos != -1)) {
 85:         $pos = index($link,":80");
 86:         if ($pos != -1) { # cut off Port 80 if necessary
 87:           $link = substr($link,0,$pos).
 88:                   substr($link,$pos+3,length($link)-3-$pos); }
 89:         # add ending slashes to directories or root if necessary
 90:         if ((substr($link, -1, 1) ne '/')
 91:               and ((rindex($link, '/') > rindex($link, '.'))
 92:               or (uc(substr($link, -3, 3)) eq ".DE")))
 93:           { $link = $link."/"; }
 94:         # no other ports? -> then go ahead!
 95:         if (index(uc($link),".DE:") == -1) {
 96:           $findURL->execute($link) or die $dbh->errstr();
 97:           my @counter = $findURL->fetchrow_array;
 98:           if (not defined ($counter[0])) {
 99:             push @linklist, $countfast;
100:             $insert->execute($link, $countfast) or die $dbh->errstr();
101:             $countfast++;
102:           } else {
103             for $item (@linklist)
104:               { if( $counter[0] eq $item ) { $found = 1; last; } }
105:             if ($found == 0) { push @linklist, $counter[0]; }
106:             else { $found = 0; }
107:         } # end if(95)
108:       } # end if(83)
109:     } # end foreach(81)
110:     $update->execute(join(':', sort(@linklist)), $size, $url);
```

```
111:     @links = (); @linklist = (); # Flush'em
112:   } else { $errors++; } #end if(74)
113:   if (($countslow % 100) == 0)
114:     {print $countslow," of ",$countfast," done! ",$errors," errors.\n";}
115: } until ($countfast == $countslow);
116:
117: # finish & disconnect from database
116: $update->finish();
117: $insert->finish();
118: $selnextURL->finish();
119: $findURL->finish();
120: $dbh->disconnect();
```

Literaturverzeichnis

[1] James Abello, Adam L. Buchsbaum, Jeffery Westbrook, **A Functional Approach to External Graph Algorithms**, Proceedings of the 6th Annual European Symposium on Algorithms, LNCS 1461, Seite 332–343, Springer 1998; http://www.research.att.com/~alb/biblio/func-esa98.ps.Z

[2] James Abello, Irene Finocchi, Jeffrey Korn, **Graph Sketches**, Proceedings of InfoVis 2001, Seite 67–70; http://www.mgvis.com/Papers/Visualization/graph_sketches.pdf

[3] James Abello, Jeffrey Korn, **MGV: A System for Visualizing Massive Multidigraphs**, IEEE Transactions on Visualization and Computer Graphics 2002 Vol. 8 No. 1, Seite 21–38; http://www.research.att.com/areas/visualization/papers_videos/papers/2002ak.pdf

[4] James Abello, Mauricio G.C. Resende, Sandra Sudarsky, **Massive Quasi-Clique Detection**, Latin American Theoretical INformatics 2002, LNCS 2286, Seite 598–612; http://www.research.att.com/~mgcr/doc/latin2002.pdf

[5] V. Batagelj, A. Mrvar, M. Zaveršnik, **Partitioning Approach to Visualization of large Graphs**, Proceedings of the 7th International Graph Drawing Symposium 1999, LNCS 1731, Seiten 90–97; http://vlado.fmf.uni-lj.si/pub/networks/doc/part/CorePart.pdf

[6] G. di Battista, P. Eades, R. Tamassia, I.G. Tollis, **Graph Drawing**, Prentice Hall 1999

[7] Endika Bengoetxea, **Inexact Graph Matching Using Estimation of Distribution Algorithms**, PhD Thesis, 2002; http://www.sc.ehu.es/acwbecae/ikerkuntza/these/thesis.pdf

[8] Jacques Bertin, **Graphics and Graphic Information-Processing**, Walter de Gruyter 1981

[9] T. Bladh, D. A. Carr, J. Scholl, **Extending Tree-Maps to Three Dimensions — A Comparative Study**, Proceedings of the 6th Asia-Pacific Conference on Computer-Human Interaction (APCHI 2004); http://www.sm.luth.se/csee/csn/publications/APCHI04Web.pdf

[10] Susanne Biermann, **Entwicklung eines Visualisierungsansatzes zur Unterstützung der Simulation in der Systembiologie**, Diplomarbeit 2003, Institut für Informatik, Universität Rostock

[11] S. Biermann, L. Uhrmacher, H. Schumann, **Supporting Multi-level Models in System Biology by Visual Methods**, Proceedings of the 18th European Simulation Multiconference (ESM 2004); http://wwwicg.informatik.uni-rostock.de/~schumann/papers/2004+/bierman.pdf

[12] Ulrik Brandes, **A Faster Algorithm for Betweenness Centrality**, Journal of Mathematical Sociology 2001, Seiten 163–177; http://www.inf.uni-konstanz.de/algo/publications/b-fabc-01.pdf

[13] Ulrik Brandes, Dorothea Wagner, **visone — Analysis and Visualization of Social Networks**, in Michael Jünger und Petra Mutzel (Hrsg.): Graph Drawing Software, 321–340, Springer-Verlag, 2003; http://www.inf.uni-konstanz.de/algo/publications/bw-vavsn-03.pdf

[14] Andreas Brandstädt, **Graphen und Algorithmen**, Teubner, 1994

[15] C. Buchheim, M. Jünger, S. Leipert, **Improving Walker's Algorithm to Run in Linear Time**, Proceedings of the 10th International Graph Drawing Symposium 2002, LNCS 2528, Seite 344–353; http://www.zaik.uni-koeln.de/~paper/unzip.html?file=zaik2002-431.ps

[16] Thomas Bürger, **Magic Eye View: Eine neue Fokus + Kontext Technik zur Darstellung von Graphen**, Diplomarbeit 1999, Institut für Informatik, Universität Rostock

[17] T. Carpenter, G. Karakostas, D. Shallcross, **Practical Issues and Algorithms for Analyzing Terrorist Networks**, Proceedings of the Western Simulation MultiConference 2002; http://www.cas.mcmaster.ca/~gk/papers/wmc2002.pdf

[18] Hsichun Chen, et al., **Crime Data Mining: A General Framework and Some Examples**, IEEE Computer, 37(4), 50–56, 2004; http://www.business.hku.hk/~mchau/papers/CrimeDataMiningFramework.pdf

[19] Martin Dodge, **Mapping Peer-to-Peer Networks**, 11. April 2003; http://www.cybergeography.org/maps/maps26.html

[20] Peter Eades, Mao Lin Huang, **Navigating Clustered Graphs using Force-Directed Methods**, Journal of Graph Algorithms and Applications Vol. 4 No. 3 (2000), Seite 157–181; http://www.cs.brown.edu/publications/jgaa/accepted/00/EadesHuang00.4.3.pdf

[21] J. Edachery, A. Sen, F. J. Brandenburg, **Graph Clustering using Distance-k Cliques**, Proceedings of the 7th International Graph Drawing Symposium 1999, LNCS 1731, Seiten 98–106; http://www.public.asu.edu/~halla/papers/gd996.ps

[22] David Eppstein, Joseph Wang, **Fast Approximation of Centrality**, Poceedings of the Symposium on Discrete Algorithms 2001, Seite 228–229; http://www.ics.uci.edu/~eppstein/pubs/EppWan-SODA-01.pdf

[23] Y.-H. Fua, M.O. Ward, E.A. Rundensteiner, **Navigating Hierarchies with Structure-Based Brushes**, Proceedings of InfoVis 1999, Seite 58–64; http://davis.wpi.edu/~xmdv/docs/infovis99_SBB.pdf

[24] George W. Furnas, **The FISHEYE view: A new look at structured files**, Bell Laboratories Technical Memorandum #82-11221-22, October 1982; http://www.si.umich.edu/~furnas/Papers/FisheyeOriginalTM.pdf

[25] Thomas M.J. Fruchterman, Edward M. Reingold, **Graph Drawing by Force-directed Placement**, Software — Practice and Experience Vol. 21 No. 11 (1991), Seiten 1129–1164; http://www.cs.ubc.ca/local/reading/proceedings/spe91-95/spe/vol21/issue11/spe060tf.pdf

[26] Pawel Gajer, Stephen G. Kobourov, **GRIP: Graph dRawing with Intelligent Placement**, Proceedings of the 8th International Graph Drawing Symposium 2000, LNCS 1984, Seiten 222–228; http://www.cs.arizona.edu/~kobourov/grip_demo.pdf

[27] M. Ganesh, E.H. Han, V. Kumar, S. Shekhar, J. Srivastava, **Visual Data Mining: Framework and Algorithm Development**, Technical Report TR-96-021, Department of Computer Science, University of Minnesota, Minneapolis, 1996; ftp://ftp.cs.umn.edu/dept/users/kumar/datavis.ps

[28] Frank van Ham, Jarke J. van Wijk, **Beamtrees: Compact Visualization of Large Hierarchies**, InfoVis Proceedings 2002, Seite 93–100; http://www.win.tue.nl/~fvham/beamtrees/Downloads/vanHamBeamtrees.pdf

[29] I. Herman, M.S. Marshall, G. Melançon, **Graph Visualization and Navigation in Information Visualization: a Survey**, IEEE Trans. on Visualization and Computer Graphics 2000, Vol. 6, No. 1, Seite 24–43; http://homepages.cwi.nl/~ivan/AboutMe/Publications/StarGraphVisuInInfoVis.pdf

[30] I. Herman, M.S. Marshall, G. Melançon, **Automatic generation of interactive overview diagrams for the navigation of large graphs**, Reports of the Centre for Mathematics and Computer Sciences, INS-0014 (2000); http://ftp.cwi.nl/CWIreports/INS/INS-R0014.pdf

[31] Vaughan Hobbs, Darius Pfitzner, David Powers, **A Unified Taxonomic Framework for Information Visualization**, Proceedings of the Australian Symposium on Information Visualisation 2003, Seite 57–66; http://crpit.com/confpapers/CRPITV24Pfitzner.pdf

[32] Daniel A. Keim, **Information Visualization and Visual Data Mining**, IEEE Transactions on Visualization and Computer Graphics 2002, Vol. 8 No. 1, Seite 1–8; http://fusion.cs.uni-magdeburg.de/pubs/TVCG02.pdf

[33] Y. Koren, L. Carmel, D. Harel, **ACE: A Fast Multiscale Eigenvector Computation for Drawing Huge Graphs**, InfoVis Proceedings 2002, Seite 137–144; http://www.research.att.com/~yehuda/pubs/ace_journal.pdf

[34] Matthias Kreuseler, Norma López, Heidrun Schumann, **A Scalable Framework for Information Visualization**, InfoVis Proceedings 2000, Seite 27–36; http://wwwicg.informatik.uni-rostock.de/~mkreusel/SInVis/vis00.pdf

[35] Matthias Kreuseler, Heidrun Schumann, **A Flexible Approach for Visual Data Mining**, IEEE Transactions on Visualization and Computer Graphics 2002 Vol. 8 No. 1, Seite 39–51; http://csdl.computer.org/comp/trans/tg/2002/01/v0039abs.htm

[36] Matthias Kreuseler, Thomas Nocke, Heidrun Schumann, **A History Mechanism for Visual Data Mining**, IEEE Information Visualization, InfoVis'04, Austin, Texas, USA, October 2004; http://wwwicg.informatik.uni-rostock.de/~nocke/download/KreuselerNockeSchumannHMfVDMInfoVis04.pdf

[37] Michihiro Kuramochi, George Karypis, **Discovering Frequent Geometric Subgraphs**, Proceedings of IEEE International Conference on Data Mining (ICDM'02), Seite 258–265; http://www-users.cs.umn.edu/~karypis/publications/Papers/PDF/gFSG.pdf

[38] John Lamping, Ramana Rao, Peter Pirolli, **A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies**, Conference on Human Factors in Computing Systems 1995; http://www.ramanarao.com/papers/startree-chi95.pdf

[39] W. Luebke, M. Richmond, J. Somervell, D.S. McCrickard, **An Extensible Framework for Information Visualization and Collection**, Proceedings of the ACM Southeast Conference (ACMSE '03), Seite 365–370; http://people.cs.vt.edu/~mccricks/papers/luebke-acmse03.pdf

[40] Peter Lyman, Hal R. Varian, **How Much Information 2003?**, UC Berkeley School of Information Management and Systems; http://www.sims.berkeley.edu/research/projects/how-much-info-2003/printable_report.pdf

[41] Jock D. Mackinlay, **Automating the Design of Graphical Presentations of Relational Information**, ACM Transactions on Graphics 5(2), April 1986, Seite 110–141; http://www2.parc.com/istl/projects/uir/pubs/items/UIR-1986-02-Mackinlay-TOG-Automating.pdf

[42] Joe Mattis, Steven F. Roth, **Data Characterization for Intelligent Graphics Presentation**, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 1990, Seite 193–200; http://portal.acm.org/citation.cfm?id=97273 (full doc available at CiteSeer)

[43] D.W. Matula, **Subtree isomorphism in $\mathbf{O(n^{5/2})}$**, Annals of Discrete Mathematics 1978, Seite 91–106.

[44] Wolfgang Müller, Heidrun Schumann, **Visual Data Mining**, NORSIGD Info 2/2002, Seite 4–7; http://wwwicg.informatik.uni-rostock.de/~schumann/papers/2002+/VDM_Norw.pdf

[45] Tamara Munzner, **H3: Laying Out Large Directed Graphs in 3D Hyperbolic Space**, Proceedings of InfoVis 1997, Seite 2–10; http://graphics.stanford.edu/papers/h3/h3.ps.gz

[46] Mark Newman, Michelle Girvan, **Community structure in social and biological networks**, Proceedings of the National Academy of Sciences (2002), Seiten 7821–7826; http://www.santafe.edu/sfi/publications/Working-Papers/01-12-077.pdf

[47] Thomas Nocke, **Metadatengewinnung und -spezifikation für Visualisierungsentscheidungen**, Diplomarbeit 2000, Institut für Informatik, Universität Rostock

[48] Thomas Nocke, Heidrun Schumann, **Goals of Analysis for Visualization and Visual Data Mining Tasks**, CODATA Prague Workshop Information Visualization, Presentation and Design, 29–31 March 2004; http://www.cgg.cvut.cz/infoviz/papers/nocke.pdf

[49] Maria Christina Ferreira de Oliveira, Haim Levkowitz, **From Visual Data Exploration to Visual Data Mining: A Survey**, IEEE Transactions on Visualization and Computer Graphics 2003, Vol. 9, No. 3, Seite 378–394; http://www2.cs.uregina.ca/~yang/CS805/FromVisual.pdf

[50] G. G. Robertson, J. D. Mackinlay, S. K. Card, **Cone Trees: Animated 3D Visualizations of Hierarchical Information**, Proceedings of ACM Conference on Human Factors in Computing Systems 1991, Seiten 189-194

[51] Arno Scharl, **Adaptive Web Representation**, Human Computer Interaction Development & Management, Ed. T. Barrier. Hershey, London: IRM Press. 255-260, 2002.

[52] Heidrun Schumann, Wolfgang Müller, **Informationsvisualisierung: Methoden und Perspektiven**, Information Technology 46 (2004) 3, Oldenbourg Verlag 2004

[53] Ron Shamir, Dekel Tsur, **Faster Subtree Isomorphism**, Israel Symposium on Theory of Computing Systems1997, Seite 126–131; http://www.math.tau.ac.il/~rshamir/papers/subtree.ps

[54] Jianbo Shi, Jitendra Malik, **Normalized Cuts and Image Segmentation**, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'97), Seiten 731–737; http://www.cs.berkeley.edu/~malik/papers/shi-malik97.pdf

[55] Ben Shneiderman, **Tree Visualization with Tree-Maps: A 2-D Space-Filling Approach**, ACM Transactions on Graphics, Vol. 11 No. 1 Januar 1992, Seite 92–99; http://portal.acm.org/citation.cfm?id=949654 (full doc available at CiteSeer)

[56] Ben Shneiderman, **The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations**, Technical Report 1996; http://techreports.isr.umd.edu/reports/1996/TR_96-66.pdf

[57] Robert Spence, **Information Visualization**, Addison Wesley 2001

[58] J. Stasko, R. Catrambone, M. Guzdial, K. McDonald, **An Evaluation of Space-Filling Information Visualizations for Depicting Hierarchical Structures**, International Journal of Human-Computer Studies, Vol. 53 No. 5, Nov. 2000, Seite 663–694; http://www.cc.gatech.edu/~john.stasko/papers/ijhcs00.pdf

[59] Soon Tee Teoh, Kwan-Liu Ma, **RINGS : A Technique for Visualization Large Hierarchies**, Proceedings of the 10th International Graph Drawing Symposium 2002, LNCS 2528, Seite 268–275; http://graphics.cs.ucdavis.edu/~steoh/research/graph/GD02.pdf

[60] Volker Turau, **Algorithmische Graphentheorie**, Addison-Wesley 1996

[61] Gabriel Valiente, Conrado Martínez, **An Algorithm for Graph Pattern-Matching**, Proceedings of the 4th South American Workshop on String Processing, Vol. 8 of International Informatics Series, Carleton University Press (1997), Seite 180–197; http://se2c.uni.lu/tiki/se2c-bib_download.php?id=946

[62] Denny Voigt, **WWW-basierte Darstellung komplexer Informationsstrukturen**, Diplomarbeit 2001, Institut für Informatik, Universität Rostock

[63] J.Q. Walker, II, **A node-positioning algorithm for general trees**, Software-Practice&Experience, Vol. 20 No. 7, Juli 1990, Seite 685–705; http://portal.acm.org/citation.cfm?id=79026

[64] M. Ward, J. Yang, **Interaction Spaces in Data and Information Spaces**, Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization, Konstanz 2004

[65] Christopher Westphal, Teresa Blaxton, **Data Mining Solutions: Methods and Tools for Solving Real-World Problems**, John Wiley & Sons, New York, 1998

[66] J. van Wijk, H. van de Wetering, **Cushion Treemaps: Visualization of Hierarchical Information**, Proceedings of InfoVis 1999, San Francisco, Seite 73–78; http://www.win.tue.nl/~vanwijk/ctm.pdf

[67] Veikko Wünsche, **Visualisierung strukturierter Informationen mit VRML**, Diplomarbeit 1997, Institut für Informatik, Universität Rostock

[68] K.-P. Yee, D. Fisher, R. Dhamija, M. Hearst, **Animated Exploration of Dynamic Graphs with Radial Layout**, Proceedings of InfoVis 2001, Seite 43–50; http://bailando.sims.berkeley.edu/papers/infovis01.pdf

Abbildungsverzeichnis

1.1 Darstellung von Telefonverbindungen 4

2.1 Ein Beispiel für eine implizit gegebene Informationsstruktur . . . 8
2.2 Automatisch generierte Euler-Venn-Diagramme zur Visualisierung einfacher Hypergraphen 9
2.3 Illustration des Ausgabe-Engpasses 13

3.1 Das Anfügen eines Knotens an einen 2-Baum und an einen „klassischen“ 1-Baum 19
3.2 Beispiel für ein Dendrogramm 20
3.3 Beispiel einer k-Core Dekomposition 22
3.4 Layout-Beispiel 1 24
3.5 Layout-Beispiel 2 25
3.6 Beispiel einer Visualisierung mit StepTree 26
3.7 Schematische Darstellung des Cushion-Prinzips anhand einer binären Intervallzerlegung und Visualisierung mit SequoiaView . . . 27
3.8 Sunburst-Darstellung zweier Verzeichnisstrukturen 27
3.9 2- und 3-dimensionale Darstellung mit der Beamtree-Technik . . . 28
3.10 Darstellung von Clusterhierarchien mit Hilfe des Magic Eye Views 29
3.11 Darstellung eines ConeTrees mit der Visualisierungssoftware FAHAM 30
3.12 Beispiele für Hyperbolic Browser und H3 30
3.13 Clusterhierarchien 31
3.14 Graph Sketch mit Überblicks- und Detailansicht 32
3.15 Zwei Intervalldarstellungen ein und desselben Graphen 32
3.16 Beispiel eines Permutationsgraphen und seines Permutationsdiagramms 33
3.17 Schematische Darstellung eines hierarchischen Federkraftmodells . 35
3.18 Darstellung von Netzwerken mit der Baumvisualisierungstechnik Magic Eye View 36
3.19 Die fünf Abhängigkeitstypen der Daten 38
3.20 Londons U-Bahnplan mit dem Stadtzentrum im Fokus 40
3.21 Beispiel der Kindsenke mit der Visualisierungstechnik RINGS . . 40
3.22 Interactive Overview Diagram mit Inhalts- und Navigationsansicht 41

3.23 Beispiel einer visuellen Anfrage mit der Software BibleWorks 4.0 . 43

4.1 Architekturvorschlag für ein Framework zum visuellen Data Mining 46
4.2 Modell des visuellen Data Mining Prozesses auf großen Strukturen 47
4.3 Framework zum visuellen Data Mining und konkrete Verfahren . . 49

5.1 Beispiele für die grafischen Ausgabemöglichkeiten von Cyram NetMiner 2.5 58
5.2 Beispiel für die grafische Ausgabe von Ucinet 6.0 59
5.3 Beispiel für die grafische Ausgabe von Pajek 1.0 59
5.4 Funktionsplot der Laufzeitkomplexitäten 62
5.5 Beispiel für eine Instanz der Klasse `cTable` 62
5.6 Die umgesetzten Module im Kontext des Frameworks 69
5.7 Beispiel zum Algorithmus zur Berechnung der Abhängigkeitszahlen 72
5.8 Beispiel für die exponentielle Zunahme der Zahl kürzester Wege . 73
5.9 Beispiele für die Inhaltsansicht und Interaktionstechniken 75
5.10 Steuerung komplexer Operationen über Dialoge 75
5.11 Die Detailansicht unter LINUX 77
5.12 Selektion eines Clusters und neue Ansicht mit dem selektierten Cluster als Wurzelknoten 78

6.1 Ein Ausschnitt der Textdatei mit den Daten des EAT 80
6.2 k-Neighborhood-Diagramm für das Wort PEANUTS 81
6.3 Explizite Darstellung eines markierten Clusters 81
6.4 k-Neighborhood-Größen der Startseite und eine gefilterte Inhaltsansicht 83
6.5 Rearrangement des Graphen aus Abbildung 6.4 84

Glossar

Ähnlichkeitsmaß 17, 38, 52ff., 85
Allrelation 7
Annotation 42, 68, 75
Attributfunktion 8
Attributmenge 8
Ausdruck, regulärer 42
average-linkage-Verfahren 22
Azyklizität 51

Baum, aufspannender 52
Baumähnlichkeit 18, 36ff., 51, 57, 67, 70, 76, 80, 85
Baumdarstellung 25ff., 85
Baumweite 19
Beamtree 28
betweenness 17, 73, 76
Binärsuche 61ff.
BinSort 70
breadth first search (BFS) 52, 66, 70ff.
Breitensuche 52, 66, 70ff.
brushing 39

cache 64
center 17
Clique 7, 50, 77, 84
closeness 16
Clustering 21, 52
  bottom-up 22
  top-down 21
compactness 18
Cone Tree 29f.
connectivity 17
Crime Data Mining 4
cross edge 36, 52, 76
Cushion Tree-Map 26

Darstellungsraum 16
Data Mining 3
  visuelles 4
data relationship 38, 51
data scrubbing 43
Daten, multirelationale 9, 61, 65
Datenaufbereitung 43
Deadlinks 55
degrees of interest (DOI) 41
Dekomposition 52
Dendrogramm 20, 53
density 18, 41, 52
dependency 17, 67, 71
diameter 17
Dichte 18, 41, 52
distance-k-clique 21, 50
Distanzmaß 21, 53
Durchmesser 17
Durchschnittsmaß 18

eccentricity 16
ends, dangling 55, 66, 80
expression, regular 42

feature visualization 59
Federkraftmodell 33ff., 52
  hierarchisches 35f.
Feedback-Arc-Set-Problem 12
Filterung 39, 68, 83
flow-betweenness 17, 50
Fokus+Kontext 39
Fruchterman-Reingold-Algorithmus 33ff., 67, 74

Graph 7, 10
dense 12
directed acyclic 12
schlichter 66
sparse 12, 53
graph drawing 15, 33
graph matching 23, 42, 54, 85
Graph Sketch 31f., 53

H3 30f.
Hamming-Abstand 53
Hash-Tabelle 64
history 42, 55, 69
Hyperbolic Browser 30f.
Hypergraph 7, 65
Hyperkante 7, 9, 65

Ikone 16, 32
information hiding 39
Informationsmenge 7
Informationsmodell 7f.
Informationsobjekt 7
Informationsraum 8
Informationsstruktur 8
implizite 8
I/O-Bottleneck 13, 39ff.
Interaktionstechniken 16, 42, 54
Intervallgraph 32

k-Baum 18
partieller 19
k-core 21, 67, 70, 79, 82f., 85
k-Nachbarschaft 16, 67, 70, 77, 81
k-neighborhood 16, 67
Kantenmenge 7
Kindsenken 40
Knotengrad 16
Knotenmenge 7
Knowledge Discovery 3
Kompaktheit 18
Komplementgraph 57
Kreisfreiheit 51

Laufzeitkomplexität 12, 50, 58, 60, 62, 73
Layout-Verfahren 24f., 38f.

MagicEye View 28f., 68, 77
Maße
globale 18
strukturelle 16
Maximalfluß 17
Maximum-Acyclic-Subgraph-Problem 12
Maximum-Clique-Problem 12, 23
Menge, unabhängige 7
Metadaten 16, 48
Minimum-Feedback-Arc-Set-Problem 12

Navigation 42
Netzwerkdarstellung 25, 31ff., 85
Nicht-Baum-Kanten 36, 52, 76
node degree 16
normalized association 22
normalized cut 21
NP-vollständig 11, 23
Nullrelation 7

Objektorientierung 60
overview+detail 41

pattern matching 23
Peer-To-Peer-Netzwerk 11, 86
Permutationsgraph 33
Petri-Netz 11
Post-Processing 54
Präsentationsraum 16

Quasicliquen 23
QuickSort 61

Radius 17
Rearrangement 42, 68, 75, 84
Relation 7
response 79ff.

Screen Bottleneck 13
Selektion 42, 54, 68, 78
simulated annealing 34, 74
single-linkage-Verfahren 22
single-source-shortest-path 66
Sortierung, topologische 51
Speicherkomplexität 12, 52, 60
stimulus 79ff.
Struktur
    dynamische 11, 86
    statische 11
Strukturdeskriptor 48, 51, 67
Strukturmenge 9
subgraph, frequent 23, 55
Subgraph-Isomorphism-Problem
        12, 23
Sunburst 27
swaps 12

Trägermenge 7
Transformationsoperation 57
treelikeness 18
Tree-Map 26f.
treewidth 19

Überblick+Detail 41
Undo+Redo 42, 68f.
uniform resource locator (URL) 82
Universalrelation 7

Visualisierungsverfahren 15, 24, 53

Walker-Algorithmus 29
Web Mining 3f.
Wertebereich 8

Zentralitätsmaß 17, 25, 41, 52f., 85
Zentrum 17, 25

Selbständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, den 30. September 2004

Thesen

1. Prozesse zur Entscheidungsfindung basieren heutzutage zunehmend auf Informationen, die aus umfangreichen elektronischen Datenbanken gewonnen wurden. Dabei ist nicht automatisch diejenige Entscheidung die beste, welche auf den meisten Daten basiert; vielmehr ist der Erfolg an eine geschickte Auswertung und Interpretation der Datenbasis geknüpft. Verfahren zur Unterstützung dieser Datenanalyse sind Gegenstand aktueller Forschung.

2. Dazu zählt auch der moderne Ansatz des visuellen Data Minings (VDM), der automatische Berechnungsmethoden und Techniken der Informationsvisualisierung verbindet. Die Untersuchung struktureller Abhängigkeiten innerhalb der Daten mit Hilfe des VDM wurde bislang jedoch weder untersucht, noch umfassend implementiert.

3. Diese strukturellen Abhängigkeiten können durch Relationen beschrieben und damit als Graphen modelliert werden. Multirelationale Datensätze werden dabei von verschiedenen Arten struktureller Abhängigkeit gebildet.

4. Während beim herkömmlichen VDM in erster Linie statistische Methoden zur Charakterisierung der Daten verwendet werden, eignen sich beim VDM auf Strukturen zu diesem Zweck graphentheoretische Verfahren.

5. Die Darstellung der Strukturen kann mit allen bekannten Visualisierungstechniken für Hierarchien oder Netzwerke erfolgen. Eine flexible Parametrisierung dieser Verfahren ist z.B. mit Baumähnlichkeitsmaßen möglich.

6. Diese Ideen greift das entwickelte Framework für das VDM auf Strukturen auf und vereint sie in einem allgemeinen Konzept. Der Hauptfokus des Frameworks liegt dabei vornehmlich auf der Verarbeitung besonders großer, komplexer Strukturen, wie sie auch in der Praxis auftreten.

7. Da einige Anwendungsgebiete z.T. spezielle Analyseverfahren benötigen, wurde für das flexibel einsetzbare Framework ein modular erweiterbares Softwaredesign gewählt. Eine Basisfunktionalität ist durch die Realisierung grundlegender Verfahren gegeben.

8. Die Leistungsfähigkeit dieses Ansatzes wurde durch die Analyse zweier Datensätze aus der Praxis exemplarisch belegt.