visone

Software for the Analysis and Visualization of Social Networks

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Michael Baur

aus Ravensburg

Tag der mündlichen Prüfung:	13. November 2008
Erste Gutachterin:	Frau Prof. Dr. Dorothea Wagner
Zweiter Gutachter:	Herr Prof. Dr. Ulrik Brandes

Contents

1 Zusammenfassung 7

1.1 Überblick 7

1.2 Grundlagen von visone 8

1.3 Analyse 8

1.4 Visualisierung 9

1.5 Weitere Funktionen 11

2 Motivation 13

3 visone Basics 17

3.1 Graph Theory 18

3.2 Network Model 21

3.2.1 Direction 21

3.2.2 Confirmation 22

3.2.3 Weights and Attributes 23

3.2.4 Data Representation 24

3.3 visone User Interface 24

3.3.1 Main Window 25

3.3.2 Graphical Properties and Templates 27

3.3.3 Geometric Transformations 27

3.3.4 Selections 28

3.3.5 Edge Transformations 29

3.3.6 Attributes 31

4 Analysis 33

4.1 Characteristics of Input Data 33

4.2 Size of the Graph 35

4.3 Centrality 35

4.3.1 Normalization . . . 36
4.3.2 Degree . . . 36
4.3.3 Distance . . . 37
4.3.4 Shortest Paths . . . 38
4.3.5 Shortest-Path Edge Centralities . . . 40
4.3.6 Current-Flow . . . 41
4.3.7 Feedback . . . 43
4.4 Neighborhood Density . . . 47
4.5 Cohesiveness . . . 49
4.5.1 k-core . . . 49
4.5.2 Cliques & Co . . . 50
4.6 Connectedness . . . 51
4.7 Clustering . . . 52
4.7.1 Quality Measures . . . 53
4.7.2 Clustering Methods . . . 56
4.7.3 Conclusion . . . 58
4.8 Network Statistics . . . 59

5 General Graph Layout 63
5.1 Preliminaries . . . 64
5.2 Criteria of Good Layouts . . . 64
5.3 Random Layout . . . 65
5.4 Circular Layout . . . 65
5.5 Force-Directed Layouts . . . 66
5.5.1 Classical Force-Directed Layout . . . 68
5.5.2 Classical MDS . . . 69
5.5.3 Stress Majorization . . . 70
5.5.4 Conclusion . . . 72
5.6 Spectral Layout . . . 72

6 Analytic Visualization 75
6.1 Basic Visualizations . . . 76
6.1.1 Sizes . . . 76
6.1.2 Colors . . . 77

6.1.3 Labels 80
6.1.4 z-Coordinate 81
6.2 Radial Visualization 81
6.2.1 Layout Algorithm 82
6.2.2 Final Placement 83
6.2.3 Confirmation 83
6.2.4 Scale 85
6.2.5 User Interaction 85
6.2.6 Final Remarks 86
6.3 Status Visualization 87
6.3.1 Layout Algorithm 87
6.3.2 Final Positions 89
6.3.3 Confirmation 89
6.3.4 Scale 90
6.3.5 User Interaction 90
6.3.6 Final Remarks 91
6.4 Multi-Circular Visualization 91
6.4.1 Macro-Level Layout 93
6.4.2 Micro-Level Layout Algorithm 93
6.4.3 Final Placement 94
6.4.4 Final Remarks 95
6.5 Additional Positional Visualizations 95
6.5.1 Coordinate Visualization 95
6.5.2 Group Visualization 96
6.6 Examples 97
6.6.1 The Network of the Deutschland AG 97
6.6.2 Email Communication Network 98

7 Layout Algorithms 101
7.1 Circular Crossing Reduction 101
7.1.1 Introduction 101
7.1.2 Preliminaries 102
7.1.3 Initial Greedy Layout 103
7.1.4 Improvement by Circular Sifting 105

7.1.5 Constraint Crossing Reduction . . . 107
7.1.6 Experimental Evaluation . . . 109
7.1.7 Conclusion . . . 111
7.2 Multi-Circular Layout of Micro/Macro Graphs . . . 112
7.2.1 Preliminaries . . . 112
7.2.2 Macro Layout . . . 113
7.2.3 Related (Micro-) Layout . . . 114
7.2.4 Multi-Circular Layout . . . 115
7.2.5 Layout Algorithms . . . 118
7.2.6 Conclusion . . . 123
7.3 Visualizing Hierarchies in Two and a Half Dimensions . . . 123
7.3.1 Introduction . . . 124
7.3.2 Preliminaries . . . 124
7.3.3 Layout Method . . . 125
7.3.4 Results . . . 126
7.3.5 Conclusion . . . 129

8 Network Models 133
8.1 Fundamental Models . . . 133
8.1.1 General Random Graphs . . . 133
8.1.2 Small World . . . 135
8.1.3 Preferential Attachment . . . 136
8.2 k-Core Generation using PA . . . 137
8.2.1 Preliminaries . . . 139
8.2.2 Core Generator . . . 141
8.2.3 Modeling the AS Network . . . 147
8.2.4 Conclusion . . . 155

Bibliography 157

Index 177

Appendix 179

Résumé 189

Chapter 1

Zusammenfassung

1.1 Überblick

Seit einigen Jahren werden wirtschaftliche und gesellschaftliche Entwicklungen immer häufiger unter dem Aspekt der *“Vernetzung”* betrachtet. Für das Verständnis von Ereignissen ist dabei nicht mehr die institutionelle Position des Einzelnen, sondern die Verbindungen der Handelnden untereinander und das sich daraus ergebende Netzwerk entscheidend. Außer in sozialen Prozessen treten Netzwerke auch in vielen weiteren Bereichen auf, beispielsweise als elektronische Schaltkreise, technische Kommunikationsnetze oder als weltumspannende Transportnetzwerke. Entsprechend diesem breiten Anwendungsbereich sind die relevanten Fragestellungen sehr vielfältig und die betrachteten Netzwerke unterscheiden sich deutlich in Größe und Ausprägung. Bereiche der Informatik, die Methoden für die Netzwerkanalyse bereitstellen können, sind offensichtlich die Graphentheorie, und da sich die Visualisierungen der Netzwerke anbietet, das Graphenzeichnen, aber auch viele andere Bereiche wie Algorithmik und Kombinatorik.

Entsprechend der gestiegenen Relevanz wird auch geeignete Software zur Analyse und Visualisierung von Netzwerken wichtiger. In dieser Arbeit wird die Software visone für diese Aufgabe vorgestellt, die kostenlos von der Homepage [vpt] bezogen werden kann. Obwohl das Projekt ursprünglich in Kooperation mit Sozialwissenschaftlern gestartet wurde und deshalb den Begriff *soziale Netzwerkanalyse* im Namen führt, lag ein Schwerpunkt der Entwicklung immer auf der allgemeinen Anwendbarkeit. Da viele der verwendeten Modelle und Methoden universell sind, ist eine strikte Abgrenzung auch kaum möglich und sinnvoll. Wichtige weitere Designziele sind

- die Bereitstellung eines geeigneten universellen Dateiformats,
- exakt definierte und allgemein anwendbare Analyseverfahren,
- Netzwerkvisualisierungen, die sowohl die Analyseergebnisse exakt wiedergeben als auch die Netzwerkstruktur übersichtlich darstellen und
- eine für nicht-technische Benutzer verständliche Bedienung.

visone ermöglicht durch die vollständig integrierte graphische Anzeige- und Editorkomponente einen fast spielerischen Zugang zur Netzwerkanalyse. Speziell entwickelte Visualisierungen und eine flexible Verknüpfung mit den Analysemethoden ermöglichen schnelles Experimentieren und Überprüfen verschiedener Hypothesen.

Andere oft verwendete Programme zur Netzwerkanalyse sind UCINET und Pajek. UCINET ist ein sehr umfangreiches Programm, dass viele verschiedene Arten von Analyseverfahren bereitstellt. Das Hauptprogramm ist allerdings vollständig Text- beziehungsweise Datei-basiert und die separate Visualisierungskomponente enthält deutlich weniger Funktionalität. In Pajek ist die graphische Darstellung integriert, erlaubt aber nur eingeschränkte Editiermöglichkeiten und erschwert somit für Einsteiger die Anwendung der zahlreichen Funktionen.

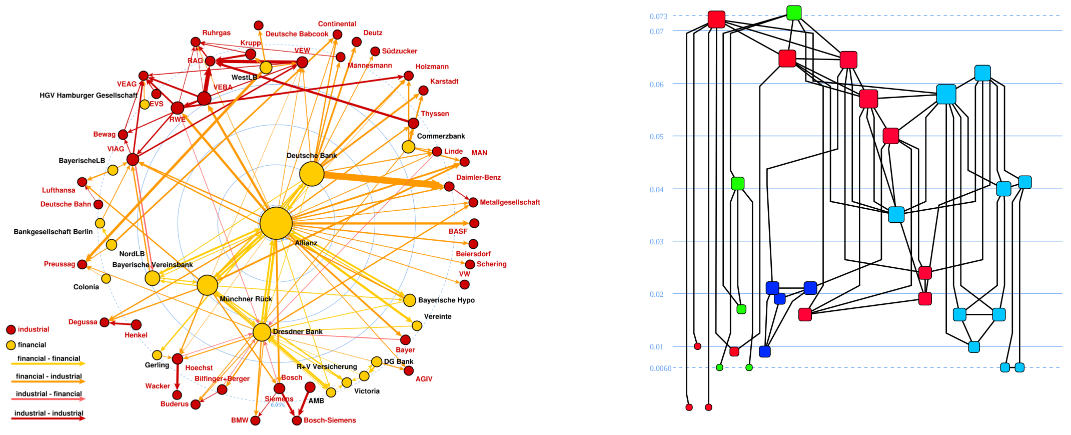
1.2 Grundlagen von visone

Da der Begriff *"Netzwerk"* unterschiedlichste Assoziationen weckt, wird im Folgenden die strenge Terminologie der Graphentheorie verwendet. Das in visone verwendete allgemeine Graphenmodell deckt fast alle denkbaren Anwendungsgebiete ab und ermöglicht unter anderem gerichtete und ungerichtete Kanten, Schleifen und Multi-Kanten zwischen einem Knotenpaar. Zusätzlich können in sogenannten *Attributen* beliebige weitere Daten für Knoten und Kanten verwaltet werden, beispielsweise Kantengewichte, vorgegebene externe Knotengruppierungen oder textuelle Bezeichnungen. Geeignete Attribute können als Eingabeparameter für Analyse- und Visualisierungsmethoden verwendet werden. Zur dauerhaften Speicherung wird das standardisierte XML-basierte Dateiformat GraphML verwendet, in dem sich alle Daten abbilden lassen.

1.3 Analyse

Für die Analyse der Netwerke stehen in visone sowohl neue, in den Arbeitsgruppen von Prof. Wagner und Prof. Brandes entwickelte Verfahren, als auch klassische, häufig verwendete Methoden zur Verfügung. Alle Analysemethoden wurden, wenn nötig, für das allgemeine Graphmodell generalisiert. Entsprechend dem analysierten Objekt werden die Verfahren oft in die drei Kategorien *Element-* (Knoten- und Kanten), *Gruppen-* und *Netzwerk-Ebene* eingeteilt. Folgende Analysearten sind in visone enthalten:

- **Elementebene:** Zentralitätsmaße, z. B. Closeness und Betweenness, und das lokale Dichtemaß Clustering Coefficient
- **Gruppenebene:** Clusterung, Indentifizierung von strukturell dichten Gruppen, z. B. Cliquen, Zusammenhangskomponenten und strukturelle Rollen



(a) Radialvisualisierung (b) Statusvisualisierung

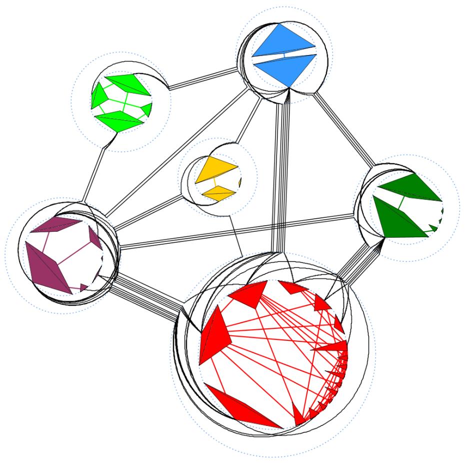
Abbildung 1.1: Beispiele für Radial- und Statusvisualisierungen. Die Knotenpositionen geben jeweils Zentralitätsmaße wieder. Durch Farbe, Form, Knotengröße und Kantenbreite werden weitere Eigenschaften visualisiert.

- **Netzwerkebene:** allgemeine Grapheigenschaften und Statistiken der Maße der Elementebene

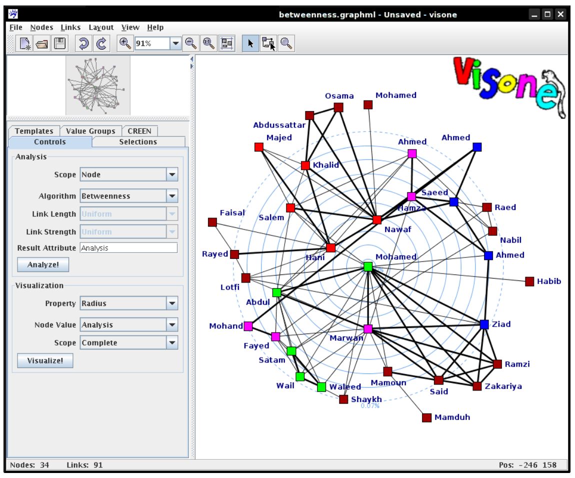
Ausgewählt wurden nur Verfahren, die häufig verwendet werden, allgemeine Relevanz besitzen und sich algorithmisch effizient berechnen lassen. Die Ergebnisse einer Analyse werden als Attribute gespeichert und sind dadurch flexibel als Eingabeparameter für Visualisierungmethoden verwendbar.

1.4 Visualisierung

Mit dem Begriff *"Visualisierung"* werden in visone Darstellungsverfahren bezeichnet, die ein Analyseergebnis bzw. Attributwerte akkurat wiedergeben. Einfache und naheliegende Visualisierungen sind die proportionale Skalierung von Knotengröße oder Kantendicke bezüglich eines numerischen Maßes und die Abbildung einer Clusterung auf verschiedene Farben. Fortgeschrittene Verfahren ändern das eigentliche Layout des Graphen, also die Position der Knoten und den Verlauf der Kanten, um Attributwerte wiederzugeben. Typischerweise werden dabei die grundlegende Layoutart und gewisse Nebenbedingungen festgelegt, in deren Rahmen für die Lesbarkeit und Ästhetik der Darstellung relevante Kriterien, etwa die Anzahl der Kantenkreuzungen oder die Varianz der Kantenlängen, optimiert werden müssen. Im Allgemeinen ergeben sich sehr komplexe Optimierungskriterien, die meist sogar $\mathcal{NP}$-schwer sind. Visualisierungen, die auf unterschiedliche graphische Eigenschaften abbilden, lassen sich problemlos kombinieren, um verschiedene Attribute gleichzeitig darzustellen.



(a) Multi-Kreis-Visualisierung



betweenness.graphml - Unsaved - visone

Scope: Node
Algorithm: Betweenness
Link Length: Uniform
Link Strength: Uniform
Result Attribute: Analysis

Property: Radius
Node Value: Analysis
Scope: Complete

Nodes: 34 Links: 91 Pos: -246 158

(b) die Benutzeroberfläche von visone

Abbildung 1.2: (a) Multi-Kreis-Visualisierung eines Netzwerks mit sechs Gruppen. Die Gruppenstruktur ist klar erkennbar. Knotenhöhe und -breite geben die Anzahl der Kanten innerhalb der Gruppe und zu anderen Gruppen an. (b) Oberfläche von visone mit großer Graphansicht, kleiner Übersichtsanzeige und Bereich für Eingabekomponenten.

Radialvisualisierung Bei der *Radialvisualisierung* werden die Knoten proportional zu einer numerischen Bewertung näher im Zentrum der Zeichnung platziert (siehe Abb. 1.1(a)), Knoten mit gleichem Wert liegen also auf dem gleichen Orbit. Optimierungskriterien sind die Reduzierung der Kreuzungsanzahl und eine gleichmäßige Verteilung der Knoten in der Fläche [BB04a].

Lagenvisualisierung Die *Lagenvisualisierung* dient ebenfalls zur Darstellung numerischer Bewertungen, Knoten mit gleichem Wert werden aber nicht wie im Radiallayout auf Orbits, sondern auf horizontalen oder vertikalen Lagen platziert (siehe Abb. 1.1(b)). Für entsprechende Layouts mit disjunkten Lagen existieren einige Verfahren zur Kreuzungsreduzierung, die auf dem bekannten Sugiyama-Framework basieren. Die neue Herausforderung besteht hier in einer guten Umwandlung der kontinuierlichen Positionen in disjunkte Lagen und der Behandlung langer Kanten.

Multi-Kreis-Visualisierung Ein Verfahren zur Darstellung von Clusterungen und anderer disjunkter Gruppierungen ist das *Multi-Kreis-Layout* [BB08], welches gleichzeitig die Elementebene des Graphen und die Gruppenstruktur, also die Cluster und ihre Verbindungen (siehe Abb. 1.2(a)) zeigt. Für die Gruppenstruktur wird ein Layout erstellt, in dem die Cluster genügend Fläche für ihre Knoten reservieren und ihre Verbindungen breit genug sind, um die entsprechenden Kanten aufzunehmen. Die Knoten jedes Clusters werden auf einem Kreis innerhalb der reservierten Fläche platziert. Kanten innerhalb eines Clusters verlaufen innerhalb dieser Kreise und Kanten zwischen Clustern winden sich um diese Kreise herum, um dann dem Verlauf der Cluster-Verbindungen zu folgen. Eine kombinatorische Beschreibung dieser Layouts erlaubt es, Verfahren zur Reduzierung der Kreuzungsanzahl anzuwenden.

Allgemeine Gruppenvisualiserung Die Visualisierung nicht-disjunkter Gruppen wie der Cliquenzugehörigkeit ist ein schwieriges Problem, da sich üblicherweise viele paarweise Überlappungen mit komplexen Abhängigkeiten ergeben. In visone gibt es deshalb eine *Gruppenvisualisierung*, die in einem festen Layout jeweils eine ausgewählte Gruppe besonders hervorhebt. Dazu wird zuerst ein Kreislayout des Gesamtgraphen bestimmt und die Knoten der jeweils ausgewählten Gruppe unter Beibehaltung der anderen Positionen auf einem zweiten Kreis in der Mitte platziert. Zur Optimierung dieses Layouts können die Kreuzungsreduzierungstechniken der Radialvisualisierung verwendet werden.

1.5 Weitere Funktionen

Neben den unter die beiden genannten Bereiche fallende Methoden sind in visone noch viele andere für die Netzwerkanalyse wichtige Verfahren implementiert. Allgemeine Graphlayoutverfahren wie Kreis-, Spektral- und MDS-Layout vermitteln einen Eindruck der Struktur des Graphen und liefern teilweise schwache Analyseaussagen, das Spektrallayout etwa platziert strukturell ähnliche Knoten nahe beieinander. Durch umfangreiche Selektionsmöglichkeiten nach strukturellen und Attribut-basierten Kriterien lassen sich nahezu beliebige Teilgraphen bestimmen. Graphgeneratoren für die bekannten Modelle $G_{n,p}$, Small World, Preferential Attachment und für Graphen mit vorgegebener k-Core-Struktur [BGG$^+$07] erlauben das Erstellen von Zufallsgraphen, etwa um neue Methoden zu erproben.

Chapter 2

Motivation

Over the past years, “networking” has become a common aspect in the analysis of political and social processes. From this point of view, the key to understanding certain events is no longer the institutional role of an individual but the interconnections of actors and the resulting network. Besides social sciences, networks also occur in many other fields of application, for example in the design of integrated circuits, in telecommunication, in the global linkage of financial transactions, or in the planning of transportation routes. In accordance with this broad range of application, the relevant questions are manifold and the resulting networks are of great diversity in size and characteristic.

Along with the increased relevance of network analysis and the growing size of considered networks, adequate software for social network analysis is becoming more and more important. In this dissertation, we present the software tool visone[1], which was developed in the homonymous DFG-funded project, aiming to bring together efficient algorithms for methods of analysis and suitable graph drawing techniques for the visualization of networks.

The beginning of the visone project dates back to June 1996 when Ulrik Brandes, Patrick Kenis, Jörg Raab, Volker Schneider, and Dorothea Wagner set up an interdisciplinary project group at University of Konstanz for the visualization of social networks. In the course of this project, the ideas for analytic visualizations based on circular and hierarchical graph layouts were born [BKR+99, BW00b]. After experimental studies endorsed the usefulness and exploratory power of these methods [BRW01], the first version of the visone software was created at the Algorithmics Group in Konstanz [BBB+02] which complemented the visualizations with shortest-path and feedback centrality indices and basic graph layout algorithms. While this version served its purpose very well, limitations of the utilized user interface library prevented the addition of new features. Finally, in 2004, it was decided to develop the more comprehensive version of the software presented in this work on a more permissive basis.

[1] visone is available free of charge for academic and non-commercial purpose from the homepage `http://visone.info`.

visone was and is intended both as a testbed for the work of our groups and as an everyday tool for students and researchers in network analysis. Therefore, we adapt all algorithms to a consistent and comprehensive graph model and put in great efforts to provide a simple but flexible user interface hiding unnecessary complexity. In contrast to common tools like UCINET [Ana08], which present to the user only a matrix representation of the data, we build on the expressiveness and explanatory power of graph layouts and provide a complete graphical view of the network (see Figure 1.2(b)). Observations indicate that users enjoy the playful nature of our approach. Besides our original work which is describe below, we have included novel algorithms developed by other members of the project and by members of related groups at the universities of Karlsruhe and Konstanz, in order to cover fields like centrality indices [Bra08], clusterings [Gae07], structural similarity [Ler07], and spectral layouts [Fle07]. The functionality is completed by well-known commonly-used methods for network analysis.

Visualizing social networks is more than simply creating intriguing pictures, it is about generating learning situations: *"Images of social networks have provided investigators with new insights about network structure and have helped them communicate those insights to others"* [Fre00]. Additionally, inappropriate drawings of networks are misleading or at least confusing. Therefore, we pay special attention to the visualization of the networks. Selected general graph layout algorithms provide an uncluttered view on the network and reveal its overall structure, but the unique feature of visone are the analytic visualizations which exactly depict analysis results, like centrality scores and clusterings, by means of tailored and suggestive graph layouts (see Figures 1.1 and 1.2(a)). Combinatorial models of these visualizations allow for the optimization of esthetic properties to improve the expressiveness and exploratory power without changing their analytic signification. We focus on edge crossings, since these have a prominent effect on the layout quality and present improved crossing reduction heuristics.

Countless hours of work have been spend to the design of the software tool in order to offer a pleasant and productive user experience. Besides this, our main theoretical contributions are:

- the multi-circular visualization of partitions of the vertex set,
- an improved crossing reduction heuristic, used in the radial and status visualization,
- an experimental visualization in two-and-a-half dimensions, and
- a network model that uses a preferential-attachment process to establish a predefined nested decomposition of the vertices.

Structure

This dissertation is structured as follows.

Chapter 3: visone Basics After a brief compilation of fundamental graph-theoretical notation and concepts, we introduce the general network model used throughout visone and present the main features – apart from analysis and visualization – of the software tool.

Chapter 4: Analysis The available analysis methods, including centrality indices, identification of connected or cohesive groups, clustering algorithms, and network statistics, cover a wide range of applications. We paid special attention to utilizing efficient algorithms and we strived to implement them as general as possible, i. e., they should not impose unnecessary restrictions on the input data.

Chapter 5: General Graph Layout General graph layout algorithms convey a first impression of the structure of the network, and some of them even give a deeper structural insight. In visone, a circular, a spectral, and various force-directed layout algorithms are available.

Chapter 6: Analytic Visualization In this chapter we describe drawing methods for the exact representation of given values for the vertices or edges, e. g., previously computed analysis results. Besides basic visualizations of values as size or color of the elements, we also present layout models unique to visone which convey both the structure of the graph and the given values in suggestive and expressive drawings, namely the *radial*, the *status*, and the *multi-circular* visualization.

Chapter 7: Layout Algorithms Our advanced visualizations are partially based on new layout models, or at least on improved algorithms. We explicate an improved crossing reduction heuristic for circular layouts, a combinatorial model as well as crossing reduction heuristics for multi-circular layouts, and an experimental visualization in two-and-a-half dimension.

Chapter 8: Network Models visone provides efficient implementations of three of the most popular network models: general random graphs $\mathcal{G}(n, p)$, *small world*, and preferential attachment. These are accompanied by our newly developed graph generator which realizes a predefined k-core structure in a preferential-attachment process.

Parts of this work have been published in [BB04a, BB04b, BBGW04, BBGW05, BGG+07, BGG+08, BB08].

Chapter 3

visone Basics

Networks are used to describe relational data of various fields of application. While the basic concepts of graph theory apply to all of these fields, every application imposes specific restrictions, e. g., directedness/undirectedness, connectedness/unconnectedness, and weighted/binary relations, on the characteristics of the network. On the other hand, each graph-theoretic method is defined for graphs of specific characteristics only.

In common tools, users sometimes face the problem that the characteristic of their network data does not fit the requirements of an analysis method simply because the implementation is more restrictively than necessary: for example, eigenvector centrality is only available for undirected graphs in UCINET. Of course, a network can be converted manually to the required format, but this task is error-prone and forces the user to manage a large number of files for the same data. In visone, special attention was paid to defining methods for the widest possible range of network characteristics. In order to achieve this, we followed a two-fold approach. On the one hand, we carefully analyzed algorithms for finding straight-forward relaxations or meaning-preserving extensions of the original method. If, nevertheless, the data does not fit the requirements, we automatically transform the network in a reasonable way most likely intended by the user. Clearly, this neither liberates users from the decision whether a method is reasonable for their application or not nor does it prohibit the usage of a custom transformation.

In this chapter, we define the theoretical concepts of visone's network model, explain how this ties together analysis and visualization in a powerful, flexible, and easy-to-understand way, and finally describe the realization of these concepts in the graphical user interface. First, however, we give some fundamental definitions of graph theory.

3.1 Graph Theory

In this section, we introduce common notation and fundamental concepts of graph theory. We closely adhere to [BE05a] but exclude concepts not used in visone.

A *graph* $G = (V, E)$ consists of a set V of *vertices* and a set $E \subseteq V \times V$ of *edges*. The cardinality of V is usually denoted by n, the cardinality of E by m. An edge $e \in E$ connects its two *endvertices* $u, v \in V$. The vertex v is said to be *adjacent* to u and the edge e is *incident* to u and v. Adjacent vertices $u, v \in V$ are called *neighbors*. An edge is either *directed* $e = \{u, v\}$ or *undirected* $e = (u, v)$. For a directed edge $e = (u, v) \in E$, v is called the *head* or *target* of e and u is called its *tail* or *source*. An edge connecting a vertex to itself, is called a *loop*. We will assume all graphs to be loop-free unless specified otherwise.

A pair of vertices is called a *dyad* regardless whether it is connected by an edge or not. A *(graph) element* $x \in X \in \{V, E\}$ is either a vertex or an edge. This notion is used whenever a statement is valid for vertices and edges.

Multi-Graphs We may allow multiple edges between the same dyad by extending the edge set E to a multiset. Such edges are called *parallel edges* and graphs with parallel edges are called *multi-graphs*. For directed graphs, only edges having the same head and the same tail are parallel. A graph without parallel edges is called *simple*.

Weighted Graphs For many applications, it is useful to associate numerical values (weights) with the edges or vertices of a graph $G = (V, E)$. Numerical weights can be represented as a function $\omega : X \to \mathbb{R}$, $X \in \{V, E\}$, that assigns to each element $x \in X$ a weight $\omega(x)$. For a given subset X' of the vertices or edges, we use $\omega(X')$ as shorthand for the total weight of the elements in X', i. e., $\omega(X') = \sum_{x \in X'} \omega(x)$.

For the vast majority of analysis methods used in network analysis, only edge weights are relevant. However, a generalization of weights to *attributes* is a powerful and flexible way to associate any type of data with vertices and edges (see Section 3.2.3).

Mixed Graphs Graphs that can have directed edges as well as undirected edges are called *mixed graphs*. Such graphs are rarely required by an application and combinatorial methods almost always require either a strictly directed or undirected graph. However, often mixed graphs can be transformed into any of these types. The offer maximum flexibility, the network model of visone allows mixed graphs (see Section 3.2).

For a mixed graph $G = (V, E)$, the *underlying undirected graph* is the undirected graph with vertex set V that has an undirected edge $\{u, v\}$ between two vertices $u, v \in V$ for every edge $(u, v) \in E$. Its *simple* underlying undirected graph does not contain any parallel edges, i. e., for each adjacent dyad $u, v \in V$ all except one edge $\{u, v\}$ are discarded.

Subgraphs A graph $G' = (V', E')$ is a *subgraph* of the graph $G = (V, E)$, denoted by $G' \subseteq G$ if $V' \subseteq V$ and $E' \subseteq E$. A *(vertex) induced* subgraph for a given subset of vertices $V' \subseteq V$ contains exactly the edges of $e \in E$ that join vertices in V' and is denoted by $G[V']$. Similarly, the *(edge-)induced subgraph* for a given subset of edges $E' \subseteq E$ contains exactly the vertices which are endvertices of at least one edge in E' and is denoted by $G[E']$.

Density The *density* $\rho(G)$ of a graph $G = (V, E)$ is the fraction of the number of present edges to the maximum number of edges in a graph with the same number of vertices, i. e., the density is $\rho(G) = 2m/(n(n-1))$ if G is undirected and $\rho(G) = m/(n(n-1))$ if the graph is directed. The density of a subgraph $G' = (V', E')$ of G is defined analogously as $\rho(G') = 2|E'|/|V'|(|V'| - 1)$ if G' is undirected and as $\rho(G') = |E'|/|V'|(|V'| - 1)$ if G' is directed.

Graphs and subgraphs of density one, i. e., every vertex is connected to every other vertex by an edge, are called *complete.*

Degree In an undirected graph $G = (V, E)$, the *degree* $\deg(v)$ of a vertex $v \in V$ is the number of its incident edges. The set of incident edges of a vertex v is denoted by $E(v)$ and the set of neighbors of v is denoted by $N(v)$.

In a directed graph $G = (V, E)$, the *out-degree* $\text{outdeg}(v)$ of $v \in V$ is the number of edges in E that have source v. Accordingly, the *in-degree* $\text{indeg}(v)$ of $v \in V$ is the number of edges with target v. The set of edges with source v is denoted by $E_{\text{out}}(v)$ and the set of edges with target v by $E_{\text{in}}(v)$. The set of neighbors of v which are the source of the connecting edge is denoted by $N_{\text{out}}(v)$ and the set of neighbors which are the target of a connecting edge by $N_{\text{in}}(v)$.

If G is a multi-graph, parallel edges are counted according to their multiplicity in E. For a graph with given edge weight, the degree is generalized by summing over the weights of the respective edges.

Paths and Cycles A *walk* from vertex x_0 to vertex x_k is an alternating sequence $x_0, e_1, x_1, e_2, x_2, \ldots, x_{k-1}, e_k, x_k$ of vertices and edges, where $e_i = \{x_{i-1}, x_i\}$ in the undirected case and $e_i = (x_{i-1}, x_i)$ in the directed case. The length of a walk is defined as its number of edges. If context permits with introducing ambiguity, walks are specified in terms of vertices or edges only. A *path* is a walk in which every edge occurs only once, i. e., $e_i \neq e_j$ for $i \neq j$, and a path is a *simple path* if no vertex occurs twice, i. e., $x_i \neq x_j$ for $i \neq j$. A path ending with its first vertex, i. e., $x_0 = x_k$, is called a *cycle.* A cycle is a *simple cycle* if $x_i \neq x_j$ for $0 \leq i < j \leq k-1$.

Distance In the context of paths, a weight is also called *length.* For a given edge length $\ell : E \to \mathbb{R}$, the length $\ell(p)$ of a path p is defined as the total length values of the edges of p. A path from u to v in G is a *shortest path* if its length is the smallest possible among all paths from u to v. The length of a shortest path from u to v is also called the (shortest-path) distance $\mathrm{d}(u, v)$ of u and v

Shortest-Path Problems The single-source shortest-paths problem (SSSP) is to find shortest paths from a given (source) vertex s to all other vertices. For non-negative edge weights, this problem can be solved in time $\mathcal{O}(m + n \log n)$ using an efficient implementation of Dijkstra's algorithm [CLRS01]. Note that SSSP is not well-defined if the graph contains a cycle of negative length. For unweighted graphs, the problem can be solved in linear time using breadth-first search (BFS).

Connected Components An undirected graph $G = (V, E)$ is *connected* if every vertex can be reached from every other vertex, i. e., if there is a path from every vertex to every other vertex, and *disconnected* otherwise. A directed graph $G = (V, E)$ is *strongly connected* if there is a directed path from every vertex to every other vertex. A directed graph is called *weakly connected* if its underlying undirected graph is connected.

For a given undirected graph $G = (V, E)$, a *connected component* of G is an induced subgraph $G[V']$ that is connected and maximal, i. e., there is no connected subgraph $G[V'']$ with $V'' \supset V'$). For directed graphs, weakly and strongly connected components are defined analogously. For all three types of connectedness, checking whether a graph is connected and computing all its connected components can be done in time $\mathcal{O}(n + m)$ using modifications of depth-first search (DFS) or breadth-first search (BFS). For more details on connectedness, see Section 4.6.

Partitions A *partition* is a subdivision of the vertex set V into pairwise disjoint, non-empty subsets $V = V_1 \dot\cup \ldots \dot\cup V_k$. An edge $e = \{u, v\} \in V_i \times \in V_j$ is called an *intra-partition edge* if $i = j$ and *inter-partition edge* otherwise. The set of intra-partition edges of a partition V_i is denoted by $E(V_i)$, or E_i for short, the set of inter-partition edges of two partitions V_i, V_j by $E(V_i, V_j)$, or $E_{i,j}$ for short. A partition is called *trivial* if either $k = 1$ (*1-partition*) or $k = n$ (*singletons*). A partition with $k = 2$ is also called a *cut*. For a given edge weight ω, the weight of a cut $(V', V \setminus V')$ is defined as the sum of the weights of its inter-partition edges, i. e., $\omega((V', V \setminus V')) = \omega(E(V', V \setminus V'))$.

Graph Isomorphism Two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* (written as $G_1 \simeq G_2$) if there is a bijection $\phi : V_1 \to V_2$ with

$$\forall u, v \in V : \ \{u, v\} \in E_1 \Leftrightarrow \{\phi(u), \phi(v)\} \in E_2 \,.$$

Such a bijection is called an *isomorphism*. An isomorphism that maps a graph onto itself is called an *automorphism*. Usually two graphs are considered to be the same if they are isomorphic. Isomorphism and automorphism for directed graphs are defined analogously.

Graph Matrices Let $G = (V, E)$ be a simple directed graph with n vertices and m edges. The *adjacency matrix* $A(G)$ of G is an $n \times n$ matrix with entries

$$a_{u,v} = \begin{cases} 1 & \text{if } (u, v) \in E \,, \\ 0 & \text{otherwise.} \end{cases}$$

The rarely used *incidence matrix* $B(G)$ of G is a $n \times m$ matrix defined by

$$b_{v,e} = \begin{cases} -1 & \text{if } v \text{ is the source of } e, \\ 1 & \text{if } v \text{ is the target of } e, \\ 0 & \text{otherwise.} \end{cases}$$

For an undirected graphs G, the adjacency matrix $A(G)$ is symmetric and has entries $a_{u,v} = a_{v,u} = 1$ if u and v are adjacent. The entries $b_{v,e}$ of its incidence matrix $B(G)$ equal one if v is incident to the edge e. For weighted graphs, the non-zero entries of all of this matrices are $\omega(e)$ rather than 1, where $e \in E$ is the respective edge of the entry.

The (weighted) Laplacian matrix $L(G) \in \mathbb{R}^{n \times n}$ of an undirected graph G with edge weight ω is defined by the elements

$$l_{u,v} = \begin{cases} \sum_{w \in V} \omega(\{u,w\}) & \text{if } u = v \ , \\ -\omega(\{u,v\}) & \text{if } u \neq v \text{ and } \{u,v\} \in E \ , \\ 0 & \text{otherwise.} \end{cases}$$

In matrix notation, this can be expressed as $L(G) = D(G) - A(G)$, where $D(G)$ is the diagonal matrix of (weighted) vertex degrees, i. e., $d_{vv} = \deg(v)$. For unweighted graphs, unit weights can be assumed.

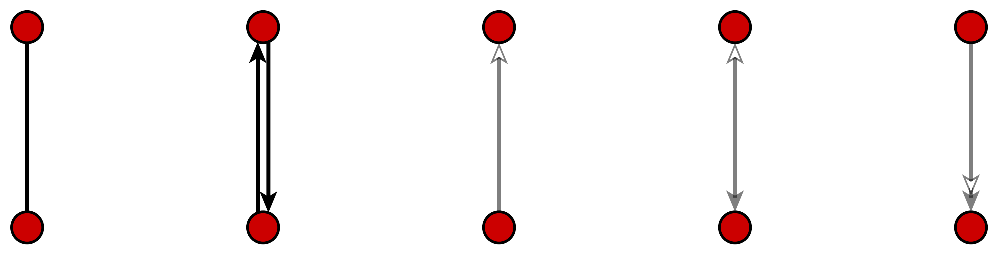
Eigenvectors Let $M \in \mathbb{C}^{n \times n}$ be a matrix. A value $\lambda \in \mathbb{C}$ is called an *eigenvalue* of M if there is a non-zero vector $x \in \mathbb{C}^n$ such that $Mx = \lambda x$. Such a vector x is called an *eigenvector* of M (with eigenvalue λ). The (multi-) set of all eigenvalues of a matrix is called its *spectrum*.

3.2 Network Model

In order to achieve maximum generality, the network model of visone is a mixed multi-graph to which an unlimited number of weights for the vertices and edges can be attached. However, algorithms are typically designed to work on undirected or directed edges only, allow or forbid parallel edges, and can handle either weighted or binary data. While we have striven to formulate the algorithms in the most general way, some methods impose inherent restrictions. In this case, graphs not adhering to the requirements have to be transformed. In the following, we describe the relevant characteristics and the respective reasonable transformations.

3.2.1 Direction

Typically, a formulation of an algorithm allowing directed edges is more general than one for undirected edges since for almost all purposes, each undirected edge $\{u, v\}$



(a) undirected, confirmed (b) directed, confirmed (c) undirected, unconfirmed (d) directed, receiver confirms (e) directed, sender confirms

Figure 3.1: Directed and undirected edges of different confirmation.

is replaced equivalently by two symmetric, contrariously directed edges (u, v) and (v, u). The transformation from directed to undirected data is not equally simple because the inverse operation is not well-defined for all undirected graphs, i. e., it fails for dyads linked in only one of the two directions. Thus, our alternative approach is to use the underlying undirected graph (see Section 3.1). Note that the number of vertices and edges stays the same in the original graph and the underlying undirected graph since each directed edge is replaced by a undirected one. In particular, two symmetric, contrariously directed edges (u, v) and (v, u) are replaced by two undirected edges $\{u, v\}$ which may be unintentional.

3.2.2 Confirmation

In many real-world applications, the inherent uncertainty of the data collection process results in another important property of edges: *confirmation*, also called *reciprocity*. Unconfirmed edges emerge for example when two actors have divergent perceptions of the existence or specificity of their relation or when one of them simply lies. Such unconfirmed connections exhibit an additional form of direction induced by the actor who testifies for it. A directed edge is called *sender confirmed* if it is confirmed by its tail and *receiver confirmed* if it is confirmed by the head. The appropriate handling of unconfirmed edges is a controversial issue in social network analysis. Typically solutions are either to ignore them completely or to treat them all as confirmed.

Common tools for network analysis do not feature a specific handling of unconfirmed relations. Thus, users are forced either to finally decide about the notion of unconfirmedness in the beginning or to manage a large number of data files in order to test different hypotheses. In either way, the original meaning of the unconfirmed edges is likely to be inapparent in the final results, especially in visualizations generated thereof.

In contrast, confirmation is modeled as explicit edge property in visone allowing simple and reproducible handling of these edges. For distinction and in order to lower their visual impact, unconfirmed edges are drawn in semi-transparent colors.

The direction of confirmation is depicted by additional hollow arrows which are easily distinguishable from the filled arrows indicating direction. Algorithms for analyses ignore unconfirmed edges by default but the ones that are selected by the user are included. This allows a quick comparison of different states and supports the user in his final decision about the handling of these edges. In visualization and layout methods, unconfirmed edges are included in the computation since they are present in the drawing anyway. In addition to the weakened conspicuousness achieved by the semi-transparent drawing, some visualizations further highlight the core layout of the confirmed subgraph (e. g., see Sections 6.2.3 and 6.3.3).

3.2.3 Weights and Attributes

Often, one is not only interested in the existence of edges but in a quantification of the interconnections. Depending on the application, this may be the frequency of meetings, the helpfulness of advice, or the costs of exchange. The common model for a quantification are numerical *weights* for the edges or, rarely, the vertices of the graph, i. e., a function $\omega : X \to \mathbb{R}$ which assigns each element $x \in \{V, E\}$ a real value $\omega(x) \in \mathbb{R}$. Typically, a large number of additional semantical data is known about the actors and links of a network, e. g., personal data of the actors like names and sex or a classification like department membership. While this may not be relevant for analysis methods, it is often desirable to include these information into a visualization.

In visone, we have consolidated weights and semantical data into *attributes*. In general, an attribute is a function $f : X \to Y$ which assigns each element $x \in \{V, E\}$ a value $f(x)$ of the specified type Y. Available types are *binary*, *integer*, *double*, *text*, and lists of these basic types. Furthermore, each attribute has a unique name for identification and, optionally, a textual description and a default value. An unlimited number of attributes for vertices and edges can be associated with each graph.

The main use cases of attributes are:

- store user data, e. g., weights and semantical element information,
- specify input parameters of analyses methods, typically as edge weights,
- store the result of analyses methods, e. g., centrality indices, a partitioning, or clique membership, and
- specify input parameters of visualizations, e. g., the index or the partitioning to depict in a drawing.

Using the type system, only valid attributes are offered as input parameters for each method to guarantee consistency and to guide the user in the analysis process. For the present analysis methods and visualization algorithms, only a small number of differing characteristics of the attributes occur. These are

- *weights*, indices, scores and so on which can be represented by numerical attributes of the types integer and decimal,

- *partitionings* like a clustering or the connected components which are induced by any attribute of a basic type through a grouping into elements of the same value, and
- *families of overlapping subgraphs*, e. g., cliques or biconnected components, which we represented as lists of the group identifiers a element is affiliated to.

Network-Level Attributes In addition to information about individual elements, many applications require to store data which applies to the complete graph, for example a textual description of the network, the number of connected components, the centralization, or any other result of a network-level analysis. For this purpose, we provide network-level attributes which store single values of one of the types named before. Additionally, they are identified by a name and optionally provide a textual description, similarly to element-level attributes.

3.2.4 Data Representation

The native file format of visone is GraphML [BEH+01], a comprehensive and extensible file format for graphs based on XML and available free of charge under the Creative Commons Attribution License. Its main features cover all requirements of our network model. In particular, GraphML supports mixed graphs with an unlimited number of attribute data, and, through its extension mechanism, edge confirmation and a comprehensive set of graphical properties of the vertices and edges.

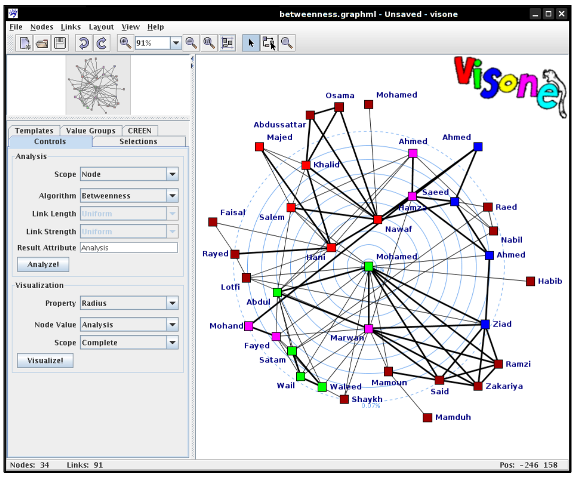
For interoperability, visone can also deal with other common file formats. Since these formats typically support only a small number of attributes and few basic graphical properties, exporting to one of them is typically accompanied by information loss. Available formats for im- and export include

- adjacency matrices in plain text files,
- DL files of UCINET [Ana08] – structured plain text files with support for vertex and edge labels which allow the representation of a graph as adjacency matrix, adjacency list, or edge list [BEF99] – and
- the NET format of Pajek [BM03].

Additionally, pictures of a network can be saved in various vector-graphic and bitmap formats, e. g., as PDF, SVG, EMF, PNG, JPEG, or GIF.

3.3 visone User Interface

The visone software is a powerful visual graph editor providing tailored means of analysis and visualization for social network analysis. The overall appearance and many concepts of user interaction are closely related to common vector graph-



Analysis
Scope: Node
Algorithm: Betweenness
Link Length: Uniform
Link Strength: Uniform
Result Attribute: Analysis

Visualization
Property: Radius
Node Value: Analysis
Scope: Complete

Nodes: 34 Links: 91 Pos: -246 158

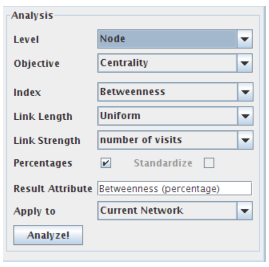
Figure 3.2: The main window of visone features an editable and zoom-able view (main view) of the graph, an overview depicting a sketch of the complete graph, a panel to control various operations, and standard elements like a menu bar, a toolbar, and a status bar.

ics software and is similar to other graph editor tools like yEd [yWo08a], Cytoscape [SMO$^+$03], or SONIVIS [tea08]. The used terminology is adapted to social network analysis and technicalities as well as complex settings are hidden whenever possible.

3.3.1 Main Window

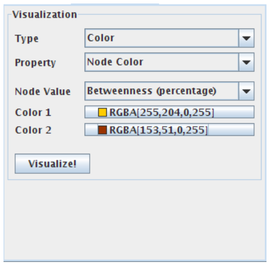
The main window contains the standard elements of typical graphical user interfaces: a *menu bar* and a *toolbar* on top and at the bottom a *status bar* (see Figure 3.2). Most of the window's area is reserved for the *main views* of the open networks. Each network is displayed in full detail in its own view in a separate tab in order to allow users to work on related data simultaneously and to compare results of different networks or even different analyses of the same network. An *overview* of the network and the *control panel* which provides fast access to frequently used methods like analyses, visualizations, and transformations are located on the left side.

Adhering to common practice, the menu bar provides access to all functions of visone (except the ones easily reachable in the controls area), grouped into the self-explanatory menus *file*, *edit*, and *view*. In addition, the menus *nodes* and *links* provide functions specific to vertices and edges, respectively. In the menu *layout*, general graph layout algorithms are available which are described in detail in Chapter 5. In the toolbar, the most frequently used operations are duplicated for convenience. The status bar continuously displays the numbers of vertices and edges and the current mouse coordinate and occasionally shows messages about the progress, success, or failure of user triggered actions.



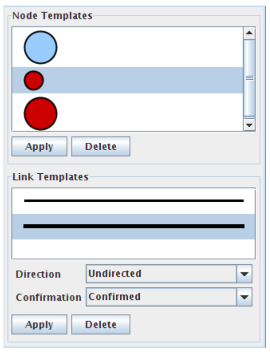
Analysis
Level: Node
Objective: Centrality
Index: Betweenness
Link Length: Uniform
Link Strength: number of visits
Percentages ✓ Standardize ☐
Result Attribute: Betweenness (percentage)
Apply to: Current Network

(a) analysis



Visualization
Type: Color
Property: Node Color
Node Value: Betweenness (percentage)
Color 1: 🟨 RGBA[255,204,0,255]
Color 2: 🟫 RGBA[153,51,0,255]

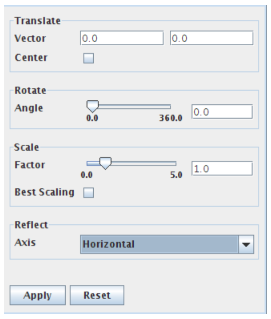
(b) visualization



Node Templates

Link Templates
Direction: Undirected
Confirmation: Confirmed

(c) templates



Translate
Vector: 0.0 0.0
Center ☐

Rotate
Angle: 0.0 (0.0–360.0)

Scale
Factor: 1.0 (0.0–5.0)
Best Scaling ☐

Reflect
Axis: Horizontal

(d) geometry

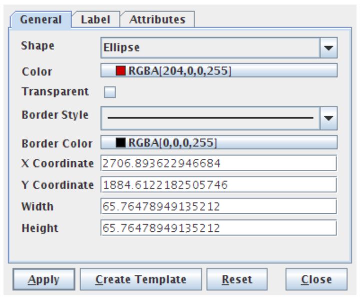
Figure 3.3: The components of the control panel to manage analyses, visualizations, templates, and geometric transformations.

Main View The main view depicts the graph in full detail and allows interactive changes to the graph structure and the graphical properties of the vertices and edges. The current point of view and the zoom level are freely adjustable by simple mouse movements. Subsets of the vertices and edges can be selected (indicated by selection markers and darker element color) to restrict the range of application of many functions to these elements. A context menu for the vertices and edges provides in-place access to frequently used functions.

In order to prevent users from accidental modifications of the graph structure, there are three modes of operation of a view. In *edit mode*, the main mouse operation is element creation and users can add, delete, and edit vertices and edges by simple mouse clicks. In *analysis mode*, new elements cannot be created since a mouse click now moves vertices or selects elements. Finally, in *navigation mode*, neither the structure nor the layout of the graph can be changed. Instead, the main operations are the moving of the drawing area and zooming in and out.

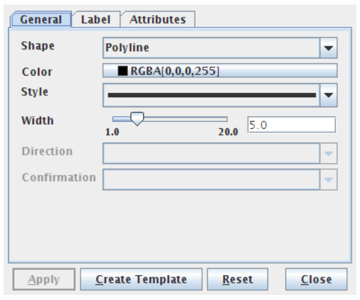
Control Panel The control panel is placed side-by-side to the main view. Therefore, it is always easily accessible and does not overlap with the display of the graph. Here, users can execute an *analysis* and a *visualization*, manage the lists of stored *selections* and *templates*, and apply *geometric transformations* to the graph (see Figure 3.3). In anticipation of the detailed definitions and algorithms for the available analysis measures and visualization methods in Chapters 4 and 6, we describe their control panels here. The other panels are described together with their respective functions in the following Sections 3.3.2-3.3.4.

The analysis control consists of three parts from which the middle one contains the components specific to a given measure and the upper and lower parts are common to all methods (see Figure 3.3(a)). The upper part covers the selection of the analysis method, divided into the three stages *level*, *objective*, and *name* as described in Chapter 4. For example, a clique analysis belongs to the group-level measures of objective cohesiveness. In the lower part, the name of the attribute to save the result to can be specified. Furthermore, users can choose to do an analysis not only for the active network but for all open ones.



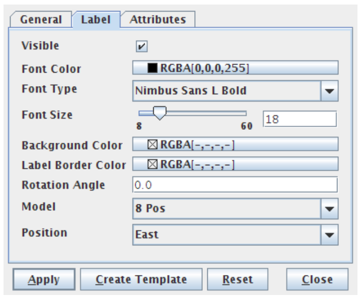
Shape	Ellipse
Color	🟥 RGBA[204,0,0,255]
Transparent	☐
Border Style	
Border Color	■ RGBA[0,0,0,255]
X Coordinate	2706.893622946684
Y Coordinate	1884.6122182505746
Width	65.76478949135212
Height	65.76478949135212

(a) properties of vertices



Shape	Polyline
Color	■ RGBA[0,0,0,255]
Style	
Width	5.0
Direction	
Confirmation	

(b) properties of edges



Visible	✓
Font Color	■ RGBA[0,0,0,255]
Font Type	Nimbus Sans L Bold
Font Size	18
Background Color	☒ RGBA[-,-,-,-]
Label Border Color	☒ RGBA[-,-,-,-]
Rotation Angle	0.0
Model	8 Pos
Position	East

(c) properties of labels

Figure 3.4: The properties dialogs for vertices, edges, and labels.

The control panel for visualizations also consists of three parts: the choice of a method, the selection of the attribute to visualize, and, optionally, components to define additional parameters of the visualization. The available visualizations are grouped according to the graphical property modified by them, e. g., color or size, following the structure of Chapter 6. Figure 3.3(b) shows exemplary the selection of the vertex color visualization.

3.3.2 Graphical Properties and Templates

The main view of a graph allows to specify various graphical properties of vertices and edges. These properties comprise the shape, the color, and the border style of vertices and the shape, the color, the style, and the width of edges. Additionally, there are extensive options to change the appearance of the labels of the vertices and edges. Figure 3.4 depicts the components to change these properties.

In visone, a *template* is a model for all graphical properties of a vertex or an edge, respectively. These models are presented in graphical lists in the template control (see Figure 3.3(c)). Templates can be either created by coping the appearance of any existing element or specified in the properties dialogs and assigned to individual elements or any selection. Newly created elements inherit their initial appearance from the active template.

Graphical properties are not only important to create visually pleasing drawings but also to depict attributes of the elements of the network like edge weights or a classification of the vertices. Methods which change graphical properties of the elements to exactly reflect a given attribute are called visualizations in visone and described in Chapter 6.

3.3.3 Geometric Transformations

A number of geometric transformations of the layout of a graph, namely translation, rotation, scaling, and reflection, are available through the geometry control panel (see Figure 3.3(d)). A geometric transformation is applied to the selected vertices or, if there are no selected vertices, to the whole graph. In either way, the base point of the transformation is the center of the bounding box of the affected vertices, e. g., for a horizontal reflection, the axis of reflection is a horizontal line through this center. In addition to the transformations according to explicit values, e. g., a given angle of rotation or a given scaling factor, there are options to translate vertices to the center of the current view and to apply the *best scaling*, i. e., the layout is scaled to fit best inside of the current view. Note that such a best-fit scaling is executed automatically after most of the graph layout algorithms, thus users can control the dilation of the layout by simply choosing a zoom level.

3.3.4 Selections

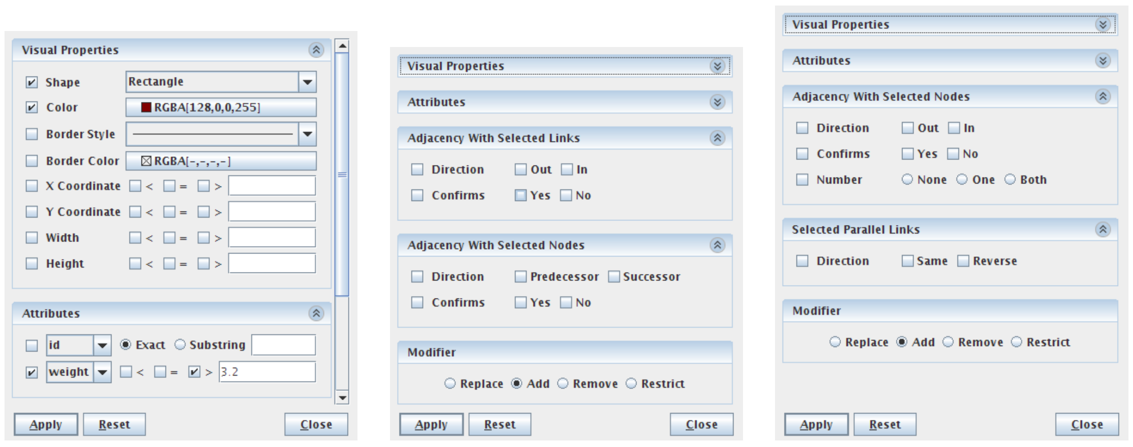
At first sight, a selection is just a user defined collection of vertices and edges. Similar to equivalent concepts in other graphics tools, elements can be selected by clicking with the mouse and, typically, methods act on all the selected elements at once. This makes it easy for example to change the graphical properties of a large number of elements or to delete them. Besides facilitation of handling, selections can play an important role in the process of analysis. For this purpose, we provide two additional components: an extensive *selection dialog* and a *list of selections*.

The selection dialog allows the selection of elements of any given visual property or attribute value, for example to select exactly the vertices which are red and of rectangular shape and have an attribute value above a specified threshold (see Figure 3.5(a)). Another available criterion is the adjacency to already selected vertices and edges which optionally considers the direction and the confirmation of the respective connection (see Figures 3.5(b) and 3.5(c)). A prominent and important application is the selection of the *ego-network* of a vertex, i. e., the subgraph induced by a given vertex and its neighborhood. Individual queries can be combined to complex selections using *modifiers*. The modifier defines how the selection defined by the settings in the dialog is merge with the selection currently present in the view. The available options are:

- *replace* the current selection by the one of the dialog, i. e., the current selection is not considered at all,
- *add* elements selected in the dialog to the current selection,
- *remove* elements selected in the dialog from the current selection, and
- *restrict* the current selection to the elements also selected in the dialog, i. e., only elements which are selected in the current view and by the dialog remain selected.

While for an individual query exactly the elements for which all specified criteria are fulfilled are selected, the selection of, for example, all vertices which are red or of rectangular shape can be composed of two simple queries using the modifier “add”.

Often one has to handle more than one selection or wants to store the current selection for later usage. For this purpose, visone provides a tabular list of named



(a) visual properties of vertices (b) adjacency of vertices (c) adjacency of edges

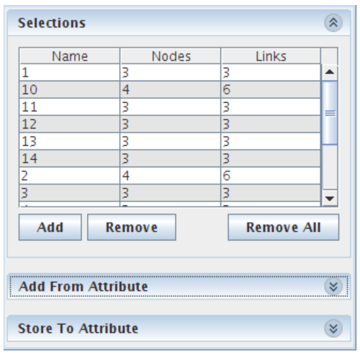
Figure 3.5: The selection dialog offers comprehensive selection criteria: visual properties, attribute values, and adjacency structure. Individual queries can be combined with the current selection in several ways, specified by the so called *modifier*.

selections in the control panel (see Figure 3.6). Besides the user-defined name, this list displays the number of affiliated vertices and edges of each selection and can be sorted according to any of these columns, e. g., to identify the largest selections. At any time, users can add the current selection to this list or re-establish a selection previously stored. Since every selection is obviously a subgraph, the entirety of selections can be seen as a family of subgraphs or, for disjoint selections, even as partition and therefore represented as attribute as described in Section 3.2.3. Conversely, any attribute represents either a partition or a family of subgraphs and can therefore be used to fill up the list. A typical example of application is the clique analysis described in Section 4.5.2. Such an analysis yields a family of non-disjoint subgraphs which is stored as attribute and can be loaded into the list of selections. There, the sizes of the cliques are immediately apparent and the members of any given clique can be selected easily.

The interplay of these two components provides fast and simple means to handle collections of vertices and edges in order either to change their visual appearance or to alter the graph structure. Since a selection itself does not change the graph in any way but clearly highlights the affected elements, its usage is safe and comprehensible. Through the conversion from and to attributes, user defined selections can even be used as as input data for analysis and visualization methods.

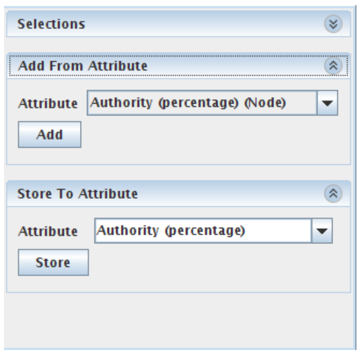
3.3.5 Edge Transformations

The network model of visone allows mixed graphs, i. e., graphs which contain directed and undirected edges, and applies automatic transformations if the current graph does not fit the requirements of an analysis method (see Sections 3.2 and 4.1).



Name	Nodes	Links
1	3	3
10	4	6
11	3	3
12	3	3
13	3	3
14	3	3
2	4	6
3	3	3

(a) the list of selections



Attribute: Authority (percentage) (Node)

Attribute: Authority (percentage)

(b) controls to convert from and to an attribute

Figure 3.6: The list of selections provides immediate access to any given subgraph, whether user defined or computed by an analysis.

If this automatism is not appropriate or the user wants to enforce a specific presentation of the connections, manual intervention is required. Therefore, we provide typical transformations to *split* edges and to *merge* edges between the same dyad.

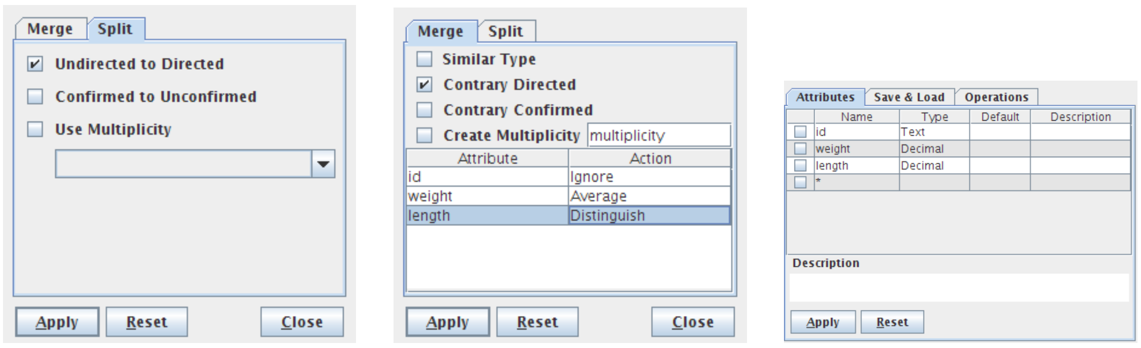
The split operation can replace an undirected edge $\{u, v\}$ by two directed edges (u, v) and (v, u) or separate a confirmed edge into the two directions of confirmation. An optional attribute specifies the multiplicity of newly create edges, i. e., a multiplicity value of three would split a confirmed edge into three unconfirmed edges in each of the two directions of confirmation. In either case, all attribute values are copied from the original edge to each of the newly created ones, thus no information is lost by a split operation. Figure 3.7(a) shows the controls of the split operation.

A merge operation combines two or more edges between the same pair of vertices into one edge. Since such an operation can become complicated if candidates for a merge have different attribute values, we first concentrate on the three different cases imposed by structural properties only, which are

- the unification of two contrary directed edges (u, v) and (v, u) of the same confirmation into an undirected edge $\{u, v\}$,
- the merging of two edges of the same direction but of different confirmation into a confirmed edge of this direction, and
- the combination of multiple edges of the same direction and confirmation into one edge of this type.

Optionally, the multiplicity of the new edges, i. e., the number of original edges merged into each new one, is stored as attribute.

If the candidates for a merge have different attribute values, various approaches may be reasonable, depending on the application and on the meaning of the attribute. Therefore, we leave this decision to the user and provide the following options. In the simplest case, an attribute is irrelevant for the merging and can be *ignored*. Contrarily, the values may *distinguish* different types of connections which prevents



[✓] Undirected to Directed
[ ] Confirmed to Unconfirmed
[ ] Use Multiplicity

[ ] Similar Type
[✓] Contrary Directed
[ ] Contrary Confirmed
[ ] Create Multiplicity multiplicity

Attribute	Action
id	Ignore
weight	Average
length	Distinguish

	Name	Type	Default	Description
☐	id	Text		
☐	weight	Decimal		
☐	length	Decimal		
☐	*			

(a) split of edges (b) merger of edges (c) list of attributes

Figure 3.7: Controls to split and to merge edges and the component to manage the list of attributes.

any merge of edges of unequal value. For numerical attributes, statistical aggregation like summation, minimum, or maximum may also be appropriate. Note that disregarding and aggregation of attribute values will result in information loss. In the example in Figure 3.7(b), symmetrically directed edges will be joined into undirected edges if they have the same value of the attribute "length". The "id" of the edges is ignored and the value of the "weight" of each new edge will be the average of its original edges.

3.3.6 Attributes

The management of attributes is equal for vertices and edges but separated into two different components. Both feature a list of all respective attributes, a panel to execute a number of operations on the values of an attribute, and the possibility to import and export attributes from and to a file, respectively. The tabular list of attributes shows the name, the type, the default value, and the description of each attribute and allows the modification, creation, and removal of attributes (see Figure 3.7(c)). We provide only a rather small number of operations which comprise copying of an attribute, ranking of its values, and basic arithmetic operations. For complex operations, the export and import allows the transfer of attributes to any spreadsheet program and back. Of course, any additional data can be imported into attributes just as well. The file format we use is the widely supported comma separated values format (CSV).

Chapter 4

Analysis

The purpose of social network analysis is to identify important actors, crucial links, roles, dense groups, and so on, in order to answer substantive questions about structure.

Hardly something is annoying users more than a program not accepting their data. visone is very liberal in handling input data but nevertheless rigorous in the computation of analysis results. We achieve this by a two-fold approach. First, we have analyzed existing algorithms carefully and relaxed restrictions on the input data if possible. Second, if the characteristics of the data does not meet the requirements of an analysis method, we apply transformations which preserve the intuition of the analysis. In Section 4.1 we describe the relevant characteristics and possible transformations in detail.

The analysis methods available in visone are divided into four main categories according to the *level* or subject of interest: *vertex*, *dyad*, *group*, and *network* level. These levels break further down into measures of the same *objective*, e. g., connectedness or cohesiveness. The purpose of this categorization is to guide the user in the analysis process and to clarify the user interface. It is not intended as a rigorously defined classification of analysis methods since some methods conform to multiple categories equally well.

The description of the analysis methods in this section follows the categorization in levels and objectives. We start with centrality and density measures on the element level in Sections 4.3 and 4.4. In the following Sections 4.5-4.7, we describe measures for the group level, namely cohesiveness, connectedness, clustering, and role assignment. We conclude by network-level statistics in Section 4.8.

4.1 Characteristics of Input Data

While every analysis algorithm has its own domain there are some criteria which are common to all graph algorithms and most relevant in the context of network analysis.

Direction Typically, analysis methods are defined either for directed or undirected data. Since visone allows the usage of mixed-mode graphs, i. e., undirected and directed edges coexist in one graph, the symmetrized directed graph or the underlying undirected graph are used when a method requires directed or undirected edges respectively (see Section 3.2).

Note that for virtually every analysis method, the transformation into the symmetrized directed graph preserves the original meaning while the underlying undirected graph contains multi-edges for oppositely directed edges which may be unintentional. In this sense, the directed version of an analysis algorithm is more general and therefore preferred in visone.

Edge Weights What is commonly referred to as edge weight often one of two different meanings, has depending on the context: *strength* (larger is better) or *length* (smaller is better). This distinction translates from the data to the analysis methods. For each method, it is clearly labeled if a weight is considered as strength or as length. For some methods, even two weights of different meaning can be specified. However, it is the user's responsibility to select a reasonable attribute as weight. In general, unit weights can be safely assumed for unweighted data and methods for unweighted data simply ignore all weights.

Since the results for negative weights are almost always hard to interpret or even undefined, we restrict ourselves to non-negative numerical weights, which we also call *score* or *index*. Sometimes negative values are used to invert the effect of a weight, which is rendered unnecessary in visone by the notions of strength and length.

Multi-Edges If an algorithm cannot handle multi-edges, the underlying simple, either directed or undirected, graph is used. Note that in principle this transformation is prohibitive then edge weights are used.

On the over hand, there is a direct correspondence between edge strength and multi-edges: duplicating an edge should have the same effect as doubling its strength. Therefore, every algorithm that supports a strength is defined for multi-edges. Multi-edges of unequal length behave differently, e. g., only the shortest edge is significant for shortest-path based methods. In general, they cannot be handled uniformly, thus, every algorithm using lengths is defined explicitly for multi-graphs in visone.

Loops An algorithm which cannot handle loops simply ignores them. Note that for many applications, self-loops are either irrelevant or dispensable, e. g., for shortest-path based methods.

Connectedness If an algorithm cannot be carried out on an unconnected graph, we provide a meaningful composition of the final result from the provisional results of the (weakly) connected subgraphs. Note that directed methods may even require strongly connectedness. Algorithms which require a higher vertex or edge connectivity are seldom used in network analysis.

4.2 Size of the Graph

The numbers of vertices and edges are simple but important characteristics of networks and are therefore continuously displayed in the status bar of visone. Moreover the number of selected elements are also shown here allowing quick inspection of the size of the currently selected group, e. g., a connected component or a set of vertices of the same attribute value. For a closer inspection of the size and membership of groups the selection dialog and the selection control panel can be used.

4.3 Centrality

Centrality is a core concept for the analysis of social networks and refers to an intuitive feeling that in most networks some vertices or edges are more central than others. The first concepts of centrality date back to the 1950s, e. g., the Bavelas index [Bav48, Bav50] or degree centrality [PL51]. Despite this long time, or maybe on that account, the term 'centrality' is by no means clearly defined.

We purposely have not striven for a complete collection of centrality indices and selected only methods for visone which are amongst the most frequently used and least controversial ones. In addition, we do not want to argue about the meaning of 'importance' and adopt the rather technical definition for centrality from [BE05b]. As a minimal requirement, we demand that the result of a centrality index is given in real numbers and depends only on the structure of the graph. This is called a structural index.

Definition 4.1 (Structural Index) *Let G and H be weighted, directed or undirected multi-graphs, let ϕ be an isomorphism between G and H, and let X represent the set of vertices or edges of G, respectively. A real-valued function s is called a structural index if and only if:*

$$\forall x \in X : G \simeq H \Rightarrow s_G(x) = s_H(\phi(x)) ,$$

where $s_G(x)$ denotes the value of $s(x)$ in G.

A centrality index c is required to be a structural index and thus induces at least a semi-order on the set of vertices or edges. By means of this order we can say that $x \in X$ is at least as central as $y \in X$ with respect to a given centrality c if $c(x) \geq c(y)$.

Centrality measures for edges are clearly outnumbered by the vast amount of centrality indices for vertices. This is reflected by the measures presented in the following which almost exclusively compute vertex centralities with the notable exception of shortest-path edge centralities in Section 4.3.5.

4.3.1 Normalization

In general, the absolute values of a centrality index have no meaning by themselves, and the difference or ratio of two centrality values of the same graph cannot be interpreted as a quantification of how much more central one element is than the other. Consequently, centrality values of vertices of different graphs are also not comparable. Often absolute values are higher in larger graphs. To overcome these problem and to achieve better manageable values, different methods of normalization have been proposed. Freeman [Fre79] suggested to divide by the maximum possible value in any network of the same number of vertices. This seems rather arbitrary since the actual values can be much smaller and, moreover, may be not well-defined for weighted or multi-graphs. For distinction, we call this method *standardization*.

We propose a different normalization independent of graph size and applicable to all types of graphs, namely dividing by the sum of all scores. The resulting indices are non-negative and have unit sums, i. e., they can be viewed as percentage values and interpreted as the share of importance of a vertex in its graph. This method is called *percentage* in the following.

Most of the centralities available in visone are originally defined for strongly connected, directed graphs (or connected undirected graphs). A common way to deal with insufficient connectivity is to compute the measure for each connected component separately and multiply each value by the size of its component. This can be problematic if the measure does not scale proportional to the graph size [PBM00]. We propose a variation, namely to scale the values of each component to sum up to the component's share of size. When used together with percentage values, results are reasonable for many graph instances.

In visone, users can choose between the computation of pure values, normalization to percentages, and, if applicable, traditional standardization.

4.3.2 Degree

Degree centrality is defensibly the most simplest reasonable centrality. Originally the centrality value of a vertex of an undirected, unweighted graph is defined as its degree [PL51]. The reformulation of degree as the number of incident edges leads to the straight-forward generalizations to weighted multi-graphs.

Definition 4.2 (Degree Centrality [PL51]) *Let $G = (V, E)$ be an undirected, weighted multi-graph with edge strength ω. The degree centrality $c_D : V \to \mathbb{R}_{\geq 0}$ of a vertex $v \in V$ is defined as the sum of the strengths of its incident edges, i. e.,*

$$c_D(v) = \sum_{\{u,v\} \in E} \omega(\{u, v\}) \ . \tag{4.1}$$

For directed networks two variants of the degree centrality may be appropriate.

Definition 4.3 (In- and Out-Degree Centrality) *Let $G = (V, E)$ be a directed, weighted multi-graph with edge strength ω. The in-degree centrality $c_{iD} : V \to \mathbb{R}_{\geq 0}$ and the out-degree centrality $c_{oD} : V \to \mathbb{R}_{\geq 0}$ of a vertex $v \in V$ are defined as the sum of the strengths of its incident incoming and outgoing edges respectively, i. e.,*

$$c_{iD}(v) = \sum_{e=(u,v)\in E} \omega(e) \text{ and } c_{oD}(v) = \sum_{e=(v,w)\in E} \omega(e) \ . \tag{4.2}$$

Since the values for a vertex v depend only on its incident edges no special handling of unconnected graphs and self-loops is needed. For unweighted graphs we can assume unit weights and get the usual equality of degree centrality and degree: $c_D(v) = \deg(v)$, $c_{iD} = \text{indeg}(v)$, and $c_{oD} = \text{outdeg}(v)$. The standardization constant for all degree centralities is $1/(n-1)$, because $n-1$ is the maximum degree in an unweighted simple graph.

There are two well-known connections between directed and undirected degree centrality. For the symmetric network $\overrightarrow{G}$ of a graph G we have $c_D = c_{iD} = c_{oD}$ and for the underlying undirected network $\overline{G}$ $c_D = c_{iD} + c_{oD}$.

Obviously, degree centrality is a very local measure, i. e., only the adjacencies of the vertex are considered. In the next sections we investigate global centrality measures.

4.3.3 Distance

In this section we describe centrality measures based on the assumption that a vertex is more central when its distance to others is small. Closeness centrality, defined as the reciprocal of the total distance, is by far the most commonly used centrality of this type.

Definition 4.4 (Closeness Centrality [Bea65, Sab66]) *Let $G = (V, E)$ be a directed, strongly connected multi-graph with edge length ℓ. The closeness centrality $c_C : V \to \mathbb{R}_{\geq 0}$ of a vertex $v \in V$ is defined as the reciprocal of the sum of the distances to all other vertices $v \neq u \in V$, i. e.,*

$$c_C(v) = \frac{1}{\sum\limits_{v\neq u\in V} d(v,u)} \ . \tag{4.3}$$

There are two variants of this approach. Hage and Harary [HH95] exchanged the total distance to all other vertices by the eccentricity $\varepsilon(v)$, i. e., the maximum distance to any other vertex $v \neq u \in V$.

Definition 4.5 (Eccentricity Centrality [HH95]) *Let $G = (V, E)$ be a directed, strongly connected multi-graph with edge length ℓ. The eccentricity centrality $c_E : V \to \mathbb{R}_{\geq 0}$ of a vertex $v \in V$ is defined as the reciprocal of the maximum of the distances to all other vertices $v \neq u \in V$, i. e.,*

$$c_E(v) = \frac{1}{\max\limits_{v\neq u\in V} d(v,u)} = \frac{1}{\varepsilon(v)} \ . \tag{4.4}$$

In both closeness and eccentricity the multiplicative inverse is used to achieve increasing centrality values with decreasing distance. Valente and Foreman [VF98] proposed to use a reverse value instead, i. e., to subtract the distance from the diameter of the graph.

Definition 4.6 (Radiality Centrality [VF98]) *Let $G = (V, E)$ be a directed, strongly connected multi-graph with edge length ℓ. The radiality centrality $c_E : V \to \mathbb{R}_{\geq 0}$ of a vertex $v \in V$ is defined as the sum of reverse distances to all other vertices $v \neq u \in V$, i. e.,*

$$c_R(v) = \sum_{v \neq u \in V} diam(G) + 1 - d(v, u) \; . \tag{4.5}$$

Note that the edge length ℓ is not used explicitly in the above definitions but affects the computation of vertex distances and graph diameter. Neither self-loops nor multi-edges conflict with the definition of distance, so no special handling for graphs containing such connections is needed. While no standardization is needed for eccentricity since the maximum possible value is one, a common standardization factor for closeness is $n_C = n - 1$. For both centralities the maximum value is achieved for a vertex directly connected to all other vertices. Sometimes, radiality is defined with a normalization factor $n_R = 1/(n - 1)$ but generally this does not bound the values to the unit interval $[0, 1]$.

Since all these centralities base on the reachability between all pairs of vertices, weakly connected and unconnected graphs need special attention. Often, the distance between unreachable pairs of vertices is defined to be infinity but this will render these centrality indices meaningless. Therefore, we only take the distances between vertices of the same connected or weakly connected component into account. In order to account for varying sizes of the components, we assume that each component's share of the total centrality is proportional to its number of vertices minus one and scale the values accordingly.

Implementation All these distance based centralities can be computed directly from the solution of the all-pairs shortest-paths problem (APSP). APSP can be efficiently computed from n shortest-paths trees, one for each vertex $v \in V$ as source. Using Dijkstra's algorithm (see Section 3.1 for details) yields a total running time of $\mathcal{O}(nm + n^2 \log n)$ for non-uniform edge length and $\mathcal{O}(nm)$ otherwise. If intermediate results are not stored the required storage space is $\mathcal{O}(n + m)$.

4.3.4 Shortest Paths

The measures under consideration in this section, betweenness [Ant71, Fre77] and stress [Shi53], implicitly assume that communication is conducted along shortest paths only. Intuitively speaking, an actor is regarded to be central if it is part of many of these paths. Clearly, distance and shortest paths are closely related, namely the length of a shortest path between two vertices $u, v \in V$ is the distance $\mathrm{d}(u, v)$ between u and v. But in contrast to the previous section, these measures consider the number of shortest paths and not their length.

First, we want to recall some of the definitions from Section 3.1. Let $\sigma(s,t)$ be the number of shortest (s,t)-paths and denote by $\sigma(s,t|v)$ the number of shortest (s,t)-paths passing through a vertex v. By convention, let $\sigma(s,t)=1$ if $s=t$, and let $\sigma(s,t|v)=0$ if $v\in\{s,t\}$.

Definition 4.7 (Betweenness Centrality [Ant71, Fre77]) *Let $G=(V,E)$ be a directed multi-graph with edge length ℓ and edge strength ω. The betweenness centrality $c_B : V \longrightarrow \mathbb{R}_{\geq 0}$ is defined as*

$$c_B(v) = \sum_{s,t\in V} \frac{\sigma(s,t|v)}{\sigma(s,t)} \,, \tag{4.6}$$

where $\frac{0}{0}=0$ by convention.

Since loops have no influence on distance – and shortest paths – they can be ignored. The definition of betweenness applies directly to disconnected graphs [Fre77] since the number of shortest paths between unconnected pairs of vertices is zero, resulting in the desired contribution of zero. A common standardization for unweighted betweenness is multiplying a factor of $n_B = 1/(n-1)(n-2)$ but for the reasons described in Section 4.3.1 we propose to scale to percentage values instead.

Note that neither the edge length ℓ nor the edge strength ω are mentioned explicitly in the above definition. Similar to distance based centralities, edge length is concealed in the definition of shortest paths. The influence of the edge strength has to conform to the correspondence between multi-edges and integer edge strength explained in Section 4.1. If there are multi-edges between more than one pair of vertices on a path, any edge between one pair combined with any edge between the other pair yields a different path, so that the total number of paths obtained is the product of the multiplicities of its edges. Therefore, the strength of a shortest path is the product of the strength values of its edges. Brandes [Bra08] pointed out that alternative measures of betweenness for weighted graphs [FBW91, New05] are not based on shortest paths and therefore are not proper generalizations of shortest-path betweenness.

A "simplification" of betweenness does not consider the fraction of shortest paths passing through a vertex but their absolute number. Sometimes this approach is also referred to as betweenness [LGH06, PS06], but in fact, it yields a different measure called *stress* by Shimbel [Shi53] long before the introduction of betweenness.

Definition 4.8 (Stress Centrality [Kat53]) *Let $G=(V,E)$ be a directed multi-graph with length ℓ and edge strength ω. The stress centrality $c_S : V \longrightarrow \mathbb{R}_{\geq 0}$ is defined as*

$$c_S(v) = \sum_{s,t\in V} \sigma(s,t|v) \,. \tag{4.7}$$

All remarks for betweenness about self-loops, disconnected graphs, and edge length and edge strength apply also to stress centrality.

Implementation A naive implementation of these centrality measures has two steps. First, compute the length and number of shortest paths between each pair of vertices and store them in a table. In the second step, for each vertex $v \in V$ look-up for all other pairs of vertices s, t the fraction of shortest s-t-paths passing through v and sum these fractions to get the betweenness value. Clearly, this approach has a running time in $\mathcal{O}(n^3)$ and needs space in $\mathcal{O}(n^2)$ for the look-up table.

By considering the recursive dependency of each vertex's shortest paths on the dependencies of its neighbors [Bra01a], Brandes incorporated all computations into the traversal of n directed acyclic graphs, each consisting of the shortest paths from a selected source to all other vertices. This avoids the time and space penalty of the table look-ups resulting an a running time in $\mathcal{O}(nm+n^2 \log n)$ for non-uniform edge length and $\mathcal{O}(nm)$ otherwise. Since intermediate results are not stored the required storage space is in $\mathcal{O}(n+m)$. Recently, Brandes extended his original betweenness algorithm into a framework for shortest-path centralities and pointed out extentions for weighted graphs and the computation of stress centrality [Bra08].

4.3.5 Shortest-Path Edge Centralities

The intuition of shortest-path centralities naturally extends to measures for edges by replacing $\sigma(s,t|v)$ by $\sigma(s,t|e)$, the number of shortest (s,t)-paths passing through an edge $e \in E$.

Definition 4.9 (Edge Betweenness Centrality [Ant71]) *Let $G = (V, E)$ be a directed multi-graph with edge length ℓ and edge strength ω. The edge betweenness centrality $c_B : E \to \mathbb{R}_{\geq 0}$ is defined as*

$$c_B(e) = \sum_{s,t \in V} \frac{\sigma(s,t|e)}{\sigma(s,t)} , \tag{4.8}$$

where $\frac{0}{0} = 0$ by convention.

In an analogous manner, we can replace the fraction of shortest paths passing through an edge by the absolute number to define edge stress centrality.

Definition 4.10 (Edge Stress Centrality) *Let $G = (V, E)$ be a directed multi-graph with edge length ℓ and edge strength ω. The stress centrality $c_S : E \to \mathbb{R}_{\geq 0}$ is defined as*

$$c_S(e) = \sum_{s,t \in V} \sigma(s,t|e) . \tag{4.9}$$

Clearly, all remarks for shortest-path vertex centrality about self-loops, disconnected graphs, and edge length and edge strength apply also to edge centralities. Furthermore, the computation of edge betweenness and edge stress can be easily incorporated into the general framework by Brandes [Bra08].

4.3.6 Current-Flow

Neglecting flow of information along non-shortest paths by betweenness and other shortest path based measures is a disadvantage commonly complained about which limits their applicability. Recently, a variation of betweenness based on the flow of electrical current has attracted considerable attention [New05] and inspired a generalization of closeness in a similar way [BF05]. Before we give the definition of these centrality indices, we briefly describe the employed concept of electrical networks.

Electrical Networks An *electrical network* N is an undirected, simple, connected graph $G = (V, E)$ together with positive edge weights $\omega : E \to \mathbb{R}_+$ indicating the *conductance* or strength of an edge. Equivalently, the network could be defined in terms of positive edge weights $\ell : E \to \mathbb{R}_+$ indicating the *resistance* or length of an edge, where conductance and resistance are related by $\omega(e) = 1/\ell(e)$ for all $e \in E$.

A *supply* function $b : V \to \mathbb{R}$ specifies there external electrical current enters and leaves this network. The amounts of entering and leaving currents have to be equal, i. e., $\sum_{v \in V} b(v) = 0$. For our purpose, we consider only unit s-t-supplies, i. e., a unit current enters the network at vertex s and leaves it at vertex t, that is, $b_{st}(s) = 1, b_{st}(t) = -1$, and $b_{st}(v) = 0$ for all $v \in V \setminus \{s, t\}$.

Since it is useful to talk about the direction of a current in the undirected graph, each edge $e \in E$ is arbitrarily oriented to obtain an oriented edge $\overrightarrow{e}$, which results in an oriented edge set $\overrightarrow{E}$.

A function $I : \overrightarrow{E} \to \mathbb{R}$ is called a (electrical) current in $N = (V, E, \omega)$ if

$$\sum_{(v,w) \in \overrightarrow{E}} I(v, w) - \sum_{(w,v) \in \overrightarrow{E}} I(w, v) = b(v) \text{ for all } v \in V$$

and

$$\sum_{e \in C} I(\overrightarrow{e}) = 0 \text{ for every (undirected) cycle } C \subseteq E.$$

The former equation is known as Kirchhoff's current law, and the latter as Kirchhoff's potential law. Negative values of I are to be interpreted as current flowing against the direction of an oriented edge.

Alternatively to the current I, an electrical flow can also be represented by potentials. A function $U : V \to \mathbb{R}$ is a (electrical) potential if $U(v) - U(w) = I(v, w)/\omega(v, w)$ for all $(v, w) \in \overrightarrow{E}$. As an electrical network $N = (V, E, \omega)$ has a unique current I for any supply b, it also has a potential U that is unique up to an additive factor [Bol98].

Recall that the weighted Laplacian matrix $L = L(N)$ of the electrical network N is defined as

$$L_{vw} = \begin{cases} \sum_{e \in E(v)} \omega(e) & \text{if } v = w \\ -\omega(e) & \text{if } e = \{v, w\} \in E \\ 0 & \text{otherwise} \end{cases}$$

for $v, w \in V$. Then, a potential U for an electrical network $N = (V, E, \omega)$ and a supply b can be found by solving the linear system $LU = b$.

Current-Flow Centralities In electrical networks, the fraction of a unit s-t-current flowing through a vertex v is the correspondent of the fraction $\sigma_{st}(v)$ of shortest s-t-paths passing through that vertex. Given an s-t-current, the *throughput* of a vertex $v \in V$ is therefore defined as

$$\tau_{st}(v) = \frac{1}{2} \left(-|b_{st}(v)| + \sum_{e \in E(v)} |I(\overrightarrow{e})| \right).$$

The term $-|b_{st}(v)|$ ensures that a given unit s-t-current counts only for the throughput of inner vertices $v \notin \{s, t\}$, and the term $\frac{1}{2}$ adjusts for the fact that the summation counts both the current into and out of the vertex v.

Definition 4.11 (Current-Flow Betweenness Centrality [New05]) *Let $G = (V, E)$ be a undirected multi-graph with edge length ℓ and edge strength ω. The current-flow betweenness centrality $c_{CB} : V \to \mathbb{R}_{\geq 0}$ is defined as*

$$c_{CB}(v) = \sum_{s,t \in V} \tau_{st}(v). \tag{4.10}$$

In the original definition of current-flow betweenness, the same standardization as for shortest path betweenness is proposed, i. e., multiplication by the constant factor of $n_{CB} = n_B = 1/(n-1)(n-2)$.

Current-flow betweenness is called *random-walk betweenness* in [New05] because of the following correspondence. A simple random s-t-walk is a random walk that starts at s, ends in t, and continues at vertex $v \neq t$ by picking an incident edge $e \in E$ with probability $\omega(e) / \sum_{e' \in E(v)} \omega(e')$. Then, given an s-t-current, the amount of current flowing through a particular edge $\overrightarrow{e}$ equals the expected difference of the number of times that the simple random s-t-walk passes edge $\overrightarrow{e}$ along and against its orientation (see, e. g., [Bol98]).

The difference in electrical potential of vertices in electrical networks can be seen as the analog of shortest-path distance in distance-based networks and is therefore used for the definition of a correspondent closeness measure.

Definition 4.12 (Current-Flow Closeness Centrality [BF05]) *Let $G = (V, E)$ be a undirected multi-graph with edge length ℓ and edge strength ω. The current-flow* *closeness centrality $c_{CC} : V \to \mathbb{R}_{\geq 0}$ is defined as*

$$c_{CC}(v) = \frac{1}{\sum_{t \neq v} U_{vt}(v) - U_{vt}(t)}. \tag{4.11}$$

The classical standardization constant $n_C = n - 1$ of shortest path closeness does not translate to current-flow closeness because of different possible maximum values. Instead a factor of $n_{CC} = 2(n-1)/n$ can be used for normalization.

Brandes and Fleischer [BF05] showed that current-flow closeness is equal to *information centrality* defined by Stephenson and Zelen [SZ89].

Computation For the computation of the current-flow centralities the efficient algorithm of Brandes and Fleischer [BF05] is used. It is based on the inversion of the restricted Laplacian matrix of the graph G. The total running time for current-flow betweenness is in $\mathcal{O}(M(n-1) + mn \log n)$ and for current-flow closeness in $\mathcal{O}(M(n-1) + n$, respectively, where $M(n)$ is the time required to compute the inverse of an $n \times n$-matrix. Using Gaussian elimination for matrix inversion we have $M(n) \in \mathcal{O}(n^3)$. Since social networks are typically sparse, a method better suited for this case can be used resulting in a running time for matrix inversion in $M(n) \in \mathcal{O}(mn^{1.5})$. The memory requirement for the computation of closeness is in $\mathcal{O}(m)$. In the worst case, for betweenness the memory requirement is $\mathcal{O}(n^2)$ but the authors give a heuristic method to reduce this significantly in many cases.

4.3.7 Feedback

In this section we present centrality indices in which a vertex's score depends on the score of its neighbors, i. e., a vertex is the more central the more central its neighbors are. In general, such feedback relations can be specified in terms of linear systems. Special care has to be taken to get solvable and properly defined systems.

Since feedback centralities do not count concrete structural properties like shortest paths or distances, usually no standardization is used. On the over hand, some kind of normalization is advisable since the results can differ by a constant factor depending on the method of calculation. Therefore, we have the following lemma.

Convention 4.13 *All feedback centralities are normalized to percentages, i. e., given a non-normalized measure $c' : V \to \mathbb{R}_{\geq 0}$, the normalized centrality is defined as*

$$c(v) = \frac{c'(v)}{\sum\limits_{u \in V} c'(u)} \,. \tag{4.12}$$

Status The first realization of the concept of feedback was given by Katz [Kat53] in 1953. His idea can be interpreted as a generalization of degree centrality, where the number of neighbors at distance k is also counted but attenuated by α^k, where $\alpha > 0$ is a damping factor. Recall from Section 3.1 that $(A^k)_{uv}$ holds the number of paths of length k from u to v, where A is the adjacency matrix of the graph G. Therefore, Katz's centrality can be defined as

$$c_{KS}(v) = \sum_{k=1}^{\infty} \sum_{u \in V} a^k (A^k)_{uv} \ ,$$

if the infinite sum converges. We reformulate this definition to emphasize the feedback nature of the centrality.

Definition 4.14 (Status Centrality [Kat53]) *Let $G = (V, E)$ be a weakly connected, directed multi-graph with edge strength ω. The status centrality $c_{KS} : V \to \mathbb{R}_{\geq 0}$ is defined as the normalized values of*

$$c_{KS}(v) = \alpha \cdot \sum_{(u,v) \in E(v)} \omega\big((u,v)\big) \cdot \big(1 + c_{KS}(u)\big) \ , \tag{4.13}$$

where $1/\alpha = \min\big\{\max_{v \in V} indeg_\omega(v), \max_{v \in V} outdeg_\omega(v)\big\}$.

Note that in contrast to the original definition by Katz, we define status in terms of incoming edges to conform to the other centralities in this section.

Eigenvector While status depends on both the number of neighbors and their scores, Bonacich [Bon72] introduced a measure which considers solely the centrality values of the neighbors, i. e., which is a solution of the equation system

$$c(v) = \sum_{u \in V} A_{uv} \cdot c(u)$$

or, equally, $\boldsymbol{c} = A \cdot \boldsymbol{c}$, where the vector $\boldsymbol{c} \in \mathbb{R}^n$ contains the values $c(v)$ for all $v \in V$. If G is a weakly connected multi-graph, the largest eigenvalue λ_1 of the corresponding Eigenvalue problem $\lambda \boldsymbol{c} = A\boldsymbol{c}$ is simple and the entries of the eigenvector for λ_1 are of the same sign. Therefore, we can define eigenvector centrality in the following way.

Definition 4.15 (Eigenvector Centrality [Bon72]) *Let $G = (V, E)$ be a directed, weakly connected multi-graph with edge strength ω. The eigenvector centrality $c_E : V \to \mathbb{R}_{\geq 0}$ is defined as the normalized values of*

$$c_E(v) = \frac{1}{\lambda} \cdot \sum_{(u,v) \in E(v)} \omega\big((u,v)\big) \cdot c_E(u) \ , \tag{4.14}$$

where λ is the largest eigenvalue of the adjacency matrix A of G.

Hubs & Authorities Closely related to eigenvector centrality are *Hubs & Authorities* introduced by Kleinberg [Kle99] for scoring Web pages with respect to two different purposes, where

"a good hub is a page that points to many good authorities"

and

"a good authority is a page that is pointed to by many good hubs".

Despite their original scope there are no concerns against their general usage.

Definition 4.16 (Hubs & Authorities [Kle99]) *Let $G = (V, E)$ be a directed, weakly connected multi-graph with edge strength ω. The authorities centrality $c_A : V \to \mathbb{R}_{\geq 0}$ is defined as the normalized values of*

$$c_A(v) = \frac{1}{\lambda} \cdot \sum_{(u,v) \in E(v)} \omega((u,v)) \cdot \big(\sum_{(u,w) \in E_u} \omega((u,w)) \cdot c_E(w) \big) \ , \tag{4.15}$$

where λ is the largest eigenvalue of the matrix $A^T A$, and the hubs centrality $c_H : V \to \mathbb{R}_{\geq 0}$ is defined as the normalized values of

$$c_H(v) = \frac{1}{\lambda} \cdot \sum_{(v,w) \in E(v)} \omega((v,w)) \cdot \big(\sum_{(u,w) \in E_w} \omega((u,w)) \cdot c_E(u) \big) \ , \tag{4.16}$$

where λ is the largest eigenvalue of the matrix AA^T.

Intuitively speaking, a vertex v is the more central in terms of authority centrality the more vertices u point both to u and to other central vertices w. Similarly, its hub score gets higher if more important vertices u point to successors w of v. There is also a descriptive analogy of Hubs & Authorities and eigenvector centrality.

Observation 4.17 *Let $G = (V, E)$ be a directed multi-graph. The symmetric multi-graph $\mathcal{B}(G)$ with adjacency matrix $A_{\mathcal{B}(G)} = A_G A_G^T$ is called* bibliographic coupling *and the symmetric multi-graph $\mathcal{C}(G)$ with adjacency matrix $A_{\mathcal{C}(G)} = A_G^T A_G$ is called* co-citation*, respectively. Hub centrality $c(G)_H$ in G equals eigenvector centrality $c(\mathcal{B}(G))_E$ in $\mathcal{B}(G)$ and authority centrality $c(G)_A$ in G equals eigenvector centrality $c(\mathcal{C}(G))_E$ in $\mathcal{C}(G)$, i. e., for all $v \in V$*

$$c(G)_H(v) = c(\mathcal{B}(G))_E(v) \text{ and } c(G)_A(v) = c(\mathcal{C}(G))_E(v) \ .$$

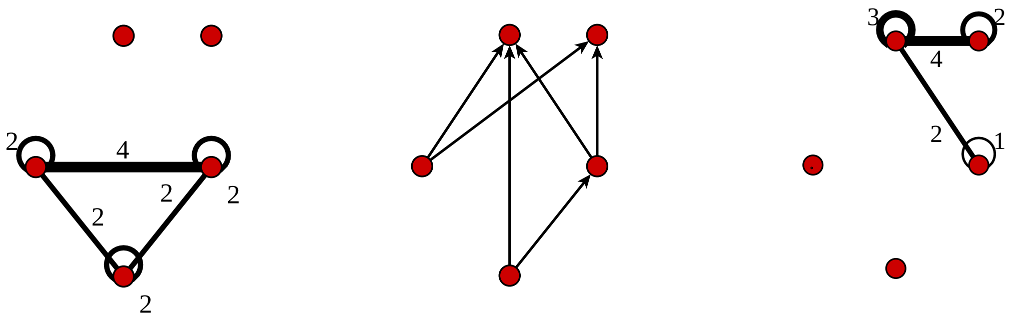
Figure 4.1 gives an example of bibliographic coupling and co-citation graphs.

PageRank Since the initial stage of Google, *PageRank* [PBMW99] is one of the main ingredients of the search engine [BP98]. Over the years, the original measure was refined and extended to keep up with new developments in the Web like dubious score optimization techniques, but for general social networks the measure has lost nothing of its value. The basic idea is to combine eigenvector centrality with a uniform initial score assigned to each vertex.

Definition 4.18 (PageRank Centrality [BP98]) *Let $G = (V, E)$ be a directed multi-graph with edge strength ω. The PageRank centrality $c_{PR} : V \to \mathbb{R}_{\geq 0}$ is defined as the normalized values of*

$$c_{PR}(v) = \alpha \frac{1}{n} + (1 - \alpha) \sum_{(u,v) \in E(v)} \omega((u,v)) \cdot c_{PR}(u) \ , \tag{4.17}$$

where $0 < \alpha < 1$ is a free parameter.



(a) bibliographic coupling (b) original graph (c) co-citation

Figure 4.1: Bibliographic coupling and co-citation for the example graph shown in the middle. The multiplicity of edges is shown as labels and reflected by the line width.

Insufficient Connectivity All these centralities except PageRank are only well-defined on at least weakly connected graphs. The first term in the definition of PageRank can be interpreted as an edge of low weight between each dyad, therefore this measure assumes a connected graph implicitly. For the other measures, we use a generalization similar to the distance-based centralities, i. e., we compute the centrality separately for each connected component and account for varying sizes of the components by scaling each component's share proportionally to its number of vertices minus one.

Computation The exact computation of eigenvectors is a non-trivial problem. Since we are only interested in the largest absolute eigenvalue and its corresponding eigenvector the rather simple *power iteration* can be used. Starting with arbitrary initial values, e. g., $c_E(v) = 1$ for all $v \in V$, this methods iteratively calculates better approximated values. Exploiting the fact that the value of each vertex depends only on its neighbors, the running time for one iteration is $\mathcal{O}(n+m)$. The overall running time depends on the convergence rate and a good stop criterion. Note that special care has to be taken if the graph is bipartite. Details on the power iteration can be found in many textbooks on linear algebra, e. g., Wilkinson [Wil65] and for advanced techniques and algorithms for special matrices, see textbooks like [GVL96, PTVF92].

For the computation of hubs & authorities, we do not use the analogy to eigenvector centrality because, even for a sparse graph G, coupling $\mathcal{B}(G)$ and co-citation $\mathcal{C}(G)$ are typically dense, resulting in an unnecessary increase in running time and memory requirement. Instead, we use the directed iteration proposed by Kleinberg [Kle99] which exploits the following mutual dependence of hub and authority scores:

$$\boldsymbol{c}_H^k = A\boldsymbol{c}_A^{k-1} \text{ and } \boldsymbol{c}_A^k = A^T\boldsymbol{c}_H^k \ .$$

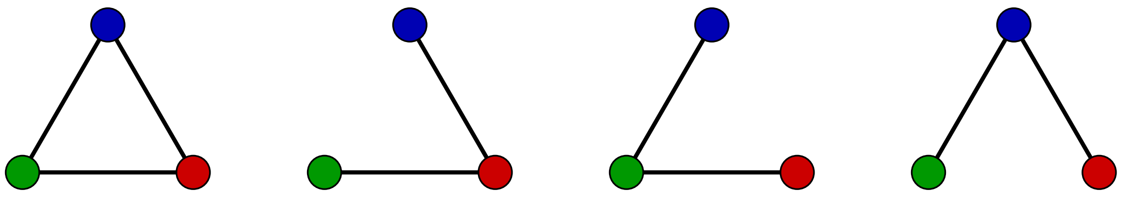


Figure 4.2: A triangle contains exactly three triples.

4.4 Neighborhood Density

For a long time, density has been a frequently used concept to characterize groups and complete networks. In 1998, Watts and Strogatz [WS98] introduced the clustering coefficient, a local density measure for the neighborhood of a vertex. The clustering coefficient $\mathrm{cc}(v)$ of a vertex $v \in V$ represents the likeliness that two neighbors of v are connected.

Definition We will define the clustering coefficient in terms of triangles and triples of a vertex. Let G be a simple, undirected graph. A *triangle* $\triangle$ is a complete subgraph of three vertices and a *triple* is a subgraph of three vertices and two edges, i. e., a path of length 2 (see Figure 4.2). A triangle $\triangle$ is a *triangle of vertex* v if v is any of the vertices of $\triangle$ and a triple is a *triple at vertex* v if v is incident to both edges of the triple.

The clustering coefficient of a vertex $v \in V$ is defined as the fraction of the number of triangles $\lambda(v)$ of v and the number of triples $\tau(v)$ at v, i. e.,

$$\mathrm{cc}(v) = \frac{\lambda(v)}{\tau(v)} , \tag{4.18}$$

where $\frac{0}{0} := 0$ by convention. Note that the number of triples $\tau(v)$ at a vertex v can be formulated in terms of its degree $\deg(v)$ as

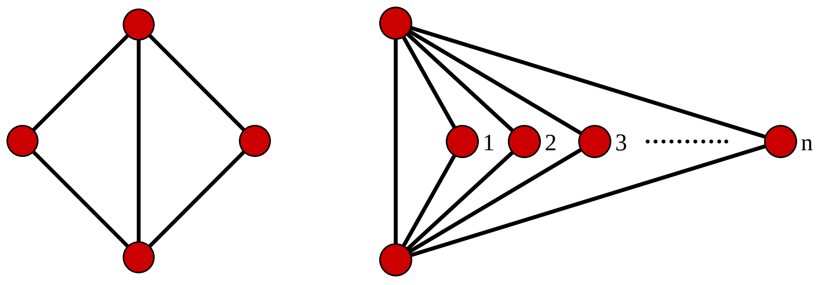
$$\tau(v) = \binom{\deg(v)}{2} = \frac{\deg(v)^2 - \deg(v)}{2} . \tag{4.19}$$

Graph Measures Watts and Strogatz [WS98] defined the clustering coefficient $\mathrm{cc}(G)$ of a graph G as the average of the clustering coefficients $\mathrm{cc}(v)$ of all vertices $v \in V$ with degree $deg(v) \geq 2$, i. e.,

$$\mathrm{cc}(G) = \frac{1}{|V'|} \sum_{v \in V'} \mathrm{cc}(v) = \frac{1}{|V'|} \sum_{v \in V'} \frac{\lambda(v)}{\tau(v)} ,$$

where $V' = \{v \in V \mid \deg(v) \geq 2\}$.

This average has become a popular measure in network analysis, provoking ‘alternative formulations’ by Barrat and Weigt [BW00a] and by Newman et al. [NSW02]. Their reformulations really turned out to be equivalent to each other but not to the



(a) graph with 4 vertices (b) family of graphs

Figure 4.3: Clustering coefficient and transitivity are different. The graph on the left has clustering coefficients $\mathrm{cc}(a) = \mathrm{cc}(c) = 2/3$, $\mathrm{cc}(b) = \mathrm{cc}(d) = 1$, $\mathrm{cc}(G) = (2 + 4/3)/4 \approx 0.83$ and transitivity $t(G) = 3 \cdot 2/8 = 0.75$. For the family of graphs on the right we have $t(G) \to 0$ and $\mathrm{cc}(G) \to 1$ for $n \to \infty$.

original clustering coefficient. The *transitivity* of a graph is defined dependent on the number of triangles $\lambda(G)$ in G and the number of triples $\tau(G)$ in G as

$$t(G) = 3\frac{\lambda(G)}{\tau(G)} = 3\frac{\sum_{v \in V} \lambda(v)}{\sum_{v \in V} \tau(v)} ,$$

where the factor of 3 accounts for the exactly three triples in each triangle (see Figure 4.2). Figure 4.3 illustrates this difference. The example graph exhibits similar values of clustering coefficient and transitivity while in the given family of graphs the values approach one and zero, respectively, for increasing n.

Computation The number of triples $\tau(v)$ for all vertices $v \in V$ can be computed in linear time using Equation 4.19. The common method to compute the number of triangles $\lambda(v)$ for all vertices iterates over all vertices and checks whether the edge between any two neighbors is present. This algorithm has running time in $\mathcal{O}(n \cdot \Delta(G)^2)$ where $\Delta(G) = \max\{\deg(v) \mid v \in V\}$ if the test for edge existence requires constant time, e. g., by using an efficient hashing schema as proposed by Schank [Sch07].

Another approach is to use matrix multiplication since the diagonal elements of A^3, where A is the adjacency matrix of a graph G, contain two times the number of triangles of the corresponding vertex. The resulting running time is in $\mathcal{O}(n^\gamma)$, where γ is the matrix multiplication coefficient. For advanced methods, $\gamma \leq 2.376$ [CW90]. An algorithm by Alon, Yuster, and Zwick [AYZ97] combines both methods by using the standard approach for low degree vertices and fast matrix multiplication on the subgraph induced by the high degree vertices, resulting in a running time of $\mathcal{O}(m^{3/2})$.

Note that for the transitivity, in principle, it suffices to compute $\lambda(G)$ for the whole graph but it is unknown whether there is an algorithm that is asymptotically faster in computing the number of triangles globally than iterating locally over all vertices.

4.5 Cohesiveness

Groups of actors can be defined according to extrinsic properties like common goals, interests, preferences, or other similarities. In contrast, we are interested in groups defined by structural properties of the network. A vast number of semantically very different concepts for defining structural groups have been developed in the past. In the following sections, we describe connectedness, clustering, and role assignment in more detail. In this section, we concentrate on the discovery of cohesive groups.

Two concepts of cohesive groups have been implemented in visone. First, we describe the k-core decomposition in Section 4.5.1 which computes a sequence of nested groups of increasing cohesiveness. Thereafter, concepts of complete and nearly complete groups like cliques are presented in Section 4.5.2. In contrast to cores, these groups are typically overlapping and not all-encompassing, i. e., vertices can be members of several groups and not every vertex belongs to a group.

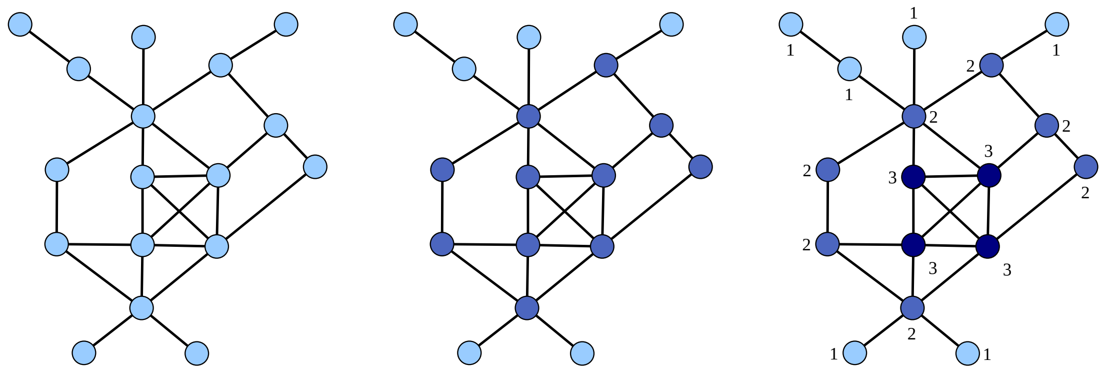
The groups in this section are defined according to a graph-theoretic property. Such a property is called *local* if the membership of a vertex depends only on the connections within the group, i. e., the group exists also in the induced subgraph of the vertices of this group. Typically, only maximal groups are considered. A group is called *maximal*, if it is not contained in a larger group.

In this section, all algorithms are defined for the case of unweighted, undirected simple graphs $G = (V, E)$. Note that *group* is used as intuitive notation for subgraph in the following and has no implicit meaning.

4.5.1 k-core

The concept of *cores* was originally introduced by Seidman [Sei83] and generalized by Batagelj and Zaveršnik [BZ02b]. Constructively speaking, the *i-core* of an undirected graph is defined as the unique subgraph obtained by iteratively removing all vertices of degree less than i. This procedural definition immediately gives rise to a construction algorithm that can easily be implemented. Moreover, it is equivalent to the closed definition of the i-core as the set of all vertices with at least i adjacencies to other vertices in the i-core. Obviously, this is a local and maximal property as described above. The *core number* of a graph is the smallest i such that the $(i + 1)$-core is empty, and the corresponding i-core is called the *core* of a graph. Since each $(i + 1)$-core is complete contained in the i-core, the cores can be seen as a decomposition of the graph in nested groups of increasing cohesiveness. Figure 4.4 depicts the core decomposition of an example graph with a core number of 4. The core decomposition can be computed in linear time with respect to the graph size [BZ02a].

A vertex has *coreness i* if it belongs to the i-core but not to the $(i + 1)$-core. Note that the shells form a partition of the vertices. Informally speaking, the coreness of a vertex can be seen as a robust version of the degree, i. e., a vertex of coreness i retains its coreness even after the removal of an arbitrary number of vertices of



(a) 1-core (b) 2-core (c) 3-core

Figure 4.4: A k-core decomposition with core number 3. Core-shells are indicated by vertex colors from light blue to dark blue.

smaller coreness.

4.5.2 Cliques & Co

A subset $U \subseteq V$ is a *clique* if the induced subgraph $G[U]$ is a complete graph [LP49]. Often, we identify the vertex set u and the induced subgraph $G[U]$ of a clique. A clique u is a *maximal* clique, if there is no other clique $U' \subseteq V$ with $U \subsetneq U'$, i. e., U is not contained in a larger complete subgraph of G. A *maximum* clique is a maximal clique of the maximum number of vertices.

Since every member of a clique is adjacent to every other member, cliques have a number of desirable properties. The density of cliques has the maximum value of 1. The distance between ever pair of vertices is the minimum value of 1. The vertex and edge connectivity of a clique $U = \{u_1, \ldots, u_k\}$ is the maximum value of $k - 1$, i. e., at least $k - 1$ vertices or $k - 1$ edges have to be removed in order to disconnect $G[U]$.

In social network analysis, one is typically interested in either a maximum clique or an enumeration of all maximal cliques. Despite the rather simple structure of cliques, both problems are $\mathcal{NP}$-hard to solve [Kar72, LLK80]. Therefore, we have implemented an algorithm for the enumeration of all maximal cliques with a polynomial delay of $\mathcal{O}(n^3)$ [TIAS77], i. e., the delay until the first output, between consecutive outputs, and past the last output is bounded by $\mathcal{O}(n^3)$. Furthermore, this algorithm requires only linear space $\mathcal{O}(n+m)$. Note that the total running time is not polynomial since a tight upper bound for the number of maximal cliques is $3^{\lceil \frac{n}{3} \rceil}$ [MM65].

Relaxations Since the structural requirements of cliques are very strong, it is likely that there are only few cliques of large size in real-world networks, especially if the existence of edges is afflicted with uncertainty. Typical approaches to handle this problem relax the requirement either of direct connectivity to connections by paths of a given length k or of completeness to the allowance of a small number k of missing edges per vertex. An example for the latter are k-plexes [SF78, Sei80].

In visone, two relaxations of the first type are available: k-cliques and k-clans. A subset $U \subseteq V$ is a *k-clique* if for all vertices $u, v \in U$, $\mathrm{d}_G(u, v) \leq k$. A *$k$-clan* U is a k-clique with $\mathrm{diam}(G[U]) \leq k$. Note that k-cliques are defined with respect to distances in G and not in $G[U]$, thus, it is not a local property. In fact, k-cliques can even be disconnected for $k > 1$. Since many real-world networks have small diameter, large values of maximum distance k are rarely reasonable.

4.6 Connectedness

There are many concepts for the strength of the connectivity of a graph. Common measures consider for example the minimum number of vertices or edges which have to be removed in order to split the graph into disconnected components or the minimum number of vertex- or edge-disjoint paths between dyads. We provide only a small number of rather basic methods to investigate connectivity in visone. These comprise the identification of connected and biconnected components.

Recall from Section 3.1 the definitions of connected components. The connected component of an undirected graph can be computed in linear time using any graph search algorithm like DFS or BFS. All vertices reachable from a single source vertex belong to the same component. As long as unvisited vertices exist, one of them is chosen as next source vertex. By ignoring the direction of the edges, the same algorithm can be used for the calculation of weakly connected components of an directed graph.

Strongly Connected Components The computation of the strongly connected components of a directed graph requires a more elaborate method. The basic idea of a linear-time algorithm presented first by Tarjan [Tar72] is to detect simple cycles during the execution of a DFS since vertices belong to the same strongly connected component if they are part of at least one common (directed) cycle. Note that, if DFS finds an edge (u, v) to a visited vertex v which lies on the current search path from the source, the search path below v obviously induces a cycle. Using a numbering of the vertices in traversal order from 1 to n, we can identify during back-tracking vertices v of the lowest number of any vertex in the same strongly connected component. Finally, these vertices can be used to divide the search tree into the strong components.

Biconnected Components A vertex whose removal increases the number of connected components of an undirected graph is called a *cut-vertex*. A *cut-edge* is defined analogously. A *biconnected component*, also called *block* is a maximal subgraph which cannot be disconnected by removing only one vertex. In contrast to connected components, the biconnected components do not imply a partition of the vertices since the cut-vertices belong to more than one biconnected component.

A linear-time algorithm for the computation of the biconnected components by Hopcroft and Tarjan [HT73] is based on the same idea of cycle detection in DFS as the algorithm for strongly connected components.

4.7 Clustering

The purpose of clustering is the decomposition of a set of entities into 'natural groups'. Originally, clustering was introduced in the data mining research for automatic classification of patterns into groups [JD88]. In the classic clustering theory, entities are embedded in metric spaces and their similarity is derived from the distances between them. Thus, all pairwise similarities were known at the beginning. In social network analysis, the networks are typically sparse. Therefore, tailored concepts and algorithm are needed which take advantage of this special setting.

According to [Gae07], there are two main combinatorial aspects of clustering: the measuring of the quality of a given clustering and the identification of good clusterings. In this section, we follow this classification. First, we present measurements for the quality of clusterings based on the intuitive paradigm of intra-cluster density versus inter-cluster sparsity, i. e., in a good clustering, the subgraph induced by each cluster is dense while there are only few edges between different clusters. Then, we describe selected algorithms for the identification of clusterings. Typically, an algorithm uses (some of) these measures in its computation to decide whether to keep or discard an intermediate state. Based on the findings of Gaertler [Gae07], we restrict the domain of input parameters to appropriate values – or even preset them to a fixed value – in order to simplify their application. For a comprehensive introduction to the clustering of networks, including various concepts not mentioned here, we refer to [Gae07, Gae05].

Preliminaries Let $G = (V, E)$ be an undirected graph. A *clustering* $\mathcal{C} = (C_1, \dots, C_k)$ of G is a partition of the vertex set V into k pairwise disjoint, non-empty subsets C_i. Since every clustering is also a partition, the notation introduced for partitions in Section 3.1 can be used without modifications. Furthermore, the number of intra-cluster edges is denoted by $m(\mathcal{C})$ and the number of inter-cluster edges by $\overline{m}(\mathcal{C})$. In the following, we often identify a cluster C with its induced subgraph in G, i. e., the graph $G(C) = (C, E(C))$. Recall that a cut is a clustering into exactly two parts.

The set of all possible clusterings is denoted by $\mathcal{A}(G)$. The set $\mathcal{A}(G)$ is partially ordered with respect to inclusion. Given two clusterings $\mathcal{C}_1 = (C_1, \dots, C_k)$ and $\mathcal{C}_2 = (C'_1, \dots, C'_l)$, the partial ordering is defined as

$$\mathcal{C}_1 \leq \mathcal{C}_2 \colon \iff \forall\, 1 \leq i \leq k \colon \exists\, j \in \{1, \dots, \ell\} \colon C_i \subseteq C'_j \tag{4.20}$$

Clustering $\mathcal{C}_1$ is called a *refinement* of $\mathcal{C}_2$, and $\mathcal{C}_2$ is called a *coarsening* of $\mathcal{C}_1$.

Typically, we consider graphs where a positive edge weight $\omega : E \rightarrow \mathbb{R}_+$ represents the similarities of the incident vertices. For the unweighted case, a uniform weight of one is assumed. For any set of edges E', we use $\omega(E')$ as shorthand for $\omega(E') = \sum_{e \in E'} \omega(e)$. For simplicity, we require $\omega(E') \neq 0$. Methods for negative similarities or dissimilarities are not available in visone and therefore not considered in the following.

4.7.1 Quality Measures

All presented clustering measures are based on the paradigm of intra-cluster density versus inter-cluster sparsity. More formally, each measure can be expressed as a combination of two independent functions $f, g : \mathcal{A}(G) \rightarrow \mathbb{R}_{\geq 0}$ which measure the density inside the clusters and the sparsity between clusters, respectively, in the following general schema:

$$\text{index}(\mathcal{C}) = \frac{f(\mathcal{C}) + g(\mathcal{C})}{\max\{f(\mathcal{C}') + g(\mathcal{C}') \mid \mathcal{C}' \in \mathcal{A}(G)\}} , \tag{4.21}$$

where $\text{index}(\mathcal{C}) = 0$ by convention if all clusterings $\mathcal{C}' \in \mathcal{A}(G)$ are evaluated to zero. Note that $\text{index}(\mathcal{C}) \in [0, 1]$ by definition. If a measure considers only one aspect, either density or sparsity, the respective other function f or g is constantly zero. Through the normalization of the quality measures by the maximum possible value for any clustering of this graph, they can be used to compare clustering of different graphs.

Typically, a quality measure is based on some intuition of the structure of a good clustering and, consequently, highlights this idea. In order to achieve a more balanced result, different measures can be used in combination with appropriate ratios.

Coverage

The coverage $\text{cov}(\mathcal{C})$ of a given clustering $\mathcal{C}$ is defined as the fraction of the weights of intra-cluster edges of the total edge weight, i. e.,

$$\text{cov}(\mathcal{C}) = \frac{\omega(E(\mathcal{C}))}{\omega(E)} = \frac{\sum_{e \in E(\mathcal{C})} \omega(e)}{\sum_{e \in E} \omega(e)} . \tag{4.22}$$

Using $f(\mathcal{C}) = \omega(E(\mathcal{C}))$ and $g \equiv 0$, coverage fits into the general schema given in Equation 4.21.

Optimum values of coverage are achieved for clusterings of only one cluster or if all clusters are unions of connected components. Furthermore, if more than one cluster is required for a connected graph, the clustering of optimal coverage value corresponds to a minimum cut, which often separates only a small portion of the graph. In order to avoid this undesired behavior, coverage is seldomly used as the only quality measure.

Conductance Cuts

Although minimum cuts are often not appropriate for clustering, cuts in general are a valuable concept. In this section, we consider the cut measure *conductance*. Let $\mathcal{C} = \{C_1, C_2 = V \setminus C_1\}$ be a cut. The conductance evaluates the total weight of the edges $e \in E(C_1, C_2)$ between the two parts and the total weight of the edges $e' \in E(C_i, V)$ incident to one of the two parts of the cut. More formally, the conductance-weight $a(C_i)$ of the part C_i of the cut is defined as

$$a(C_i) = \sum_{\{u,v\} \in E(C_i,V)} \omega(\{u,v\}),\ i \in \{1,2\},$$

and the conductance $\varphi(\mathcal{C})$ of the cut $\mathcal{C}$ is defined as

$$\varphi(\mathcal{C}) = \frac{\omega(E(C_1, C_2))}{\min\{a(C_1), a(C_2)\}}\ . \tag{4.23}$$

Note that $a(C_i) > 0$ since the definition of clustering forbids empty clusters. Finally, we can define the conductance $\varphi(G)$ of a graph G as

$$\varphi(G) = \min\{\varphi(\{C_1, V \setminus C_1\}) \mid \emptyset \neq C_1 \subsetneq V\}\ . \tag{4.24}$$

The only graphs of maximum conductance $\varphi(\mathcal{C}) = 1$ are stars and connected graphs of at most three vertices. The calculation of the conductance of a graph is $\mathcal{NP}$-hard [ACG+02] but there are approximation algorithms with a guarantee of $\mathcal{O}(\log n)$ [KVV00] and $\mathcal{O}(\sqrt{\log n})$ [ARV04], respectively.

Now, we can define two quality measures based on conductance. The intra-cluster conductance $\alpha(\mathcal{C})$ [KVV00] of a clustering $\mathcal{C}$ is defined as the minimum conductance of the subgraphs $G[C]$ induced by the clusters $C \in \mathcal{C}$, i. e., the measuring functions are

$$f(\mathcal{C}) = \min\{\varphi(G[C]) \mid C \in \mathcal{C}\} \text{ and } g(\mathcal{C}) \equiv 0\ .$$

and, since the maximum of $f + g$ is one,

$$\alpha(\mathcal{C}) = f(\mathcal{C}) = \min\{\varphi(G[C]) \mid C \in \mathcal{C}\}\ . \tag{4.25}$$

Since the conductance of a subgraph is small if there is a cut of low value for the respective cluster, a low intra-cluster conductance can be evidence for a too coarse overall clustering and the minimum conductance cut itself may be a good split.

In contrast to intra-cluster conductance, which evaluates cuts inside of clusters, the inter-cluster conductance $\delta(\mathcal{C})$ [BGW07] considers the cuts induced by a clustering $\mathcal{C}$ in the original graph G, i. e., the measuring functions are

$$f(\mathcal{C}) \equiv 0 \text{ and } g(\mathcal{C}) = 1 - \max\{\varphi(\{C, v \setminus C\}) \mid C \in \mathcal{C}\}\ ,$$

and, since the maximum of $f + g$ is also one,

$$\delta(\mathcal{C}) = g(\mathcal{C}) = 1 - \max\{\varphi(\{C, v \setminus C\}) \mid C \in \mathcal{C}\}\ , \tag{4.26}$$

where $g(\{V\}) = 1$ by convention. A low value of inter-cluster conductance indicates a clustering in which at least one cluster $C \in \mathcal{C}$ is strongly connected to the remainder $V \setminus C$.

Performance

The intuition of performance [vD00] is very close to the paradigm of intra-cluster density versus inter-cluster sparsity. For a given clustering, it counts the pairs of vertices which are 'correctly' assigned, i. e., pairs of adjacent vertices in the same cluster and pairs of non-adjacent vertices in different clusters. According to the general schema in Equation 4.21, the resulting index can be formalized as

$$\begin{aligned} f(\mathcal{C}) &= \sum_{C\in\mathcal{C}} |E(C)| \text{ and} \\ g(\mathcal{C}) &= \left|\Big\{\{u,v\} \in V\times V \mid \{u,v\} \notin E \text{ and } u \in C_i, v \in C_j, i \neq j\Big\}\right| . \end{aligned}$$

Calculating the maximum of $f + g$ is $\mathcal{NP}$-hard [SST02], thus, a the trivial upper bound of $n(n-1)/2$, the number of different pairs of vertices, is used in the computation of performance instead. Using some simplifications, the performance $\mathsf{perf}\,(\mathcal{C})$ of a given clustering $\mathcal{C}$ can be defined as

$$\mathsf{perf}\,(\mathcal{C}) = 1 - \frac{2}{n(n-1)} \cdot m(1 - 2\frac{m(\mathcal{C})}{m}) + \frac{1}{2}\sum_{C\in\mathcal{C}} |C|(|C|-1) . \tag{4.27}$$

In contrast to coverage, there is no general characterization of clusterings with maximum performance. Typically, clusterings of many small clusters tend to have a high value of performance. The main disadvantage of performance is the handling of very sparse graphs since $f(\mathcal{C})$, which evaluates only present edges, tends to be negligible in comparison to $g(\mathcal{C})$.

For the generalization to weighted performance, we have to define how to take non-adjacent pairs of vertices into account since, obviously, there is no weight defined for them. The straight-forward approach is to assume a uniform weight of $M \in \mathit{Real}$ for each of these dyads. M should be a reasonable upper bound for the edge weights, usually, their maximum is a good choice. The resulting generalized measuring functions are

$$\begin{aligned} f(\mathcal{C}) &= \omega(E(\mathcal{C})) \text{ and} \\ g(\mathcal{C}) &= \sum_{u\in C_i, v\in C_j, i\neq j, \{u,v\}\notin E} M . \end{aligned}$$

Since this approach neglects the weights of the inter-cluster edges, we use the slight variation

$$\begin{aligned} g'(\mathcal{C}) &= g(\mathcal{C}) + \sum_{u\in C_i, v\in C_j, i\neq j, \{u,v\}\in E} M - \omega(\{u,v\}) \\ &= \overline{m}\,(\mathcal{C}) \cdot M - \sum_{u\in C_i, v\in C_j, i\neq j, \{u,v\}\in E} \omega(\{u,v\}) , \end{aligned}$$

in which the weight of the actual inter-cluster edges is subtracted from the weight which would be counted if no such edges are present at all.

Modularity

The modularity measure [CNM04] has recently gained a lot of attention. Its intuition is to compare the ratio of intra-cluster edges and an approximation of this fraction in a random model. The modularity $\mathsf{q}(\mathcal{C})$ of a given clustering $\mathcal{C}$ is defined as

$$\mathsf{q}(\mathcal{C}) = \sum_{C \in \mathcal{C}} \left(\frac{\omega(E(\mathcal{C}))}{\omega(E)} - \frac{1}{4\omega(E)^2} \left(\sum_{v \in C} \sum_{\{u,v\} \in E} \omega(\{u,v\}) \right)^2 \right) . \qquad (4.28)$$

Note that modularity does not directly realize the intra-cluster density versus inter-cluster sparsity paradigm but an experimental evaluation of Gaertler [Gae07] showed that the resulting clusterings also conform to it. In order to fit modularity into the schema of Equation 4.21, we have to relax the range of the function $g(\mathcal{C})$ to include negative values and then define

$$f(\mathcal{C}) = \omega(E(\mathcal{C})) \text{ and } g(\mathcal{C}) = -\frac{1}{4\omega(E)^2} \left(\sum_{v \in C} \sum_{\{u,v\} \in E} \omega(\{u,v\}) \right)^2 .$$

Also the value of modularity as a whole can become negative. For an unweighted, undirected graph, the following inequalities hold: $-0.5 \leq \mathsf{q}(\mathcal{C}) \leq 1$. Recently Brandes et al. [BDG⁺06] showed that maximizing modularity is $\mathcal{NP}$-hard but some properties of clusterings of maximum modularity are known: there is always a clustering of maximum modularity in which each cluster induces a connected subgraph and all maximum clusterings are of this type if the graph contains no isolated vertices.

4.7.2 Clustering Methods

In this part, we give a short summary of general clustering strategies and then describe the algorithms available in visone. There are three main concepts for computing a clustering, which can often be found in literature: greedy linkage, greedy splitting, and shifting.

In greedy linkage, one starts with a given fine clustering, e. g., singletons, and iteratively merges two clusters until the 1-clustering is reached. In every step, a cheapest merge operation according to some measure should be performed, which can be evaluated locally, i. e., only the two affected clusters are considered, or globally, i. e., for the complete clustering. Finally, the best intermediate clustering according to a clustering measure as presented in Section 4.7.1 is selected.

In greedy splitting, the process of greedy linkage is reversed. Starting from an initial coarse clustering, e. g., the 1-clustering, iteratively one cluster is selected and split into two parts. Again, the split operation should be cheapest according to some measure. Note that, in general, splitting is more complex than linkage if every non-trivial cut of each cluster is considered. A common method to overcome this drawback is to evaluate only appropriate candidates.

Shifting is a more local approach. A given initial clustering is iteratively modified using simple operations until a local optimum is achieved. Typical operations are: move a vertex from one cluster to another, assign a vertex to a newly created cluster, and exchange two vertices of different clusters at once. Methods based on shifting are rarely ever used on their own but are common post-precessing steps of greedy algorithms.

Markov Clustering (MCL)

The basic intuition of Markov Clustering (MCL) is that a *"random walk that visits a dense cluster will likely not leave the cluster until many of its vertices have been visited"* [vD00]. Actually, MCL does not simulate random walks but modifies a matrix M of transition probabilities by a series of *expansion* and *inflation* operations until a recurrent state is reached.

In more detail, the matrix M is initially defined as the normalized adjacency matrix $M(G)$ of G, i. e., $M = M(G) = D(G)^{-1}A(G)$ where $A(G)$ is the (weighted) adjacency matrix and D(G) is the diagonal matrix of the (weighted) node degrees. Note, that $M(G)$ corresponds to random walks of a length of at most one. Then, alternately, an expansion operation simulates e steps of a random walk by taking M to the power of $e \in \mathbb{N}_{>1}$ and the inflation operation takes every entry M_{ij} to the power of $r \in \mathbb{R}_+$ and re-normalizes M. The algorithm is stopped when a previous state of the matrix M recurs. According to [vD00], it is likely that if a recurrent state is reached once it repeats in every iteration. Finally, the clustering is induced by the connected components of the graph underlying the recurrent matrix.

probabilities of

The running time of the computation of MCL is dominated by the matrix multiplication in the expansion operation. To drastically reduce the cubic complexity of a straight-forward implementation, van Dongen [vD00] suggests to maintain the matrix M sparse by keeping only a fixed number of largest values per row. For the user's convenience, we predefine all parameters in accordance with the experimental evaluation of Gaertler [Gae07] and set the exponent for the expansion $e = 2$ and the exponent for the inflation $r = 1.5$.

Iterative Conductance Cutting (ICC)

A prototypical example of a splitting algorithm is Iterative Conductance Cutting (ICC) [KVV04]. As the name suggests, the basic idea is to iteratively split clusters using minimum conductance cuts. Since finding a cut with minimum conductance is $\mathcal{NP}$-hard, a poly-logarithmic approximation algorithm is used. For a cluster $C \in \mathcal{C}$, the vertices are ordered according to an eigenvector of the second largest eigenvalue of the normalized adjacency matrix $M(C)$ of C. Among all cuts that split this order into two parts, one of minimum conductance is chosen. Clusters are split as long as the approximation value of the conductance is below a threshold $\alpha \in [0, 1]$.

The overall running time of ICC depends on the number of iterations and on the eigenvector computation in each iteration which dominates the conductance cut approximation. The runtime of such an eigenvector computation highly depends on the structure of a subgraph and is in $\mathcal{O}((n+m)\log n)$ in the worst case.

The threshold α can be used to adjust the granularity of the final clustering. Since not all possible values are reasonable, its domain is restricted to $[0, \frac{1}{2}]$ in visone.

Geometric MST Clustering (GMC)

Geometric MST Clustering (GMC) combines spectral positioning and a geometric clustering technique [Gae02, BGW07]. First, a d-dimensional embedding is constructed from d distinct eigenvectors $x_1, \ldots, x_d$ of $M(G)$ associated with the largest eigenvalues less than 1. Then, edge lengths ℓ are defined by the distances in the embedding and a minimum spanning tree (MST) of the weighted graph is determined. Finally, for a threshold value τ, a clustering is defined by the connected components of the forest $F(T,\tau)$ which emerges when all edges $e \in E(T)$ with $\ell(e) > \tau$ are excluded from T.

Note that the connected components of $F(T,\tau)$ induce a clustering for each threshold τ and that there are at most $n-1$ thresholds resulting in distinct clusterings. Furthermore, the final clustering depends only on the threshold and is independent of the MST [Gae02]. Therefore, we chose a threshold which optimizes a quality measure like the one described in Section 4.7.1.

The running time of GMC depends on the computations of the eigenvectors, the MST, and quality measure. In contrast to ICC, only one eigenvector computation is needed. Assuming that the quality measure can be computed fast, the asymptotic time and space complexity of the main algorithm is dominated by the MST computation.

In visone, the range of the dimension is restricted to $d \in [3,5]$. Furthermore, four combinations of the quality measures described in Section 4.7.1 are predefined for the calculation of the threshold τ:

- **Coverage**: only Coverage is considered.
- **Coverage and Performance**: the combination of Coverage and Performance is considered with a weighting of $1:2$.
- **Modularity**: only Modularity is considered.
- **Mean of all Three**: the combination of Coverage, Performance, and Modularity is considered with a weighting of $1:2:2$.

4.7.3 Conclusion

Gaertler [Gae07] gives an extensive experimental evaluation of the properties of the presented clustering algorithms for different classes of graphs. For dense graphs, all algorithms tend to coarse clusterings containing a low number of clusters only. For sparse graphs, ICC and MCL compute clusterings of a large number of clusters. While it is unclear if, or how, these two algorithms can be modified to calculate a more balanced number of clusters, GMC is easily extended to respect upper or lower bounds, or even both, for the number of clusters by limiting the search space of the threshold τ.

In general, all three algorithm compute clusterings with high values of the presented quality measures. While ICC and MCL have advantages with respect to performance, GMC outperforms MCL with respect to inter-cluster conductance and ICC with respect to coverage.

4.8 Network Statistics

Network statistics are used to describe essential properties of vertices, edges, groups, or the whole graph with reduced information, e. g., in a single value or a series of numbers. Thereby, we can disregard the probably large structure of the underlying graph and concentrate on a restricted set of statistics. Another common application is the differentiation between dedicated classes of graphs there the main focus is not the reduction of information but to decide whether a graph belongs to a class or not.

In general, network statistics can have very different peculiarities. Four different types are identified in [BS05] and can be described by two pairs of exclusive attributes:

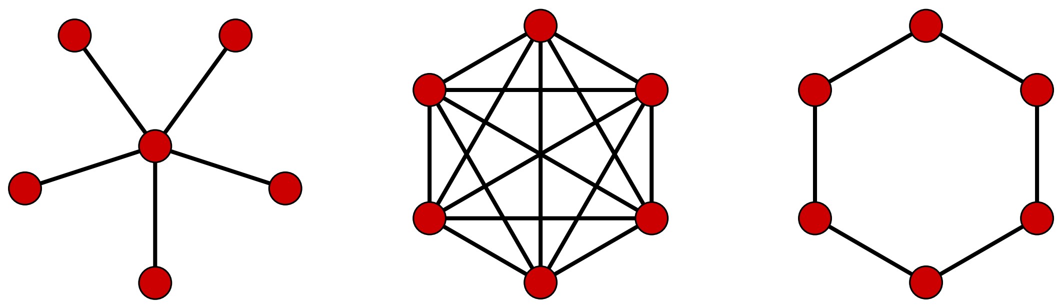
- single-valued vs. distribution and
- global vs. local,

where the term *distribution* denotes multiple-valued statistics and *local* refers to any kind of local graph item like a vertex, an edge, or any subset of vertices and edges. In this section, we concentrate on global (single-valued) statistics, i. e., methods which assign a single value to the whole graph. Examples of local single-valued statistics are all centralities and the clustering coefficient for vertices (see Sections 4.3 and 4.4).

We have already seen some examples of network statistics in previous sections although not explicitly called this way, namely

- the number of vertices and edges of the graph in Section 4.2,
- the clustering coefficient and the transitivity in Section 4.4,
- the core number of a graph in Section 4.5.1, and
- the number of connected components in Section 4.6.

Aggregation of Element-Level Indices In principle, a global statistic can be derived for every element-level index $f : X \to \mathbb{R}_{\geq 0}$ by statistical methods of aggregation like summation, averaging, the calculation of the variance of all $f(x)$, or



(a) a star (b) a clique (c) a cycle

Figure 4.5: Graphs with high (star) and low centralization (cycle, clique). Centrality scores are visualized as radial positions and vertex sizes (see Section 6.2). As a rule of thumb, keep in mind that classes of graphs whose name starts with the character 'c' feature a uniform centralization.

the maximum/minimum operator. This approach is called *globalization* in [BS05]. The core number and the clustering coefficient are such statistics while, for example, the transitivity is not. In the following, we describe additional statistics aggregated from element-level indices which have attracted special attention in the literature.

Centralization Intuitively, it is clear that some graphs are more centralized than others, i. e., they are more biased on the most central vertex than others. A star is a prototypical example of a very centralized graph (for reasonably defined centrality indices) whereas uniform graphs like cliques or cycles feature a perfectly balanced centralization (see Figure 4.5).

Definition 4.19 (Centralization [Fre77]) *Let $c_\chi : X \to \mathbb{R}_{\geq 0}$ be a centrality index of a graph G. The centralization $c_\chi(G)$ of G with respect to $c_\chi : X \to \mathbb{R}_{\geq 0}$ is defined as*

$$c_\chi(G) = \frac{1}{n-1} \sum_{x \in X} \overline{c_\chi} - c_\chi(x) \, , \tag{4.29}$$

where $\overline{c_\chi}$ is the maximum score.

Intuitively speaking, the centralization is defined as the average difference of the centrality scores from the maximum. Note that this method can be used to calculate a global statistic for any other element-level index equally well and that also other statistical methods are occasionally called centralization.

Distance A number of commonly used, intuitive statistics are based on shortest-path distances. In order to highlight the derivation from centrality indices, we state the directed definitions and also give reformulations in terms of the distance-based centralities introduced in Section 4.3.3.

The *average* or *characteristic distance* $\overline{\mathrm{d}}(G)$ of a graph G is the arithmetic mean of all distances in the graph, i. e.,

$$\overline{\mathrm{d}}(G) = \frac{1}{n^2 - n} \sum_{u \neq v \in V} \mathrm{d}(u,v) = \frac{1}{n^2 - n} \sum_{v \in V} \frac{1}{c_C(v)} \; .$$

For disconnected graphs, we obviously have $\overline{\mathrm{d}} = \infty$, hence it might be useful to restrict the averaging to all connected pairs $P = \big\{\{u,v\} \in V \times V \mid u \neq v \text{ and } \mathrm{d}(u,v)\big\}$, leading to the *average connected distance*

$$\overline{\mathrm{d}}(G) = \frac{1}{|P|} \sum_{\{u,v\} \in P} \mathrm{d}(u,v) \; .$$

We denote both statistics with $\overline{\mathrm{d}}$, since they are equal for (strongly) connected graphs and only the second is meaningful otherwise.

Recall the definition of the eccentricity $\varepsilon(v)$ of the vertex v as the maximum distance to all other vertices $v \neq u \in V$. The *radius* $\mathrm{rad}(G)$ of a graph G is defined as the minimum eccentricity of all vertices, i. e.,

$$\mathrm{rad}(G) = \min_{v \in V} \varepsilon(v) = \min_{v \in V} \frac{1}{c_E(v)} \; .$$

In contrast, the *diameter* of a graph G is defined as the maximum eccentricity of all vertices, i. e.,

$$\mathrm{diam}(G) = \max_{v \in V} \varepsilon(v) = \max_{v \in V} \frac{1}{c_E(v)} \; .$$

Conclusion The running time for the calculation of global statistics of an element-level index is dominated by the computation of the index itself since the relevant statistical operations for aggregation require linear time only. The distance-based statistics presented above require the solution of the all-pairs shortest-paths problem (see Section 3.1), thus, they can be computed from the respective centrality indices. Since no additional costs are introduced, visone calculates the described global statistics automatically whenever the respective local measure is computed.

Chapter 5

General Graph Layout

Network visualization deals with all aspects of representing relational structures and spans a diverse field ranging from matrix-like representations to figurative drawings of nodes connected by lines. The use of visual images is common in many branches of science and can also be found in social network analysis since the very beginning. While accuracy, expressiveness, and workmanship of the drawings has increased with the broad availability of computational power, there is a lack of automatic drawing routines tailored for the special needs of social network analysis.

In visone, we concentrate on the representation most common in practice, figurative two-dimensional drawings of vertices as graphical symbols and edges as straight lines, curves, etc. connecting vertices [Fre00]. The entirety of the defining topological and geometric features of such a representation is commonly called a *graph layout*. For example, the layout is fully defined by the positions of the vertices if only straight-line edges are allowed. Properly chosen graphical features like colors, shapes, and relative sizes of the elements are very important for the overall appeal of the drawing and an inadequate selection can drastically reduce the usefulness of an otherwise informative layout. They are not part of the methodological core of the layouts but can either be incorporated in the algorithms or be considered in post-processing.

We distinguish two types of drawings of graphs, *general layouts* and *analytic visualizations*. General graph layout techniques like force-directed or spectral layouts depend only on the link structure (and maybe present edge weights) of the network. Clearly these layouts can be used to get a first impression and an overview of the general graph structure but often they also convey deeper analytical insights. For example, force-directed methods can be used to distinguish sparse and dense subgroups and to display similarities. In this chapter, the describe the circular, force-directed, and spectral layout algorithms present in visone.

Analytic visualization refers to drawings which express the exact results of an analysis. Basic visualizations can be as simple as scaling the width of the edges proportionally to numerical values or coloring the vertices according to a clustering but more elaborate visualizations incorporate a constrained layout, i. e., analytic result are mapped to the positions of the vertices and a special edge routing is employed. We present analytic visualizations in Chapter 6 and detail our novel algorithms used in the visualizations in Chapter 7.

5.1 Preliminaries

For simplicity we consider only connected, undirected simple graphs $G = (V, E)$ in this chapter. Since none of the presented methods considers the direction of edges, direction is safely ignored by resorting to the underlying undirected graph. Multi-edges can be added either in a post-processing step in parallel to existing edges or are easily handled by straight-forward modifications of the presented algorithms. Unless otherwise noted, disconnected components of a graph are laid out separately and afterwards arranged side by side.

Edges are represented as straight lines by all layouts presented in this section, therefore a two-dimensional layout is fully defined by a function $p : V \longrightarrow \mathbb{R}^2$ which assigns each vertex $v \in V$ a two-dimensional position $p(v) = (x(v), y(v))$. Sometimes we use the equivalent vector notation $p_v = (x_v, y_v)$ instead.

For the illustration of the layout algorithms in this section, we use the underlying undirected graph uGCN1996 of the German company network of 1996 from a study of by Höpner and Krempel [HK03]. From the network of capital interlocks of the 100 largest German-based companies in 1996, this graph is formed by the largest connected component (for a more detailed description, see Section 6.6). It consists of 60 vertices and, in its simple, undirected version, of 125 edges.

5.2 Criteria of Good Layouts

Drawings must neither be misleading nor hard to read. Hence, there are two obvious criteria for the quality of social network visualizations:

- Is the information represented accurately?
- Is the information conveyed efficiently?

Following Brandes et al. [BKR$^+$99], these criteria translate to three aspects which should be carefully considered when creating visualizations:

- the *substance* the viewer is interested in,
- the *design* which maps the data to graphical features, and
- the *algorithm* employed to realize the design.

For network visualization, frequently named common criteria for layout quality are:

- Vertices should spread well on the drawing area.
- Adjacent vertices should be close and non-adjacent vertices should not.
- Vertices and edges should not cover each other.

Besides these, concrete properties of good layouts in graph drawing literature are small edge length, few edge crossings, and high angular resolution of incident edges.

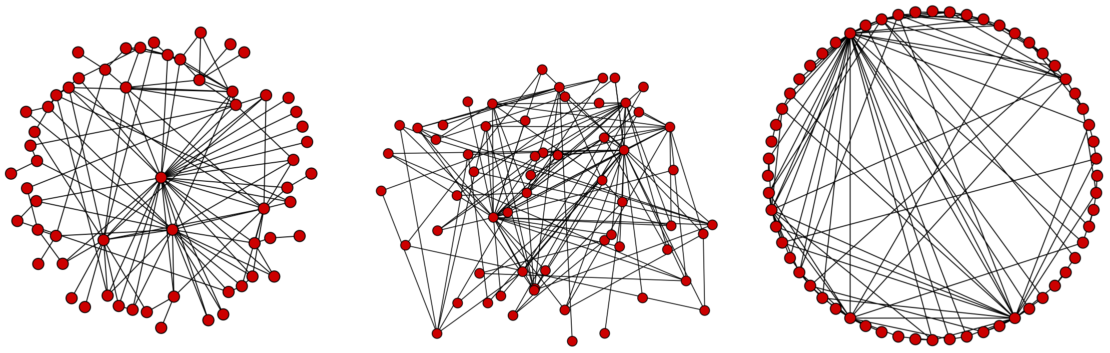

(a) radial visualization of stress centrality (b) random layout (c) circular layout

Figure 5.1: A radial visualization of stress centrality, a random layout, and a circular layout of the underlying undirected graph of the German company network of 1996 (uGCN1996) with 60 vertices and 125 edges.

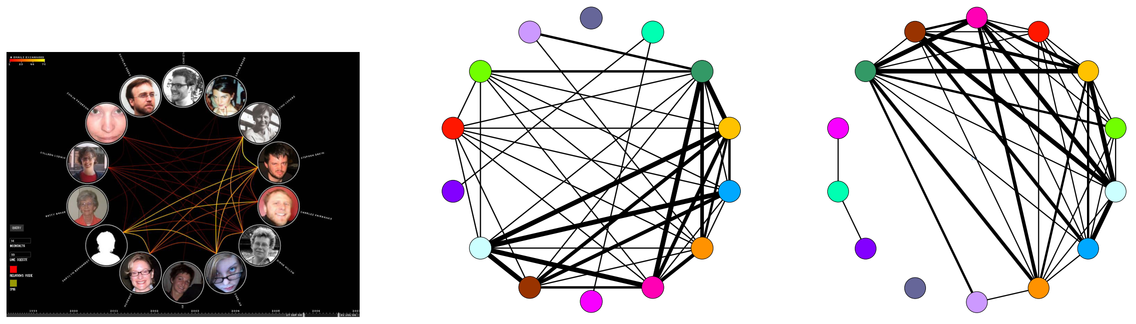
5.3 Random Layout

As the name suggests, vertices are placed at random positions in a random layout. The applicability of this method is very limited because typically resulting layouts are cluttered even for small, sparse graphs (see Figure 5.1(b)). In visone, a random layout is used for the drawing of networks imported from file formats which do not contain positional information like plain-text adjacency matrices or UCINET's DL.

5.4 Circular Layout

In a circular layout, the vertices are constrained to distinct positions along the perimeter of a circle. An important optimization criterion is to minimize the number of edge crossings in such layouts. As a side effect, this tends to reduce the total edge length, since shorter edges typically cross fewer edges. Note that the number of crossings depends only on the cyclic order of the vertices and not on the exact positions. Nevertheless, circular crossing minimization is $\mathcal{NP}$-hard [MKNF87]. In Section 7.1, we present a two-phase heuristic for crossing reduction in circular layouts which is fast, conceptually simpler than previous heuristics, and our extensive experimental results indicate that it also yields fewer crossings.

Circular layouts are a very popular representation of social networks because the uniform placement induces no implicit hierarchy between the vertices, edges and vertices are clearly separated and do not cover each over, and last but not least they are easy to understand. Although the visual appeal of circular layouts can be increased dramatically by a good crossing reduction algorithm, many applications, e. g., the nowadays ubiquitous visualizations of Web 2.0 services, obviously fail to apply one (for an example, see Figure 5.2). This is not only an esthetic issue but also



(a) original drawing (b) layout of the original drawing (c) optimized layout

Figure 5.2: Circular layouts of an email communication network. The original layout is produced by the software *email map* of Christopher Paul Baker [Bak]. For an example of suboptimal placement, observe the vertices of degree 1 which are part of a small disconnected subgroup.

promotes incorrect analytic assumptions, for example a miss-perception of individual connections and global density.

The circular layout algorithm available in visone applies the described crossing reduction heuristics to compute a suitable order of the vertices. For the user's convenience, it chooses the diameter of the circle to fit the screen best. Finally, the vertices are positioned on the perimeter either equidistantly or with respect to their sizes, i. e., the share of the length of the perimeter of each vertex is proportional to the size of an enclosing disc around it.

Note that a circular layout is also the base for the radial and group visualizations and, with extensions, for the multi-circular visualization.

5.5 Force-Directed Layouts

Force-directed approaches use a physical analogy to model the graph drawing problem, namely a system of forces acting on the vertices. Typically, there are forces striving to maintain given distances between adjacent vertices and repulsive forces between all other pairs of vertices. Then, the objective is to find a drawing where the net force acting on each vertex is zero. Equivalently, a potential energy is associated with the drawing and a configuration is sought for which this energy is local minimum. Therefore, a force-directed approach consists of two parts: the energy model that quantifies the interactions of the vertices (and maybe the edges) and an optimization algorithm for computing an (approximated) equilibrium configuration of the model.

Common Characteristics Force-directed layouts are very popular for a couple of reasons. Often, they lead to fairly good layouts, especially for medium-sized graphs. For such graphs, the interplay of attracting and repelling forces realizes the first two criteria of good layouts stated above. Additionally, they are very intuitive because of their relation to the real physical world. Furthermore, it is comparably easy to incorporate custom constraints into these models, e. g., to consider not only ‘points and lines’ but also the area of the elements or varying preferred distances between different pairs of vertices.

Generally speaking, force-directed layouts are particularly suited for sparse graphs with few shortcuts only, i. e., only some edges connect very different regions of the graph. Resulting layouts often expose the inherent symmetric and clustered structure of a graph, and feature a well-balanced distribution of vertices and few edge crossings. Contrarily, in dense graphs or graphs of low diameter, vertices tend to cluster in the central area of the drawing.

Fundamental Methods Early applications of force-directed methods for circuit layout [FI65] date back to the 1960s. The seminal work of Eades in 1984 on his *‘spring embedder’* [Ead84] caused large interest in force-directed methods lasting until today. In his model, edges attract their endvertices following a logarithmic force law like imaginary springs and vertices repel each other with an inverse-squared force law like positively charged electrical items. For nearby positioned adjacent vertices the repulsive force dominates the attraction resulting in an implicitly defined rest length of the edges. Eades’s optimization algorithm iteratively improves an initial random layout by moving all vertices simultaneously in proportion to the net force acting on them.

In the following, endless modifications of the original spring embedder have been developed in order to speed up computation and to improve layout quality. Fruchterman and Reingold [FR91] enhanced the original model by modifying the force laws to allow faster evaluation. Furthermore, their algorithm ignores the repulsive forces between vertices far away from each other and introduces a cooling parameter which increasingly limits vertex movement in later iterations. These modifications introduce a trade-off between quality and computation time: the algorithm converges faster to a stable state but this state is typically not a local optimum. Other variations proposed incorporate various constraints into the model, e. g., avoidance of overlapping vertices, restricted vertex positions, and clustering constraints that cause the algorithm to treat subgraphs independently of each other [HM98, WM96].

Kamada and Kawai [KK89] discarded the electrical charges and instead introduced springs acting in accordance with the more realistic Hooke’s Law between every pair of vertices. These springs are characterized by individual rest lengths and exert a force linearly proportional to their deformation on their endvertices, i. e., vertices are attracted if their distance is larger than the rest length and repelled otherwise. For defining the rest length, Kamada and Kawai used the shortest-path distance between the vertices, which is backed by the assumption that every path in the graph is best represented by a straight line. Instead of moving all vertices in the direction indicated by the force acting on them, their optimization reduces the inherent potential energy of the model by changing one vertex at a time. Later, Cohen [Coh97] pointed out that this approach is closely related to multidimensional scaling (MDS) which shifted attention from custom optimization algorithms to powerful numerical optimization methods like stress majorization [GKN05]. These techniques improve computational speed and sometimes even layout quality.

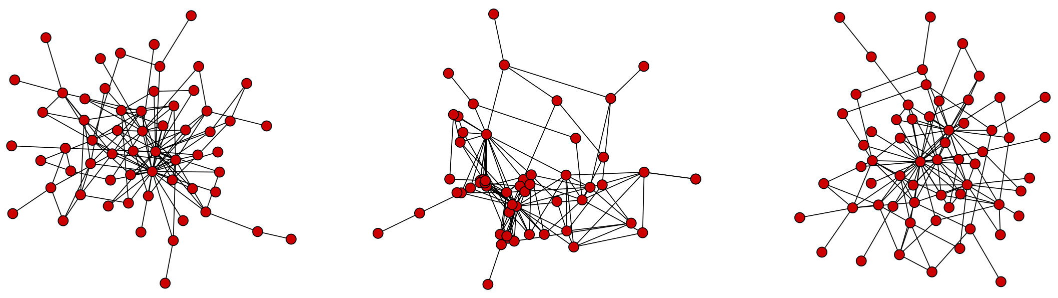
In the following we describe the three force-directed methods available in visone, namely a highly customizable classical force-directed layout and layouts based on classical MDS and on stress majorization. See Figure 5.3 for a comparison of the layouts generated by these methods.

5.5.1 Classical Force-Directed Layout

The classical force-directed layout algorithm is based on the *smart organic layouter* available in the yfiles graph library [yWo08b], which extends the model of Fruchterman and Reingold [FR91] by support for various constraints, e. g., preferred edge lengths and avoidance of overlapping vertices, and incorporates heuristics to speed-up the optimization. For convenience, we hide the technical constraints from the user of visone and instead provide more intuitive parameters.

Typically, the dilation of drawings created by similar methods depends on the interplay of parameters like the strength of the repulsive and attractive forces, the preferred edge lengths, and specific compaction methods. We liberate users from tampering with these parameters. In order to keep the dilation of the drawing roughly constant, the sum of the preferred edge length L_p is set to equal the current total edge length L, where $L = \sum_{\{u,v\}\in E} \mathrm{d}(u,v)$. Since this cannot guarantee perfect fit, the layout is rotated and scaled to the currently visible viewing area. Nevertheless, experienced users can customize the exact behavior by changing the following properties of the algorithm.

- The *scope* defines whether the algorithm is allowed to move all vertices, only selected ones, or mainly selected vertices. The last option allows the movement of unselected vertices by a small degree only.
- If *preferred edge length* is set to *uniform*, the preferred length of each edge equals $1/m \cdot L$, Otherwise, an edge index ℓ is selected and the preferred length of an edge e is set proportional to its share of the total value of ℓ, i. e., it equals $\ell(e)/\sum_{e'} \ell(e') \cdot L$.
- The *dilation* d is an exponential scaling for the total preferred edge length L_p, i. e., $L_p = 1/32 \cdot 2^{10d} \cdot L$, where $0 \leq d \leq 1$. Note that $L_p = L$ for $d = 0.5$.
- If the *vertex size* is considered, the length of the edges is measured between the borders of the vertices. Normally, the distance is measured from center to center.
- By default, *vertex overlaps* are avoided by the algorithm. Note that the final scaling phase can move vertices on top of each other again but typically overlaps are reduced significantly by this option.



(a) classical force-directed (b) classical MDS (c) stress majorization

Figure 5.3: Drawings of the network uGCN1996 using the three different force-directed methods of visone. Uniform edge lengths and uniform vertex sizes are used for all methods.

- The *quality/time ratio* allows to balance the layout quality versus the time consumption. Higher values result in a nicer layout but also in an increase of running time.
- The *maximum duration* imposes a strict upper bound on the running time of the algorithm by aborting the optimization process independently of the current convergence ratio.
- In *deterministic mode*, the algorithm is initialized in the same way as in the preceeding run, therefore, for an unchanged network, the same layout is produced.

5.5.2 Classical Multidimensional Scaling

Multidimensional scaling (MDS) refers to a family of techniques that are used to represent high-dimensional data in a lower dimensional space. Given a symmetric matrix $\Delta \in \mathbb{R}^{n \times n}$ of metric distances or *dissimilarities* δ_{uv} between all pairs of items $u, v \in \{1, \dots, n\}$, the objective is to find positions $p_u \in \mathbb{R}^d$ in d-dimensional space for each item u, such that $\|p_u - p_v\| \approx \delta_{uv}$, i. e., the distances in the d-dimensional space are close to the given distances. In graph drawing, the dimension d is either two or three and the dissimilarities typically equal the shortest-path distance, i. e., $\delta_{uv} = \mathrm{d}(u, v)$.

According to Kadushin, MDS methods have been used for social network visualization as early as in the 1960s [Bra01b], and they are still in use today [KBM94, KBM06]. As for general force-directed methods, a large number of variations for specific application domains and to speed up computation have been proposed. Interestingly, the model of Kamada and Kawai [KK89] corresponds to a variant of MDS introduced by Kruskal [Kru64], the so-called *distance scaling*. Reformulated in terms of MDS, its objective is to fit Euclidian geometric distances directly to given shortest-path distances, i. e., the considered matrix consists of the shortest-path distances as mentioned above.

The term *classical MDS* or *classical scaling* refers to the first MDS algorithm proposed by Torgerson [Tor52]. Its objective is to fit inner products rather than distances. In contrast to distance scaling, its global optimum can be computed directly by a spectral decomposition using power-iteration. Since the running time is $\mathcal{O}(nm)$ for the shortest-path computation and $\mathcal{O}(n^2)$ for each iteration, without modifications, this method is not suitable for large graphs but satisfactory for many typical applications. Recently, sampling-based approximations for classical MDS were proposed which yield layout algorithms fast enough even for very large graphs [BP07, dST02].

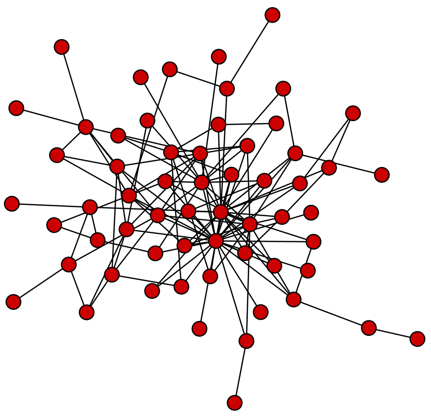
visone contains only a basic implementation of classical MDS which does not support advanced features like the specification of preferred edge lengths since stress majorization, described in the next part, is a similar but superior method.

5.5.3 Stress Majorization

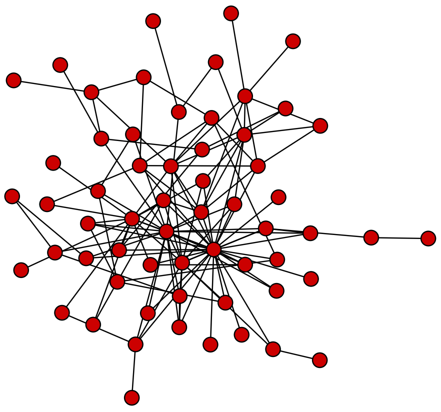
In the MDS community, the potential energy of the model of Kamada-Kawai [KK89] is known as *stress function* and a powerful method for its optimization – *stress majorization* – has been used for more than two decades now. Gansner et al. [GKN05] adapted this technique for graph drawing and showed empirically that it is superior to the original optimization both in terms of layout quality and running time. Furthermore, they proposed some smaller extensions to the original model which for example improve the usage of the drawing area. Many extension of the original model [DKM06, DM08] and applications for the drawing of large graphs [BBP08] showed the usefulness and efficiency of this method.

The basic idea is to iteratively minimize an appropriate upper bound of the stress function, the so-called majorization. This upper bound decreases monotonically until it converges at the same optimum as the stress itself. Its iterative minimization is efficiently computed using simple matrix operations which are available in highly optimized libraries. A small experimental evaluation of Gansner et al. [GKN05] showed that the number of iterations increases only moderately with the number of vertices, making this method suitable for graphs of thousands of vertices. Regarding the quality, they claim that this method and the method of Kamada and Kawai achieve in many cases about the same stress level and that majorization is sometimes significantly superior.

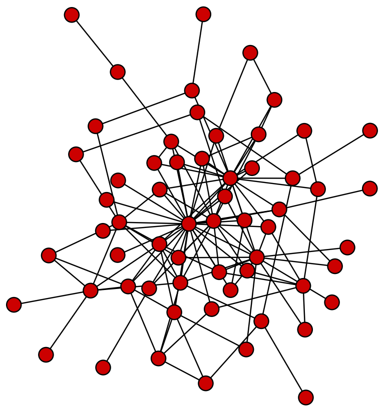
Experiments by Gansner et al. [GKN05] and application of stress majorization to draw the AS network [BBP08] suggest that a uniform preferred edge length *"makes the neighborhood of high degree vertices too dense"* [GKN05] if the degree distribution is very skewed. Instead, they both propose to scale edges by the degree of their endvertices. Therefore, the implementation of stress majorization in visone allows varying preferred edge lengths specified by attribute values (see Figure 5.4). Note that other force-directed methods like the classical force-directed layout can benefit from such a scaling as well. Furthermore, users can restrict the layout algorithm to change the positions in one dimension only.



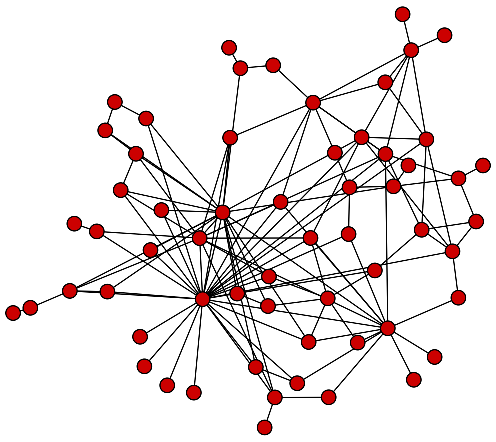
(a) classical force-directed



(b) with adjustment



(c) stress majorization



(d) with adjustment

Figure 5.4: Drawings of the network uGCN1996 using original and adjusted force-directed layouts, i. e., preferred edge lengths are set proportional to the degree of the incident vertices. For this graph, the benefit for stress majorization is apparent while our classical force-directed layout improves only marginally.

5.5.4 Conclusion

Force-directed methods often expose the inherent symmetric and clustered structure of a graph, and feature a well-balanced distribution of vertices and few edge crossings, especially for sparse graphs with few shortcuts. The advantage of the classical force-directed method is its customizability, in particular the attention to the sizes of the vertices, while stress majorization combines a low computation time and a good layout quality. Both methods can benefit significantly from adjusted preferred edge lengths. The classical MDS layout can be useful for specific applications but is inferior to the other methods in general. Since the absolute positions of the vertices and the absolute lengths of the edges are typically insignificant, all layouts are rotated and scaled to fit the current visible viewing area best.

5.6 Spectral Layout

Spectral layout, first introduced in 1970 by Hall [Hal70], denotes the use of eigenvectors of graph-related matrices such as the adjacency or the Laplacian matrix as coordinate vectors. In visone, this method is defined for simple, undirected graphs $G = (V, E)$ with positive edge weights $\omega : E \to \mathbb{R}_+$. The weight represents similarity of vertices and can therefore be seen as an edge strength in terms of Section 4.1. For multi-graphs, parallel edges are deleted and their weights are accumulated.

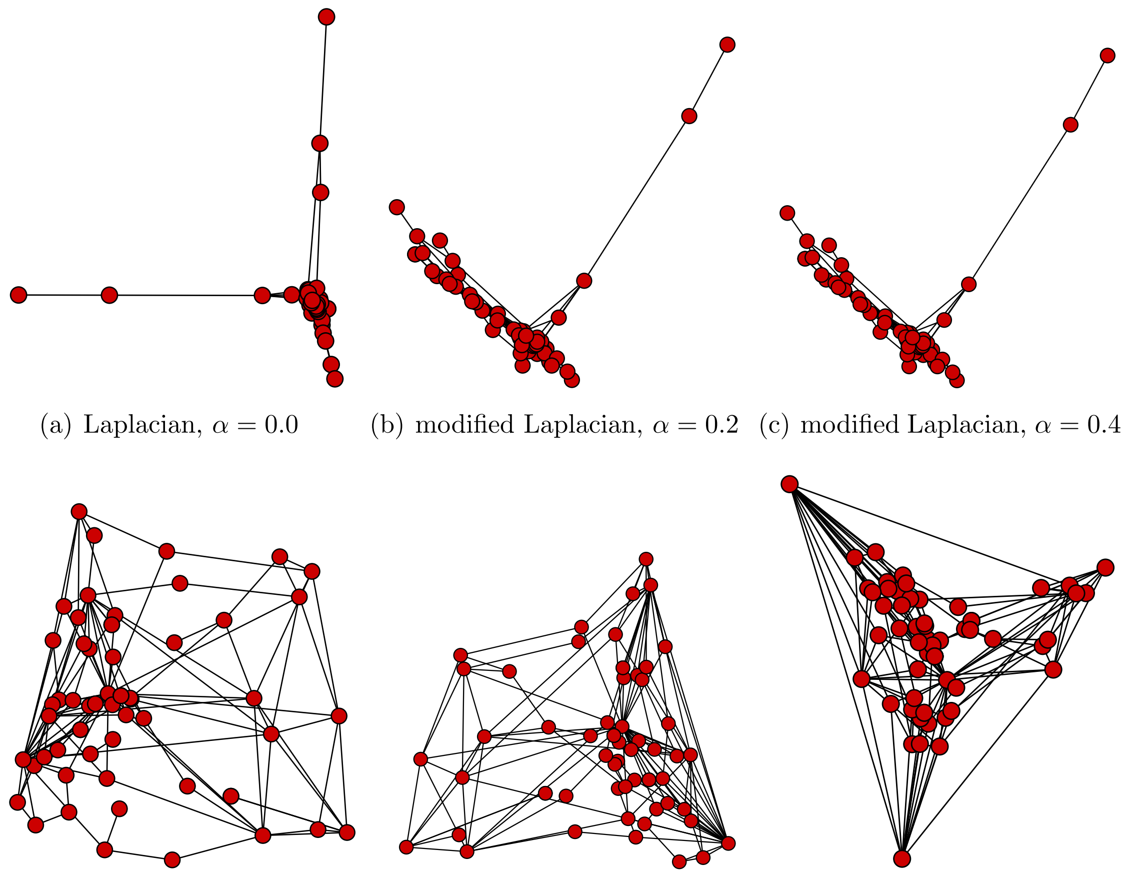
Spectral layouts are good at displaying symmetries of the graph, and structurally equivalent vertices (vertices with identical neighborhood) are even placed in the same location.

Definition For any graph-related matrix $M(G)$, a (two-dimensional) spectral layout of G is defined by two eigenvectors x and y of $M(G)$. We will only consider layouts derived from the Laplacian matrix $L = L(G)$ of G, which is defined by the elements

$$l_{u,v} = \begin{cases} \sum_{w \in V} \omega(\{u,w\}) & \text{if } u = v \,, \\ -\omega(\{u,v\}) & \text{if } u \neq v \text{ and } \{u,v\} \in E \,, \\ 0 & \text{otherwise.} \end{cases}$$

In matrix notation, this can be expressed as $L(G) = D(G) - A(G)$, where $A(G)$ is the (weighted) adjacency matrix and $D(G)$ is the diagonal matrix of (weighted) vertex degrees.

The Laplacian matrix has many interesting properties (see, e. g., [Moh91]), which are useful for the computation of layouts. The rows of $L(G)$ sum up to 0, thus, the vector $\mathbf{1} := (1, \ldots, 1)^T$ is a trivial eigenvector for the eigenvalue 0. Since G is undirected and its weights are non-negative, $L(G)$ is symmetric, all eigenvalues λ_i are non-negative real numbers, and the corresponding eigenvectors are pairwise orthogonal. Hence, the spectrum can be written as $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ with



(a) Laplacian, $\alpha = 0.0$ (b) modified Laplacian, $\alpha = 0.2$ (c) modified Laplacian, $\alpha = 0.4$

(d) modified Laplacian, $\alpha = 0.6$ (e) modified Laplacian, $\alpha = 0.8$ (f) adjacency matrix, $\alpha = 1.0$

Figure 5.5: Drawings of the network uGCN1996 by spectral layouts derived from the modified Laplacian for different values of α. Extremal values result in a dense central cluster while middle values diffuse the vertices in a larger area.

corresponding unit eigenvectors $u_1, \ldots, u_n$.

In a spectral layout based on the Laplacian, the two-dimensional coordinates $p_v = (x_v, y_v)$ of each vertex v are defined by its corresponding entries of the unit eigenvectors u_2 and u_3 as $x_v = (u_2)_v$ and $y_v = (u_3)_v$, where u_2 and u_3 are the orthogonal unit eigenvectors to the second and third smallest eigenvalues λ_2 and λ_3 of the corresponding Laplacian matrix $L(G)$. Since these eigenvectors are also orthogonal to $\mathbf{1}$, these layouts are centered around the origin.

Modified Laplacian If the graph is unbalanced, the spectral layout of the Laplacian matrix clusters most vertices in the center of the drawing and places only some loosely connected vertices far away. This effect can be countered by a modified Laplacian

$$L'(G) = (1 - \alpha)D(G) - A(G) ,$$

in which the diagonal is weakened by a constant factor α, $0 \leq \alpha \leq 1$. Note that this approach includes spectral layouts derived from the unmodified Laplacian for $\alpha = 0$

and from the adjacency matrix for $\alpha = 1$ respectively. Figure 5.5 highlights the effect of different values of α on the graph uGCN1996.

Computation For sparse graphs of moderate size like the ones addressed by visone, a practical method to determine the corresponding eigenvectors is power iteration. As proposed in [BFP07], we use the matrix $\hat{L} = g \cdot I - L$ instead of L to increases the numerical stability. Note that $\hat{L}$ has the same eigenvectors as L but in reversed order. After each step, orthogonality is restored. Spectral layouts of larger graphs can be computed efficiently using multi-scale methods [KCH03].

Appliance The spectral layout algorithm in visone leaves the choice of the value of α to the user to allow application-specific optimization of the drawing. Furthermore, good values are not easily computed from structural properties of the graph. For unexperienced users, a feasible value of $\alpha = 0.6$ is preset and the common settings $\alpha = 0.0$ and $\alpha = 1.0$ can be selected by their familiar names 'Laplacian' and 'adjacency' matrix respectively.

Chapter 6

Analytic Visualization

In network analysis, drawings of networks are often used in combination with an explicit analysis result, e. g., a centrality score. Therefore, an adequate visualization should highlight the graph structure and the result of the analysis measure simultaneously. In such a drawing, a graphical feature like the position of a vertex or the width of an edge should depict the measure under consideration as exact as a table. In contrast to a general graph layout, we call such drawing methods *visualizations*.

Before the first ideas for visone were described [BKR+99], there were no appropriate layout algorithms which could take such scores into account. Pajek [BM98] provided the option to map vertex scores to layout dimensions but had no dedicated layout algorithm to produce readable drawings given such constraints. Alternative visualizations, available also in other tools, allow the mapping of scores to vertex sizes and numerical labels or separated from the drawing in an additional table.

According to Cleveland and McGill [CM84], positions are the most accurate means for visually representing numbers. Among the most notable features of visone are therefore the dedicated visualization methods based on graph layouts – the radial and layered visualization for numerical vertex weights and the multi-circular visualization for clustering information. These are accompanied by visualizations for arbitrary weights as basic graphic properties like sizes and colors.

The explanatory power of the presented advanced visualization methods stems to a large extent from the usage of intuitive geometric counterparts of analytical concepts like centrality, status, and grouping.

Type of Input Parameter The visualizations in visone can be categorized according to the type of input parameter they can depict, either numerical or categorical. Visualizations of numerical parameters exploit the linear order of the elements and typically feature a representation proportional to the given weights. Clearly, numerical parameters comprise exactly the integer and decimal attributes, i. e., vertex and edge weights $\omega : G \to [\mathbb{R}|\mathbb{Q}]$. In contrast, categorical visualizations only rely on differences of values and not on an order but can be applied on any type of attribute.

The numerical visualizations of visone are vertex size, edge width, (linear) vertex and edge color, coordinate in z-position, radial visualization, and parallel visualization. Additionally, there are categorical visualizations as vertex and edge labels, as (categorical) vertex and edge colors, for clusters in a multi-circular layout and for general groups.

6.1 Basic Visualizations

Basic graphical features like the color and size of the elements are very important for the overall appearance of network drawings. Visualizations based on these features thus offer great potential for creating nice drawings. Furthermore, these visualizations are very powerful when used in combination with sophisticated positional visualizations like the status visualization to display the results of different analyses simultaneously in one drawing.

6.1.1 Sizes

One of the most commonly used visualization methods is most likely the scaling of the size of elements proportionally to a given weight. In visone, there are methods to scale the width, the height, and the area of vertices, the edge width, and the font size of vertex and edge labels linearly proportional to a given weight (see Figure 6.1). Typically, such a linear scaling is defined by the sizes of the two extremal weights. For convenience, we have chosen a different approach which goes without any parameter and instead maintains the total of the sizes under consideration. This keeps the appearance of the drawing and users do not need to specify detailed parameters.

In more detail, let $\omega : G \to \mathbb{R}_{\geq 0}$ be a non-negative element weight to be visualized as element size $s : G \to \mathbb{R}_{\geq 0}$. Then, the new size of an element $x \in G$ is defined as

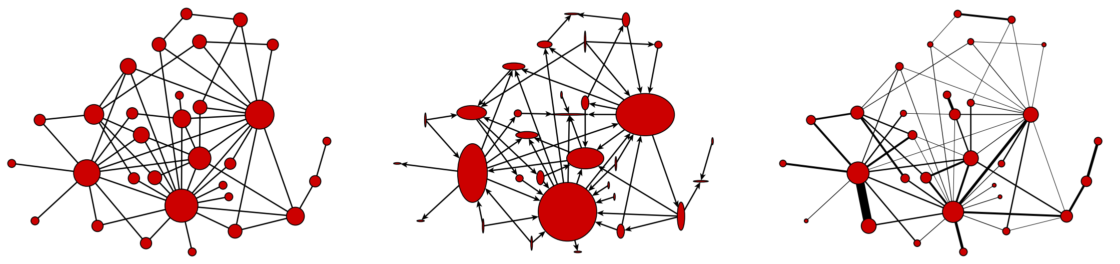
$$s(x) := \frac{\omega(x)}{\sum_{z \in G} \omega(z)} \cdot \sum_{z \in G} s(z) \ ,$$

if this is larger than a preset minimum element size $s_{\min}$. Otherwise, the new size is set to $s(x) = 0.75 s_{\min}$, keeping the element visible and depicting its small value at the same time. Table 6.1 gives an overview of the preset minimum sizes for the different visualizations. Note that application of the minimum size can change the total size but this effect is typically negligible.

At first sight, the visualization as area of the vertices may seem to be a redundant combination of the respective visualizations as width and height. In fact, the area is linear to the weight only for the direct method and quadratic for the combination of width and height. Furthermore, the direct method keeps the aspect ratio of the vertices constant.

graphical feature	minimum $s_{\min}$
vertex width	5
vertex height	5
vertex area	25
edge width	1
label font size	10

Table 6.1: Preset minima of the different size visualizations.



(a) degree as area of vertices (b) in- and out-degree as width and height of vertices (c) external weights as width of edges

Figure 6.1: Visualizations of degree centrality as vertex sizes and of an external edge weight as edge width.

6.1.2 Colors

The problem of choosing colors for data visualization is expressed by this quote from information visualization guru Edward Tufte: "... avoiding catastrophe becomes the first principle in bringing color to information: Above all, do no harm." [Tuf90] Color used well can enhance and clarify a presentation while color used poorly will obscure, muddle, and confuse.

visone assists the user in the selection of good colors and offers three methods to map dissimilar colors to elements of different weight: the categorical coloring, the unichrome color gradient, and the dichromatic color gradient. All these methods can be used likewise to color the area of vertices and the lines of edges, their borders, and their labels. Note that the objective of this visualization is different from the classic graph coloring problem in algorithmic graph theory (e. g., see [BM76]). Given a graph G, the traditional version of this problem is to color the vertices of G with as few colors as possible so that adjacent vertices always have different colors.

Categorical Coloring For the categorical coloring, elements of different weight should be colored to be as dissimilar as possible. Choosing such a coloring adequately is a well-known problem of information visualization [Sto03, Tuf90, Tuf07] for which various sophisticated methods have been proposed (e. g., see [Hea96]). Many of these methods concentrate on the selection of colors which can be rapidly distinguished

Red	Green
Yellow	Blue
Black	White
Pink	Cyan
Grey	Orange
Brown	Purple

Dark Red 🟥	Dark Green 🟩
Dark Blue 🟦	Orange 🟧
Light Red 🟥	Light Green 🟩
Light Blue 🟦	Yellow 🟫
Brown 🟨	Purple 🟪
Pink 🟪	Medium Grey ⬜
Light Grey ⬜	Dark Grey ⬛

Table 6.2: From the twelve basic colors proposed by Ware [War07] (left), we derived 14 colors used in visone (right).

and neglect a visual pleasing appeal. Furthermore, experiments by Healey [Hea96] showed that only up to ten different colors can be perceived rapidly. Therefore, visone comes with a preset palette of 14 handpicked colors derived from the set of twelve basic colors proposed by Ware [War07]. Table 6.2 lists these preset colors and Figure 6.2(a) gives an example coloring. In the rare cases when even more colors are needed, we create HSV colors with random hue and maximum saturation and value. The HSV color space is described in the next paragraph.

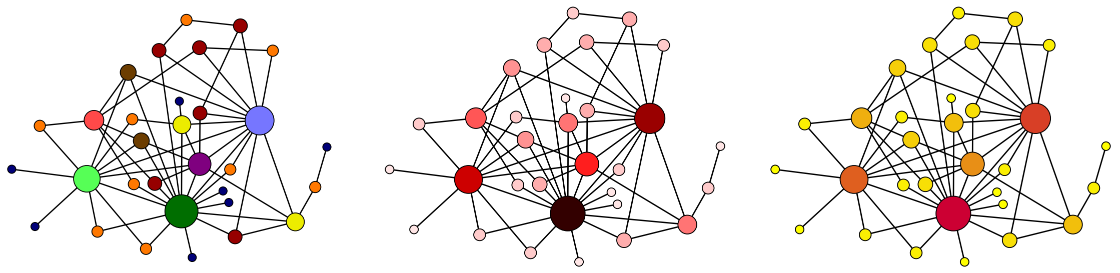
Unichrome Color Gradient The unichrome color gradient uses one of the most basic properties of perception – the lightness – to represent numerical weights. Generally, darker colors are used to represent higher values: light-to-dark for low-to-high. Similar concepts are used for example for the display of numerical data in maps [Bre99]. For a given color, its representation as HSV color is used to compute a gradient from one of its light shapes to a dark one. HSV stands for hue, saturation, value and is a representation of an RGB color space which describes perceptual color relationships more accurately than pure RGB while remaining computationally simple. Hue, saturation, and value typically range between 0 and 1.

In more detail, let $\omega : G \to \mathbb{R}_{\geq 0}$ be a non-negative element weight and let (H, S, V) be a given color in HSV representation. Furthermore, we normalize the weights to the unit interval by $\omega'(x) := \big(\omega(x) - \min_z \omega(z)\big)/\big(\max_z \omega(z) - \min_z \omega(z)\big)$. Then the new color $c(x) = (h, s, v)$ of an element $x \in G$ is defined by

$$c(x) = (h, s, v) := \begin{cases} \big(H,\ 2\omega'(x) \cdot (1 - s_{\min}) + s_{\min},\ 1\big) & \text{if } \omega'(x) < 0.5\ , \\ \big(H,\ 1,\ 2(1 - \omega'(x)) \cdot (1 - v_{\min}) + v_{\min}\big) & \text{otherwise,} \end{cases}$$

where $s_{\min}$ and $v_{\min}$ are preset minimum saturation and value, respectively. Intuitively speaking, the value of hue H is kept constant and saturation and value are linearly scaled from $(H, s_{\min}, 1)$ via $(H, 1, 1)$ to $(H, 1, v_{\min})$ (see Figure 6.2(b)). The boundaries $s_{\min}$ and $v_{\min}$ are introduced, since colors of low saturation and low value are very similar to white and black, respectively, and hard to distinguish. They are preset to $s_{\min} = 0.15$ and $v_{\min} = 0.2$.

For the selection of the defining color of the color gradient, visone offers a graphical color selection item so that users are not troubled by color space considerations.



(a) discrete coloring (b) unichrome color gradient for red (c) dichromatic color gradient for yellow and red

Figure 6.2: Color visualizations of the degree of the vertices. For clarification, the degree is also depicted as size of the vertices.

Dichromatic Color Gradient The dichromatic color gradient uses a linear transition between two given colors. Let $\omega : G \to \mathbb{R}_{\geq 0}$ be a non-negative element weight, let $\omega'(x) := \big(\omega(x) - \min_z \omega(z)\big)/\big(\max_z \omega(z) - \min_z \omega(z)\big)$ be its normalization into the unit interval and let (R_1, G_1, B_1) and (R_2, G_2, B_2) be two given colors in RGB representation. Then the new color $c(x) = (r, g, b)$ of an element $x \in G$ is defined by

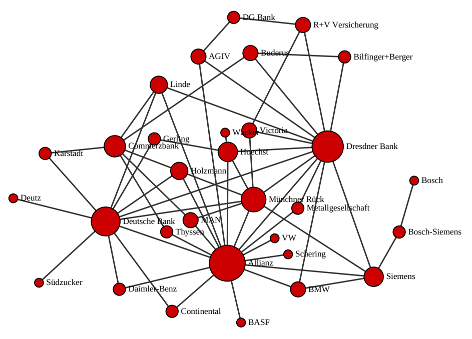
$$\begin{aligned} c(x) = (r,g,b) := \big(&R_1 + \omega'(x)(R_2 - R_1), \\ &G_1 + \omega'(x)(G_2 - G_1), \\ &B_1 + \omega'(x)(B_2 - B_1)\big) \ . \end{aligned}$$

Intuitively speaking, each component of the color is scaled proportionally to the weight between the first and second reference color (see Figure 6.2(c)).

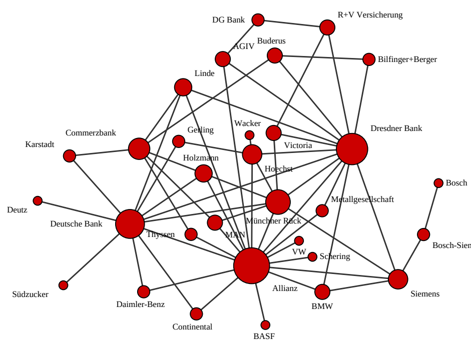
Appliance All three color visualizations are controlled by the same operation panel in visone. This panel offers two graphical color selection items. Depending on how many colors are selected by the user, the categorical, unichrome, or dichromatic variant is executed.

Note that all color visualizations fit their available range of colors to the actual range of weights. In particular, the unichrome variant assigns the lightest color to the smallest existing weight and not to an (imaginary) weight of zero. This is for example in contrast to the size visualization but common practice in information visualization.

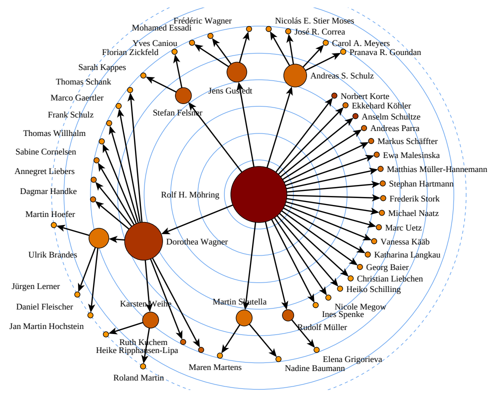
In rare cases, it can be useful to apply the methods based on color gradients to categorically weighted elements. visone allows this by creating a virtual numerical weight defined by equidistantly spacing the lexicographically ordered categorical weights.



(a) uniform label placement



(b) optimized label placement



(c) optimized label placement

Figure 6.3: Labels are powerful means of visualization but without an optimized placement even good layouts appear cluttered. The automatic label placement largely improves readability and overall appearance.

6.1.3 Labels

The label visualization displays weights as textual labels of the vertices or edges. Unlike all other visualizations, this is a permanent mapping of the weight to the label, i. e., a label is immediately updated if the value of the weight changes. Furthermore, there is no alternative method to change the text of labels. Together this prevents the accidental display of incorrect or old values of a weight.

Textual labels are a powerful way to display textual and exact numerical information but introduce an additional graphical element to the drawing and must be carefully placed to not occlude the vertices and edges. The task of manual label placement is very tedious. Therefore, visone offers a powerful automatic label placement algorithm which disburdens users from this task and significantly improves the readability of the drawing. This algorithm can be customized in various ways: the scope can be restricted to the labels of vertices, edges, or selected elements, overlaps of the labels and vertices or edges can be forbidden, and the preferred optimization criterion can be specified.

6.1.4 z-Coordinate

A graphical aspect of layouts which is often disregarded is the z-coordinate of the elements, i. e., which elements lies one top of the other if they overlap. This is especially important if the size of the vertices is skewed so that smaller vertices are potentially entirely covered by larger ones. Let $\omega : V \to \mathbb{R}$ be a numerical vertex weight. The visualization assigns to each vertex v a z-coordinate $z(v)$ such that

$$\forall u \in V : z(u) \leq z(v) \iff \omega(u) \leq \omega(v) ,$$

i. e., the vertex v is drawn above all vertices of lower weight and below all vertices of larger weight. Vertices of the same weight are ordered arbitrarily.

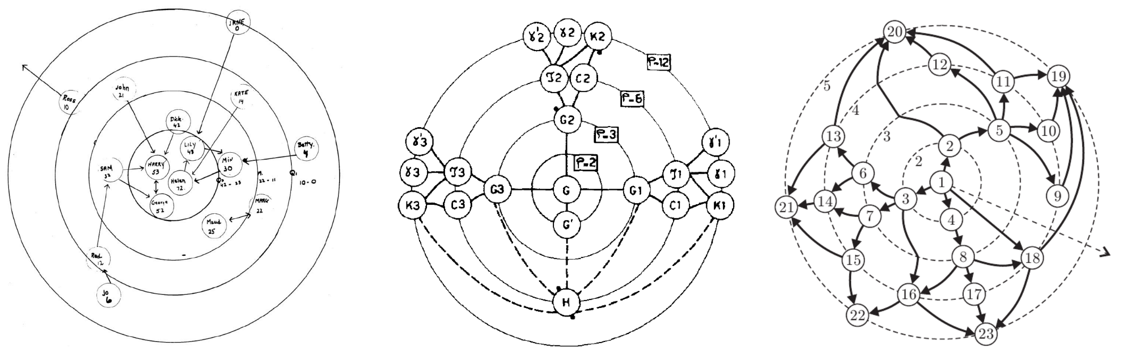
Edges are handled separately and are independent of the z-coordinates of the vertices. They are drawn below all vertices by default, and above all vertices otherwise.

6.2 Radial Visualization

The original inspiration for this type of visualization was the notation of centrality which provides an immediate model for graphical representation – geometric centrality. We therefore place vertices such that their distance from the center of the drawing is proportional to their centrality score, i. e., they are positioned at a fixed radius. While this type of visualization is clearly suited for centrality scores [BKR06, BKR$^+$99, BKW03], it can be used for any non-negative vertex weight equally well.

A historic precursor of this idea is the *target diagram* of Northway from 1940 [Nor40] which we discovered after the original model had been set up. She placed actors inside of rings corresponding to centrality quantiles (see Figure 6.4(a)) and arranged them manually. Some general graph layout methods like the force-directed variants sometimes happen to place vertices of high score close to the center of the drawing but this is a rough approximation of the actual scores at best. Typically, they are more often misleading than not.

The design described so far constrains vertices to lie on radii proportional to the given weight. We still have to determine the actual position of the vertices which should be used to achieve a good readability of the drawing, i. e., a uniform distribution of vertices and few edge crossings. An energy-based placement was used for this purpose in a previous version of visone [BKW03] which yielded appealing layouts but was too slow for an interactive tool. Therefore, we switched to a purely combinatorial approach based on a two-phase crossing reduction algorithm for a derived logical circular layout.



(a) target diagram (b) ring diagram (c) radial level layout

Figure 6.4: Drawings of networks similar to the radial visualization: a target diagram of Northway [Nor40], a ring diagram of Reggiani and Marchetti [RM88], and a radial level layout of Bachmaier [Bac07].

6.2.1 Layout Algorithm

For layout optimization, we assume all vertices to be constrained to distinct positions along the perimeter of a single circle. This allows us to use combinatorial methods for the reduction of the number of edge crossings, since in such circular layouts their number depends only on the order and not the actual coordinates of the vertices. Studies indicate that the number of crossings significantly affects the readability of a drawing [Pur98]. Note that typically, the values of scores are not uniformly distributed but dominated by a large number of small values. Therefore, many vertices will lay on the outer parts of the radial visualization and the circular model is legitimate.

Since crossing minimization is an $\mathcal{NP}$-hard problem even in such rather simple circular layouts [MKNF87], we propose a two-phase heuristic for obtaining circular layouts with few crossings. In the first phase, vertices are iteratively added to either end of a linear order. For the second phase, we adopt an optimization procedure presented originally for layered layouts, sifting [MSM00], to the circular case. Roughly speaking, sifting picks one vertex at a time and determines the locally optimal position within the order by probing all of them. Our algorithm is described in detail in Section 7.1. Since it allows for weighted crossing reduction, an adequate handling of visual or semantically important edges can be enforced by an appropriate edge strength, i. e., high weights are assigned to important edges to discourage crossings of them.

Closely related types of graphical presentations are *ring diagrams* introduced by Reggiani and Marchetti [RM88] and *radial level layouts* presented by Bachmaier [Bac07] (see Figure 6.4). In both cases, vertices are constrained to circumferences of circles which correspond to levels in a hierarchy and vertices are placed iteratively one level at a time. Unfortunately, the first method is designed only for graphs in which the edges do not span layers. In contrast, Bachmaier proposes a complete drawing framework similar to Sugiyama's method [STT81] and the layout algorithm of Brandes and Köpf [BK02] for (linear) layered layouts. In principle, radial level layouts can be used as a basis for radial visualizations using methods similar to the ones proposed for the status visualization in Section 6.3. Unfortunately, we had no time to implement and test this.

6.2.2 Final Placement

The position of each vertex $v \in V$ in a radial visualization can be described in polar coordinates $p(v) = (r(v), \varphi(v))$, where $r(v)$ is the radius and $\varphi(v)$ is the angle with respect to a fixed direction from the center. We set the radii by allocating the vertices at equidistant angles of $2\pi/n$ in the order given by the layout algorithm. The radius $r(v)$ of a vertex $v \in V$ with $\omega(v) > 0$ is fixed at

$$r(v) := \frac{\overline{\omega} - \omega(v)}{\overline{\omega} - \underline{\omega}} ,$$

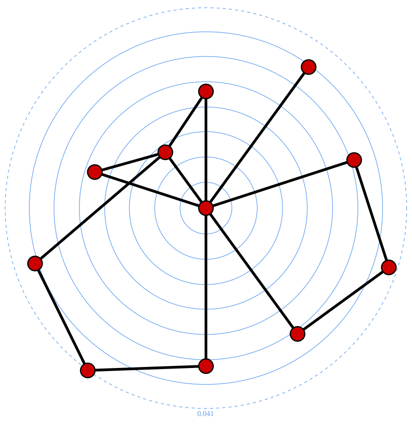
where $\overline{\omega}$ and $\underline{\omega}$ are the maximum and minimum non-zero weights, respectively. If the two highest weights differ only marginally, the radii of the vertices of maximum weight are increased by a fixed offset and the range of radii is reduced accordingly to avoid vertex overlaps in the center. Vertices of weight zero are placed at an outer orbit.

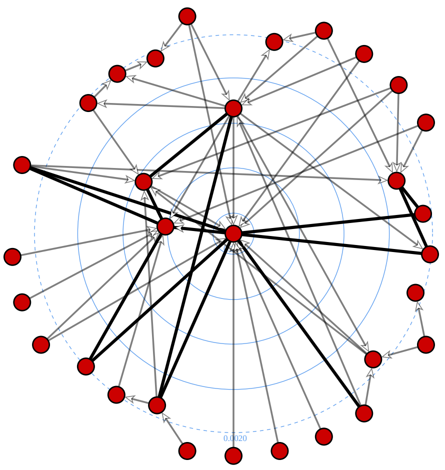
6.2.3 Confirmation

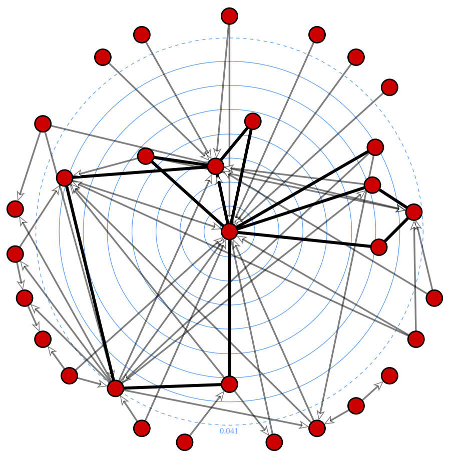
One of the unique features of visone is the differentiation between confirmed and unconfirmed edges. This distinction is implemented not only in the analysis measures but also immanent to each drawing since confirmed edges are rendered more conspicuously in opaque colors than unconfirmed edges which are drawn semi-transparent. This visual dominance renders uniform treatment of all edges inappropriate. Therefore, we first establish a core layout of the confirmed subgraph, i. e., the subgraph $G[E_c] = (V_c, E_c)$ induced by the confirmed edges E_c. Then, we insert the unconfirmed edges while maintaining the core layout.

In more detail, in the first step, we apply both phases of our layout algorithm to $G[E_c]$ to compute an order of its vertices V_c. In the second step, we first use *insertion sifting* to complete the layout order by all other vertices $v \in V \setminus V_c$, i. e., vertices are incrementally added at one end of the order and sifted to their currently optimal position (see Section 7.1.5). At each iteration, a vertex with the largest number of already placed neighbors is selected, where ties are broken in favor of vertices with fewer unplaced neighbors, i. e., the processing sequence equals the one of the greedy phase (see Section 7.1.3). Then, we apply sifting to improve this order where vertices $v \in V_c$ are skipped to maintain the core layout and a higher weight is assumed for confirmed edges $e \in E_c$ to discourage crossings with these edges.

Figure 6.5 illustrates the importance of a good layout of the confirmed subgraph. Note that this may be achieved at the expense of other parts of the final layout.


(a) confirmed subgraph


(b) uniform treatment of all edges


(c) preferential treatment of confirmed edges

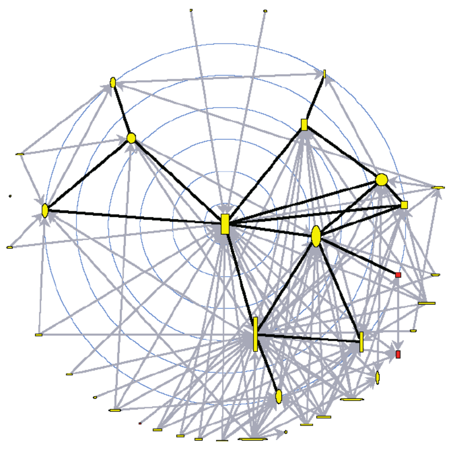

(d) preferential treatment in the previous layout algorithm

Figure 6.5: Layout of the core subgraph and effect of preferential and non-preferential treatment of confirmed edges. For comparison, the preferential treatment in the previous layout algorithm is also shown (for a different graph).

weight range δ	distance of levels	typical levels
$0.1 \le \delta < 0.2$	0.02	$0.02, 0.04, 0.06, 0.08, \dots$
$0.2 \le \delta < 0.5$	0.05	$0.05, 0.1, 0.15, 0.2, \dots$
$0.5 \le \delta < 1.0$	0.1	$0.1, 0.2, 0.3, 0.4, \dots$

Table 6.3: Levels of the scale of the radial and status visualizations for a range $\delta = \overline{\omega} - \underline{\omega}$ of the weight ω of $0.1 \le \delta < 1.0$. For other ranges, the levels are scaled by appropriate powers of ten.

Similar to the computation of analysis measures, the user can indicate that certain unconfirmed edges should be considered as confirmed simply by selecting them.

6.2.4 Scale

Showing levels as thin circles in the background of the drawing allows us to compare scores exactly so that no tabular presentation is needed. If the number of levels is too small, metering is inexact, and if the number is too large, individual levels are hard to follow. Therefore, we aim for five to ten equidistant levels marking prominent values and choose the distance between levels accordingly (see Table 6.3). Furthermore, we mark the minimum non-zero weight by a dashed circle to separate the outer orbit. If a fixed offset for the radii is used because the two highest weights differ only marginally, this offset is also marked by a dashed circle.

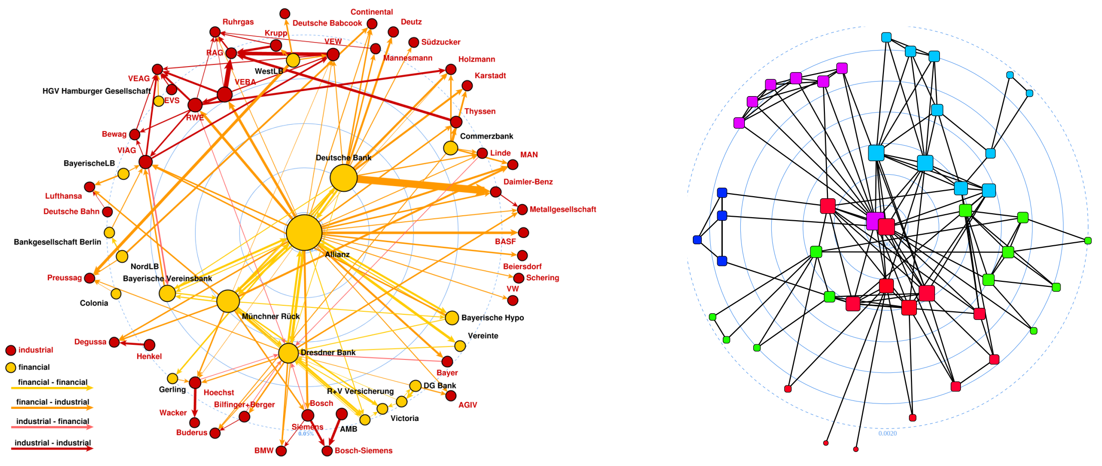
6.2.5 User Interaction

Since we decoupled the centrality constraints from the readability criteria, we are able to provide the following forms of interaction which proved particularly useful for explanatory centrality analysis.

Snap to Orbits All vertices are moved to have a distance from the center of the drawing that is determined by the selected weight. This movement is along a ray emanating from the center and passing through the current position of the vertex. If a vertex is currently in the center, a randomly oriented ray is chosen.

Improve Distribution Prior to the movement of ‘snap to orbits’, the vertices are positioned at equidistant angles, preserving the current order around the center of the drawing.

Improve Layout Instead of computing a layout from scratch, the first phase of the layout algorithm is skipped and the second phase is initialized by the current order around the center of the drawing. Subsequently, the vertices are positioned at equidistant angles and moved to their orbits as described in ‘improve distribution’. This option is particularly useful if the optimization algorithm became stuck in a



(a) German company network

(b) extract of an email communication network

Figure 6.6: Fully featured examples of the radial visualization.

local minimum which the user can improve, for example by exchanging more than two vertices.

Complete Layout The complete layout algorithm as described above, including both optimization phases, is carried out.

6.2.6 Final Remarks

The exploratory and explanatory power of the radial visualization is exemplified in a study by Brandes, Kenis, and Raab [BKR06]. Since their study we were able to further improve the quality and the running time of the radial visualization due to the novel layout algorithm. The most obvious drawback of the previous method was the tendency to accumulate non-core vertices in one sector of the drawing (see Figure 6.5(d)).

Figure 6.6 gives two examples of radial visualizations. The left drawing shows a radial visualization of status centrality of the German company network. Additionally, other characteristics of the network are depicted by basic visualizations in order to further increase the information density. This drawing is described in more detail in Section 6.6.

The drawing on the right shows a subgraph of small institutes of the email communication network described in Section 6.6. Here, a radial visualization of current flow centrality is enhanced by visualizations of institute affiliation as colors and number of connections as vertex size.

6.3 Status Visualization

Organizational charts are well-known, widely spread visualizations of interconnections in hierarchical networks, e. g., to display the chain of command between the departments of a public authority or a company. All these drawings take advantage of the everyday notion of 'higher' and 'lower' status by mapping status to vertical positions (see Figure 6.7(a)).

Already some early network visualizations employed the same notion of status as vertical positions. In his famous book 'Street Corner Society' [Why43], Whyte arranged actors vertically so that positions indicate relative actor status (see Figure 6.7(b)). It should be noted, however, that the network does not form the basis for this hierarchy, since status is determined by other factors and that levels are discrete like in organizational charts. Northway used horizontal stripes to indicate the quartile in which her measure, the sociometric choice status of an actor, lies. However, she only shows the ego-network of a particular actor, probably because the arrangement of an entire network in this fashion was too cumbersome.

We adopt this intuitive graphical representation of status for the visualization of any non-negative vertex weight $\omega : V \to \mathbb{R}_{\geq 0}$ as described in Brandes et al. [BRW01]. In this status visualization, vertices are placed at vertical positions that exactly represent their score and horizontal positions are determined algorithmically in such a way that the overall visualization is readable.

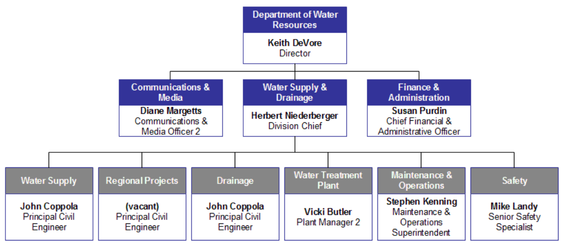
The drawings in Figure 6.7 show that the suggestive power of a status sociogram largely depends on the placement of the vertices on distinctive levels which can be further backed by an appropriate edge routing. The drawing of Northway lacks both features and constitutes the impression of status solely by drawing scale levels. Since a typical score imposes a continuous placement of the vertices, we utilize a conspicuous edge routing backed by the drawing of scale levels to clarify the meaning of the drawing.

The description of the visualization in the remainder of this part adds some additional features to the original method presented in [BRW01]. Since a combinatorial layout model similar to the radial visualization is used, we can adopt the methods for interaction and reciprocation and employ an improved crossing reduction algorithm.

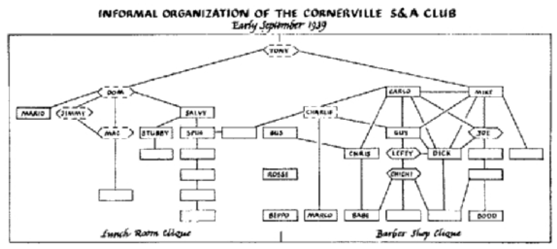
6.3.1 Layout Algorithm

The most commonly used framework for linear layered drawings of graphs is presented in Sugiyama et al. [STT81]. It consists of the following generic steps:

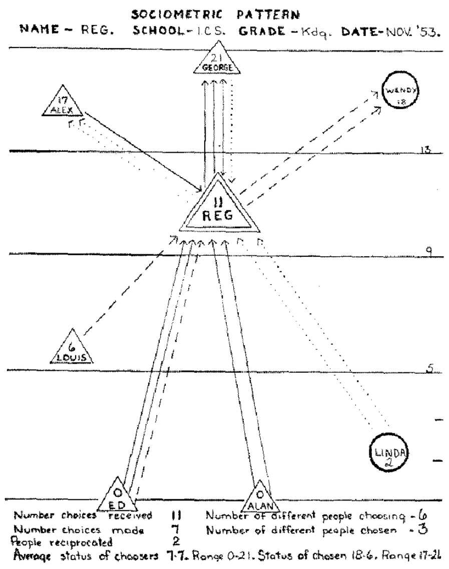
1. assign each vertex to a layer, then
2. subdivide each edge by dummy vertices at each layer it crosses and determine the order of the vertices for each layer, and finally
3. assign horizontal and vertical coordinates to each vertex and replace dummy vertices by bend points.



(a) organizational chart



(b) status ordered network


SOCIOMETRIC PATTERN
NAME – REG. SCHOOL – I.C.S. GRADE – Kdg. DATE – NOV. '53.
21 GEORGE
17 ALEX
WENDY 18
13
11 REG
9
6 LOUIS
5
LINDA 2
0 ED
0 ALAN
Number choices received 11
Number choices made 7
People reciprocated 2
Number of different people choosing – 6
Number of different people chosen – 3
Average status of choosers 7·7. Range 0-21. Status of chosen 18·6. Range 17-21

(c) sociometric choice status

Figure 6.7: Drawings of networks similar to the status visualization: a typical organizational chart, status ordered network by Whyte [Why43], and a sociometric choice status of Northway [Nor54].

The separation of steps 2 and 3 enables the use of combinatorial methods for the optimization of the layout in the second step. Since edge crossings significantly affect the readability of a drawing [Pur98], our objective is to reduce their number. Through the introduced dummy vertices, the number of crossings between two adjacent layers is fully determined by the relative order of the vertices, independent of the actual coordinates.

Layer Assignment A trivial layer assignment is to partition the vertices into sets of equal weight and to place each set in its own layer, vertically ordered with respect to the score. Analysis scores often differ only marginally leading to a large number of close layers that can increase the number of dummy vertices significantly. Furthermore, since these dummy vertices are replaced by bend points in the final layout, this results in perceptual problems like several crossing or non-crossing line segments running almost horizontally.

Instead, we cluster the vertices according to their weights and assign all vertices in the same cluster to the same logical layer. In principle, any clustering scheme can be used. To achieve a good spacing between layers, we establish a fixed number of clusters, e. g., ten or less, of equal range.

Crossing Reduction In this step, we are given a layering of the vertices and introduce additional dummy vertices where edges need to cross a layer. Our goal is to find horizontal orders of all the vertices in each layer such that the number of crossing edges is small.

Finding orders that minimize the number of crossings is an $\mathcal{NP}$-hard problem [GJ83], even if there are only two layers and one of them is fixed [EW94]. Commonly, the global problem is split into two-layer problems by the layer-by-layer sweep. Initially, the order in the first layer is fixed and the second layer is reordered. Then, iteratively, the layer reordered last is fixed and the order of the next layer is improved. After reaching the last layer, the process is reversed and repeated up and down the layering until no further improvement is made.

There are a couple of heuristics for a simplified two-layer problem without intra-layer edges, i. e., no edge connects vertices of the same layer (for an overview, see, e. g., [ML03]). Furthermore, this problem can be solved optimally for medium-size instances using computationally involved methods [JM97, ZB07]. Since in general we cannot avoid intra-layer edges and the overall number of crossings will not be minimum anyway, we use simpler heuristics. In order to obtain the initial order, each vertex is placed at the average position of its neighbors in the next layer, similar to the barycenter heuristic [STT81]. Then, the modification of our circular sifting algorithm for linearly layered layouts is applied to further reduce the number of remaining crossings (see Section 7.1.5). Roughly speaking, sifting picks one vertex at a time and determines the locally optimal position within a layer by probing all of them. In combination, these heuristics perform quite satisfactory, and our experiences suggest that the additional effort caused by sifting is indeed worth it. Again, the algorithm supports adequate handling of edge strengths by weighted crossing reduction.

Horizontal Placement Given a layering and an order of vertices and bend points within each layer, it remains to compute actual x-coordinates respecting the horizontal orders. Pleasing visualizations are obtained by ensuring that long lines run vertically as much as possible, and by reducing the horizontal distance spanned when it is not. We are using the linear-time algorithm of Brandes and Köpf [BK02] for this task.

6.3.2 Final Positions

The x-coordinates of the vertices are given by the horizontal placement of the layout algorithm. The y-coordinate of each vertex $v \in V$ of non-zero weight is defined as

$$y(v) := \frac{\omega(v) - \underline{\omega}}{\overline{\omega} - \underline{\omega}} \ ,$$

where $\overline{\omega}$ and $\underline{\omega}$ are the maximum and minimum non-zero weights, respectively. Vertices of weight zero are placed beneath at an outer level.

6.3.3 Confirmation

Since the visual dominance of confirmed edges over unconfirmed ones renders uniform treatment of all edges inappropriate, we first establish a core layout of the confirmed subgraph, i. e., the subgraph $G[E_c] = (V_c, E_c)$ induced by the confirmed edges E_c and then insert the unconfirmed edges while maintaining the core layout.

Similar to the radial visualization (see Section 6.2.3), we compute an initial order of the vertices of $G[E_c]$ by applying both phases of our layout algorithm. Then, we insert all remaining vertices into this order by 'insertion sifting' and improve this order by sifting where vertices $v \in V_c$ are skipped to maintain the core layout. Furthermore, high logical weights of confirmed edges $e \in E_c$ impede crossings between them and unconfirmed edges.

6.3.4 Scale

We support metering of the scores by showing levels as thin lines in the background. Similar to the radial visualization we aim for five to ten equidistant levels marking prominent values and choose the distance between levels accordingly (see Table 6.3). Note that these levels typically do not correspond to the clusters used for the layer assignment which can be bounded by arbitrary values. Furthermore, we mark the maximum and the minimum non-zero weight by dashed lines to separate outlying areas.

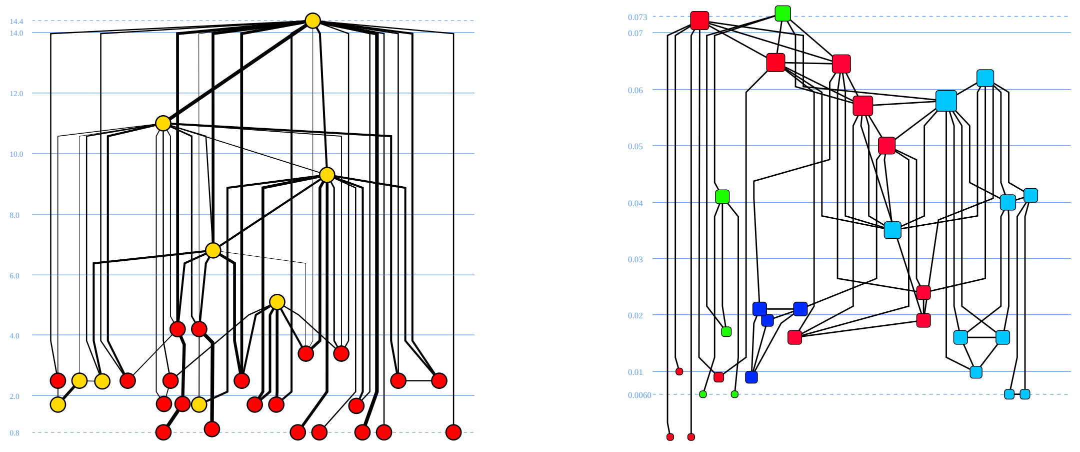
6.3.5 User Interaction

The separation of combinatorial layout and exact placement allows us to provide the following forms of interaction for the status visualization which are similar to the ones provides for radial visualizations.

Snap to Levels All vertices are moved vertically to have a distance from the top of the drawing that is determined by the given weight and an appropriate edge routing is established. To achieve this, the crossing reduction phase of the layout algorithm is skipped and the vertices are ordered in their respective layer according to their current x-coordinates instead.

Improve Layout This option is similar to 'snap to levels' but rather than skipping the crossing reduction phase entirely, the orders given by the current x-coordinates of the vertices are improved by sifting crossing reduction. This allows user to restart the algorithm after manual improvements, e. g., to resolve a bad local minimum.

Complete Layout The complete layout algorithm as described above, including both optimization phases, is carried out.



(a) subgraph of the German company network

(b) subgraph of the email communication network

Figure 6.8: Fully featured examples of the status visualization.

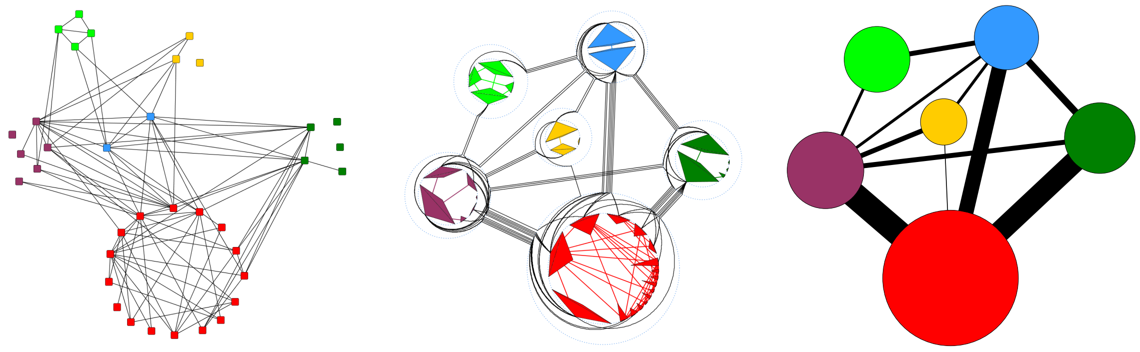
6.3.6 Final Remarks

As an additional feature, the orientation of the status visualization can be changed from horizontal to vertical, i. e., the layout is mirrored at the diagonal and the score of the vertices increases from left to right. The usefulness of the status visualization for the exploration and communication of social networks has been outlined in various studies [BKR06, BRW01, BKR+99]

6.4 Multi-Circular Visualization

An important aspect in the visualization of many types of networks is the interplay between fine- and coarse-grained structures. While the micro-level graph is given, a macro-level graph is induced by a partitioning of the micro-level vertices. For example it may originate from a group-level network analysis such as a clustering (see Sections 4.7, from an attribute-based partitioning of the vertices, or may just be given in advance.

visone includes a tailored visualization for networks with micro/macro structure like a partitioning based on a novel multi-circular drawing convention. Given a layout of the macro-level graph with large nodes and thick edges, each vertex of the micro-level graph is drawn in the area defined by the macro-vertex it belongs to, and each micro-edge is routed through its corresponding macro-edge. In more detail, each micro-vertex is placed on a circle inside of the area of its corresponding macro-vertex and micro-edges whose end vertices belong to the same macro-vertex are drawn inside of these circles. All other micro-edges are then drawn inside of their corresponding macro-edges and at constant but different distances from the border



(a) geometric grouping and straight-line edges

(b) multi-circular layout

(c) corresponding macro graph

Figure 6.9: (a) Example organizational network with geometric grouping and straight-line edges (redrawn from [Kre96]). In our multi-circular layout (b), all details are still present and the macro-structure induced by the grouping becomes visible. Additionally, the height and width of the vertices reflects the number of connections within and between groups.

of the macro-edge, i. e., in straight-line macro-edges they are drawn as parallel lines. These edges must also be routed inside the area of macro-vertices to connect to their endpoints, but are not allowed to cross the circles. Figure 6.9 shows a concrete example of this model. Micro-edges connecting vertices in the same macro-vertex are drawn as straight lines. Inside of macro-vertices, the other edges spiral around the circle of micro-vertices until they reach the area of the macro-edge. For this micro-level layout we define a combinatorial multi-circular embedding and present corresponding layout algorithms based on edge crossing reduction strategies.

We do not impose restrictions on the macro-level layout other than sufficient thickness of edges and vertices, so that the micro-level graph can be placed on top of the macro-level graph, and provide layout algorithms and tailored means of interaction to support the generation of appropriate macro-layouts.

While the drawing convention consists of proven components (geometric grouping is used, e. g., in [Kre96, ST04], and edge routing to indicate coarse-grained structure is proposed in, e. g., [BD07, Hol06], our approach is novel in the way we organize micro-vertices to let the macro-structure dominate the visual impression without cluttering the micro-level details too much. Note also that the setting is very different from layout algorithms operating on structure-induced clusterings (e. g., [AMA07, KW03]), since we cannot make any assumptions on the structure of clusters (they may even consist of isolates). Therefore, we neither want to utilize the clustering for a better layout, nor do we want to display the segregation into dense subregions or small cuts. Our aim is to represent the interplay between a (micro-level) graph and a (most likely extrinsic) grouping of its vertices.

6.4.1 Macro-Level Layout

We do not require a specific layout strategy for the macro-graph as long as its elements are rendered with sufficient thickness to draw the underlying micro-graph on top of them. For example, post-processing can be applied to any given layout [GN99] or methods which consider vertex size (e. g., [HK02, WM96]) and edge thickness (e. g., [DEKW02]) have to be used. Since, depending on the particular application domain and other contexts, different layout methods will be appropriate for the macro-graph, we provide interactive means for its creation.

Given a partition assignment $\phi : V \to \{0, \dots, k-1\}$, the corresponding macro-graph $Q(G, \phi) = (V_Q, E_Q)$, called *quotient graph*, contains a vertex for each partition of G and two vertices $V_i, V_j \in V_Q$ are connected if and only if E contains at least one edge between a vertex in V_i and a vertex in V_j (see Figure 6.9 for an example). visone displays this graph in an additional view along with the micro-graph and propagates changes to the position and thickness of its elements to the micro-layout. This view behaves like any other view offering all kinds of interactive and automatic layout improvement to users. Initially, we set vertex size and edge width in the quotient graph to be proportionally to the number of corresponding micro-level elements and compute a layout by the classical force-directed layout with appropriate preferred edge length and option 'avoid vertex overlaps' enabled (see Section 5.5). For convenience, additional graphical features like colors and shapes are also adopted by the micro-graph.

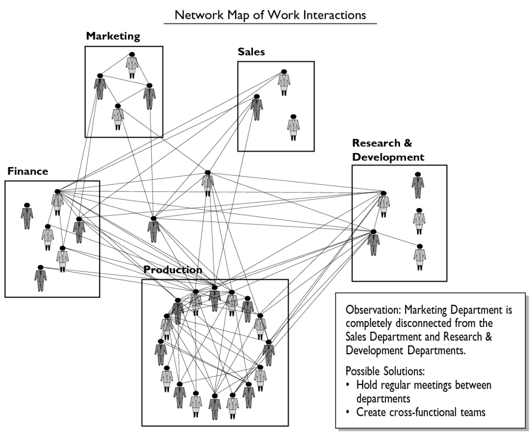
6.4.2 Micro-Level Layout Algorithm

Recall that our multi-circular layout convention requires each vertex of the micro-level graph to be drawn in the area defined by the macro-vertex it belongs to and each micro-edge is routed through its corresponding macro-edge. Through a combinatorial model of this layout convention, we can employ crossing reduction algorithms to improve the drawing [Pur98]. Again, we use a local optimization strategy in combination with a simple initial placement.

Our model is an extension and combination of ideas for (single) circular and radial layer layouts and consists of three parts:

1. circular layouts of the subgraphs inside of each macro-vertex, i. e., a circular vertex order for each partition,
2. the (fixed) circular orders of the incident macro-edges of each macro-vertex given by the macro-layout, and
3. the winding of micro-edges between different partitions around the circular positions of the vertices.

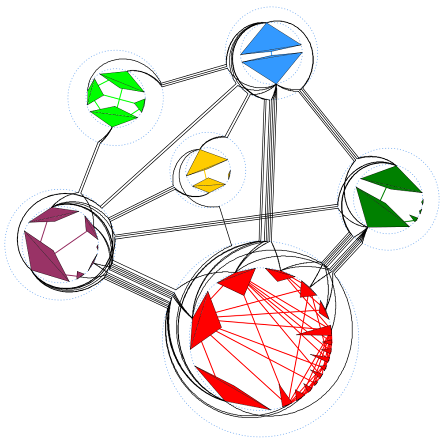
In order to meter windings, we introduce for each partition a ray from its center to infinity and determine how often and in which direction each edge crosses these rays.



Observation: Marketing Department is completely disconnected from the Sales Department and Research & Development Departments.

Possible Solutions:
- Hold regular meetings between departments
- Create cross-functional teams

(a) original drawing



(b) multi-circular

Figure 6.10: Original drawing of an organizational network by Krebs [Kre96] and the corresponding multi-circular visualization. The box reads *Marketing [light green] is completely disconnected from Sales [yellow] and from Research & Development [dark green]*, which is much more apparent in the right layout.

Crossing minimization in multi-circular layouts is an $\mathcal{NP}$-hard problem since it is already hard in (single) circular layouts [MKNF87]. Therefore, we employ an initial barycenter placement for each partition separately. For optimization, we again adapt the sifting procedure [MSM00] which is also used for radial and status visualizations. Note that for multi-circular layouts, exclusive relocation of the vertices is not sufficient since the number of crossings depends also on the winding of the edges. Roughly speaking, multi-circular sifting picks one vertex at a time, probes for every position within the order reasonable windings of its incident edges, and finally sets position and windings of the locally optimal state. Our algorithm which again allows for weighted crossing reduction, is described in detail in Section 7.2.

6.4.3 Final Placement

From the micro-layout algorithm we get a combinatorial description of the layout, i. e., circular vertex orders for each partition and edge windings, which allows arbitrarily rotating each circular order without introducing new crossings (see Section 7.2.4). Therefore, for each partition, we choose a rotation which minimizes the total angular span of the inter-partition edges, reserve space for the drawing of these edges, and place the vertices accordingly at a uniform distance from the border of the macro-vertex.

Another critical factor is the transition of the route of inter-partition edges from arcs to straight-lines at the border of the macro-vertices. If many edges bend exactly at the cut of macro-edge and -vertex, the individual curves are hard to distinguish. Therefore, it might be advisable to use rounded bends but this will occupy additional space.

6.4.4 Final Remarks

Similar to the radial and the status visualization, the combinatorial layout model in combination with the sifting crossing reduction algorithm enables a different handling of confirmed and unconfirmed edges and interactive layout improvement. Due to time constraints, this is not implemented yet. Clearly, multi-circular visualizations can be enriched further when used in combination with basic visualizations.

The redrawing of a network of work interactions from a study of Krebs [Kre96] in Figure 6.10 exemplifies the usefulness of this visualization. One of his findings is that *Marketing is completely disconnected from Sales and from Research & Development*, a conclusion which is not highlighted by his drawing. In fact, the proximity of marketing and sales suggests a connection at first. In the multi-circular layout of the network, the disconnection is apparent and, furthermore, more communication deficits become visible, e. g., sales (the yellow group) is also not connected to Research & Development (dark green). Another, larger example is given in Section 6.6.2.

6.5 Additional Positional Visualizations

visone features two more methods to depict attribute data as positions which we describe in this section. The coordinate visualization is a rather simple assignment of the horizontal or vertical coordinate of the vertices proportional to a given non-negative numerical weight. The group visualization can be used for interactive exploration of an ambiguous grouping.

6.5.1 Coordinate Visualization

The coordinate visualization assigns the horizontal or vertical coordinate of the vertices proportionally to a given non-negative numerical weight $\omega : V \to \mathbb{R}_{\geq 0}$ while keeping the other coordinate unchanged. The new coordinate $c(v)$ of each vertex $v \in V$ of non-zero weight is defined equally to the status visualization as

$$c(v) := \frac{\omega(v) - \underline{\omega}}{\overline{\omega} - \underline{\omega}} ,$$

where $\overline{\omega}$ and $\underline{\omega}$ are the maximum and minimum non-zero weights, respectively, and vertices of weight zero are just as well placed at an outer level, either beneath or left of the others. Furthermore, an appropriate scale is drawn.

When the horizontal and vertical coordinate visualization are executed one after the other, the resulting layout equals a two-dimensional scatter plot. As an additional feature, the scale displays both axes simultaneously in this case.

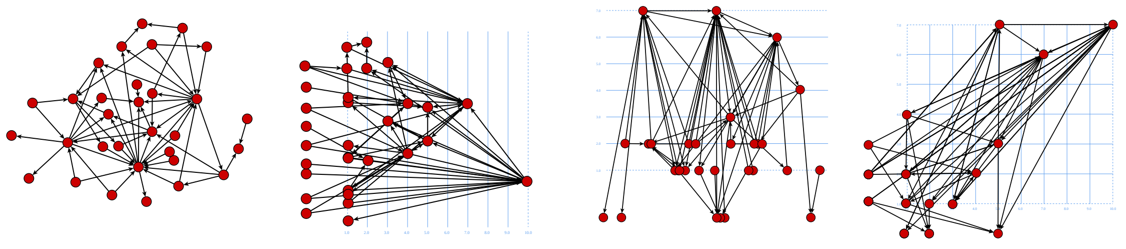

(a) original layout (b) in-degree as x-coordinate (c) out-degree as y-coordinate (d) 'scatter plot'

Figure 6.11: Examples of the coordinate visualization. The right-most figure shows a scatter plot created by combining visualizations in vertical and horizontal direction.

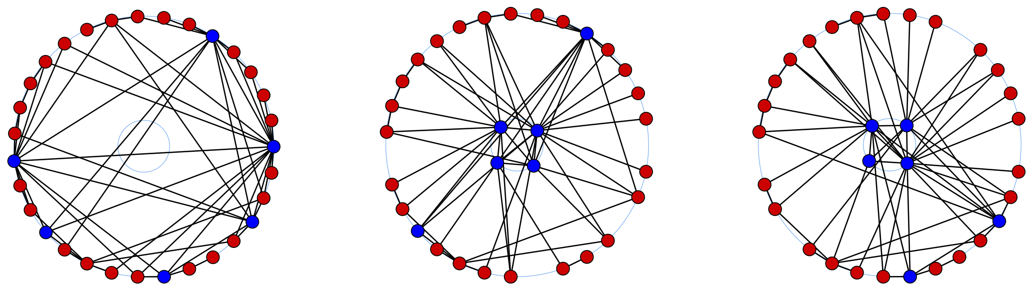

(a) circular layout (b) highlighting a clique (c) highlighting another clique

Figure 6.12: Example of the group visualization. Two different groups, in this case cliques, are highlighted while the fundamental circular layout is preserved. Note that selections are not exported to images and are therefore not visible in the figures.

6.5.2 Group Visualization

visone provides the sophisticated multi-circular visualization to depict a partitioning, i. e., a grouping in which each vertex belongs to a unique group. While many common analysis methods yield a partitioning, e. g., clusterings and structural roles, others do not. Prominent examples of ambiguous grouping are cliques and related structurally dense groups. Since the affiliation of vertices to ambiguous groups can instantiate complex interconnections, we propose an interactive method which maintains the overall layout of the graph and highlights only one group at a time.

The fundamental layout of this visualization is a circular order of all vertices on an outer circle. When a group is chosen by the user, its vertices are placed on the perimeter of an inner concentric circle while the positions of all other vertices are kept fixed on the outer circle (see Figure 6.12). The method described so far can thus be seen as a radial visualization of appropriate scores (see Section 6.2). Its distinctive features are an animated movement of the vertices of the chosen group from the outer to the inner circle and back when another group is chosen, the highlighting of the neighborhood of the group as selection, and an improved placement of the inner vertices. They are relocated to equidistant angles in the order given by the circular layout and exact positions are chosen such that the total length of their incident edges is minimal.

6.6 Examples

In the following, we give some examples for the power and usefulness of our visualizations, especially then used in combination.

6.6.1 The Network of the Deutschland AG

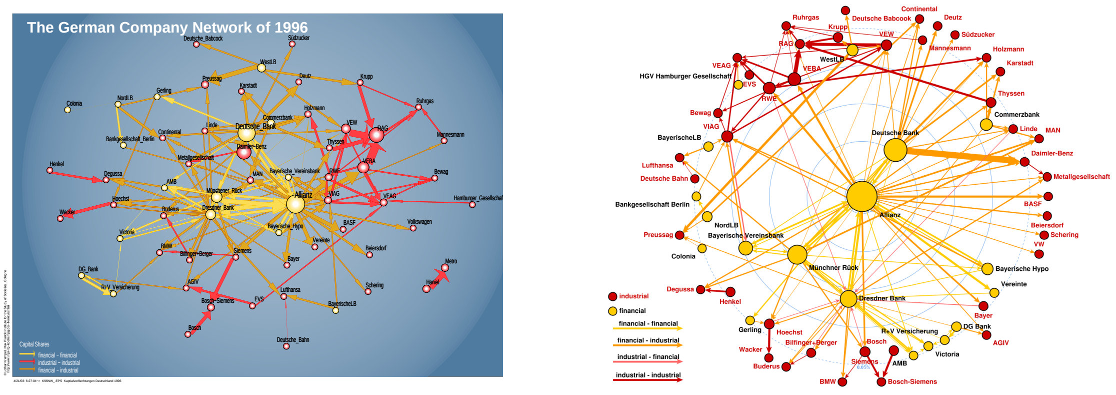
Since the 1990s an obvious and highly discussed erosion of what was previously known as the *Deutschland AG* has taken place. A review of this process presented by Höpner and Krempel in 2003 [HK03, HK04] attracted a lot of attention not only because of its profound theoretical analysis but also because of its demonstrative, colorful and visually pleasing drawings of the network.

The studied network consists of all companies, among the 100 largest German-based companies, which were entangled with others of these companies by capital interlocks, i. e., a source company holds shares of a target company. Besides the edge structure Höpner and Krempel depict additional interesting characteristics of the data by various graphical features:

- The colors of vertices and edges reflect affiliation to different sectors, financial or industrial.
- The width of an edge corresponds to the value of the shares.
- The size of the vertices *"represents degrees of involvement in the network"*, i. e., the sum of in- and out-degree.
- Important vertices are placed in the center of the layout.

While the pictures clearly depict the decline of the connectedness of the network they also exhibit some visual and analytic shortcomings: edges do not start and end in the middle of the vertices, labels hide edges even if there is enough space, the local vertex placement is not good (see the yellow edges of 'GD Bank' and 'R+V Versicherung' in the lower left), and the global layout is misleading since vertices of lesser importance are positioned in the center like the *Big Players*. This indicates that the pictures were created in a time-consuming process *by hand* using a general diagramming software and not in a tailored network visualizing tool.

The basic visualizations available in visone can be used to depict the characteristics described above in a similar fashion as in the original drawing without any manual



(a) original drawing (b) radial visualization

Figure 6.13: Drawings of the German company network as of 1996, the original image by Höpner and Krempel [HK03] and a replication using a combination of the radial and additional basic visualizations.

action. Furthermore, the radial visualization provides an accurate and expressive placement of the vertices.

6.6.2 Email Communication Network

The strength of a multi-circular layout is the coherent drawing of vertices and edges at two levels of detail. It reveals structural properties of the macro-graph and allows identification of micro-level connections at the same time. The showcase for the benefits of our micro/macro layout is an email communication network of a department of the Universität Karlsruhe (TH). The micro-graph consists of 442 anonymized department members and 2 201 edges representing at least one email communication in the considered time frame of five weeks. At the macro-level, a grouping into 16 institutes is given, resulting in 66 macro-edges. In the following drawings, members of the same institute are colored identically.

We start by inspecting drawings generated by a general force-directed approach similar to the method of Fruchterman and Reingold [FR91] and by multidimensional scaling (MDS) [CC01], see Figure 6.14. As described in Section 5.5, both methods tend to place adjacent vertices near each other but ignore the additional grouping information. Therefore, it is not surprising that the drawings do not show a geometric clustering and the macro-structure cannot be identified. Moreover, it is difficult or even impossible to follow edges since they massively overlap each other.

More tailored for the drawing of graphs with additional vertex grouping are the layout used by Krebs [Kre96], and the force-directed attempts to assign vertex positions by Six and Tollis [ST04] and Krempel [Kre05]. All three methods place the vertices of each group on circles inside of separated geometric areas. While some efforts are made to find good vertex positions on the circles, edges are simply drawn as straight lines. Figure 6.15(a) gives a prototypical example of this layout style.

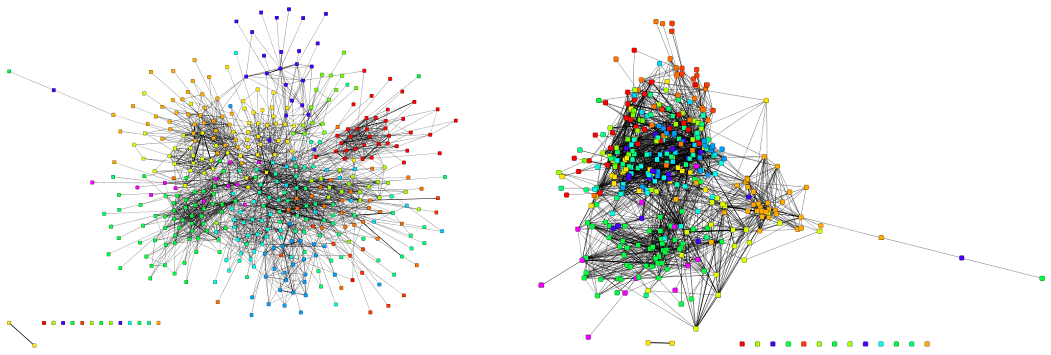
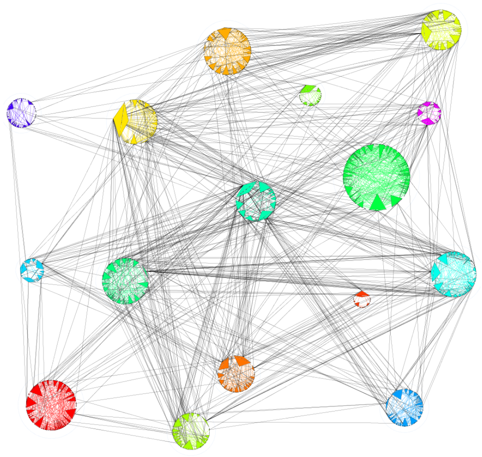


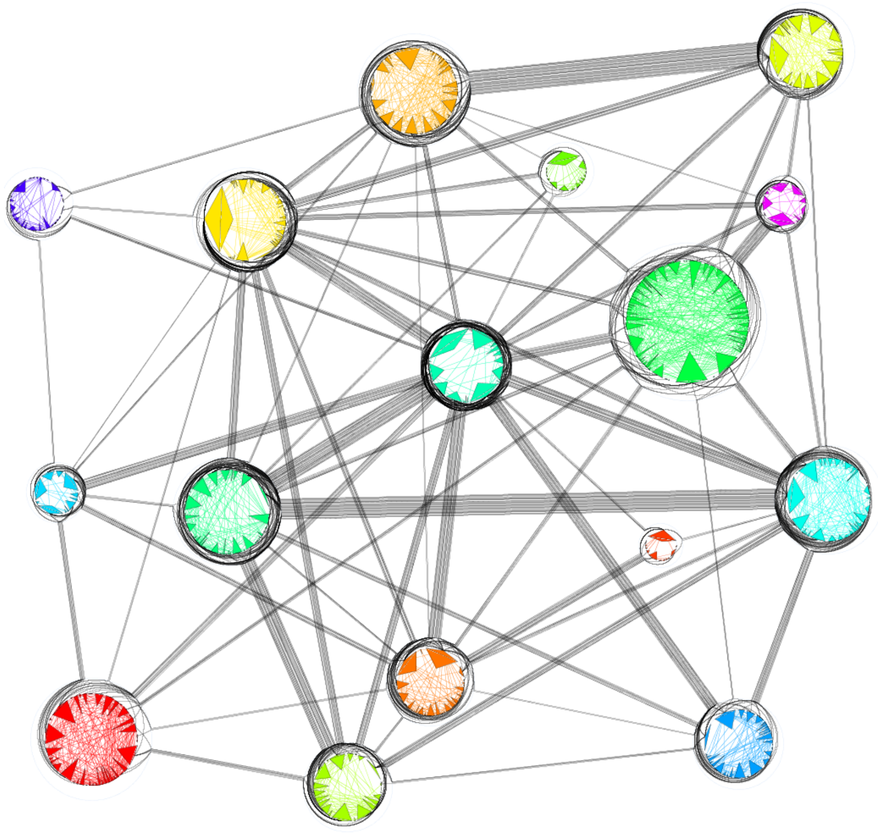
Figure 6.14: Drawings of the email network generated by a force-directed method (left) and by multidimensional scaling (MDS, right). The colors of the vertices depict the affiliation to institutes.

Although these methods feature a substantial progress compared to general layouts and macro-vertices are clearly visible, there is no representation of macro-edges and so the overall macro-structure is still not identifiable.

Finally, we layouted the email network with our multi-circular visualization. Its combinatorial descriptions allows for enrichments with analytical visualizations of the vertices. For this special application, we experimented with visualizations not available in visone by default. In Figure 6.15 the angular width of the circular arc a vertex covers is proportional to its share of the total inter-partition edges of this group. The height from its chord to the center of the circle reflects the fraction of present to possible intra-edges.



(a) straight-line layout



(b) multi-circular layout

Figure 6.15: Straight-line circular and multi-circular layout of the slightly modified email network (some small groups are pruned).

Chapter 7

Layout Algorithms

In this chapter, we give detailed descriptions of the layout algorithms used for the radial and the multi-circular visualization. Furthermore, we present a layout convention for two and a half dimensional visualization which we have not included in the visone software due to its requiring of a three-dimensional layout engine. Instead we exemplify its usefulness in a visualization of the Internet at the level of Autonomous Systems.

7.1 Circular Crossing Reduction

In this section, we present a two-phase heuristic for crossing reduction in circular layouts. While the first algorithm uses a greedy policy to build a good initial layout, an adaptation of the sifting heuristic for crossing reduction in layered layouts is used for local optimization in the second phase. Both phases are conceptually simpler than previous heuristics, and our extensive experimental results indicate that they also yield fewer crossings. An interesting feature is their straightforward generalization to the weighted case. Furthermore, we present various modifications of the second phase which proved useful in practice and allow the handling of intra-layer edges in layered layouts, a case which was excluded so far.

7.1.1 Introduction

In circular graph layouts, the vertices of a graph are constrained to distinct positions along the perimeter of a circle, and an important objective is to minimize the number of edge crossings in such layouts. Since circular crossing minimization is $\mathcal{NP}$-hard [MKNF87], several heuristics have been devised [Mäk88, DMM97, ST99]. Moreover, there is a factor $\mathcal{O}(\log^2 |V|)$ approximation algorithm [SSSV94].

We propose a two-phase approach for obtaining circular layouts with few crossings. In the first phase, vertices are iteratively added to either end of a linear layout.

This leaves three degrees of freedom: the start vertex, the insertion order, and the end at which to append the next vertex. For the different strategies tried, empirical evidence suggests that a particular one outperforms both the others and previous heuristics.

For the second phase, we adapt a local optimization procedure for layered layouts, sifting [MSM00], to the circular case. Note that, similar to two-layer layouts, the number of crossing is completely determined by the (cyclic) order of vertices. The thus related one-sided crossing minimization problem in two-layer drawings of bipartite graphs is $\mathcal{NP}$-hard as well [EW94], but significantly better understood. It turns out that circular sifting reduces the number of crossings both with respect to our first phase and previous heuristics.

After defining some terminology in Section 7.1.2, we describe our greedy append and circular sifting algorithms for the phases in Sections 7.1.3 and 7.1.4. Both are evaluated experimentally in Section 7.1.6.

7.1.2 Preliminaries

Throughout this part, let $G = (V, E)$ be a simple undirected graph with $n = |V|$ vertices and $m = |E|$ edges. Furthermore, let $N(v) = \{u \in V : \{u, v\} \in E\}$ denote the neighborhood of a vertex $v \in V$. A *circular layout* of G is a bijection $\pi : V \to \{0, \ldots, n-1\}$, interpreted as a clockwise sequence of distinct positions on the circumference of a circle. By selecting a reference vertex $s \in V$, we obtain linear orders $\prec_s^\pi$ from π by defining

$$u \prec_s^\pi v \iff (\pi(u) - \pi(s) \mod n) < (\pi(v) - \pi(s) \mod n)$$

for all $u, v \in V$, i. e., u is encountered before v in a cyclic traversal starting from s. We say that $u, v \in V$ are *consecutive*, denoted by $u \curvearrowright_\pi v$, if $\pi(v) - \pi(u) \equiv 1 \mod n$. A subset $W \subset V$ is *consecutive* if there is an order of the vertices of W so that $w_0 \curvearrowright_\pi w_1 \curvearrowright_\pi \ldots \curvearrowright_\pi w_{|W|-1}$, $w_i \in W$.

Let

$$\chi_\pi(\{u_1, v_1\}, \{u_2, v_2\}) = \begin{cases} 1 & \text{if } u_1 \prec_{u_1}^\pi u_2 \prec_{u_1}^\pi v_1 \prec_{u_1}^\pi v_2 \ , \\ 0 & \text{otherwise} \ . \end{cases} \tag{7.1}$$

for all $\{u_1, v_1\}, \{u_2, v_2\} \in E$ and w.l.o.g. $\pi(u_i) < \pi(v_i)$. We say that $e_1, e_2 \in E$ *cross* in π if and only if $\chi_\pi(e_1, e_2) = 1$, i. e., the endvertices of e_1, e_2 are encountered alternately in a cyclic traversal. The *crossing number* of a circular layout π is

$$\chi(\pi) = \sum_{e_1, e_2 \in E} \chi_\pi(e_1, e_2)$$

and $\chi(G) = \min_\pi \chi(\pi)$ is called the *circular crossing number* of G. We will omit π from our notation whenever the circular layout is clear from context.

Theorem 7.1 ([MKNF87]) *Circular crossing minimization is $\mathcal{NP}$-hard.*

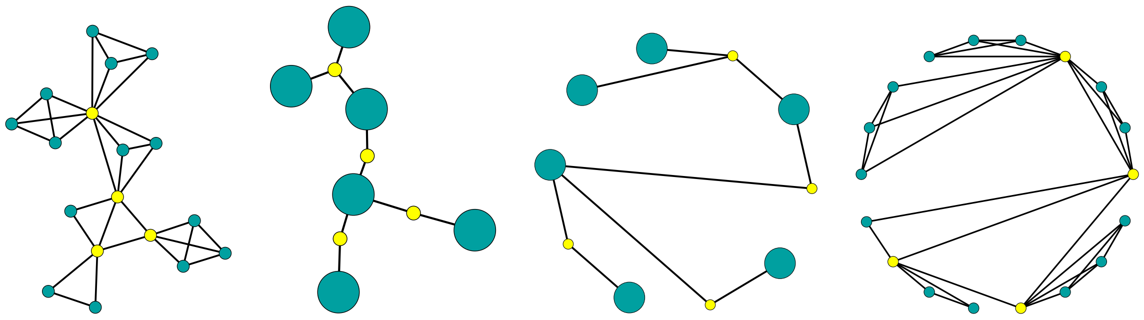


Figure 7.1: The circular crossing number of a graph is the sum of those of its biconnected components (cutpoints shown in lighter color.)

On the other hand, a graph has a circular layout with no crossings if and only if it is outerplanar. A linear time recognition algorithm for outerplanar graphs [Mit79] is easily extended to yield a crossing-free circular layout [ST99].

Since, in particular, trees have circular layouts with no crossings, it is possible to consider the biconnected components of a graph separately, and insert their circular layouts into a crossing-free layout of the block-cutpoint-tree without producing additional crossings. See Figure 7.1 for an illustration. Hence, only biconnected graphs are used in the experimental evaluation summarized in Section 7.1.6.

7.1.3 Initial Greedy Layout

Our approach for an initial layout is inspired by a heuristic algorithm for the minimum total edge length problem in circular layouts [Mäk88]. This problem is somewhat related to crossing minimization, since shorter edges tend to cross few other edges.

The basic idea is simple: start with a layout consisting of a single vertex and place the other vertices, one at a time, at either end of the current (linear) layout (see Algorithm 1). After all vertices are inserted, the final layout is considered to be circular. This method leaves us with three parameters to choose:

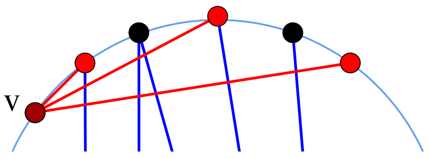
- the start vertex s,
- the processing sequence, and
- the end to append the next vertex at.

Note that the processing sequence need not to be fixed in the beginning, but may be determined while the algorithm proceeds. Since, in our experiments, the rules for choosing a start vertex had little influence on the final result, it is chosen at random. In the following we describe instantiations for the other two parameters.

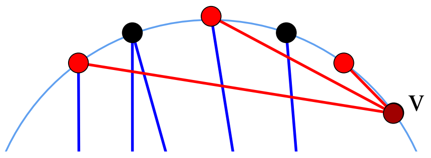
During the algorithm some vertices are already placed while others are not. An edge is called *open* if it connects a placed vertex with an unplaced one, and *closed* if both its vertices have been inserted.

Algorithm 1: Greedy-Append Heuristic

```
place start vertex $s \in V$ arbitrarily;
$V \leftarrow V \setminus \{s\}$;
while $V \neq \emptyset$ do
    greedily choose $v \in V$;
    append $v$ at either end of the current layout;
    $V \leftarrow V \setminus \{v\}$;
```



(a) adding v on the left



(b) adding v on the right

Figure 7.2: Incident edges of an inserted vertex v (red) cross open edges (blue).

Four rules for determining an insertion order are investigated. The rationale behind these heuristics is to keep the number of open edges low because they tend to result in crossings later on.

1. *Degree.* Vertices are inserted in non-increasing order of their degree.
2. *Inward Connectivity.* At each step, a vertex with the largest number of already placed neighbors is selected, i. e., a vertex which closes the most open edges.
3. *Outward Connectivity.* At each step, a vertex with the least number of unplaced neighbors is selected, i. e., a vertex which opens the fewest new edges.
4. *Connectivity.* At each step, a vertex with the largest number of already placed neighbors is selected, where ties are broken in favor of vertices with fewer unplaced neighbors.

The other degree of freedom left is the selection of an end of the current layout at which to append the next vertex. Again, four rules of choice are investigated.

1. *Random.* Select the end at which to append randomly each time.
2. *Fixed.* Always append to the same end.
3. *Length.* Append each vertex to the end that yields the smaller increase in total edge length.
4. *Crossings.* Append each vertex to the end that yields fewer crossing of edges being closed with open edges. In Figure 7.2, there are eight such crossings for the left end and only six for the right end. Note that crossings with closed edges not incident to the currently inserted vertex do not need to be considered because they are the same for both sides. It should also be noted that crossings with open edges are independent of the positions at which the unplaced vertex will eventually be placed.

The experiments outlined in Section 7.1.6 show that the combination of the *Connectivity* insertion order with *Crossings* outperforms all other combinations, and it can be implemented efficiently.

Theorem 7.2 *The Greedy-Append heuristic with* Connectivity *insertion order and end-to-append selection based on* Crossings *can be implemented to run in* $\mathcal{O}((n + m)\log n)$ *time.*

Proof The insertion sequence can be realized by storing all unplaced vertices in a two-dimensional priority queue, in which the first key gives the number of already placed neighbors and the second the number of unplaced neighbors. With an efficient implementation, update and extract operations require $\mathcal{O}(\log n)$ time. Since each vertex is extracted once, and each edge triggers exactly one update, the total running time for determining the insertion order is $\mathcal{O}((n + m)\log n)$.

The number of crossings with open edges can be determined from prefix and suffix sums over vertices already in the layout. These can be maintained efficiently using a balanced binary tree storing in its leaves the number of open edges incident to a placed vertex, and in its inner nodes the sum of the values of its two children. The prefix sum at a vertex is the sum of all values in the left children of nodes on the path from the corresponding leaf to the root. The suffix sum is determined symmetrically. Insertion of a vertex thus requires $\mathcal{O}(\log n)$ time to determine the crossing numbers from prefix and suffix sums and $\mathcal{O}(\deg(v)\log n)$ for updating the tree. The total is again $\mathcal{O}((n + m)\log n)$. $\square$

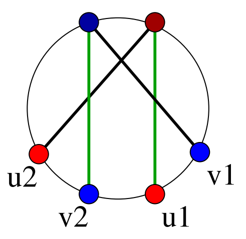
Note that the heuristic is easily generalized to weighted graphs. In the next section we show how to further reduce the number of crossings, given an initial layout.

7.1.4 Improvement by Circular Sifting

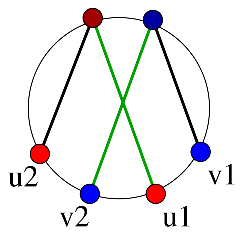
Sifting was originally introduced as a heuristic for vertex minimization in ordered binary decision diagrams [Rud93] and later adapted for the one-sided crossing minimization problem [MSM00]. The idea is to keep track of the objective function while moving a vertex along a fixed order of all other vertices. The vertex is then placed in its (locally) optimal position. The method is thus an extension of the greedy-switch heuristic [EK86].

For crossing reduction the objective function is the number of crossings between the edges incident to the vertex under consideration and all other edges. The efficient computation of crossing numbers in sifting for layered layouts is based on the crossing matrix. Its entries correspond to the number of crossings caused by pairs of vertices in a particular linear order and are computed easily in advance. Whenever a vertex is placed in a new position only a smallish number of updates is necessary.

It is not possible to adapt the crossing matrix to the circular case since two vertices cannot be said to be in a (linear) order in general. Thus, we define the crossing



(a) no crossings between the green edges



(b) only the green edges cross

Figure 7.3: After swapping consecutive vertices $u \curvearrowright v$, exactly those pairs of edges cross that did not before.

number

$$c_{uv}(\pi) = \sum_{x \in N(u)} \sum_{y \in N(v)} \chi_\pi(\{u,x\},\{v,y\}) \tag{7.2}$$

only for pairs of consecutive vertices $u \curvearrowright v \in V$ and use the following exchange property, which is the basis for sifting and holds nevertheless.

Lemma 7.3 *Let $u \curvearrowright v \in V$ be consecutive vertices in a circular layout π and let π' be the layout with their positions swapped, then the number of crossings in the new layout is*

$$\begin{aligned}\chi(\pi') &= \chi(\pi) - c_{uv}(\pi) + c_{vu}(\pi') \\ &= \chi(\pi) - \sum_{x \in N(u)} \left|\{y \in N(v) : y \prec^\pi_x u\}\right| + \sum_{y \in N(v)} \left|\{x \in N(u) : x \prec^{\pi'}_y v\}\right|\end{aligned}$$

Proof Since u and v are consecutive, edges incident to neither u nor v do not change their crossing status. The first equality follows immediately. For the second equality, observe that the sums are obtained from (7.2) by inserting (7.1). See Figure 7.3 for an illustration. □

Based on the above lemma, the locally optimal position of a single vertex can be found by iteratively swapping the vertex with its neighbor and recording the change in crossing count, which is computed by considering only edges incident to one of these two vertices. After the vertex has been moved past every other vertex, it is placed where the intermediary crossing counts reached their minimum. Repositioning a vertex in this way is called *sifting a vertex* and sifting every vertex once in this way is called a *round of sifting.*

If adjacency lists are ordered according to the current layout, the sums in Lemma 7.3 are over suffix lengths in these lists. Updating the crossing count therefore corresponds to merging the adjacency lists, where the length of the remaining suffix is added or subtracted.

Theorem 7.4 *One round of circular sifting takes $\mathcal{O}(nm)$ time.*

Proof Sorting the adjacency lists according to the vertex order is easily done in $\mathcal{O}(m)$ time (traverse the vertices in order, and add each to the adjacency lists of

Algorithm 2: Circular Sifting

```
for ($u \in V$) do
    let $v_0 = u \prec_u v_1 \prec_u \ldots \prec_u v_{n-1}$ denote the current layout;
    for ($v \in V$) do
        sort adjacency list of $v$ according to the current layout;
    $\chi \leftarrow 0$;  $\chi^* \leftarrow 0$;  $v^* \leftarrow v_{n-1}$;
    for ($k \leftarrow 1, \ldots, n-1$) do
        let $x_0 \prec_{v_k} \ldots \prec_{v_k} x_{r-1}$ denote the adjacency list of $u$ without $v_k$;
        let $y_0 \prec_{v_k} \ldots \prec_{v_k} y_{s-1}$ denote the adjacency list of $v_k$ without $u$;
        $c \leftarrow 0$;  $i \leftarrow 0$;  $j \leftarrow 0$;
        while ($i < r$ and $j < s$) do
            if ($x_i \prec_{v_k} y_j$) then
                $c \leftarrow c - (s - j)$;  $i \leftarrow i + 1$;
            else if ($y_j \prec_{v_k} x_i$) then
                $c \leftarrow c + (r - i)$;  $j \leftarrow j + 1$;
            else
                $c \leftarrow c - (s - j) + (r - i)$;  $i \leftarrow i + 1$;  $j \leftarrow j + 1$;
        $\chi \leftarrow \chi + c$;
        if ($\chi < \chi^*$) then $\chi^* \leftarrow \chi$;  $v^* \leftarrow v_k$;
    move $u$ so that $v^* \curvearrowright u$;
```

its neighbors). If adjacency lists are stored cyclically, a head pointer yields $\prec_v$ for arbitrary v, i. e., the adjacency lists need not be reordered before a swap. The final relocation of u takes constant time.

When swapping u with neighbor v_k, the time for the traverse of the adjacency lists is in $\mathcal{O}(d_G(u) + d_G(v_k))$. Since

$$\sum_{u \in V} \sum_{v \in V} \big(\deg_G(u) + \deg_G(v)\big) = \sum_{u \in V} \sum_{v \in V} \deg_G(u) + \sum_{u \in V} \sum_{v \in V} \deg_G(v) = 2 \cdot n \cdot 2m$$

the total running time is in $\mathcal{O}(nm)$. □

At the end of the outer loop each vertex is placed at its locally optimal position, so that circular sifting can only decrease the number of crossings. Our experiments outlined in Section 7.1.6 suggest that a few rounds of sifting suffice to reach a local minimum.

7.1.5 Constraint Crossing Reduction

Applications sometimes impose specific constraints on the circular layout. For example, the handling of unconfirmed edges in the radial visualization requires maintaining a given order for the core vertices (see Section 6.2.3). Our algorithm can be altered to consider the following constraints.

Edge Weights

For a given edge weight $\omega : E \to \mathbb{R}_{\geq 0}$, we can define the *weighted crossing number* by counting each crossing with the product of the two edge weights involved, i. e., Equation 7.1 can be generalized to

$$\chi_\pi(\{u_1, v_1\}, \{u_2, v_2\}) = \begin{cases} \omega(\{u_1, v_1\}) \cdot \omega(\{u_2, v_2\}) & \text{if } u_1 \prec^\pi_{u_1} u_2 \prec^\pi_{u_1} v_1 \prec^\pi_{u_1} v_2 \ , \\ 0 & \text{otherwise .} \end{cases} \tag{7.3}$$

In order to adapt Lemma 7.3 to the weighted case, we replace suffix cardinalities by suffix sums of weights, i. e.,

$$\begin{aligned} \chi(\pi') &= \chi(\pi) - c_{uv}(\pi) + c_{vu}(\pi') \\ &= \chi(\pi) - \sum_{x \in N(u)} \omega(\{u, x\})\Big(\sum_{\{y \in N(v) \,:\, y \prec^\pi_x ur\}} \omega(\{v, y\})\Big) \\ &\qquad + \sum_{y \in N(v)} \omega(\{v, y\})\Big(\sum_{\{x \in N(u) \,:\, x \prec^{\pi'}_y v\}} \omega(\{u, x\})\Big) \ . \end{aligned}$$

Modifying the algorithm accordingly is straightforward.

Insertion Sifting

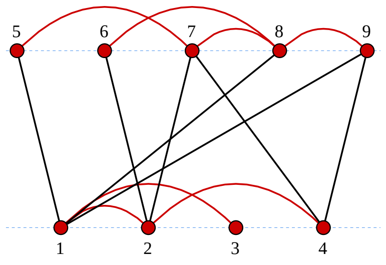
Sifting can be used to compute an initial layout by iteratively adding a new vertex to the current layout and sifting it, i. e., we relocate it to all positions and finally place it there the running crossing count reached the minimum. Since the greedy phase is faster and avoids some pathologic cases, it is preferable for computing an order from sketch. However, insertion sifting can be used when an initial order of a subset of the vertices is given which should be preserved (see also the next paragraph).

Fixed Vertices and Forbidden Positions

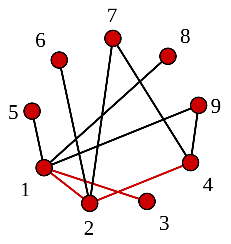
The fundamental idea of sifting, iteratively select a vertex and find a better position for it while keeping all other vertices fixed, allows for not moving every vertex and not testing every position at the cost of an increase in the crossing number. Therefore, a given initial order of a subset of the vertices can be respected by simply not sifting the contained vertices. Similarly, some positions can be prohibitive for specific vertices and are therefore not considered as target for the relocation to the minimum position.

Linear Layered Layouts

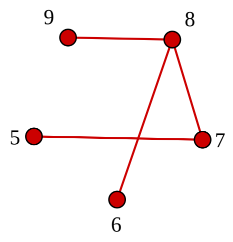
The modification for linear layered layouts is an ably combination of fixed vertices and prohibited positions. According to Sugiyama's framework which is described



(a) two-layered layout



(b) circular layout for both layers



(c) circular layout for the upper layer

Figure 7.4: A (linear) two-layer layout with intra-layer edges and its corresponding circular layouts.

in more detail in Section 6.3.1, typical algorithms iteratively compose the overall layout from sub-layouts for two consecutive layers. Edges connecting vertices of the same layer often occur in practice, e. g., in the layer assignment of the status visualization, but are commonly ignored by crossing reduction algorithms for such layouts.

When drawn as straight lines, intra-layer edges overlap both each other and the vertices. Therefore, we draw them as curves and, for simplicity, restrict them to run above the layer of their endvertices. Figure 7.4 gives an example of this layout convention and also suggests how to form corresponding circular layouts by connecting the appropriate endings of the linear orders. In order to maintain the base layered layout during sifting, we fix all vertices of the reference layer and forbid all positions between them for the vertices of the layer to reorder.

For multi-layered graphs, the classical layer-by-layer sweep considers in each step two consecutive layers and reorders one of them while keeping the other one fixed. Crossings of edges between the current layer and its other neighboring layer are disregarded in this step. Sifting allows to take both adjacent layers into account by simply summing the individual crossing counts of each side.

Bachmaier and Forster [BF06] introduce the same layout convention but propose a more complicated, separated treatment of inter- and intra-layer edges. Their experiments suggest that considering intra-edges significantly reduces the number of crossings compared to previous methods.

7.1.6 Experimental Evaluation

We performed extensive experiments to determine the relative behavior of the different variants of our heuristics. As a base reference we use CIRCULAR [ST99], the currently most effective heuristic for circular crossing minimization. CIRCULAR consists of two phases as well: an initial placement (CIRCULAR 1) derived from a recognition algorithm for outerplanar graphs [Mit79], and a subsequent improvement phase (CIRCULAR 2) that probes alternative positions for each vertex

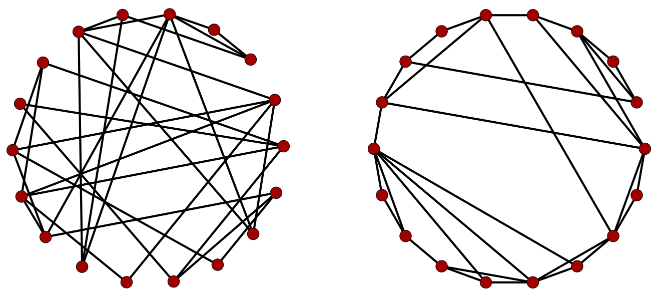


Figure 7.5: Circular layouts for a random order and for our algorithm.

and relocates if the number of crossings is reduced. The second phase is similar to circular sifting but vertices are restricted to specific candidate positions and thus may miss optimum positions. Note also that CIRCULAR 2 actually counts crossings (rather than just changes) so that its running time depends on the number of crossings. When restricting replacements to a subset of positions, circular sifting simulates CIRCULAR 2 with an improved worst-case performance but in our experiments we rather implemented an improved method for counting crossings since realistic graphs have relatively few crossings anyway.

We have implemented all algorithms in C++ using LEDA [MN99]. Our experiments were carried out on a standard desktop computer with 1.5 GHz and 512 MB running Linux. Each data point is the average of 10 runs with different internal initializations (in particular, permuted adjacency lists).

The experiments were run on three families of undirected, biconnected graphs (recall from Section 7.1.2 that crossings between edges in different biconnected components can be avoided altogether):

- *Rome graphs.* A set of 10 541 biconnected components with 10 to 80 vertices used in [DBGL+97]. These are sparse real-world graphs with $m \approx 1.3n$.
- *Fixed average degree.* Three sets of random graphs with 10 to 200 vertices and variable edge probability of $\frac{3}{n-1}$, $\frac{5}{n-1}$, and $\frac{10}{n-1}$, resulting in graphs with expected average degree of 3, 5, and 10.
- *Fixed density.* Three sets of random graphs with 10 to 200 vertices and fixed edge probability of 0.02, 0.05, and 0.1, resulting in graphs with expected density of 0.02, 0.05, and 0.1, respectively.

A comprehensive selection of results is given in the appendix. We here summarize our conclusions and show a layout computed by the combination of greedy-append and circular sifting for a sample graph (see Figure 7.5).

Initialization using Greedy Append The performance of various combinations of insertion orders for greedy append is shown in Figure 8.7 relative to CIRCULAR 1. While for some rules of choice the results depend on number of edges in the graph, the *Connectivity* variant consistently outperforms all others, including CIRCULAR 1.

The results in Figure 8.8 indicate that appropriate placement is indeed important, but has a much smaller effect than the insertion order. On random graphs, the combinations of *Connectivity* insertion with *Length* and *Crossings* selection criteria perform almost equally well, with a slight advantage for *Crossings*.

The two best combinations, *Connectivity* with *Length* or *Crossings*, compare favorably with CIRCULAR 1, both in terms of the resulting number of crossings and running time (see Figures 8.10, 8.11, and 8.15). Note that the running time of the initialization methods is negligible when compared to the improvement strategies.

Subsequent Improvement using Circular Sifting Circular sifting reaches a local minimum in few rounds. As can be expected, the improvement is larger in early rounds and the number of rounds required depends on the initial configuration (see Figure 8.9). It can be concluded that the improvement algorithms (circular sifting and CIRCULAR 2) should not be used by themselves but only in combination with a good initialization method.

With any of the good initialization strategies identified in the previous subsection, circular sifting is able to further reduce the number of crossings produced by CIRCULAR 2 as can be seen in Figures 8.10 and 8.13 and is also confirmed by an independent study of He and Sýkora [HS04]. This suggests that the additional positions considered for relocation indeed pay off. However, there is a slight runtime penalty if sifting is run until there is no further improvement (Figure 8.14).

7.1.7 Conclusion

We have presented an approach for circular graph layouts with few crossings. It consists of two phases: in the first phase, we greedily append vertices to either end of a partial (linear) layout according to some criteria, and in the second we further reduce the number of crossings by repeatedly sifting each vertex to a locally optimal position. Furthermore, we have shown that our algorithm can easily be adopted for some modified circular crossing reduction problems relevant in practice and even for two-layer layouts with existing intra-layer edges.

Our experimental evaluation clearly shows that the method of choice is to initialize circular sifting with a greedy-append approach using the *Connectivity* insertion order with the *Crossings* placement rule and that this combination consistently outperforms previous heuristics. They also show that both phases are necessary. While circular sifting yields a substantial improvement over the initial layouts, a good initialization significantly reduces the number of rounds required and thus the overall running time at essentially no extra cost.

7.2 Multi-Circular Layout of Micro/Macro Graphs

We propose a layout algorithm for micro/macro graphs, i. e., relational structures with two levels of detail. While the micro-level graph is given, the macro-level graph is induced by a given partition of the micro-level vertices. A typical example is a social network of employees organized into different departments. We do not impose restrictions on the macro-level layout other than sufficient thickness of edges and vertices, so that the micro-level graph can be placed on top of the macro-level graph.

For the micro-level graph we define a combinatorial multi-circular embedding and present corresponding layout algorithms based on edge crossing reduction strategies. Each micro-vertex is placed on a circle inside of the area of its corresponding macro-vertex and micro-edges whose endvertices belong to the same macro-vertex are drawn inside of these circles. All other micro-edges are then drawn inside of their corresponding macro-edges and at constant but different distances from the border of the macro-edge, i. e., in straight-line macro-edges they are drawn as parallel lines. These edges must also be routed inside the area of macro-vertices to connect to their endpoints but are not allowed to cross the circles. In principle, an arbitrary layout strategy can be used as long as it complies with these requirements. Figure 7.6 shows a concrete example of this model. Micro-edges connecting vertices in the same macro-vertex are drawn as straight lines. Inside of macro-vertices, the other edges spiral around the circle of micro-vertices until they reach the area of the macro-edge.

We give a combinatorial description of the above model and then focus on the algorithmically most challenging aspect of these layouts, namely crossing reduction by cyclic ordering of micro-vertices and choosing edge winding within macro-vertices.

After defining some basic terminology in Section 7.2.1, we state required properties for macro-graph layout in Section 7.2.2 and recapitulate related micro layout models in Section 7.2.3. Multi-circular micro-graph layout is discussed in more detail in Section 7.2.4 and crossing reduction algorithms for it are given in Section 7.2.5.

7.2.1 Preliminaries

Throughout this part, let $G = (V, E)$ be a simple undirected graph with $n = |V|$ vertices and $m = |E|$ edges. Furthermore, let $E(v) = \{\{u, v\} \in E : u \in V\}$ denote the incident edges of a vertex $v \in V$, let $N(v) = \{u \in V : \{u, v\} \in E\}$ denote its neighbors, and let $\text{sgn} : \mathbb{R} \to \{-1, 0, 1\}$ be the signum function.

Since each micro-vertex is required to belong to exactly one macro-vertex, the macro-structure defines a partition of the micro-vertices. Contrary to this top-down approach, we can also start from the bottom. Recall from Section 3.1 that a *partition assignment* $\phi : V \to \{0, \dots, k-1\}$ for G subdivides the (micro-)vertex set V into k pairwise disjoint, non-empty subsets $V = V_0 \dot\cup \dots \dot\cup V_{k-1}$, where $V_i = \{v \in V : \phi(v) = i\} = \phi^{-1}(i)$. An edge $e = \{u, v\} \in V_i \times \in V_j$ is called an *intra-partition edge*

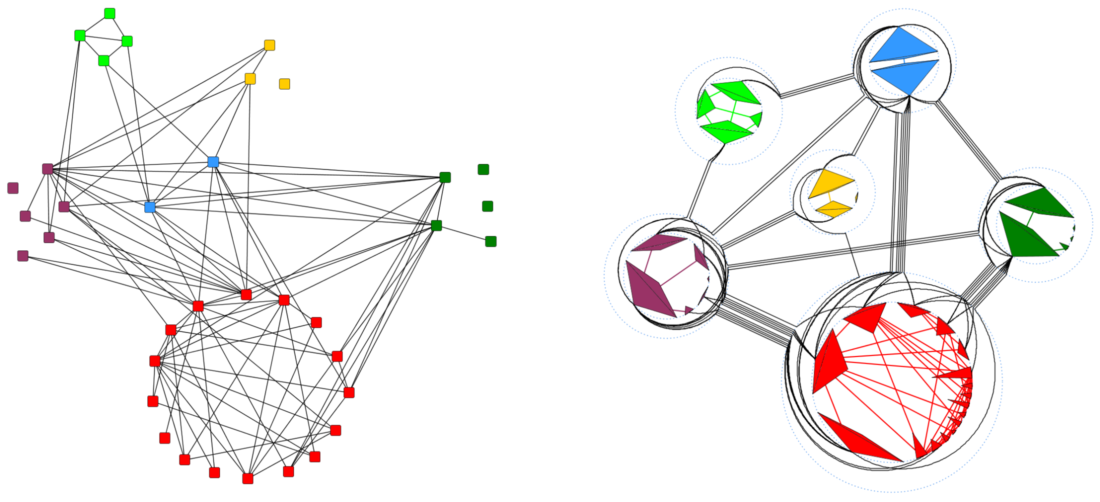

(a) geometric grouping, straight-line edges (b) multi-circular layout

Figure 7.6: (a) Example organizational network with geometric grouping and straight-line edges (redrawn from [Kre96]). In our multi-circular layout (b), all details are still present and the macro-structure induced by the grouping becomes visible. The height and width of the vertices reflects the number of connections within and between groups.

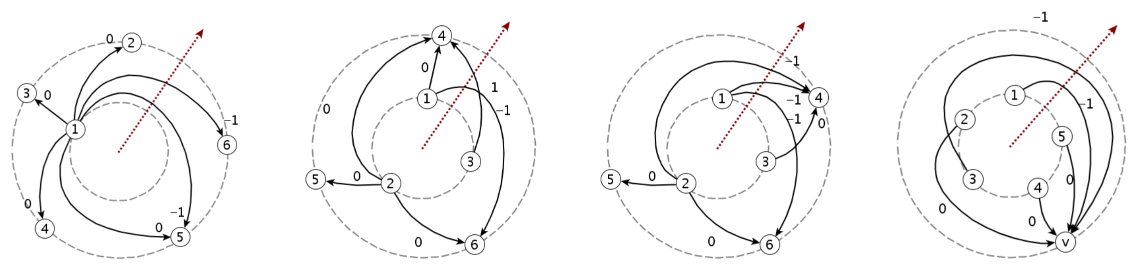
if and only if $i = j$, otherwise it is called an *inter-partition edge*. The set of intra-partition edges of a partition V_i is denoted by E_i, the set of inter-partition edges of two partitions V_i, V_j by $E_{i,j}$. We use $G = (V, E, \phi)$ to denote a graph $G = (V, E)$ and a related partition assignment ϕ.

A *circular order* $\pi = \{\pi_0, \ldots, \pi_{k-1}\}$ defines for each partition V_i a vertex order π_i as a bijective function $\pi_i : V_i \to \{0, \ldots, |V_i| - 1\}$ with $u \prec v \Leftrightarrow \pi_i(u) < \pi_i(v)$ for any two vertices $u, v \in V_i$. An order π_i can be interpreted as a counter-clockwise sequence of distinct positions on the circumference of a circle.

7.2.2 Macro Layout

A prototypical macro-graph, the *quotient graph*, is defined by a partition assignment. Given a partition assignment $\phi : V \to \{0, \ldots, k-1\}$, the corresponding quotient graph $Q(G, \phi) = (\mathcal{V}_Q, \mathcal{E}_Q)$ is defined as $\mathcal{V}_Q = \{V_0, \ldots, V_{k-1}\}$ and $\mathcal{E}_Q = \big\{\{V_i, V_j\} : \exists u \in V_i, \exists v \in V_j, \{u, v\} \in E\big\}$, i. e., $Q(G, \phi)$ contains a vertex for each partition of G and two vertices $V_i, V_j \in \mathcal{V}_Q$ are connected if and only if E contains at least one edge between a vertex in V_i and a vertex in V_j.

We do not require a specific layout strategy for the macro-graph as long as its elements are rendered with sufficient thickness to draw the underlying micro-graph on top of them. In order to achieve this, post-processing can be applied to any given layout [GN99] or methods which consider vertex size (e. g., [HK02, WM96]) and edge thickness (e. g., [DEKW02]) have to be used.



(a) some incident edges

(b) node 4 is at position 0

(c) node 4 rotated to position 2

(d) without parting

Figure 7.7: Examples of Radial layouts. Edges are labeled with their winding value.

From a macro-layout we get *partition orders* $\Pi_i : N_Q(V_i) \to \{0, .., \deg(V_i) - 1\}$ for each partition V_i, defined by the sequence of its incident edges in $Q(G, \phi)$, and a partition order $\Pi = \{\Pi_0, \ldots, \Pi_{k-1}\}$ for G. For each macro-vertex this can be seen as a counter-clockwise sequence of distinct docking positions for its incident macro-edges on its border.

7.2.3 Related (Micro-) Layout

Before we discuss the multi-circular layout model for the micro-graph, let us recall the related concepts of (single) circular and radial embeddings. In *(single) circular layouts* all vertices are placed on a single circle and edges are drawn as straight lines. Therefore, a *(single) circular embedding* ε of a graph $G = (V, E)$ is fully defined by a vertex order π, i. e., $\varepsilon = \pi$ [BB04a]. Two edges $e_1, e_2 \in E$ *cross* in ε if and only if the endvertices of e_1, e_2 are encountered alternately in a cyclic traversal.

In *radial level layouts* the partitions are placed on nested concentric circles (*levels*) and edges are drawn as curves between consecutive partitions. Therefore, only graphs $G = (V, E)$ with a *proper* partition assignment $\phi : V \to \{0, \ldots, k-1\}$ are allowed, i. e., $|\phi(u) - \phi(v)| = 1$ for all edges $\{u, v\} \in E$. Note that this prohibits intra-partition edges and edges connecting non-consecutive partitions. For technical reasons, edges are considered to be directed from lower to higher levels.

Recently, Bachmaier [Bac07] investigated such layouts. They introduced a *ray* from the center to infinity to mark the start and end of the circular vertex orders. Using this ray, it is also possible to count how often and in which direction an edge is wound around the common center of the circles. We call this the *winding* $\psi : E \to \mathbb{Z}$ of an edge (Bachmaier called this *offset*). $|\psi(e)|$ counts the number of crossings of the edge with the ray and the sign reflects the mathematical direction of rotation. See Figure 7.7 for some illustrations. Finally, a *radial embedding* ε of a graph $G = (V, E, \phi)$ is defined to consist of a vertex order π and an edge winding ψ, i. e., $\varepsilon = (\pi, \psi)$.

There is additional freedom in radial drawings without changing the crossing number: the rotation of a partition V_i. A *rotation* moves a vertex v with extremal position in π_i over the ray. In Figure 7.7 layout (c) is a clockwise rotation of layout (b). Rotations do not modify the cyclic order, i. e., the neighborhood of each vertex on its radial level is preserved. However, the winding of the edges incident to v and all positions of π_i must be updated.

Crossings between edges in radial embeddings depend on their winding and on the order of the endvertices. There can be more than one crossing between two edges if they have very different windings. We denote the number of crossings between two edges $e_1, e_2 \in E$ in an radial embedding ε by $\chi_\varepsilon(e_1, e_2)$. The (radial) crossing number of an embedding ε and a level graph $G = (V, E, \phi)$ is then naturally defined as

$$\chi(\varepsilon) = \sum_{\{e_1,e_2\}\in E, e_1\neq e_2} \chi_\varepsilon(e_1, e_2)$$

and $\chi(G) = \min\{\chi(\varepsilon) : \varepsilon \text{ is a radial embedding of } G\}$ is called the *radial crossing number* of G.

Theorem 7.5 ([Bac07]) *Let $\varepsilon = (\pi, \psi)$ be a radial embedding of a two-level graph $G = (V_1 \dot\cup V_2, E, \phi)$. The number of crossings $\chi_\varepsilon(e_1, e_2)$ between two edges $e_1 = (u_1, v_1) \in E$ and $e_2 = (u_2, v_2) \in E$ is*

$$\chi_\varepsilon(e_1, e_2) = \max\left\{0, \left|\psi(e_2) - \psi(e_1) + \frac{b-a}{2}\right| + \frac{|a| + |b|}{2} - 1\right\} ,$$

where $a = \operatorname{sgn}(\pi_1(u_2) - \pi_1(u_1))$ and $b = \operatorname{sgn}(\pi_2(v_2) - \pi_2(v_1))$.

Bachmaier also states that in crossing minimal radial embeddings every pair of edges crosses at most once and adjacent edges do not cross at all. As a consequence, only embeddings need to be considered where there is a clear *parting* between all edges incident to the same vertex u. The parting is the position of the edge list of u that separates the two subsequences with different winding values. See Figure 7.7 for layouts with and without proper parting. Furthermore, only embeddings with small winding are considered because large winding values correspond to very long edges which are difficult to follow and generally result in more crossings.

7.2.4 Multi-Circular Layout

Unless otherwise noted, vertices and edges belong to the micro-level in the following. In the micro-layout model each vertex is placed on a circle inside of its corresponding macro-vertex. Intra-partition edges are drawn within these circles as straight lines. Inter-partition edges are drawn inside their corresponding macro-edges and at constant but different distances from the border of the macro-edge. To connect to their incident vertices, these edges must also be routed inside of macro-vertices. Since they are not allowed to cross the circles, they are drawn as curves around them. We call such a drawing a *(multi-)circular layout.* Since intra- and inter-partition edges cannot cross, all crossings of intra-partition edges are completely defined by the vertex order π_i of each partition V_i. Intuitively speaking, a vertex order defines a circular layout for the intra-partition edges. In the following we thus concentrate on inter-partition edges.

The layout inside each macro-vertex V_i can be seen as a two-level radial layout. The orders can be derived from the vertex order π_i and the partition order Π_i. Similar to radial layouts we introduce a *ray* for each partition and define the beginning of the orders and the edge winding according to these rays. Note that for each edge $e = \{u, v\} \in E$, $u \in V_i$, $v \in V_j$, two winding values are needed, one for the winding around partition V_i denoted by $\psi_i(e) = \psi_u(e)$, and one for the winding around partition V_j denoted by $\psi_j(e) = \psi_v(e)$. If the context implies an implicit direction of the edges, we call windings either source or target windings, respectively. Since radial layouts can be rotated without changing the embedding, rays of different partitions are independent and can be directed arbitrarily. Finally, a *multi-circular embedding* ε is defined by a vertex order π, a partition order Π, and the winding of the edges ψ, i. e., $\varepsilon = (\pi, \Pi, \psi)$.

Observation 7.6 *For each partition V_i in a multi-circular embedding $\varepsilon = (\pi, \Pi, \psi)$ a two-level radial embedding $\varepsilon_i = ((\pi_i, \pi'), \psi_i)$ is defined by the vertex order π_i, the partition order Π_i, and the edge winding ψ_i, where $\pi'(v) = \Pi_i(\phi(v)), v \in V \setminus V_i$.*

There is another connection between radial and multi-circular layouts. A two-level radial layout can easily be transformed into a two-partition circular layout and vice versa. Given a graph $G = (V_1 \dot{\cup} V_2, E, \phi)$ and a radial embedding $\varepsilon = (\pi, \psi)$ of G, the two-partition circular embedding $\varepsilon^* = (\pi^*, \Pi^*, \psi^*)$ defined by $\pi_1^* = \pi_1$, $\pi_2^* = -\pi_2$, $\Pi_1^* = 0$, $\Pi_2^* = 0$, and $\psi_1^*(e) = \psi(e)$, $\psi_2^*(e) = 0$ realizes exactly the same crossings (see Figure 7.8 for an example). Intuitively speaking, the topology of the given radial embedding is not changed if we drag the two circles apart and reverse one of the vertex orders. If a two-partition circular embedding $\varepsilon^* = (\pi^*, \Pi^*, \psi^*)$ is given, a related radial embedding $\varepsilon = (\pi, \psi)$ is defined by $\pi_1 = \pi_1^*$, $\pi_2 = -\pi_2^*$, and $\psi(e) = \psi_1(e) - \psi_2(e)$.

Observation 7.7 *There is a one-to-one correspondence between a two-level radial embedding and a two-circular embedding.*

Crossings in the micro-layout are due to either the circular embedding or crossing macro-edges. Since crossings of the second type cannot be avoided by changing the micro-layout, we do not consider them in the micro-layout model. Obviously, pairs of edges which are not incident to a common macro-vertex can only cause crossings of this type. For pairs of edges which are incident to at least one common macro-vertex we can define corresponding two-level radial layouts using Observations 7.6 and 7.7 and compute the number of crossings by modifications of Theorem 7.5.

Theorem 7.8 *Let $\varepsilon = (\pi, \Pi, \psi)$ be a multi-circular embedding of a graph $G = (V, E, \phi)$ and let $e_1 = \{u_1, v_1\}$, $e_2 = \{u_2, v_2\} \in E$ be two inter-partition edges. If e_1 and e_2 share exactly one common incident macro-vertex, e. g., $V_i = \phi(u_1) = \phi(u_2)$,*

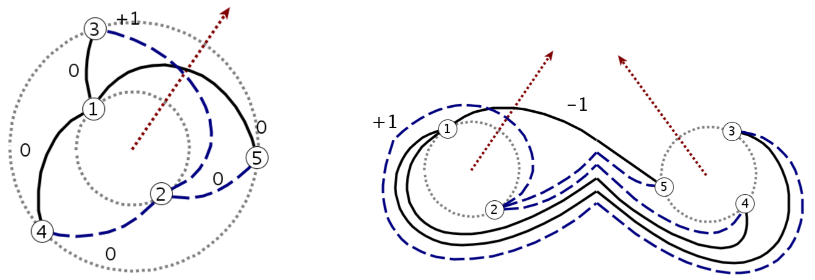


Figure 7.8: A two-level radial layout and its corresponding two-circular layout.

$\phi(v_1) \neq \phi(v_2)$, then the number of crossings of e_1 and e_2 is

$$\chi_\varepsilon(e_1, e_2) = \max\left\{0, \left|\psi_i(e_2) - \psi_i(e_1) + \frac{b-a}{2}\right| + \frac{|a|+|b|}{2} - 1\right\},$$

where $a = \operatorname{sgn}(\pi_i(u_2) - \pi_i(u_1))$ and $b = \operatorname{sgn}(\Pi(\phi(v_2)) - \Pi(\phi(v_1)))$.

Proof Let $e_1 = \{u_1, v_1\}$, $e_2 = \{u_2, v_2\} \in E$ be two edges with exactly one common end partition, e. g., $V_i = \phi(u_1) = \phi(u_2), \phi(v_1) \neq \phi(v_2)$. All crossings between e_1 and e_2 not caused by the macro layout occur in the macro-vertex V_i. According to Observation 7.6, we can regard the fraction of the layout in V_i as a two-level radial layout defined by $\varepsilon' = (\pi_i, \Pi_i \circ \phi)$. Applying Theorem 7.5 to the embedding ε', the theorem follows. □

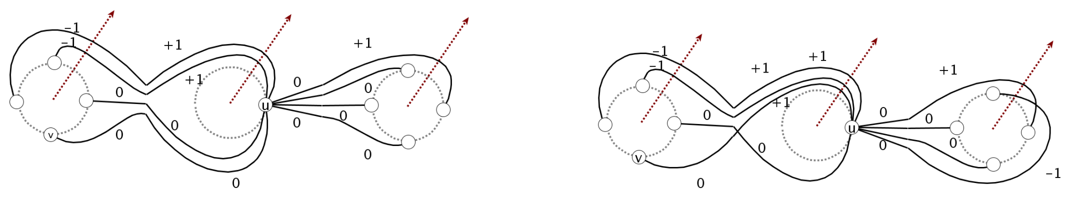
Theorem 7.9 *Let $\varepsilon = (\pi, \Pi, \psi)$ be a multi-circular embedding of a graph $G = (V, E, \phi)$ and let $e_1 = \{u_1, v_1\}$, $e_2 = \{u_2, v_2\} \in E$ be two inter-partition edges. If e_1 and e_2 belong to the same macro-edge, e. g., $V_i = \phi(u_1) = \phi(u_2)$, $V_j = \phi(v_1) = \phi(v_2)$, then the number of crossings between e_1 and e_2 is*

$$\chi_\varepsilon(e_1, e_2) = \max\left\{0, \left|\psi'(e_2) - \psi'(e_1) + \frac{b-a}{2}\right| + \frac{|a|+|b|}{2} - 1\right\},$$

where $a = \operatorname{sgn}(\pi_i(u_2) - \pi_i(u_1))$, $b = \operatorname{sgn}(\pi_j(v_1) - \pi_j(v_2))$ and $\psi'(e) = \psi_i(e) + \psi_j(e)$.

Proof Let $e_1, e_2 \in E$ be two inter-partition edges which belong to the same macro-edge. Since only two partitions are involved, we can define a two-level radial embedding ε' for e_1 and e_2 according to Observation 7.7. In ε' the two edges e_1 and e_2 cause the same crossings than in ε. Applying Theorem 7.5 to the embedding ε', the theorem follows. □

Similar to radial layouts, in a crossing minimal multi-circular embedding incident edges do not cross and there is at most one crossing between every pair of edges. Therefore, only embeddings need to be considered where there is a clear *parting* between all edges incident to the same vertex $u \in V_i$. Since in multi-circular layouts winding in different macro-vertices can be defined independently, we split the edge list $E(u)$ of u by target partitions and get edge lists $E(u)_j = \{\{u, v\} \in E(u) : v \in$



(a) parting (b) edge $\{u, v\}$ violates parting

Figure 7.9: Not all winding combinations for the incident edges of u result in a good layout.

$V_j\}$. For each list $E(u)_j$, we get a position ℓ_j that separates the two subsequences with different values of winding ψ_j and defines the parting for this partition. Furthermore, there is also a parting for V_i defined on the edge list $E(u)$. The order of $E(u)$ for this parting depends on the partings ℓ_j in the target partitions V_j. Edges are sorted by the partition order and for edges to the same partition V_j, ties are broken by the reverse vertex order started not at the ray but at the parting position ℓ_j. Then, the parting for V_i is the position ℓ_i which separates different values of winding ψ_i in the so ordered list. See Figure 7.9 for a layout with parting and a layout where the edge $\{u, v\}$ violates the parting.

Corollary 7.10 *Multi-circular crossing minimization is $\mathcal{NP}$-hard.*

Proof Single circular and radial crossing minimization [Bac07, MKNF87] are $\mathcal{NP}$-hard. As we have already seen, these two crossing minimization problems are subproblems of the multi-circular crossing minimization problem, proving the corollary. □

As a consequence, we do not present exact algorithms for crossing minimization in multi-circular layouts. Instead, we propose extensions of some well-known crossing reduction heuristics for horizontal and radial crossing reduction.

7.2.5 Layout Algorithms

Since the drawing of inter-partition edges inside a macro-vertex can be seen as a radial drawing, a multi-circular layout can be composed of separate radial layouts for each macro-vertex (for instance using the techniques of [ST04, GK07, Bac07]). However, such a decomposition approach is inappropriate since intra-partition edges are not considered at all and inter-partition edges are not handled adequately due to the lack of information about the layout at the adjacent macro-vertices. For example, choosing a path with more crossings in one macro-vertex can allow a routing with much less crossings on the other side.

Nevertheless, we initially present in this section adaptations of radial layout techniques because they are quite intuitive, fast, and simple, and can be used for the evaluation of more advanced algorithms.

Barycenter and Median Layouts

The basic idea of both the barycenter and the median layout heuristic is the following: each vertex is placed in a central location computed from the positions of its neighbors - in either the barycenter or the median position - to reduce edge lengths and hence the number of crossings. For a two-level radial layout, the *Cartesian Barycenter* heuristic gets the two levels and a fixed order for one of them. All vertices of the fixed level are set to equidistant positions on a circle and the component-wise barycenter for all vertices of the second level is computed. The cyclic order around the center defines the order of the vertices and the edges are routed along the geometrically shortest-path. The *Cartesian Median* heuristic is defined similar. Running time for both heuristics is in $\mathcal{O}(|E| + |V| \log |V|)$.

Both heuristics are easily extended for multi-circular layouts. The layout in each macro-vertex V_i is regarded as a separate two-level radial layout as described in Observation 7.7 and the partition orders Π_i are used to define the orders of the fixed levels. Because of the shortest-path routing, no two edges cross more than once and incident edges do not cross at all in the final layout. On the other hand, the used placement and winding strategies are based on edge length reduction and avoid crossings only indirectly.

Multi-Circular Sifting

To overcome the drawbacks of the radial layout algorithms described before, we propose an extension of the sifting heuristic which computes a complete multi-circular layout and considers edge crossings for optimizing both vertex order and edge winding, and thus is expected to generate better layouts.

Recall from Section 7.1.4 the basic idea of sifting. A vertex is moved along the fixed order of all other vertices. For each position, the number of crossings between the edges incident to the vertex under consideration and all other edges is computed. Finally, the vertex is placed in its (locally) optimal position. In multi-circular layouts, the number of crossings depends on both the vertex order and the edge winding. Therefore, we have to find for each position of a vertex the winding values for its incident edges which result in the minimal crossing number.

The efficient computation of crossing numbers in sifting for layered and single circular layouts is based on the locality of crossing changes, i. e., swapping consecutive vertices $u \curvearrowright v$ only affects crossings between edges incident to u with edges incident to v. In multi-circular layouts this property clearly holds for intra-partition edges since they form (single-)circular layouts. For inter-partition edges the best routing path may require an update of the windings. Such a change can affect crossings with all edges incident to the involved partitions.

Since swapping the positions of two consecutive vertices (and keeping the winding values) only affects incident edges, the resulting change in the number of crossings can be computed efficiently. Therefore, we need an efficient update strategy for

Algorithm 3: Multi-Circular Sifting

```
for ($0 \leq i < k$) do
    for ($u \in V_i$) do
        let $n = |V_i|$;
        let $v_0 = u \prec_u v_1 \prec_u \ldots \prec_u v_{n-1}$ denote the current layout;
        // set initial windings to 1
        for ($\{u, v\} \in E(u)$) do
            $\phi_u(e) \leftarrow 1$, $\phi_v(e) \leftarrow 1$;
        // set counters for position and crossing number
        $p^* \leftarrow 0$, $c^* \leftarrow c \leftarrow 0$, initialize counters for parting;
        // find best position and corresponding winding for u
        for ($p \leftarrow 0, \ldots, n - 1$) do
            repeat
                improve parting of source partition according to step 1
            until no crossing reduction ;
            repeat
                improve parting of target partition according to step 2
            until no crossing reduction ;
            repeat
                improve parting of target partition according to step 3
            until no crossing reduction ;
            $c \leftarrow$ c + crossing number change;
            // store best position
            if ($c < c^*$) then
                $p^* \leftarrow p$, $c^* \leftarrow c$, store counters for parting;
            // swap vertices u and $u_{p+1}$
            $c \leftarrow c - c(u, v)$;
            $\pi(u) \leftarrow p + 1$, $\pi(v_{p+1}) \leftarrow p$;
            $c \leftarrow c + c(v, u)$;
        move $u$ to position $p^*$;
        apply edge windings according to counters for parting;
```

edge windings while $u \in V_i$ moves along the circle. We do not consider each possible combination of windings for each position of u but keep track of the parting of the edges. Note that we have to simultaneously alter the parting for the source partition and all the partings for the target partitions because for an edge, a changed winding in the source partition may allow a better routing with changed winding in the target partition. Intuitively speaking, the parting in the source partition should move around the circle in the same direction as u but on the opposite side of the circle, while the parting in the target partitions should move in the opposite direction. Otherwise, edge lengths increase and with them the likelihood of crossings. Thus, we start with winding values $\psi_u(e) = 1$ and $\psi_v(e) = 1$ for all $e = \{u, v\} \in E(v)$ and iteratively move parting counters around the circles and mostly decrease this values in the following way:

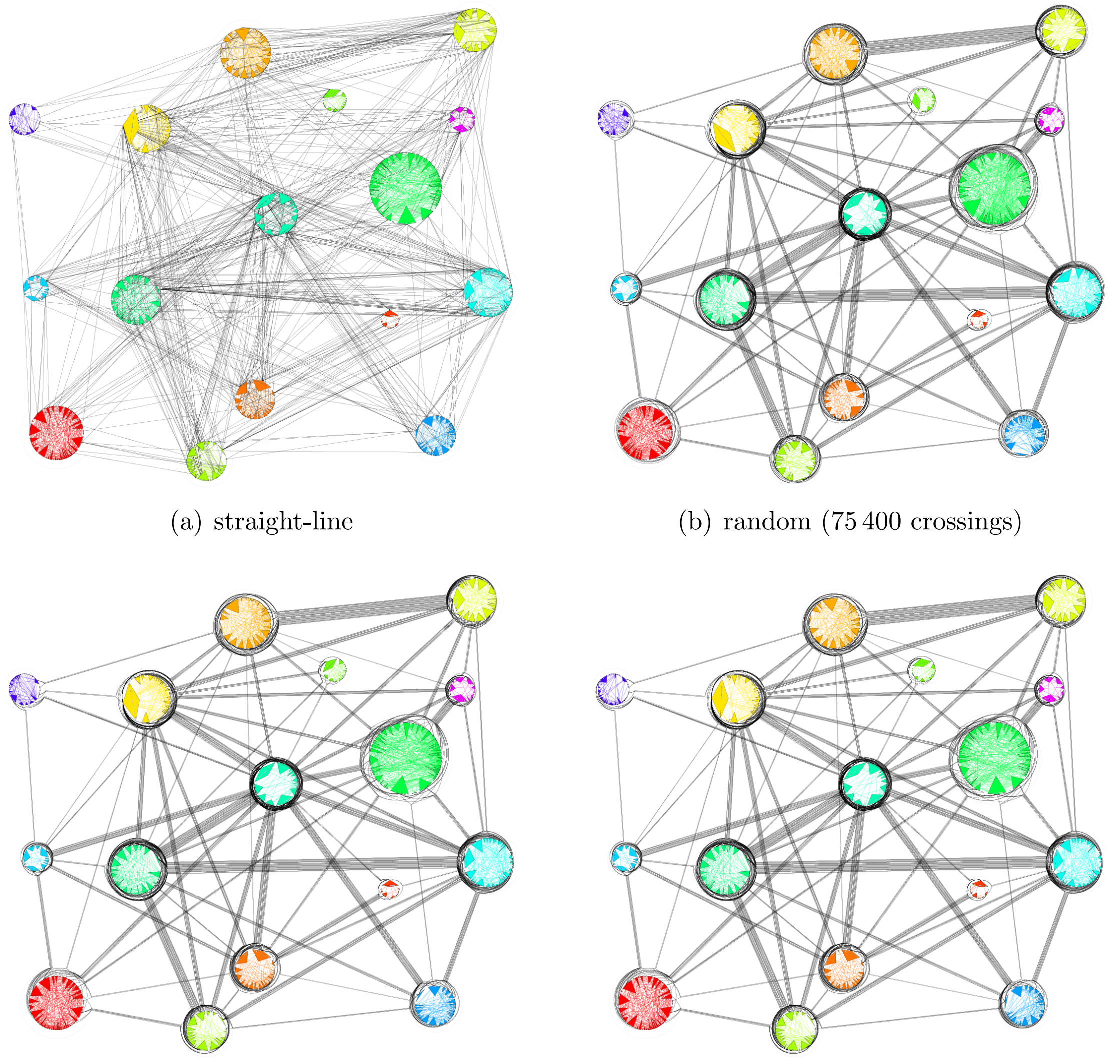
1. First try to improve the parting at V_i, i. e., iteratively, the value of ψ_u for the current parting edge is decreased and the parting moves counter-clockwise to the next edge until this parting can no longer be improved.
2. For edges whose source winding are changed in step one, there may be better target windings which cannot be found in step three because the value of ψ_j has to be increased, i. e., for each affected edge, the value of ψ_j for the edge is increased until no improvement is made any more.
3. Finally try to improve the parting for each target partition V_j separately, i. e., for each V_j, the value of ψ_j for the current parting edge is decreased and the parting moves clockwise to the next edge until this parting can not be improved any further.

After each update, we ensure that all counters are valid and that winding values are never increased above 1 and below -1. Algorithm 3 gives an overview of this procedure.

Based on the above, the locally optimal position of a single vertex can be found by iteratively swapping the vertex with its neighbor and updating the edge winding while keeping track of the change in crossing number. After the vertex has passed each position, it is placed where the intermediary crossing counts reached their minimum. Repositioning each vertex once in this way is called a *round of sifting.*

Theorem 7.11 *The running time of multi-circular sifting is in $\mathcal{O}(|V||E|^2)$.*

Proof Computing the difference in cross-count after swapping two vertices requires $\mathcal{O}(|E|^2)$ running time for one round of sifting. For each edge, the winding changes only a constant number of times because values are bounded, source winding and target winding are decreased in steps one and three, respectively, and the target winding is only increased for edges whose source winding decreased before. Counting the crossings of an edge after changing its winding takes time $\mathcal{O}(|E|)$ in the worst-case. Actually, only edges incident to the at most two involved macro-vertices have to be considered. For each vertex $u \in V$, the windings are updated $\mathcal{O}(|V| \cdot \deg(u))$ times, once per position and once per shifted parting. For one round, this results in $\mathcal{O}(|V||E|)$ winding changes. Together, the running time is in $\mathcal{O}(|V||E|^2)$. $\square$


(a) straight-line

(b) random (75 400 crossings)

(c) barycenter (68 300 crossings)

(d) sifting (57 400 crossings)

Figure 7.10: Multi-circular layouts of the email network

Short Evaluation

Here, we give a primary comparison of the presented crossing reduction techniques. A detailed discussion of the benefits of the multi-circular drawing convention is given in Section 6.6.2. Again, our test-case is the email communication network of a department of the Universität Karlsruhe (TH) consisting of 442 vertices and 2 201 edges at the micro-level and 16 vertices and 66 edges at the macro-level.

To investigate the effect of improved vertex orders and appropriate edge windings, we compare two variations of multi-circular layouts: shortest-path edge winding combined with random vertex placement and with barycenter vertex placement, see Figure 7.10. The macro-structure of the graph is apparent at first sight. Since the placement of the vertex circles is the same as in Figure 7.10 (a), this improvement clearly follows from the grouping of micro-edges. A closer look reveals the drawback of random placement: edges between different groups have to cover a long distance around the vertex circles and are hard to follow. Also a lot of edge crossings are generated both inside of the groups and in the area around the vertex placement circles. Assigning vertex positions according to the barycenter heuristic results in a clearly visible improvement and allows the differentiation of some of the micro-edges. Using sifting improves the layout even further, resulting from a decrease of the number of crossings from more than 75 000 to 57 400 in the considered email network. The time for computing the layout of this quiet large graph is below 10 seconds.

7.2.6 Conclusion

We have presented a drawing convention for micro/macro graphs where micro-level elements are drawn on top of the elements of the coarse macro-graph so that the contribution of micro-level elements to macro-level structure becomes apparent. Since there is no need to place restrictions on the layout of the macro-graph, we assumed it is given and focused on layouts of the micro-graph. Furthermore, we have presented a multi-circular layout model and investigated layout strategies based on crossing reduction techniques for it.

Backed by the visualizations of the email communication network computed by an initial implementation of our algorithms, we claim that the grouping of micro-edges into macro-edges according to the micro/macro drawing convention exhibits benefits over layouts which group the vertices. Furthermore, since vertex orders and edge windings have a large effect on the readability of multi-circular layouts, it is justified to spend a larger effort to improve them.

A major benefit of the multi-circular layout is its combinatorial description since it allows the combination with other visualization techniques to highlight some graph properties or to further improve the visual appearance. A very interesting aspect would be the combination with Holten's [Hol06] edge bundling technique.

7.3 Visualizing Hierarchies in Two and a Half Dimensions

In this part, we propose a method for the visualization of hierarchical information in a 2.5D graph layout. The levels of the hierarchy are represented by 2D layouts whose interdependence for increasing height is displayed by the third dimension. Initially, a layout is chosen for the maximum level thus emphasizing on the most important part of the hierarchy. The lower levels are added iteratively by force-based methods.

In this study, we consider the core hierarchy of the AS network but in principle, any network and hierarchy can be used. In contrast to alternative approaches to visualize AS network data, our method illustrates the entire AS network structure. Moreover, it is generic with regard to the hierarchy displayed by the third dimension.

7.3.1 Introduction

Current research activities in computer science and physics are aiming at understanding the dynamic evolution of large and complex networks like the physical Internet, World Wide Web, peer-to-peer systems and the relation between Autonomous Systems (AS). The design of adequate visualization methods for such networks is an important part of this research. As these graphs are on one hand large or even huge, on the other hand evolving, customized visualizations concentrating on their intrinsic structural characteristics are required.

For a given hierarchy, our method first obtains a two-dimensional spectral layout to display the maximum level and then iteratively adds the lower cores by force-based methods. Using 2.5D graph visualization, we then represent the hierarchy by stacking the induced 2D layouts of the levels in increasing height on top of each other in the third dimension.

For the demonstration of our method, we have chosen the AS network which recently attracted much attention in the complex systems community. A few samples of visualizations of AS networks are already available. However, they either focus on the geographic location of the AS [CAI03], on the routing structure seen from a selected AS [DBMPP04, CAI02], or on a high level view created by clustering the vertices [SW04]. In contrast, our method displays the entire AS network structure without using external information.

Previous attempts to analyze the structure of the AS network propose the existence of meaningful central vertices that are highly connected to a large fraction of the graph [GP04]. It seems that this structural peculiarity is interpreted very well by the notion of k-cores [Sei83, BZ02b]. This concept is already rudimentarily used for initial cleaning in [GMZ03]. Accordingly, our approach is based on the hierarchical core decomposition of the AS network.

Other visualizations of network data in 2.5D have been proposed recently, for example to display other graph hierarchies [BDS04, EF97] or evolving graphs over time [BC03a].

The new 2.5D visualization method for AS networks is explained in Section 7.3.3. In Section 7.3.4 we present and discuss the results obtained for various AS network data sets and Section 7.3.5 gives the conclusions.

7.3.2 Preliminaries

Let $G = (V, E)$ be an undirected, connected graph and let $V = V_0 \supset V_1 \supset \ldots \supset V_k \neq \emptyset$ be a given hierarchy of the vertices of G with levels V_i. A vertex has *levelness* i if it belongs to the level i but not to the level $(i+1)$. We call the collection of all vertices having levelness i the *i-shell*. An edge $\{u, v\}$ is an *intra-shell edge* if both u and v have the same levelness, otherwise it is an *inter-shell edge*. For our algorithm, we assume that the induced subgraph $G[V_i]$ of every level i, $0 \leq i \leq k$, is connected as well.

Recall from Section 4.5.1 that the k-core decomposition yields a hierarchy. The *i-core* of a graph is defined as the unique subgraph obtained by iteratively removing all vertices of degree less than i. In general, the core decomposition can result in disconnected levels but for the AS network, all i-cores stay connected.

7.3.3 Layout Method

Before describing our layout method, we state out layout paradigm. Then, we introduce a generic method to generate a 2.5D layout of a hierarchical decomposition of the graph based on a $2D$ layout. Finally, we describe how to set parameters to fulfill the requirements induced by the specific structure of AS networks.

Layout Paradigm Visualizations of hierarchies are often based on the abstraction to the levels of the hierarchy. However, this abstraction is typically accompanied by a loss of information that should be avoided. Therefore, we establish the following layout paradigm:

1. all vertices and edges are displayed,
2. the levels of the hierarchy are emphasized, and
3. inter- and intra-shell connections are made clear.

Layout Algorithm The first step of the algorithm constructs a spectral layout for the highest level of the hierarchy. Then, iteratively, the lower levels are added using a combination of barycentric and force-directed placement. Algorithm 4 gives a formal description of this procedure based on the core hierarchy.

Studies indicate that a spectral placement does not lead to a satisfactory layout of the AS network as a whole [Gae07]. However, the results improve for increasing core value. We therefore choose a spectral layout as initial placement for the core

Algorithm 4: Generic AS layout algorithm

```
Input: graph $G = (V, E)$
let $k \leftarrow$ maximum coreness, $G_l \leftarrow$ the $l$-core, $C_l \leftarrow$ $l$-core layer
calculate spectral layout for $G_k$
for $l \leftarrow k-1, \dots, 1$ do
    if $C_l \neq \emptyset$ then
        calculate barycentric layout for $C_l$ in $G_l$, keeping $G_{l+1}$ fixed
        calculate force-directed layout for $C_l$ in $G_l$, keeping $G_{l+1}$ fixed
        calculate force-directed layout for $G_l$
```

of the graph. Then, for the iterative addition of the other level of hierarchy, we first calculate a barycentric placement in which all new vertices are placed in the barycenter of their neighbors in this level.

Unfortunately, barycentric layouts also have a number of drawbacks. Firstly, vertices that are structurally equivalent in the current subgraph are assigned to the same position. Secondly, all vertices are placed inside the convex hull of the already positioned vertices. In particular this means that the outermost placed vertices are those in the highest level which is clearly contradictory to the intuition of importance. To overcome these difficulties, we use the barycentric layout as an initial placement for a subsequent force-directed refinement step, where only newly added vertices are displaced. In addition, a force-directed approach is applied for all vertices in order to relax the whole graph layout. However, the number of iterations and the maximal movement of the vertices is carefully restricted not to destroy the previously computed layout. A special feature of this relaxation step is the use of non-uniform preferred spring lengths $l(u, v)$, where $l(u, v)$ scales with the smaller core value of the two incident vertices u and v. Thus, the effect of a barycentric layout is modeled since edges between vertices of high coreness are longer than edges between vertices of low coreness. Accordingly, these springs prevent vertices with high coreness from drifting into the center of the layout.

Fitting the Parameters Beside the choice of the hierarchical decomposition, the algorithm offers a few more degrees of freedom that allow an adjustment to a broad range of applications. Our choice of parameters originates from the core structure of the AS network. For the spectral layout we propose a modified Laplacian matrix $L' = 1/4 \cdot D - A$ [BC03b]. Our experiments showed that the normalized adjacency matrix results in comparably good layouts while the standard Laplacian matrix performs significantly worse.

The force-directed placement is computed by a variant of the algorithm from Fruchterman and Reingold [FR91]. Unlike the original algorithm, we calculate the displacement only for one vertex at a time and update its position immediately. Furthermore, we use the original forces but with non-uniform preferred edge lengths $l(u, v)$ proportional to the square of $\min\{\text{level}(u), \text{level}(v)\}$. For the local refinement step we perform at most 50 iterations and for the global roughly 20 iterations.

7.3.4 Results

We illustrate the results of our method for real AS data sets as well as for generated graphs. For a more detailed discussion, we also refer to [BBGW04]. The section is concluded by techniques to aid the human perception.

Our real world data consist of three AS networks collected by the Oregon Routeview Project [Rou] on June 1st in 2001, 2002, and 2003, respectively. The sizes of these networks are given in Table 7.1. In addition, we used the Internet topology generator INET 3.0 [WJ02] to create artificial graphs that should exhibit a similar topology.

	AS 2001-06-01	AS 2002-06-01	AS 2003-06-01
Number of Nodes	11 211	13 315	15 415
Number of Edges	23 689	27 703	34 716
Core Number	19	20	25

Table 7.1: Sizes of the AS network snapshots.

We discuss two different two-dimensional types of figures, the 2D layout produced by Algorithm 4 and the projection of the 2.5D layout into one of the full dimensions, also referred to as *level projection.* Vertices are represented by ellipses of size decreasing with coreness and with colors fading from black to white. Edges are always drawn as straight lines.

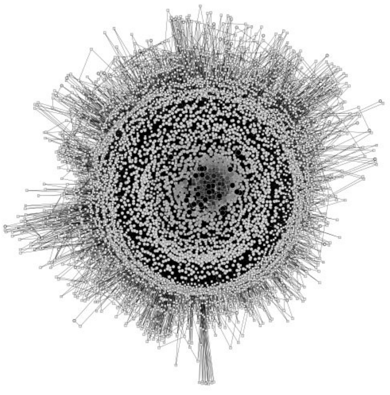
AS Network The 2D layouts are dominated by the vertices with small coreness leading to a huge periphery. This is reflected in Figures 7.11(a), 7.11(c), and 7.11(e) that show a large amount of small and bright vertices in the outer regions. On the other hand, most vertices with higher coreness are contained in the convex hull of the core, which is apparent in Figures 7.11(b), 7.11(d), and 7.11(f).

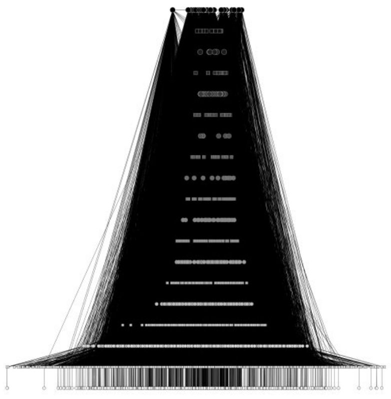
A closer examination reveals three almost separated radial areas around the center. The first one mainly contains the 3-core shell, while the 2-core shell forms the second and third area that are distinguished by their density (see Figure 7.12(a)-7.12(c)). This reflects the heterogenous importance distribution within these areas. In contrast, a large part of the 1-core shell is attracted to the central region. These properties can be observed for all three instances. The well-known growth of the AS network affects especially the 2- and 3-core shells. We observe that the spatial distances of these two shells decrease over time.

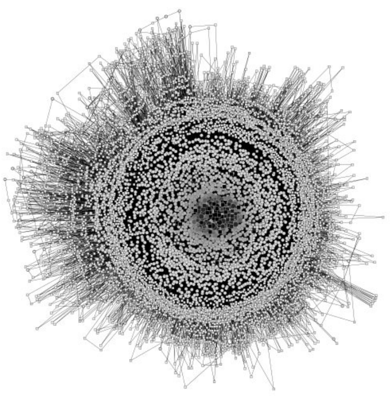
Generated Graphs There are significant differences of the generated graphs to the real AS networks, e. g., in the number of edges (35 300 vs. 23 700) and core levels (8 vs. 19), a finding that is confirmed by a later study on graph generators (see Section 8.2.3). An obvious difference of the generated graphs is the more uniform distribution of cardinalities of the core shells (Figure 7.13). Accordingly, the separation of the different core shells is less visible in the layout.

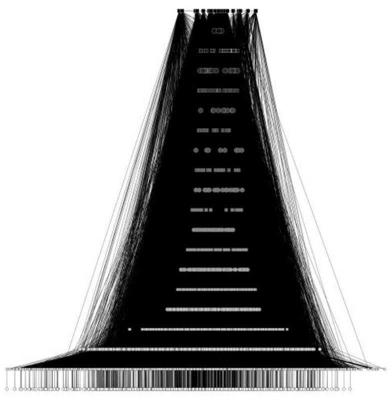
Supporting Perception There are several means for visual aid in 2.5D layouts, i. e., choice of perspective (in 3D), additional geometric objects emphasizing the levels of hierarchy, and colors. The choice of perspective is very powerful. We have already used this feature when presenting only the 2D layout and the level projection, respectively. More general, a user can focus on individual aspects, i. e., a global oriented view, a hierarchical version, or a mixture of both. A beneficial consequence might be that unintended information is automatically masked out by the perspective.

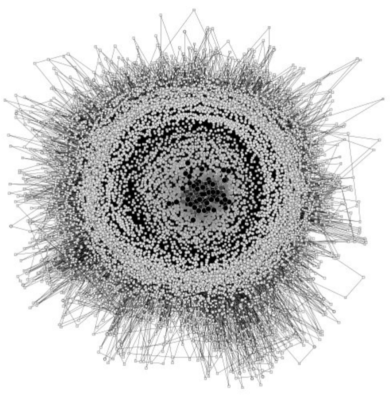
In order to simplify navigation in the three dimensional space, one can also introduce additional objects that mark the levels of hierarchy, i. e., rectangles, discs, or planes.


(a) 2D layout (2001)


(b) level projection (2001)


(c) 2D layout (2002)


(d) level projection (2002)


(e) 2D layout (2003)

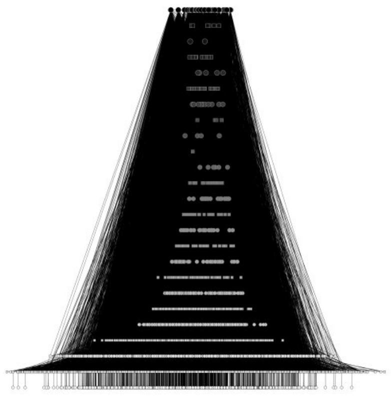

(f) level projection (2003)

Figure 7.11: 2D layout and level projection of the AS network in 2001, 2002, and 2003.

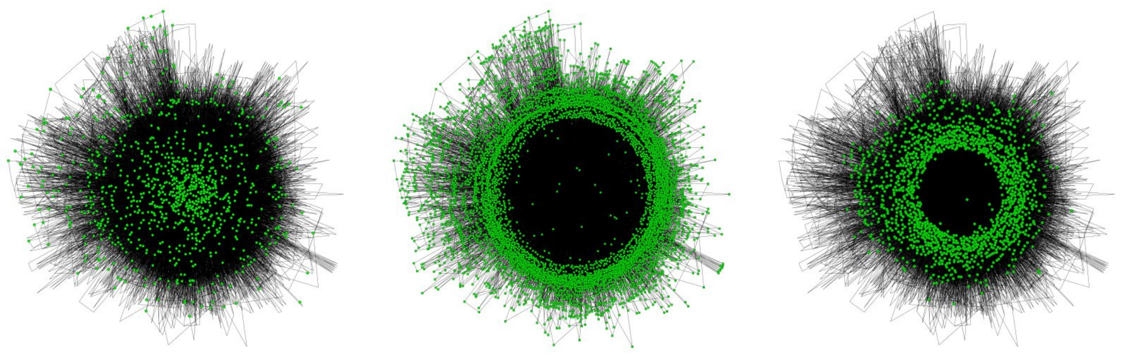

(a) 1-core shell only (b) 2-core shell only (c) 3-core shell only

Figure 7.12: 2D layout and level projection of the AS network 2002-06-01.

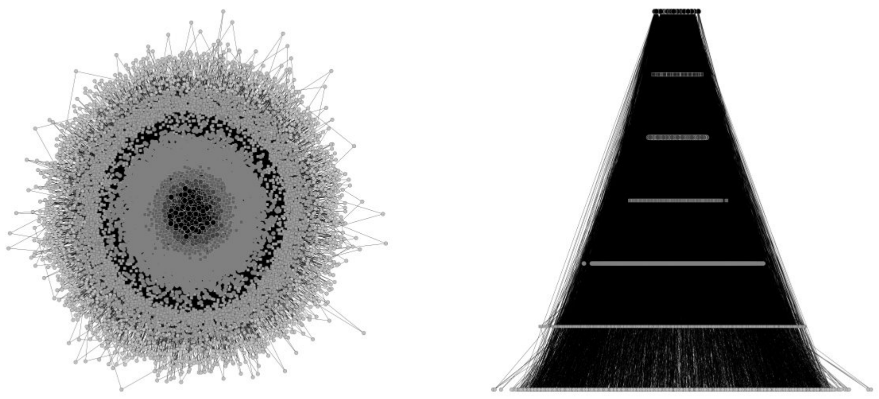

(a) 2D layout (b) level projection

Figure 7.13: Layouts of the generated graph for June 1st, 2002 with 11 211 vertices.

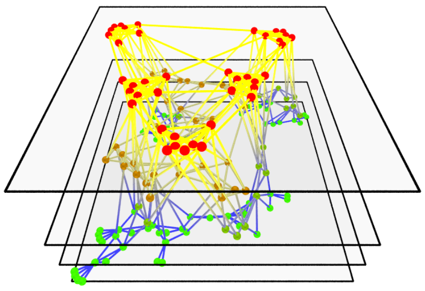
Transparency or filters might even increase their effectiveness. Color can be used in various ways, e. g., to highlight vertices and edges of special interest, to code the levels of hierarchy, or to improve the overall perception.

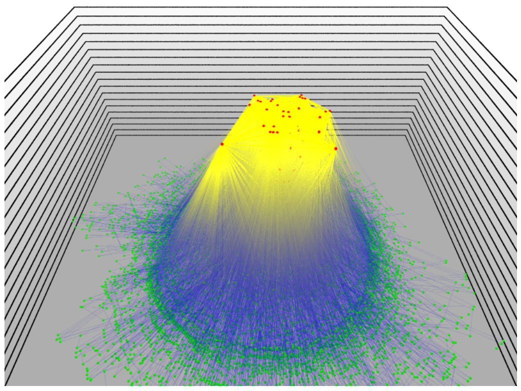
We used semi-transparent rectangles to mark levels and colored the vertices according to their coreness. The color of the edges are determined by a linear interpolation between the colors of their endvertices (see Figure 7.14).

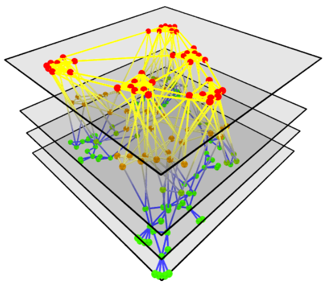
7.3.5 Conclusion

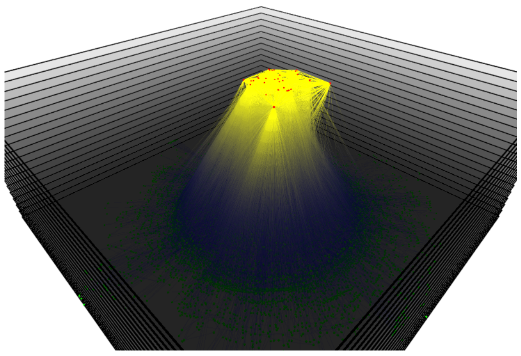
Representing the hierarchy of a network on levels of increasing height is a suggestive and obvious visualization method. We have introduced a layout paradigm for drawings in two and a half dimensions which implements this idea and overcomes drawbacks inherent to common methods like barycenter placement.

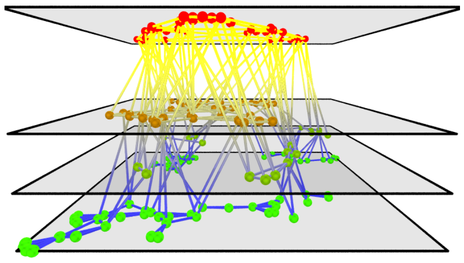
For the k-core decomposition of the AS network, we have explicated how our visualization supports the recognition of the details of the hierarchy. Especially, it


(a) view from above, light levels


(b) view from above, light levels


(c) diagonal view from above


(d) diagonal view from above


(e) horizontal view to one side

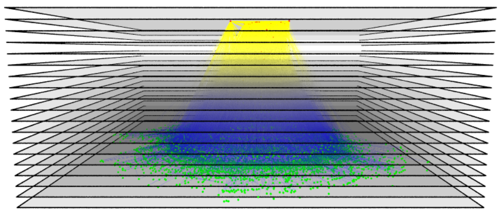

(f) horizontal view to one side

Figure 7.14: Drawings of a small example and the AS network of 2001 using graphical features to support perception. The level of the vertices and edges are marked by colors.

emphasizes the characteristics of the lower core shells and their connections with the highest shells. Furthermore, we have been able to recover known effects of the evolution of the AS network in the drawings and to distinguish generated and real AS networks by their specific, observable effects on the layout.

Chapter 8

Network Models

The first models for the generation of random graphs by Gilbert [Gil59] and Erdős and Rényi [ER59] created edges entirely uniformly at random. While such graphs have a small average distance over all vertices, real-world networks often exhibit significant other characteristics not obtainable by these models, e. g., a high local density or a specific degree distribution. Therefore, a plethora of models for random graphs was proposed thereafter. These procedures model the natural growth of networks by means of iteratively adding vertices, geometric positioning information, a definition of link connectivity based on the preference for nearest neighbors or already highly connected vertices, or combine several of these approaches.

In Section 8.1 we present some of the most prominent and fundamental models for the generation of random networks, namely $\mathcal{G}(n,p)$ [Gil59], small world [WS98], and preferential attachment [BA99]. These three methods are implemented in visone using the efficient linear time algorithms of Batagelj and Brandes [BB05].

In Section 8.2 we introduce our novel model that brings together the well-know concepts of k-cores [Sei83, BZ02b] and of preferential attachment. Recent studies exposed the significant k-core structure of several real world systems, e.g., the AS network of the Internet. We present a simple and efficient method for generating networks which at the same time strictly adhere to the characteristics of a given k-core structure, called core fingerprint, and feature a power-law degree distribution. We showcase our algorithm in a comparative evaluation with two well-known AS network generators.

8.1 Fundamental Models

8.1.1 General Random Graphs $\mathcal{G}(n,p)$

One of the first models for random graphs was introduced by Gilbert in 1959 [Gil59]. In his model $\mathcal{G}(n,p)$, the $n(n-1)/2$ potential edges of an undirected simple graph with n vertices are included independently with probability $0 < p < 1$. The probability p is usually chosen depending on the number of vertices n since for a fixed edge probability p the expected number of edges is proportional to n^2.

Shortly after Gilbert, Erdős and Rényi [ER59] introduced a closely related model $\mathcal{G}(n,m)$, in which all simple undirected non-isomorphic graphs with n vertices and exactly m edges are equally probable. For $m = pn(n-1)/2$ these two models stochastically converge as $n \to \infty$.

Efficient Implementation The direct implementation of the model $\mathcal{G}(n,p)$ – independently for each candidate edge draw uniformly at random a number $r \in (0,1)$ and insert the edge if $r < p$ – yields a running time in $\Theta(n^2)$, regardless of the number of edges actually generated. For small values of p, i. e., for the generation of sparse graphs, this method is not suitable since most trials will be unsuccessful.

In visone we therefore use the algorithm proposed by Batagelj and Brandes [BB05] which samples the waiting time, i. e., the number of rejected edges until the next accepted edge is found instead of drawing edge probabilities. Using the geometric method [FMR62], the running time for generating a graph with n vertices and m edges is in $\mathcal{O}(n+m)$, which is optimal.

Algorithm 5: $\mathcal{G}(n,p)$

```
**Input**: number of vertices $n$, probability $0 < p < 1$
**Output**: undirected graph $G = (V,E)$
$V \leftarrow \{0,\ldots,n-1\}$; $E \leftarrow \emptyset$;
$v \leftarrow 1$; $w \leftarrow -1$;
while $v < n$ do
    $r \leftarrow$ draw from $[0,1)$ uniformly at random;
    $w \leftarrow w + 1\lfloor \log(1-r)/\log(1-p) \rfloor$;
    while $w \leq v$ and $v < n$ do
        $w \leftarrow w - v$;
        $v \leftarrow v + 1$;
    if $v < n$ then
        $E \leftarrow E \cup \{\{v,w\}\}$;
```

The geometric method requires an enumeration of the candidate edges, for example in lexicographical order. In each step, the probability for accepting the next edge after k trials is

$$q(k) = (1-p)^{k-1}p \, ,$$

i. e., waiting times are geometrically distributed. In order to sample waiting times, each positive integer k is assigned an interval I_k of length $q(k)$ and these intervals are ordered consecutively starting at 0. Therefore, we have

$$\sum_{k=1}^{\infty} q(k) = 1$$

and the interval I_k ends at $1-(1-p)^k$. In each step, we accept the k-th next edge for which a randomly chosen $r \in [0,1)$ is contained in I_k, i. e., we choose $k = 1 + \lfloor \log(1-r)/\log(1-p) \rfloor$, since $r < 1-q^k \Leftrightarrow k > \log(1-r)/\log(1-p)$.

The pseudocode in Algorithm 5 implements a row-wise traversal of the lower half of the adjacency matrix which corresponds to an enumeration of undirected candidate edges in lexicographical order. The algorithm is easily modified to generate other classes of graphs by enumerating different sets of candidate edges. In visone we use a row-wise traversal of both halfs of the adjacency matrix to generate directed graphs. Furthermore, we take care of the cases $p \in \{0,1\}$, which are omitted in the algorithm, since the term $\log(1-r)/\log(1-p)$ is undefined for $p \in \{0,1\}$.

8.1.2 Small World

The defining characteristics of a *small world* networks are a small average distance over all vertices combined with a high local density. Intuitively speaking, a small world shows a distinct local clustering but some shortcuts connect very different parts of the network. More precisely, in a small world the average shortest-path distance grows only logarithmically with the number of vertices while the clustering coefficient (see Section 4.4) is high.

One of the first studies highlighting short average distance in a large social network is the famous experiment by Milgram [Mil67]. He asked several people in the US to deliver a message by passing it on only to people they knew personally. The initial senders knew only the name, the locality, and the profession of the recipients. The letters which reached their destination traveled on average over six mediators but only around 20% of them reached their target at all. Nevertheless, the popular claim of six degrees of vicinity was coined. In a recent and more elaborate study, Leskovec and Horvitz [LH08] found the average distance to be 6.6 in the communication network of the MSN instant messenger community, which consisted of 180 million vertices and 1.3 billion undirected edges at the time of the study in June 2006.

Many types of relationship induce graphs in which there are many direct connections between the neighbors of a vertex resulting in a high clustering coefficient. For example, in networks of acquaintance like the one considered in Milgram's experiment and the MSN instant messenger network it is likely that people one knows personally also know each other. In contrast, graphs following the model $\mathcal{G}(n,p)$ indeed show a small average distance even for small values of p whereas the clustering coefficient is small except for very high values of p.

The popular model of Watts and Strogatz [WS98] for undirected small world graphs starts at a fixed graph with a distinct local clustering and rewires a small fraction of the edges to generate shortcuts. More precisely, the initial graph C_n^k is the k-th power of a n-cycle, i. e., a cycle of n vertices in which each vertex is connected to its k neighbors to the left and to its k neighbors to the right. Then, each of the nk edges is rewired with probability p by replacing one of its endvertices by a randomly selected target.

Efficient Implementation The straight-forward implementation of this model creates in linear time an n-cycle and decides for each edge whether to rewire it or not. This method may generate loops and multi-edges. Furthermore, small values of probability p result in a large number of unsuccessful random trials. Nevertheless, since each created edge is tested only once, the asymptotic running time is still in $\mathcal{O}(n+m)$ and the few expected loops and multi-edges have an insignificant effect on the properties of the graphs. Algorithm 6 gives the pseudocode for this simple method. Batagelj and Brandes [BB05] propose an extention based on maintaining a list of candidate vertices for the replacement of an endvertex of an edge which goes without unsuccessful trials and avoids loops and multi-edges.

Algorithm 6: Small World

```
Input: number of vertices $n$, number of neighbors $1 \leq k \leq \lfloor (n-1)/2 \rfloor$,
       replacement probability $0 < p \leq 1$
Output: undirected small world $G = (V, E)$

$V \leftarrow \{0, \dots, n-1\}$; $E \leftarrow \emptyset$;
for $v \leftarrow 0$ to $n-1$ do
    for $w \leftarrow v+1$ to $v+k$ do
        $w \leftarrow w \mod n$;
        $r \leftarrow$ draw from $[0,1)$ uniformly at random;
        if $r < p/2$ then                          // replace 'target' vertex $w$
            $v' \leftarrow \lceil 2r(n-1)/p \rceil$;
            $E \leftarrow E \cup \{\{v, v+v' \mod n\}\}$;
        else if $r > 1-p/2$ then                   // replace 'source' vertex $v$
            $w' \leftarrow \lceil 2(1-r)(n-1)/p \rceil$;
            $E \leftarrow E \cup \{\{w, w+w' \mod n\}\}$;
        else
            $E \leftarrow E \cup \{\{v, w\}\}$ ;                  // no replacement
```

8.1.3 Preferential Attachment

In a number of real-world graphs some properties have been identified that are unlikely to emerge in the models presented so far, most notably a distribution of vertex degrees that roughly obeys a *power-law*, a fact that has been identified by Faloutsos et al. [FFF99]. More precisely, the number of vertices with degree d is proportional to $d^{-\gamma}$ for some constant γ. Graphs with this property are commonly referred to as *scale-free*. Barabási and Albert describe a growth process coined *preferential attachment* [AB02] that generates graphs with such a degree distribution. Starting out with an initial graph G_0, e. g., an empty graph, this process iteratively adds a new vertex that is adjacent to a fixed number d of already existing vertices. The choice of a specific neighbor is made with probability proportional to the current degree of the vertices.

This process is inspired by two typical facts of the evolution of social networks: growth and preferential attachment. Already highly connected vertices are likely to become even more connected, for example new web sites rather link to well-known sites like Wikipedia than to a small private page and popular people attract new relationships more easily – the so-called *"the rich get richer"*-phenomenon.

Efficient Implementation Based on the following observation, Batagelj and Brandes [BB05] present an efficient algorithm for the generation of scale-free graphs: in a list of all edges created so far the occurrence of each vertex is equal to its degree. Therefore, this list can be used to sample vertices according to the current degree distribution. In Algorithm 7 an array M containing the endvertices of each edge is used to implement this list. Both its running time and space requirement are in $\mathcal{O}(n+m)$. Note that the algorithm resolves the ambiguity of how to select a set of $d > 1$ neighbors in each step according to Bollobás's model [BRST01].

Algorithm 7: Preferential Attachment

```
Input: number of vertices $n$, minimum degree $d \geq 1$
Output: undirected scale-free multi-graph $G = (V, E)$
$V \leftarrow \{0, \dots, n-1\}$; $E \leftarrow \emptyset$;
array $M \leftarrow \emptyset$;
for $v \leftarrow 0$ to $n-1$ do
    for $i \leftarrow 0$ to $d-1$ do
        $j \leftarrow 2(vd+i)$;
        $M[j] \leftarrow v$;
        $r \leftarrow$ draw from $[0, \dots, j]$ uniformly at random;
        $M[j+1] \leftarrow M[r]$;
        $E \leftarrow E \cup \{\{v, M[r]\}\}$ ;   // note that $\{v, M[r]\} = \{M[j], M[j+1]\}$
```

For directed graphs, the minimum degree d can be specified separately for minimum in- and out-degree and the neighbor selection probability can be restricted to the in-, out-, or total degree distribution by sampling from even, odd, or all positions in M, respectively. In visone, d specifies the minimum out-degree and all positions are considered for sampling. In the original paper [BB05], a number of other modifications are described, e. g., an arbitrary initial graph G_0 can be used by filling M with the endvertices of the edges of G_0.

8.2 Augmenting k-Core Generation with Preferential Attachment

The interest in modeling specific classes of graphs has significantly increased by recent studies of complex systems such as the Internet, biological networks, river basins, or social networks. While random graphs have been studied for a long time, the standard models appear to be inappropriate because they do not share certain abstract characteristics observed for those systems. One of these characteristics is the k-core structure which can be interpreted as a nested decomposition separating parts of the network based on their density. This decomposition is commonly applied in order to identify central parts of the networks since it peels the network layer by layer, filtering out less important parts that are sparsely connected with the remaining graph. Example applications are network fingerprinting with LunarVis [GGW08] and LaNet-vi [AHDBV06], protein network analysis [WA05], or the exploration of modern social networks [DYNM06].

A crucial field of application of graph generators is the simulated evolution of a given network, granting insights in both its past development and its anticipated future behavior. One prominent example is the Internet at the Autonomous System (AS) level where various models have emerged over the last few years including BRITE [MLMB01], Inet [JCJ00], nem [Mag02], and various models presented by Pastor-Satorras and Vespignani [PSV04]. While this network has been observed to possess a very distinct k-core structure [AHDBV05, CHK$^+$07], kept track of over a long period of time, all generating tools so far ignore this structure, and thus largely fail to do justice to this significant and stable property [DGM06]. Overall, up to our knowledge an approach to create networks with a given k-core structure is missing so far.

To address this issue, we refine the abstract measurement of core sizes to a *core fingerprint* that additionally includes information on the inter-connectivity of each pair of shells. This allows us to design a simple and efficient method to incrementally generate randomized networks with a predefined k-core structure, starting with the maximum core. By utilizing two results on edge rewiring, we thus achieve a structure that precisely matches the core fingerprint.

Predefining the core fingerprint of a network still leaves many degrees of freedom open. Since we focus on the network of Autonomous Systems as a case study, we exploit this fact and optionally bias the randomness in the adjacency of vertices towards preferential attachment (see Section 8.1.3). This paradigm of setting up links in a network has been proven to introduce a power-law degree distribution, which has first been observed by Faloutsos et al. [FFF99] for the Internet. Our approach imposes almost no modifications on a vanilla realization of preferential attachment, a fact that is reflected by our experimental results. We thus manage to coalesce two of the most fundamental concepts in the theory of complex networks of the recent past.

The rest of this section is organized as follows: in the preliminaries we recall some definitions and state basic properties for k-core structures and on preferential attachment in Section 8.2.1, then we give the description of the network generator in Section 8.2.2. In Section 8.2.3 we evaluate our model in comparison to two well-known generators with respect to commonly used network properties. Finally, we give some concluding remarks.

8.2.1 Preliminaries

In this section, let $G = (V, E)$ be a simple, undirected graph. A subset $V' \subseteq V$ of the vertex set induces a subgraph $G[V'] = (V', E')$ where the edge set E' is defined by $E' = \{\{u, v\} \mid u, v \in V', \{u, v\} \in E\}$. A nested decomposition of G is a finite sequence $(V_0, \ldots, V_k)$ of subsets of vertices such that $V_0 = V$, $V_{i+1} \subseteq V_i$ for $i < k$, and $V_k \neq \emptyset$.

Cores, as introduced in Section 4.5.1, are a widely used realization of nested decompositions. Constructively speaking, the *i-core* of an undirected graph is defined as the unique subgraph obtained by iteratively removing all vertices of degree less than i. This procedural definition immediately gives rise to a construction algorithm that can easily be implemented. Moreover, it is equivalent to the closed definition of the i-core as the set of all vertices with at least i adjacencies to other vertices in the i-core. The core decomposition can be computed in linear time with respect to the graph size [BZ02a].

The *core number* of a graph is the smallest i such that the $(i + 1)$-core is empty, and the corresponding i-core is called the *core* of a graph. A vertex has *coreness i*, if it belongs to the i-core but not to the $(i + 1)$-core. We call the collection of all vertices having coreness i the *i-shell*. An edge $\{u, v\}$ is an *intra-shell edge* if both u and v have the same coreness, otherwise it is an *inter-shell edge*.

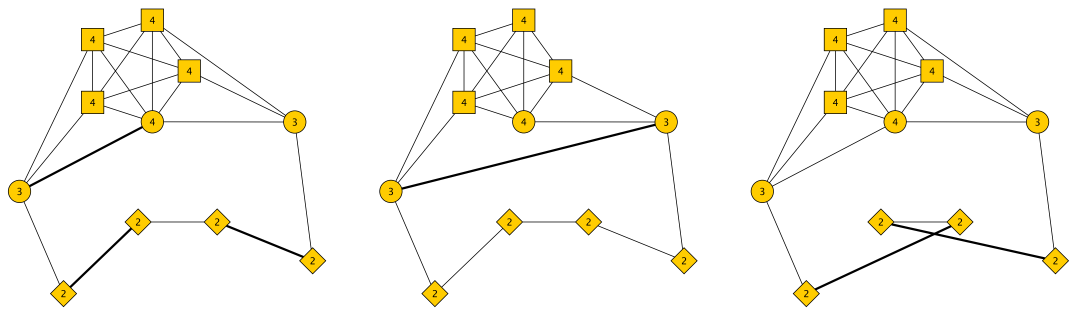
Edges in a Core Hierarchy

The following two lemmas summarize two facts about the relation of intra- and inter-shell edges. We later exploit this interaction and interchangeability of edges in our network generation algorithm.

Lemma 8.1 (Rewiring) *Let $G = (V, E)$ be a graph. Let $u, v \in V$ be two non-adjacent vertices with the same coreness and $\{u, w\}, \{v, w'\} \in E$ two edges such that* $\text{coreness}(u) < \min\{\text{coreness}(w), \text{coreness}(w')\}$. *Then $G' = (V, E')$ with $E' = E \setminus \{\{u, w\}, \{v, w'\}\} \cup \{u, v\}$ has the same core decomposition as G. Conversely, let $u, v \in V$ be two adjacent vertices with the same coreness k and with at most $k - 1$ neighbors in higher cores, and let $w, w' \in V$ be two vertices such that* $\text{coreness}(u) < \min\{\text{coreness}(w), \text{coreness}(w')\}$ *and $\{u, w\}, \{v, w'\} \notin E$. Then, $G'' = (V, E'')$ with $E'' = E \setminus \{u, v\} \cup \{\{u, w\}, \{v, w'\}\}$ has the same core decomposition as G.*

Lemma 8.2 (Swapping) *Let $G = (V, E)$ be a graph, $u, v, w, w' \in V$ be four vertices all having the same coreness, $\{u, v\}, \{w, w'\} \in E$ be two intra-shell edges, and $\{u, w\}, \{v, w'\} \notin E$. Then the graph $G' = (V, E')$ with $E' = E \setminus \{\{u, v\}, \{w, w'\}\} \cup \{\{u, w\}, \{v, w'\}\}$ has the same core decomposition as G.*

It is not hard to see that the correctness of both lemmas follows from the definition of cores. The cumbersome prerequisites can be understood more easily by the concept of a removal order that will be introduced later in Section 8.2.2. Informally speaking, Lemma 8.1 allows for most pairs of disconnected vertices of the same coreness to each



(a) original graph (b) after rewiring (c) after swapping

Figure 8.1: Rewiring and swapping edges in the left graph. The labels show the coreness of the vertices.

remove one edge to some vertices of higher coreness and instead become connected, and vice versa, without changing the decomposition. Furthermore, according to Lemma 8.2 we can swap the endvertices of intra-shell edges if this does not interfere with existing connections. Figure 8.1 illustrates these two lemmas for an example graph. Using these statements, we can now establish (tight) bounds of the sizes of cores and shells.

Lemma 8.3 (Size of i-Cores) *Let $G = (V, E)$ be a graph, $(V_0, \dots, V_k)$ its core decomposition and $G_i = (V_i, E_i) = G[V_i]$ the i-core. Then the size of every i-core is bounded as follows:*

$$i + 1 \leq |V_i| \quad \text{and} \quad \frac{(i+1)i}{2} \leq |E_i| \quad . \tag{8.1}$$

Let $n_i = |V_i \setminus V_{i+1}|$ be the number of vertices with coreness i and $m_i = |E_i \setminus E_{i+1}|$ the number of all edges whose endvertices with minimum coreness have coreness i for $0 \leq i \leq k$ (for convenience we define $V_{k+1} = \emptyset$ and $E_{k+1} = \emptyset$). Then the size of the i-shell is bounded as follows:

$$0 \leq \quad n_i \quad \leq |V| \tag{8.2}$$

$$\left.\begin{matrix} \left\lceil \frac{i \cdot |n_i|}{2} \right\rceil & , \textit{if } n_i > i \\ \binom{n_i}{2} + n_i \cdot (i - n_i + 1) & , \textit{if } n_i \leq i \end{matrix}\right\} \leq \quad m_i \quad \leq \begin{cases} i \cdot n_i & , \textit{if } i < k \\ i \cdot n_i - \frac{i^2+i}{2} & , \textit{if } i = k \end{cases} \tag{8.3}$$

Note that the bounds for the i-core (Eq. 8.1) are trivially obtained from the definition. The bounds for the i-shell (Eq. 8.2 and 8.3), however, use the above two lemmas, i. e., the shell has the minimum number of edges if it has the maximum possible number of intra-shell edges since each such edge contributes twice and a minimum number of inter-shell edges. An analogous reasoning yields the upper bounds. We omit proofs for the bounds of this lemma except of the following.

Proof [of Upper Bound 3] By definition, there exists a removal order σ that iteratively removes a vertex v from V_k with $\deg(v) \leq k$, such that eventually all vertices in V_k are removed. The maximum number of edges that still allow such an order of removal $\sigma(v)$, $v \in V_k$, is calculated by counting the maximum numbers of incident edges the removed vertices in such an removal order can have. For the first $n_k-(k+1)$ vertices (which can be zero), the removal order σ implies that the current vertex v can have a maximum degree of k. For the last $k+1$ vertices (minimum number of vertices for a k-shell) however, the number of incident edges during the removal order is even less, resulting in a $(k+1)$-clique supported by $(k^2+k)/2$ edges. Thus, there are

$$\underbrace{(n_k-(k+1))\cdot k}_{\text{by vertices beyond } k+1} + \underbrace{\frac{(k+1)\cdot k}{2}}_{\text{by clique of last } k+1 \text{ vertices}} = k\cdot n_k - \frac{k^2+k}{2} \tag{8.4}$$

edges in total, which proves the bound. It is easy to see that this bound is sharp, since our arguments induce a straight-forward construction.

Note that this bound also applies to lower shells when excluding edges to higher shells. □

8.2.2 Core Generator

In this section, we first introduce a set of relevant parameters for the construction of core structures and discuss which combinations of these lead to feasible instances, i. e., are capable of realizing a graph with a predefined core structure. Then, we describe our basic algorithm for generating such graphs and point out several variations.

As the 0-shell only contains isolated vertices and in order to reduce technical peculiarities, we restrict ourselves to generating graphs with an empty 0-shell.

Input Parameters

There are several possibilities to specify core structures. The most obvious quantitative approach is to give the number of vertices per shell, the number of intra-shell edges, and the number of inter-shell edges (for each pair of shells). This can be coded by a combination of a vector $N \in \mathbb{N}_0^k$, where n_i is the number of vertices in the i-shell, and a symmetric matrix $M \in \mathbb{N}_0^{k\times k}$, where $m_{i,j}$ contains the number of edges connecting the i-shell with the j-shell. We call this the core fingerprint. For example, the graph (omitting isolated vertices) given in Figure 4.4 has the following fingerprint:

$$N = (4,3,2,5) \quad \text{and} \quad M = \begin{pmatrix} 3 & 1 & 0 & 0 \\ 1 & 2 & 2 & 0 \\ 0 & 2 & 0 & 6 \\ 0 & 0 & 6 & 10 \end{pmatrix}$$

Clearly, the implied sizes of the shells have to respect the bounds established in Lemma 8.3. This kind of specification of core structures provides the maximum degree of freedom, i. e., the user can configure the size distribution of each shell and is only limited by constraints ensuring consistency.

One can easily relax the requirement of absolute values in the input by replacing them by parameters that correspond to the ratio of edges with respect to the tight bounds established in Equation 8.3. To further simplify the structure of the input, these ratios could be replaced by a density function. Such a function could, e.g., follow a simple power-law.

Algorithmic Approach

Our generator builds a graph by iteratively adding new shells beginning at the maximum core. When adding a new shell, we create vertices and edges according to the given core fingerprint and take care to not change the coreness of vertices in previously built higher shells. The detailed pseudocode is given in Algorithm 8. We omit in-depth explanations of supplementary operations such as APPENDALL or REMOVEALLOCCURENCES since these methods have one-to-one equivalences in most high-level programming languages.

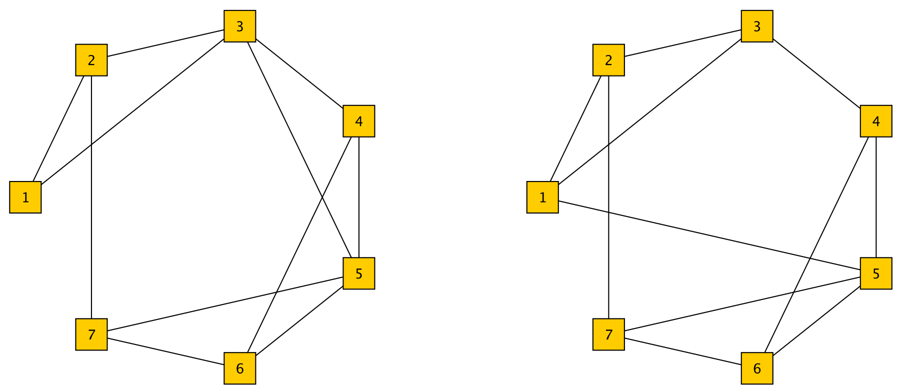
In order to guarantee that the coreness of vertices in the i-shell will not exceed i, we define an order σ_i which will be maintained as a valid removal order for this shell (line 4). It is of vital importance to ensure that for every vertex in V_i the sum of the number of neighbors in the shell i with a higher value of σ_i and the number of neighbors in higher shells does not exceed i. To model this, newly created edges are directed such that inter-shell edges point from the lower shell to the higher shell and intra-shell edges are directed in accordance to our predefined order σ_i and each vertex in V_i is restricted to a maximum out-degree of i (line 18). We are left to guarantee that the coreness is exactly i and not less. An example where this is not yet satisfied is given in Figure 8.2(a).

While lines 3 to 25, called the *element generation phase*, avoid erroneously high values of coreness, as further detailed below, the *rewiring phase* in lines 27 to 37 solves the problem of erroneously low values of coreness by a sophisticated movement of edges. We choose a vertex v with insufficient degree and a vertex w with degree greater than i (line 30). Then, we select a neighbor $c \in$ NEIGHBORS(w) which is not yet adjacent to v (lines 31 and 32) and replace this adjacency $\{w, c\}$ by a new edge $\{v, c\}$ (line 33).

Before we revisit the element generation phase in detail, we recapitulate the mechanism of preferential attachment . The network is grown from an arbitrary, small seed such as a single vertex or a triangle. Iteratively vertices are added and connected to a fixed number of neighbors. These neighbors are randomly selected from existing vertices with probability proportional to their degree. This behavior can be modeled by maintaining a list of vertices to which both endvertices of each newly inserted edge are appended. Thus, this list contains each vertex with multiplicity equal to its current degree. Drawing uniformly at random from this list is a legitimate and efficient realization of preferential attachment introduced by Batagelj and Brandes [BB05] (see also Section 8.1.3).

Algorithm 8: Core Generator

```
Input: integer $k$, vector $N \in \mathbb{N}_0^k$, valid symmetric matrix $M \in \mathbb{N}_0^{k\times k}$
Output: graph $G = (V, E)$
1  $V \leftarrow \emptyset$; $E \leftarrow \emptyset$; TARGETVERTICES $\leftarrow \emptyset$;
2  for $i \leftarrow k$ to $1$ do                                   // introduce next shell
3      list $V_i \leftarrow \{n_i$ new vertices$\}$;
4      $\sigma_i : V_i \rightarrow \{1, \ldots, n_i\}$ defined by $\sigma_i^{-1}(\ell) = V_i[\ell]$ ;         // removal order
5      $u \leftarrow V_i[n_i]$ ;                          // last vertex in removal order
6      list SOURCEVERTICES $\leftarrow V_i \setminus \{u\}$ ;   // $u$ cannot source intra-edges
7      list TARGETVERTICES$[i] \leftarrow \{u\}$ ;                      // $u$ into PA-list
8      list UNCONNECTABLE $\leftarrow \{u\}$ ;                            // see line 21
9      for $j \leftarrow i$ to $k$ do                               // select target shell
10         for $m \leftarrow 1$ to $m_{i,j}$ do                          // introduce $m_{ij}$ edges
11             $s \leftarrow$ SOURCEVERTICES[random] ;           // source of new edge
12             $C \leftarrow$ TARGETVERTICES$[j]$ ;           // target candidates list
13             $C$.REMOVEALLOCCURENCES(NEIGHBORS$(s) \cup \{s\}$);
14             if $j = i$ then                             // check removal order $\sigma$
15                 $C$.REMOVEALLOCCURENCES($\{\ell \in V_i \mid \sigma(\ell) < \sigma(s)\}$);
16             $t \leftarrow C$[random] ;                             // target of new edge
17             $E \leftarrow E \cup (s, t)$ ;
18             if $outdeg(s) = i$ then                             // source saturated
19                 SOURCEVERTICES.REMOVE($s$);
20             else if $j = i$ and $outdeg(s) \geq n_i - \sigma_i(s)$ then
21                 SOURCEVERTICES.REMOVE($s$) ;   // no more intra-targets
22                 UNCONNECTABLE.APPEND($s$) ; // store for inter-targets
23             TARGETVERTICES$[i]$.APPEND($s, t$);
24         if $j = i$ then
25             SOURCEVERTICES.APPENDALL(UNCONNECTABLE) ;   // restore
26     remove direction of edges;
27     list POORVERTICES $\leftarrow \{v \in V_i \mid \deg(v) < i\}$ ;
28     list RICHVERTICES $\leftarrow \{v \in V_i \mid \deg(v) > i\}$ ;
29     while POORVERTICES $\neq \emptyset$ do          // rewire unsaturated vertices
30         $v \leftarrow$ POORVERTICES[random]; $w \leftarrow$ RICHVERTICES[random];
31         $C \leftarrow$ NEIGHBORS$(w) \setminus$ NEIGHBORS$(v)$ ;          // pivot candidates
32         $c \leftarrow C$[random];
33         $E \leftarrow E \setminus \{\{w, c\}\} \cup \{\{v, c\}\}$;
34         if $deg(v) = i$ then                                       // $v$ saturated
35             POORVERTICES.remove($v$);
36         if $deg(w) = i$ then                               // $w$ no longer RICH
37             RICHVERTICES.remove($w$);
38     $V \leftarrow V \cup V_i$ ;                                  // shell $i$ completed
```



(a) before rewiring (b) after rewiring

Figure 8.2: Example of rewiring. The fingerprint $N = (0, 0, 7)$ and $m_{3,3} = 11$ resulted in the left hand graph. Clearly, vertex 1 has insufficient degree. In the rewiring phase we can choose either vertex 3 or 5 as the RICH vertex. For the right hand graph we selected vertex 3 and vertex 5 as the RICH vertex and the pivot vertex, respectively. Thus, we get $E = E \setminus \{\{3, 5\}\} \cup \{\{1, 5\}\}$.

Shells are created iteratively, starting with the maximum core. First, the predefined number n_i of vertices are created (line 3) together with an arbitrary removal order σ_i on them (line 4). In the element generation phase, some subtlety has been put into the choice of incident vertices of new edges. Since we only predefine the connectivity between shells, there is no fixed number of neighbors newly inserted vertices can be connected to. Instead, we maintain a list of SOURCEVERTICES which initially contains each vertex of shell i (line 6) exactly once, and an initially empty list TARGETVERTICES into which we insert the endvertices of each new edge, following the approach of Batagelj and Brandes [BB05].

We now iterate over each shell j that has already been created, starting with the very shell that has just been created ($j = i$), and create m_{ij} edges from shell i to shell j (loop starting at line 10). Each time an edge is created, we draw its source uniformly at random from SOURCEVERTICES (line 11) and check whether it now has the maximum outdegree for belonging to shell i (line 18), in which case we remove it from SOURCEVERTICES. Further, in the case $j = i$, if there are no more feasible targets for this source, i.e., it is already connected to all vertices with a higher value of σ_i, we remove it from SOURCEVERTICES (line 21). However, such a vertex is not yet saturated and therefore stored in the list UNCONNECTABLE (line 22) for later use in the case $j \neq i$ (line 25). Note that as a consequence, the highest ranking vertex $u = \text{argmax}_{v \in V_i} \sigma_i(v)$ in the current shell i is removed before the loop from SOURCEVERTICES (lines 5 and 6) and instantly added to the list UNCONNECTABLE.

Since edges can be directed towards any higher shell, we maintain the list of TARGETVERTICES[i] for each shell i throughout the algorithm. As mentioned above, these lists are the key for realizing preferential attachment. We initialize TARgetVertices[i] with u (line 7) since u is the only feasible target for all $v \in V_i$. For each choice of s in line 11, a list of feasible target vertices C is created (line 12). To this end, we prune list C of illegal choices, which are the source itself and its neighbors (line 13), and, in the case of $j = i$, vertices $v \in V_i$ with a lower value of $\sigma(v)$ (line 15). Concluding the creation of a new edge, we append its source and target to the list of targetVertices (line 23).

Analysis of the Algorithm

Based on the observations in the previous section, we prove the correctness of Algorithm 8 and analyze its running time in the following.

Observation 8.4 *Algorithm 8 generates valid core structures for the maximum number of intra-shell edges, i. e.,* $m_{ii} = i \cdot n_i - (i^2 + i)/2$ *for* $1 \leq i \leq k$.

Proof Let $m = i \cdot n_i - (i^2 + i)/2$. A vertex is removed from sourceVertices if either its out-degree is equal to i or it is connected to all vertices with a higher value of σ_i. If sourceVertices is empty, we have inserted $(n_i-(i+1))\cdot i+(i+1)\cdot i/2 = m$ edges (see Equation 8.4). □

Based on this observation, Lemmas 8.5 and 8.6 prove the correctness of Algorithm 8 inductively.

Lemma 8.5 *Given a matrix* M *belonging to a valid core fingerprint and a valid subgraph* $G[V_k \cup \cdots \cup V_{i+1}]$*, the element generation phase constructs the subgraph* $G[V_k \cup \cdots \cup V_i]$ *such that* M *is obeyed and all vertices* $u \in V_\ell$ *have* $\mathrm{coreness}\,(u) \leq \ell$*, for all* $i \leq \ell \leq k$.

Proof Let $j = i$. Lines 15 and 18 guarantee that σ_i is a valid removal order. Thus, all vertices $v \in V_i$ have $\mathrm{coreness}\,(v) \leq i$ and the coreness of all other vertices remains unchanged. Due to Observation 8.4 the upper bounds in Lemma 8.3 can be attained, thus any valid m_{ii} can be realized.

Now let $j > i$. Analogously, requiring $\mathrm{outdeg}(v) \leq i$ preserves the removal order and thus a coreness of i or less for vertices in V_i. Again, the coreness of all other vertices remains unchanged, and the upper bound in Lemma 8.3 can be attained. □

The above lemma shows that the element generation phase fits in all vertices and edges required by the fingerprint and grants to each vertex a coreness equal to or less than the required value. We are left to prove that the rewiring phase refines the edge set such that equality holds.

Lemma 8.6 *Given a matrix* M *belonging to a valid core fingerprint and a valid subgraph* $G[V_k \cup \cdots \cup V_{i+1}]$*. If* $\mathrm{coreness}\,(v) \leq i$ *holds for all* $v \in V_i$*, then the rewiring phase moves edges such that the subgraph* $G[V_k \cup \cdots \cup V_i]$ *is valid, i. e.,* M *is obeyed, and all vertices* $u \in V_\ell$ *have* $\mathrm{coreness}\,(u) = \ell$*, for all* $i \leq \ell \leq k$.

Proof We have to prove that the list poorVertices defined in line 27 is empty when the algorithm terminates. Suppose there exists at least one vertex

$v \in$ POORVERTICES. Since $\deg(v) < i$, clearly coreness$(v) < i$. Then, the list RICHVERTICES is not empty since otherwise all vertices $u \in V_i$ have $\deg(u) \leq i$ contradicting Lemma 8.3. Let $w \in$ RICHVERTICES, i. e., $w \in V_i$ and $\deg(w) > i$. Since $\deg(w) > \deg(v)$, the set of pivot candidates $C = \text{NEIGHBORS}(w) \setminus \text{NEIGHBORS}(v)$ is not empty. Choosing $c \in C$, the new set of edges $E' = E \setminus \{\{w, c\}\} \cup \{\{v, c\}\}$ still obeys M, decrements $\deg(w)$, and increments $\deg(v)$, increasing coreness(v) by at most one.

Thus, the rewiring phase maintains the invariant. Furthermore, due to the strict increase and decrease of $\deg(v)$ and $\deg(w)$, respectively, $|\text{POORVERTICES}|$ strictly decreases to 0, which terminates the algorithm. □

Since the base case, i. e., the empty graph, is trivial, Lemmas 8.5 and 8.6 inductive yield that Algorithm 8 constructs a graph in accordance with M and V_i, $0 \leq i \leq k$.

In terms of running time the crucial parts of the algorithm are the updates and random accesses of the lists SOURCEVERTICES, TARGETVERTICES, POORVERTICES, and RICHVERTICES, and the creation of the target candidate and pivot candidate lists (lines 12–15 and 31). We use array-backed lists to guarantee constant-time access to random elements. When we remove an element e, we fill its position with the last element of the list, avoiding moving all successive elements of e. Since we only have random access to the lists, preserving their orders is not required. This technique is similarly to Durstenfeld's modification [Dur64] of a Fisher-Yates shuffle [FY48].

Lemma 8.7 *The asymptotic running time of Algorithm 8 is bounded by $O((m^2 + n^2k)\log(n))$.*

Proof The runtime of the element generation phase is dominated by the assembling of target candidates in lines 12–15. Building a decision tree for the vertices to be removed in $O(n \log n)$ time, based on the ordering σ, we can prune list C in time $O(m \log n + n \log n)$ per edge, which dominates lines 3 to 25.

The running time of the rewiring phase is dominated by determining the list of pivot candidates in line 31 using $O(n \log n)$ time per rewiring. The total number of rewirings is bounded by $n \cdot k$. This dominates lines 27 to 37 as well as the element generation phase and all peripheral steps. Assuming the graph is connected, in total, both phases sum up to a running time of $O((m^2 + n^2k)\log(n))$. □

Since real-world networks seldom exhibit pathologic characteristics, we replaced the eager computation of the candidate list in lines 12–15 by a *lazy* selection from TARGETVERTICES$[i]$ that is repeated until a valid t has been drawn. Clearly, this does not improve worst-case running time but works faster for virtually all applications.

We performed our experiments on a recent standard PC, running SUSE Linux 10.2 with an implementation in Java. Absolute running times ranged between 100 and 500 milliseconds for the AS network which is comparable to BRITE. The running time of Inet is in the order of minutes. See Section 8.2.3 for the description of these generators.

Refinements

Although the core fingerprint is the prime characteristic we focus on in this work, together with the inclusion of a preferential attachment mechanism, a number of potentially describing features of a network exist. In this section, we briefly discuss other relevant features that can easily be integrated in our generator.

Connectivity is a very basic characteristic of a network, boiling down to the number of connected components. Building upon the core decomposition, this can be refined to the number of connected components *per shell*. While the whole graph or even the i-core can be connected, the i-shell can still have several disconnected components. If this is not desired, the user can specify the number and the sizes of connected components. The generator will then first create a spanning forest, where each tree is the seed of a component, and mark these edges as not rewirable. Note that requiring a specific set of connected components restricts the set of valid shell-connectivity matrices. However, this can be resolved by allowing the number of edges or the number and sizes of connected components to slightly deviate from the predefined values, depending on the user's interests.

Returning our focus to the degree distribution, the approach described in Section 8.2.2, depending on not a single parameter, can clearly be further elaborated. We tested two variants of our implementation of preferential attachment. In the first variant, we require the degree distributions of each shell as an input. Based on these we then prefill the array $\textsc{targetVertices}[i]$ in line 7 with the vertices in V_i, using the exact multiplicities as given by the degree distribution and an ordering analogous to σ. This approach clearly biases the preferential attachment process towards the desired degree distribution (see Figure 8.3). Alternatively, we can solely rely on a post-processing step. In this case we can completely abandon preferential attachment and simply apply a sequence of rewirings (Lemma 8.1) and swappings (Lemma 8.2) in order to approach a given degree distribution. Although both of these techniques yielded very good results, we exclude them from further evaluation due to their requiring rather specific parameters in addition to the core fingerprint.

8.2.3 Modeling the AS Network

An important application of a core-aware network generator is the simulation of the Internet at the AS level. In this section we compare networks generated by our method and established topology generators with three exemplary snapshots of the real AS network at the router level taken by the Oregon Routeviews project [Rou] at midnight on January 1st, 2002 (oix-full-snapshot-2002-01-01-0000), on January 1st, 2006 (oix-full-snapshot-2006-01-01-0000), and on July 1st, 2007 (oix-full-snapshot-2007-07-01-0000). Table 8.1 shows the sizes of these graphs.

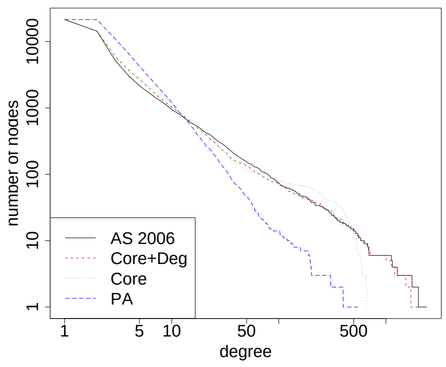


Figure 8.3: The number of vertices with degree at least d for the AS network, the original, and the refined Core generator for January 2006. A graph generated by preferential attachment of approximately the same size is shown for comparison.

	AS 2002-01	AS 2006-01	AS 2007-07
Number of Nodes	12 485	21 419	25 787
Number of Edges	25 980	45 638	53 014

Table 8.1: Sizes of the AS network snapshots.

Topology Generators

The first methods to generate networks with Internet-like structure date back to the 1990s and a multitude of techniques has been proposed since then. Among the most popular and widely used tools we have chosen Inet 3.0 [JCJ00] and BRITE [MLMB01] for our comparison since these are commonly included in other studies which cover a broader range of existing models [MP02, JCJ00]. The generator nem [Mag02] also seems promising, but we do not take it into account because of its limitation to networks not greater than 4000 vertices.

The *Internet topology generator Inet* [JCJ00] generates an AS-level representation of the Internet. Its developers claim that *"it generates random networks with characteristics similar to those of the Internet from November 1997 to February 2002, and beyond"*. Basically, Inet generates networks with a degree distribution which fits to one of the power laws originally found by Faloutsos et al. [FFF99], namely that the frequency of vertices with degree d is proportional to d raised to a power of a constant α: $f(d) \propto d^{\alpha}$. Since this law does not cover all vertices and in order to match other relevant properties as well, optimizations for various specific conditions were added to the original procedure over time. The complete generation method is explained in [JCJ00]. Since the procedures of Inet are already customized to AS networks, only a small number of input parameters can be specified: the total number of vertices, the fraction of degree-one vertices, and the size of the square used for vertex placement.

The *Boston university Representative Internet Topology gEnerator BRITE* [MLMB01] can generate networks for different levels of the Internet topology. Beside this, it offers various other options to customize the generation procedure.

Drawing area. The vertices of the generated topology are distributed in a square of a certain size.

Node distribution. In the drawing area, vertices are either distributed uniformly at random or Pareto.

Outgoing links. New vertices are connected with a specific number of outgoing links to other, already existing vertices.

Connectivity. The neighborhood of a vertex is selected based on certain guidelines such as geometric locality, preferential attachment, or a combination of both.

Procedure. Nodes can either be placed before the addition of edges or in an incremental fashion. In the latter case each new vertex introduces a number of new edges that can only connect to already existing vertices.

Characteristics

In [JCJ00], an extensive collection of characteristics is evaluated that judge the fitness of a generated graph with respect to its real world counterpart. We repeated this evaluation for a representative selection of these properties with a focus on the assessment of the core generator. In the following, we summarize the properties we employed in our analysis. Many of these statistics can be computed with visone and are described in Section 4.8.

General statistics. To see how well the generated networks fit to the most obvious characteristics, we computed some basic properties: the number of edges, the minimum and the maximum degree. Note that all models strictly meet the given number of vertices, so the number of edges corresponds to density and average degree.

Cores. The core decomposition is a significant structural property of an AS network. We compare not only the core number but also the extensive core fingerprint.

Clustering coefficient. The clustering coefficient is a measure for the local density around a vertex. It counts how many of a vertex's pairs of neighbors are themselves adjacent. These values are averaged to get a single measure for the network. Closely related characteristics are the numbers of triangles and triples and the transitivity (see Section 4.5.1).

Distance. We compare two properties based on distance: *characteristic distance*, which is the average of the distances of all vertex pairs and *average eccentricity*. The eccentricity of a vertex is its maximum distance to all other vertices. Average eccentricity then is the average of all vertices's eccentricities.

Frequency versus degree. One of the classic power laws found by Faloutsos et al. [FFF99] is $f(d) \propto d^{\alpha}$, that is, the frequency of vertices with degree d is proportional to d raised to a power of a constant α. Since this power law does not hold for nearly 2% of the highest degree vertices, we use a modified version [BT02, CCGJ02]:

$$F(d) = \sum_{i>d} f(i) \propto d^{\alpha} \quad .$$

Size of k-neighborhood. Another power law identified in [FFF99] is $\mathcal{N}(k) \propto k^{\beta}$, where $\mathcal{N}(k)$ is the sum over all vertices of their neighborhood sizes within distance k, i. e., $\mathcal{N}(k) = \sum_{u \in V} \sum_{v \in V} \mathrm{dist}_k(u, v)$, where

$$\mathrm{dist}_k(u, v) = \begin{cases} 1 & \text{, if } \mathrm{dist}(u, v) \leq k \\ 0 & \text{, otherwise.} \end{cases}$$

Note that this characteristic can also be measured as an average over all vertices, and it is also known as the *number of pairs within k hops.*

Evaluation

In the following, we detail the findings of our systematic evaluation. We gathered results on the three generators as described in Sections 8.2.2 and 8.2.3 and on the real AS network for all the properties listed in Section 8.2.3.

Based on the previous studies, we set appropriate parameters for the generators Inet and BRITE. For Inet we have chosen the default input parameters except for the number of vertices and the random seed. As the results in [MMB00] suggest, we have used preferential attachment and incremental growth for BRITE. Furthermore, we add two edges for each new vertex to fit the average degree of AS networks.

By construction, the numbers of vertices match the reference AS network, however, the numbers of edges already differ heavily. While the number of edges is only slightly lower for graphs generated by BRITE, and exactly fits the reference for our core generator (called *Core* in the following), the edge set created by Inet is larger by one third.

The well-known phenomenon of highly connected hubs in the AS network accompanied by the power-law degree distribution is regarded as one of the most significant properties of the Internet. Inet reproduces these quite well but overstates the maximum degree. In contrast, the degree distribution of Core oscillates around the reference but fails to produce high-degree vertices due to its lack of preferential attachment and the degree distribution of BRITE suggests that the preference of new vertices to connect to existing hubs is not strong enough either. These facts can be observed in Figure 8.4.

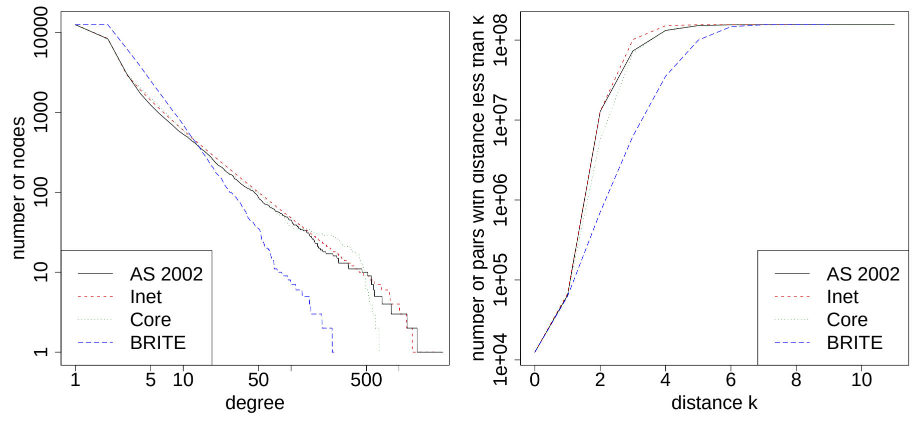
At a first glance, BRITE clearly fails to build up any kind of deep core structure (the core number is 2). The reason for this becomes evident from the incremental generation process of BRITE: the iterative addition of vertices incident to two new

	AS 2002-01	Core	BRITE	Inet
Number of Nodes	12 485	12 485	12 485	12 485
Number of Edges	25 980	25 980	24 967	27 494
Minimum Degree	1	1	2	1
Maximum Degree	2 538	644	302	2 154
Core Number	20	20	2	9
Number of Triples	7 258 817	3 140 777	347 443	6 821 628
Number of Triangles	22 832	17,272	157	11 144
Transitivity	0.009	0.016	0.001	0.005
Clustering Coeff.	0.45	0.24	0.00	0.29
Avg. Path Length	3.63	3.69	5.09	3.29
Avg. Eccentricity	8.74	9.71	8.35	6.85

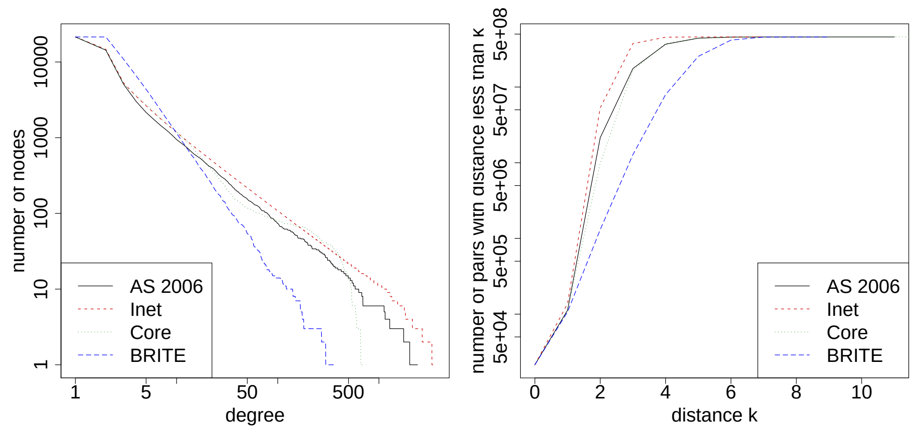
Table 8.2: Characteristics of the AS network of January 2002 and the three generators.

	AS 2006-01	Core	BRITE	Inet
Number of Nodes	21 419	21 419	21 419	21 419
Number of Edges	45 638	45 638	42 835	58 069
Minimum Degree	1	1	2	1
Maximum Degree	2 408	662	411	3 572
Core Number	26	26	2	19
Number of Triples	12 161 105	5 631 122	637 716	30 643 658
Number of Triangles	46 256	36 052	177	75 770
Transitivity	0.011	0.019	0.001	0.007
Clustering Coeff.	0.38	0.17	0.00	0.53
Avg. Path Length	3.81	3.84	5.31	3.07
Avg. Eccentricity	8.52	10.36	8.63	6.45

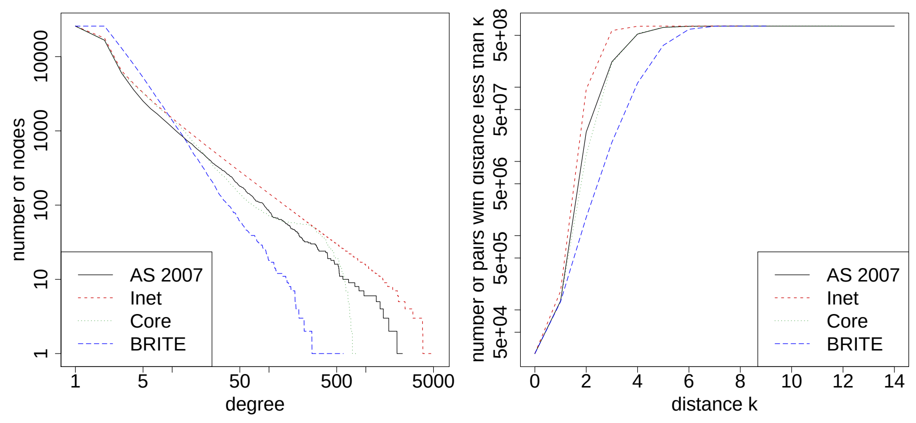
Table 8.3: Characteristics of the AS network of January 2006 and the three generators.



(a) January 1st, 2002



(b) January 1st, 2006



(c) July 1st, 2007

Figure 8.4: The number of vertices with a degree at least d (left) and the k-neighborhood for distances $k \in [0, 10]$ (right) for the AS network and the generated graphs for 2002, 2006, and July 2007.

	AS 2007-07	Core	BRITE	Inet
Number of Nodes	25 787	25 787	25 787	25 787
Number of Edges	53 014	53 014	51 571	76 467
Minimum Degree	1	1	2	1
Maximum Degree	2 391	838	393	5 168
Core Number	22	22	2	26
Number of Triples	13 889 150	6 759 443	757 653	56 514 215
Number of Triangles	39 646	29 612	174	162 889
Transitivity	0.009	0.013	0.001	0.009
Clustering Coeff.	0.33	0.15	0.00	0.65
Avg. Path Length	3.89	3.92	5.39	2.99
Avg. Eccentricity	10.24	10.64	8.72	6.52

Table 8.4: Characteristics of the AS network of July 2007 and the three generators.

edges can simply be reversed, resulting in a valid removal sequence for the 2-core that ultimately yields an empty 3-core. Figure 8.5 plots both the number of vertices and the number of edges per k-core exemplary for January 2006. Inet builds up a decent core hierarchy but fails to attain a sufficient depth for earlier snapshots, obviously resulting in larger mid-level shells, in terms of both vertices and edges. However, as Inet seems to systematically overestimate the number of edges, for later snapshots, the core hierarchy becomes too deep. By construction, Core perfectly matches the reference. The plots in Figure 8.6 show the numbers of vertices and edges per k-shell, again exemplary for January 2006. They confirm the above observations and additionally grant an insight into the absolute numbers of elements per shell.

The shallow core structure created by BRITE is accompanied by a very low transitivity alongside a negligible number of triangles and a tiny clustering coefficient, suggesting that the BRITE graph is primarily composed of a set of paths of length two. The high average path length further corroborates this conjecture since by virtue of preferential attachment hubs of high degree evolve, which, however, are interconnected via paths of length two by construction.

The absolute numbers of triples and triangles as well as the transitivity and the clustering coefficient are acceptable for both Core and Inet. The discrepancy of the latter generator from the reference can quite generally be explained by the increased number of edges. The behavior of Core with respect to these values is largely due to the absence of high-degree vertices since, intuitively speaking, star-shaped structures yield a high number of triples. The relatively high number of triangles thus yields an increased transitivity. The low clustering coefficient, however, suggests that there is a large number of vertices with a sparse direct neighborhood. Since, at the same time, Core exhibits a high number of triangles, the majority of these triangles is incident to vertices with higher degree.

Figure 8.4 depicts the size of the neighborhood within k hops (sum over all vertices).

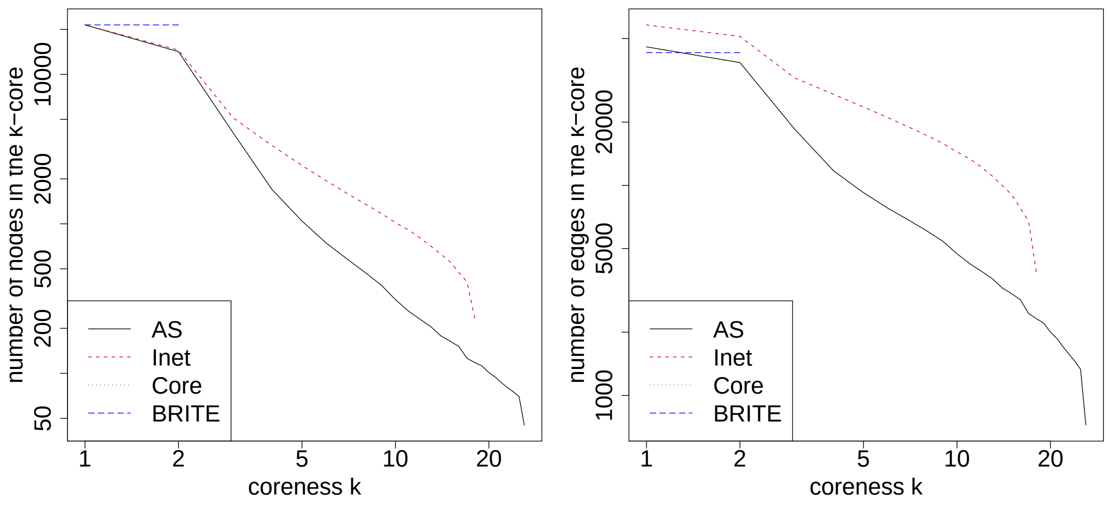


Figure 8.5: The numbers of vertices (left figure) and of edges (right figure) per k-core. Note that BRITE generates only vertices in the 2-core and that the lines of the AS 2006 and Core perfectly match by construction.

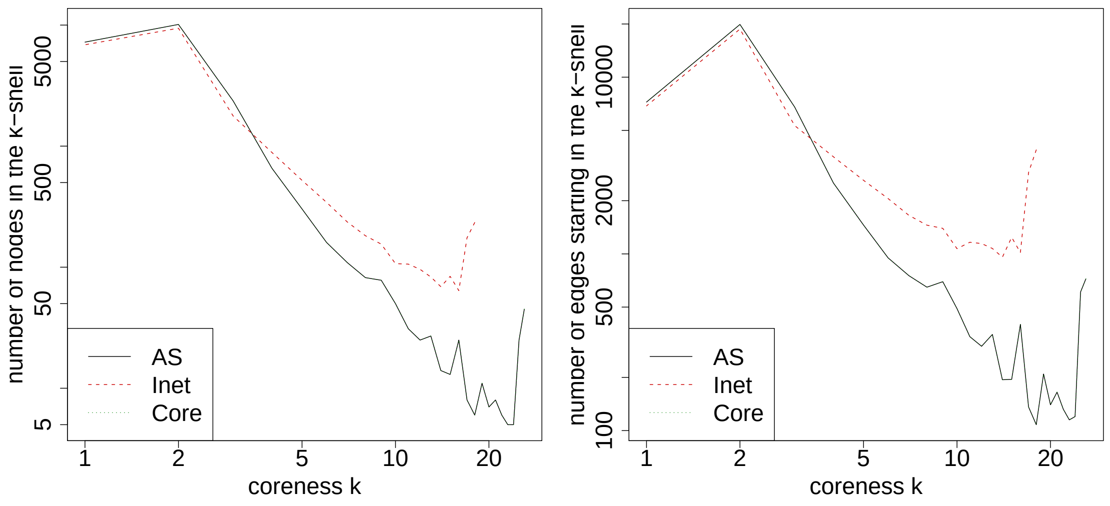


Figure 8.6: The numbers of vertices (left figure) and of edges (right figure) per k-shell (BRITE omitted). An edge is considered to belong to the k-shell if its endvertex with smallest coreness has coreness k. Note that the lines of the AS 2006 and Core perfectly match by construction.

Note that the high average path length of BRITE mentioned earlier comes along with the slow growth of the neighborhood size. The low average path length and the low average eccentricity exhibited by Inet are, again, due to the large edge set. With respect to these values, Core excels. Both the average path length and the k-neighborhood practically match the reference.

8.2.4 Conclusion

In the recent past, the core decomposition has been found to be a crucial characteristic of real world complex systems. We have presented a novel algorithm for the generation of graphs that brings together the well-known concepts of k-cores and preferential attachment. After scrutinizing and clarifying how to specify the core fingerprint of a network by examining the inter-connectivity of each pair of shells, we employ this core fingerprint to introduce a simple and efficient algorithm for the generation of random graphs based on the core decomposition.

We exemplify the feasibility of our technique in a case study using the AS network of the Internet, comparing our generator to the established topology generators BRITE [MLMB01] and Inet [JCJ00]. Our results yield that our generator is highly suitable for the simulation of AS topologies, confirming the importance of the core decomposition. Moreover, we show that BRITE largely fails to capture significant characteristics of the AS network, including its core structure, and that Inet roughly matches the reference except for its general tendency to be too densely connected. While our core generator and BRITE create a topology within seconds, a major drawback of Inet is its generation time of several minutes.

The high customizability of our rather generic core generator suggests several adaptations that can further increase the fitness to the specific peculiarities of the AS network. Such adaptations to special networks can be realized by employing a number of structural modifications such as swapping and rewiring without interfering with the core decomposition.

Bibliography

[AB02] Réka Albert and Albert-László Barabási. Statistical Mechanics of Complex Networks. *Reviews of Modern Physics*, 74(1):47–97, 2002.
Cited on page 136.

[ACG$^+$02] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, and Alberto Marchetti-Spaccamela. *Complexity and Approximation - Combinatorial Optimization Problems and Their Approximability Properties.* Springer, 2nd edition, 2002.
Cited on page 54.

[AHDBV05] José Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. k-Core Decomposition: A Tool for the Analysis of Large Scale Internet Graphs. Electronically published at `http://arxiv.org/abs/cs.NI/0511007`, November 2005.
Cited on page 138.

[AHDBV06] José Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. Large Scale Networks Fingerprinting and Visualization Using the k-Core Decomposition. In *Advances in Neural Information Processing Systems 18*, pages 41–50. MIT Press, 2006.
Cited on page 138.

[AMA07] Daniel Archambault, Tamara Munzner, and David Auber. TopoLayout: Multi-Level Graph Layout by Topological Features. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):305–317, 2007.
Cited on page 92.

[Ana08] Analytic Technologies. UCINET 6 - Social Network Analysis Software, 2008.
Cited on pages 14 and 24.

[Ant71] Jacob M. Anthonisse. The rush in a directed graph. Technical Report BN 9/71, Stichting Mathematisch Centrum, 2e Boerhaavestraat 49 Amsterdam, Oct 1971.
Cited on pages 38, 39, and 40.

[ARV04] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander Flows, Geometric Embeddings and Graph Partitioning. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing (STOC'04)*, pages 222–231. ACM Press, 2004.
Cited on page 54.

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and Counting Given Length Cycles. *Algorithmica*, 17(3):209–223, 1997.
Cited on page 48.

[BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
Cited on page 133.

[Bac07] Christian Bachmaier. A Radial Adaptation of the Sugiyama Framework for Visualizing Hierarchical Information. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):585–594, 2007.
Cited on pages 82, 114, 115, and 118.

[Bak] Christopher Paul Baker. Email map. `http://christopherbaker.net/projects/mymap/`.
Cited on page 66.

[Bav48] Alex Bavelas. A Mathematical Model for Group Structure. *Human Organizations*, 7:16–30, 1948.
Cited on page 35.

[Bav50] Alex Bavelas. Communication patterns in task oriented groups. *Journal of the Acoustical Society of America*, 22:271–282, 1950.
Cited on page 35.

[BB04a] Michael Baur and Ulrik Brandes. Crossing Reduction in Circular Layouts. In *Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'04)*, Lecture Notes in Computer Science, pages 332–343. Springer, 2004.
Cited on pages 10, 15, and 114.

[BB04b] Michael Baur and Ulrik Brandes. Crossing Reduction in Circular Layouts. Technical Report 2004-14, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2004.
Cited on page 15.

[BB05] Vladimir Batagelj and Ulrik Brandes. Efficient Generation of Large Random Networks. *Physical Review E*, (036113), 2005.
Cited on pages 133, 134, 136, 137, 142, and 144.

[BB08] Michael Baur and Ulrik Brandes. Multi-Circular Layout of Micro/Macro Graphs. In Hong et al. [HNQ08], pages 255–267.
Cited on pages 10 and 15.

[BBB$^+$02] Michael Baur, Marc Benkert, Ulrik Brandes, Sabine Cornelsen, Marco Gaertler, Boris Köpf, Jürgen Lerner, and Dorothea Wagner. visone - Software for Visual Social Network Analysis. In GD'01 [GD'02], pages 463–464.
Cited on page 13.

[BBGW04] Michael Baur, Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Drawing the AS Graph in Two and a Half Dimensions. Technical

Report 2004-12, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2004.
Cited on pages 15 and 126.

[BBGW05] Michael Baur, Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Drawing the AS Graph in 2.5 Dimensions. In GD'04 [GD'05], pages 43–48.
Cited on page 15.

[BBP08] Krists Boitmanis, Ulrik Brandes, and Christian Pich. Visualizing Internet Evolution on the Autonomous Systems Level. In Hong et al. [HNQ08], pages 365–376.
Cited on page 70.

[BC03a] Ulrik Brandes and Steven R. Corman. Visual Unrolling of Network Evolution and the Analysis of Dynamic Discourse. *Journal of Information Visualization*, 2(1), 2003.
Cited on page 124.

[BC03b] Ulrik Brandes and Sabine Cornelsen. Visual Ranking of Link Structures. *Journal of Graph Algorithms and Applications*, 7(2):181–201, 2003.
Cited on page 126.

[BD07] Michael Balzer and Oliver Deussen. Level-of-Detail Visualization of Clustered Graph Layouts. In *Asia-Pacific Symposium on Visualisation 2007 (APVIS'07)*, 2007.
Cited on page 92.

[BDG⁺06] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Höfer, Zoran Nikoloski, and Dorothea Wagner. Maximizing Modularity is hard, 2006. ArXiv physics/0608255.
Cited on page 56.

[BDS04] Ulrik Brandes, Tim Dwyer, and Falk Schreiber. Visual Understanding of Metabolic Pathways Across Organisms Using Layout in Two and a Half Dimensions. *Journal of Integrative Bioinformatics*, 002, 2004.
Cited on page 124.

[BE05a] Ulrik Brandes and Thomas Erlebach. Fundamentals. In *Network Analysis: Methodological Foundations* [BE05b], pages 7–15.
Cited on page 18.

[BE05b] Ulrik Brandes and Thomas Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *Lecture Notes in Computer Science*. Springer, February 2005.
Cited on pages 35, 159, 162, and 166.

[Bea65] Murray A. Beauchamp. An improved index of centrality. *Behavioral Scie*, 10:161–163, 1965.
Cited on page 37.

[BEF99] Stephen P. Borgatti, Martin G. Everett, and Linton Clarke Freeman. Users Guide of UCINET 5 for Windows , 1999.
Cited on page 24.

[BEH^{+}01] Ulrik Brandes, Markus Eiglsperger, Ivan Herman, Michael Himsolt, and M. Scott Marshall. GraphML Progress Report: Structural Layer Proposal. In *Proceedings of the 8th International Symposium on Graph Drawing (GD'00)*, volume 1984 of *Lecture Notes in Computer Science.* Springer, January 2001.
Cited on page 24.

[BF05] Ulrik Brandes and Daniel Fleischer. Centrality Measures Based on Current Flow. In *Proceedings of the 22nd International Symposium on Theoretical Aspects of Computer Science (STACS'05)*, volume 3404 of *Lecture Notes in Computer Science.* Springer, 2005.
Cited on pages 41, 42, and 43.

[BF06] Christian Bachmaier and Michael Forster. Crossing Reduction for Hierarchical Graphs with Intra-Level Edges. Technical Report MIP-0608, University of Passau, 2006.
Cited on page 109.

[BFP07] Ulrik Brandes, Daniel Fleischer, and Thomas Puppe. Dynamic Spectral Layout with an Application to Small Worlds. *Journal of Graph Algorithms and Applications*, 11(2):325–343, 2007.
Cited on page 74.

[BGG^{+}07] Michael Baur, Marco Gaertler, Robert Görke, Marcus Krug, and Dorothea Wagner. Generating Graphs with Predefined k-Core Structure. In *Proceedings of the European Conference of Complex Systems (ECCS'07)*, October 2007. Online proceedings `http://cssociety.org/ECCS07-Programme`.
Cited on pages 11 and 15.

[BGG^{+}08] Michael Baur, Marco Gaertler, Robert Görke, Marcus Krug, and Dorothea Wagner. Augmenting k-Core Generation with Preferential Attachment. *Networks and Heterogeneous Media*, 3(2):277–294, June 2008.
Cited on page 15.

[BGW07] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Engineering Graph Clustering: Models and Experimental Evaluation. *ACM Journal of Experimental Algorithmics*, 12(1.1):1–26, 2007.
Cited on pages 54 and 58.

[BK02] Ulrik Brandes and Boris Köpf. Fast and Simple Horizontal Coordinate Assignmen. In GD'01 [GD'02], pages 31–44.
Cited on pages 83 and 89.

[BKR^{+}99] Ulrik Brandes, Patrick Kenis, Jörg Raab, Volker Schneider, and Dorothea Wagner. Explorations into the Visualization of Policy Net-

works. *Journal of Theoretical Politics*, 11(1):75–106, 1999.
Cited on pages 13, 64, 75, 81, and 91.

[BKR06] Ulrik Brandes, Patrick Kenis, and Jörg Raab. Explanation Through Network Visualization. *Methodology*, 2(1):16–23, 2006.
Cited on pages 81, 86, and 91.

[BKW03] Ulrik Brandes, Patrick Kenis, and Dorothea Wagner. Communicating Centrality in Policy Network Drawings. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):241–253, 2003.
Cited on page 81.

[BM76] J. A. Bondy and U. S. Murty. *Graph THeory with Applications*. Macmillan, 1976.
Cited on page 77.

[BM98] Vladimir Batagelj and Andrej Mrvar. Pajek – A Program for Large Network Analysis. *Connections*, 21(2):47–57, 1998.
Cited on page 75.

[BM03] Vladimir Batagelj and Andrej Mrvar. Pajek - Analysis and Visualization of Large Networks. In Michael Jünger and Petra Mutzel, editors, *Graph Drawing Software*, pages 77–103. Springer, 2003.
Cited on page 24.

[Bol98] Béla Bollobás. *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*. Springer, 1998.
Cited on pages 41 and 42.

[Bon72] Phillip Bonacich. Factoring and Weighting Approaches to Status Scores and Clique Identification. *Journal of Mathematical Sociology*, 2:113–120, 1972.
Cited on page 44.

[BP98] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.
Cited on page 45.

[BP07] Ulrik Brandes and Christian Pich. Eigensolver Methods for Progressive Multidimensional Scaling of Large Data. In Kaufmann and Wagner [KW07], pages 42–53.
Cited on page 70.

[Bra01a] Ulrik Brandes. A Faster Algorithm for Betweenness Centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.
Cited on page 40.

[Bra01b] Ulrik Brandes. Energy-Based Placement. In Michael Kaufmann and Dorothea Wagner, editors, *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes in Computer Science*, pages 78–86. Springer, 2001.
Cited on page 69.

[Bra08] Ulrik Brandes. On Variants of Shortest-Path Betweenness Centrality and their Generic Computation. *Social Networks*, 30(2):136–145, 2008.
Cited on pages 14, 39, and 40.

[Bre99] Cynthia A. Brewer. Color Use Guidelines for Data Representation. In *Section on Statistical Graphics '99*, pages 55–60. American Statistical Association, 1999.
Cited on page 78.

[BRST01] Béla Bollobás, Oliver M. Riordan, Joel Spencer, and Gábor Tusnády. The Degree Sequence of a Scale-Free Random Graph Process. *Randoms Structures and Algorithms*, 18:279–290, 2001.
Cited on page 137.

[BRW01] Ulrik Brandes, Jörg Raab, and Dorothea Wagner. Exploratory Network Visualization: Simultaneous Display of Actor Status and Connections. *Journal of Social Structure*, 2(4), October 2001.
Cited on pages 13, 87, and 91.

[BS05] Michael Brinkmeier and Thomas Schank. Network Statistics. In Brandes and Erlebach [BE05b], pages 293–317.
Cited on pages 59 and 60.

[BT02] Tian Bu and Don Towsley. On Distinguishing between Internet Power Law Topology Generators. In INFOCOM'02 [INF02].
Cited on page 150.

[BW00a] Alain Barrat and Martin Weigt. On the properties of small-world network models. *The European Physical Journal B*, 13:547–560, 2000.
Cited on page 47.

[BW00b] Ulrik Brandes and Dorothea Wagner. Contextual Visualization of Actor Status in Social Networks. In Jan W. de Leeuw and Robert van Liere, editors, *Proceedings of the 2nd Joint Eurographics - IEEE TVCG Symposium on Visualization (VisSym'00)*, pages 13–22. Springer, 2000.
Cited on page 13.

[BZ02a] Vladimir Batagelj and Matjaž Zaveršnik. An $\mathcal{O}(m)$ Algorithm for Cores Decomposition of Networks. Technical Report 798, IMFM Ljublana, Ljubljana, 2002.
Cited on pages 49 and 139.

[BZ02b] Vladimir Batagelj and Matjaž Zaveršnik. Generalized Cores. Preprint 799, IMFM Ljublana, Ljubljana, 2002.
Cited on pages 49, 124, and 133.

[CAI02] CAIDA. Walrus – Graph Visualization Tool, 2002.
Cited on page 124.

[CAI03] CAIDA. Visualizing Internet Topology at a Macroscopic Scale, 2003.
Cited on page 124.

[CC01] Trevor F. Cox and Michael A. A. Cox. *Multidimensional Scaling.* Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, 2nd edition, 2001.
Cited on page 98.

[CCGJ02] Qian Chen, Hyunseok Chang, Ramesh Govindan, and Sugih Jamin. The Origin of Power Laws in Internet Topologies Revisited. In INFOCOM'02 [INF02], pages 608–617.
Cited on page 150.

[CHK$^+$07] Shai Carmi, Shlomo Havlin, Scott Kirkpatrick, Yuval Shavitt, and Eran Shir. A Model of Internet Topology Using k-Shell Decomposition. *Proceedings of the National Academy of Science of the United States of America*, 104:11150–11154, 2007.
Cited on page 138.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press, 2nd edition, 2001.
Cited on page 20.

[CM84] William S. Cleveland and Robert McGill. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. *Journal of the American Statistical Association*, 79(387):531–554, September 1984.
Cited on page 75.

[CNM04] Aaron Clauset, Mark E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(066111), 2004.
Cited on page 56.

[Coh97] Johnathan D. Cohen. Drawing Graphs to Convey Proximity: An Incremental Arrangement Method. *ACM Transactions on Computer-Human Interaction*, 4(3):197–229, September 1997.
Cited on page 68.

[CW90] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.
Cited on page 48.

[DBGL$^+$97] Guiseppe Di Battista, Ashim Garg, Giuseppe Liotta, Roberto Tamassia, Emanuele Tassinari, and Francesco Vargiu. An Experimental Comparison of Four Graph Drawing Algorithms. *Computational Geometry: Theory and Applications*, 7(5-6):303–325, April 1997.
Cited on page 110.

[DBMPP04] Guiseppe Di Battista, Federico Mariani, Maurizio Patrignani, and Maurizio Pizzonia. BGPlay: A System for Visualizing the Interdomain Routing Evolution. In Liotta [Lio04], pages 295–306.
Cited on page 124.

[DEKW02] Christian A. Duncan, Alon Efrat, Stephen G. Kobourov, and Carola Wenk. Drawing with Fat Edges. In GD'01 [GD'02], pages 162–177.
Cited on pages 93 and 113.

[DGM06] Sergey N. Dorogovtsev, Andrew V. Goldberg, and Jose Ferreira F. Mendes. k-Core Organization of Complex Networks. *Physical Review Letters*, 96(040601):1–4, February 2006.
Cited on page 138.

[DKM06] Tim Dwyer, Yehuda Koren, and Kim Marriott. Stress Majorization with Orthogonal Ordering Constraints. In *Proceedings of the 13th International Symposium on Graph Drawing (GD'05)*, volume 3843 of *Lecture Notes in Computer Science*, pages 141–152. Springer, January 2006.
Cited on page 70.

[DM08] Tim Dwyer and Kim Marriott. Constrained Stress Majorization Using Diagonally Scaled Gradient Projection. In Hong et al. [HNQ08], pages 219–230.
Cited on page 70.

[DMM97] Ugur Dogrusöz, Brendan Madden, and Patrick Madden. Circular Layout in the Graph Layout Toolkit. In GD'96 [GD'97], pages 92–100.
Cited on page 101.

[dST02] Vin de Silva and Joshua B. Tenenbaum. Global Versus Local Methods in Nonlinear Dimensionality Reduction. *Advances in Neural Information Processing Systems*, 15:705–712, 2002.
Cited on page 70.

[Dur64] Richard Durstenfeld. Algorithm 235: Random Permutation. *Communications of the ACM*, 7(7):420–421, July 1964.
Cited on page 146.

[DYNM06] Nicolas Ducheneaut, Nicholas Yee, Eric Nickell, and Robert J. Moore. Alone Together?: Exploring the Social Dynamics of Massively Multiplayer Online Games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06)*, pages 407–416. ACM Press, 2006.
Cited on page 138.

[Ead84] Peter Eades. A Heuristic for Graph Drawing. *Congressus Numerantium*, 42:149–160, 1984.
Cited on page 67.

[EF97] Peter Eades and Qing-Wen Feng. Multilevel Visualization of Clustered Graphs. In GD'96 [GD'97], pages 101–112.
Cited on page 124.

[EK86] Peter Eades and David Kelly. Heuristics for Reducing Crossings in 2-Layered Networks. *Ars Combinatoria*, 21(A):89–98, 1986.
Cited on page 105.

[ER59] Paul Erdős and Alfred Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.
Cited on pages 133 and 134.

[EW94] Peter Eades and Nicholas C. Wormald. Edge Crossings in Drawings of Bipartite Graphs. *Algorithmica*, 11(4), April 1994.
Cited on pages 89 and 102.

[FBW91] Linton Clarke Freeman, Stephen P. Borgatti, and Douglas R. White. Centrality in valued graphs: A measure of betweenness based on network flow. *Social Networks*, 13(2):141–154, 1991.
Cited on page 39.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the Internet topology. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 251–262. ACM Press, 1999.
Cited on pages 136, 138, 148, and 150.

[FI65] C. J. Fisk and D. D. Isett. "ACCEL" Automated Circuit Card Etching Layout. In *Proceedings of the SHARE design automation project (DAC'65)*, pages 9.1–9.31. ACM Press, 1965.
Cited on page 67.

[Fle07] Daniel Fleischer. *Theory and Applications of the Laplacian*. PhD thesis, 2007.
Cited on page 14.

[FMR62] C. T. Fun, Mervin E. Muller, and Ivan Rezucha. Development of Sampling Plans by Using Sequential (Item by Item) Selection Techniques and Digital-Computers. *Journal of the American Statistical Association*, 57(298):387–402, 1962.
Cited on page 134.

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold. Graph Drawing by Force-Directed Placement. *Software - Practice and Experience*, 21(11):1129–1164, November 1991.
Cited on pages 67, 68, 98, and 126.

[Fre77] Linton Clarke Freeman. A Set of Measures of Centrality Based Upon Betweeness. *Sociometry*, 40:35–41, 1977.
Cited on pages 38, 39, and 60.

[Fre79] Linton Clarke Freeman. Centrality in Social Networks: Conceptual Clarification I. *Social Networks*, 1:215–239, 1979.
Cited on page 36.

[Fre00] Linton Clarke Freeman. Visualizing Social Networks. *Journal of Social Structure*, 1(1), 2000.
Cited on pages 14 and 63.

[FY48] Ronald Aylmer Fisher and Frank Yates. *Statistical Tables for Biological, Agricultural and Medical Research.* Oliver and Boyd, London, 1948.
Cited on page 146.

[Gae02] Marco Gaertler. Clustering with Spectral Methods. Diplomarbeit, Fachbereich Informatik und Informationswissenschaft, Universität Konstanz, March 2002.
Cited on page 58.

[Gae05] Marco Gaertler. Clustering. In Brandes and Erlebach [BE05b], pages 178–215.
Cited on page 52.

[Gae07] Marco Gaertler. *Algorithmic Aspects of Clustering – Theory, Experimental Evaluation, and Applications in Network Analysis and Visualization.* PhD thesis, Universität Karlsruhe (TH), Fakultät für Informatik, 2007.
Cited on pages 14, 52, 56, 57, 58, and 125.

[GD'97] *Proceedings of the 4th International Symposium on Graph Drawing (GD'96)*, volume 1090 of *Lecture Notes in Computer Science.* Springer, January 1997.
Cited on page 164.

[GD'02] *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*, volume 2265 of *Lecture Notes in Computer Science.* Springer, January 2002.
Cited on pages 158, 160, and 164.

[GD'05] *Proceedings of the 12th International Symposium on Graph Drawing (GD'04)*, volume 3383 of *Lecture Notes in Computer Science.* Springer, January 2005.
Cited on pages 159 and 167.

[GGW08] Robert Görke, Marco Gaertler, and Dorothea Wagner. LunarVis - Analytic Visualizations of Large Graphs. In Hong et al. [HNQ08], pages 352–364.
Cited on page 138.

[Gil59] Horst Gilbert. Random Graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.
Cited on page 133.

[GJ83] Michael R. Garey and David S. Johnson. Crossing Number is NP-Complete. *SIAM Journal on Algebraic and Discrete Methods*, 4:312–316, 1983.
Cited on page 89.

[GK07] Emden R. Gansner and Yehuda Koren. Improved Circular Layouts. In Kaufmann and Wagner [KW07], pages 386–398.
Cited on page 118.

[GKN05] Emden R. Gansner, Yehuda Koren, and Stephen C. North. Graph Drawing by Stress Majorization. In GD'04 [GD'05], pages 239–250.
Cited on pages 68 and 70.

[GMZ03] Christos Gkantsidis, Milena Mihail, and Ellen W. Zegura. Spectral Analysis of Internet Topologies. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom)*, volume 1, pages 364–374. IEEE Computer Society Press, March 2003.
Cited on page 124.

[GN99] Emden R. Gansner and Stephen C. North. Improved Force-Directed Layouts. In Sue H. Whitesides, editor, *Proceedings of the 6th International Symposium on Graph Drawing (GD'98)*, volume 1547 of *Lecture Notes in Computer Science*, pages 364–373. Springer, January 1999.
Cited on pages 93 and 113.

[GP04] Marco Gaertler and Maurizio Patrignani. Dynamic Analysis of the Autonomous System Graph. In *IPS 2004 – Inter-Domain Performance and Simulation*, pages 13–24, March 2004.
Cited on page 124.

[GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. John Hopkins University Press, 3rd edition, 1996.
Cited on page 46.

[Hal70] Kenneth M. Hall. An r-Dimensional Quadratic Placement Algorithm. *Management Science*, 17(3):219–229, November 1970.
Cited on page 72.

[Hea96] Christopher G. Healey. Choosing Effective Colours for Data Visualization. In *Proceedings of the 7th IEEE Conference on Visualization '96 (VIS'96)*, pages 263–270. IEEE Computer Society, 1996.
Cited on pages 77 and 78.

[HH95] Frank Harary and Per Hage. Eccentricity and centrality in networks. *Social Networks*, 17:57–63, 1995.
Cited on page 37.

[HK02] David Harel and Yehuda Koren. Drawing Graphs with Non-Uniform Vertices. In *Proceedings of the 6th Working Conference on Advanced Visual Interfaces (AVI'02)*, pages 157–166. ACM Press, 2002.
Cited on pages 93 and 113.

[HK03] Martin Höpner and Lothar Krempel. The Politics of the German Company Network. Technical Report 03/9, MPI für Gesellschaftsforschung, 2003.
Cited on pages 64, 97, and 98.

[HK04] Martin Höpner and Lothar Krempel. The Politics of the German Company Network. *Competition and Change*, 8(4):339–356, Decem-

ber 2004.
Cited on page 97.

[HM98] Weiqing He and Kim Marriott. Constrained Graph Layout. *Constraints*, 3(4):289–314, October 1998.
Cited on page 67.

[HNQ08] Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors. *Proceedings of the 15th International Symposium on Graph Drawing (GD'07)*, volume 4875 of *Lecture Notes in Computer Science*. Springer, January 2008.
Cited on pages 158, 159, 164, and 166.

[Hol06] Danny Holten. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006.
Cited on pages 92 and 123.

[HS04] Hongmei He and Ondrej Sýkora. New Circular Drawing Algorithms, 2004. Unpublished manuscript.
Cited on page 111.

[HT73] John E. Hopcroft and Robert E. Tarjan. Efficient Algorithms for Graph Manipulation. *Communications of the ACM*, 16(6):372–378, June 1973.
Cited on page 52.

[INF02] *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom)*, volume 1. IEEE Computer Society Press, 2002.
Cited on pages 162 and 163.

[JCJ00] Cheng Jin, Qian Chen, and Sugih Jamin. Inet Topology Generator. Technical Report CSE-TR-433, EECS Department, University of Michigan, 2000.
Cited on pages 138, 148, 149, and 155.

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.
Cited on page 52.

[JM97] Michael Jünger and Petra Mutzel. 2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms. *Journal of Graph Algorithms and Applications*, 1:1–25, 1997.
Cited on page 89.

[Kar72] Richard M. Karp. Reducibility among Combinatorial Problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
Cited on page 50.

[Kat53] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.
Cited on pages 39, 43, and 44.

[KBM94] David Krackhardt, Jim Blythe, and Cathleen McGrath. KrackPlot 3.0: An Improved Network Drawing Program. *Connections*, 17(2):53–55, 1994.
Cited on page 69.

[KBM06] David Krackhardt, Jim Blythe, and Cathleen McGrath. KrackPlot 4.3 Network Drawing Program. 2006.
Cited on page 69.

[KCH03] Yehuda Koren, Liran Carmel, and David Harel. Drawing Huge Graphs by Algebraic Multigrid Optimization. *Multiscale Modeling and Simulation*, 1(4):645–673, 2003.
Cited on page 74.

[KK89] Tomihisa Kamada and Satoru Kawai. An Algorithm for Drawing General Undirected Graphs. *Information Processing Letters*, 31(1):7–15, April 1989.
Cited on pages 67, 69, and 70.

[Kle99] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1999.
Cited on pages 44, 45, and 46.

[Kre96] Valdis E. Krebs. Visualizing Human Networks. *Release 1.0*, pages 1–25, February 1996.
Cited on pages 92, 94, 95, 98, and 113.

[Kre05] Lothar Krempel. *Visualisierung komplexer Strukturen. Grundlagen der Darstellung mehrdimensionaler Netzwerke.* Campus, 2005.
Cited on page 98.

[Kru64] Joseph B. Kruskal. Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis. *Psychometrika*, 29(1):1–27, March 1964.
Cited on page 69.

[KVV00] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On Clusterings - Good, Bad and Spectral. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS'00)*, pages 367–378, 2000.
Cited on page 54.

[KVV04] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On Clusterings: Good, Bad, Spectral. *Journal of the ACM*, 51(3):497–515, May 2004.
Cited on page 57.

[KW03] Michael Kaufmann and Roland Wiese. Maintaining the Mental Map for Circular Drawings. In *Proceedings of the 10th International Symposium on Graph Drawing (GD'00)*, volume 2528 of *Lecture Notes in Computer Science*, pages 12–22. Springer, January 2003.
Cited on page 92.

[KW07] Michael Kaufmann and Dorothea Wagner, editors. *Proceedings of the 14th International Symposium on Graph Drawing (GD'06)*, volume 4372 of *Lecture Notes in Computer Science*. Springer, January 2007.
Cited on pages 161 and 166.

[Ler07] Jürgen Lerner. *Structural Similarity of Vertices in Networks*. PhD thesis, 2007.
Cited on page 14.

[LGH06] Stefan Lämmer, Björn Gehlsen, and Dirk Helbing. Scaling Laws in the Spatial Structure of Urban Road Networks. *Physica A*, 363(1):89–95, 2006.
Cited on page 39.

[LH08] Jure Leskovec and Eric Horvitz. Planetary-Scale Views on a Large Instant-Messaging Network. In *Proceedings of the 17th International World Wide Web Conference (WWW2008)*, pages 915–924, 2008.
Cited on page 135.

[Lio04] Giuseppe Liotta, editor. *Proceedings of the 11th International Symposium on Graph Drawing (GD'03)*, volume 2912 of *Lecture Notes in Computer Science*. Springer, January 2004.
Cited on pages 163 and 174.

[LLK80] Eugene L. Lawler, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan. Generating all maximal independent sets: $\mathcal{NP}$-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.
Cited on page 50.

[LP49] R. Duncan Luce and Albert Perry. A method of matrix analysis of group structure. *Psychometrika*, 14:95–116, 1949.
Cited on page 50.

[Mag02] Damien Magoni. nem: A Software for Network Topology Analysis and Modeling. In *Proceedings of the 10th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE Computer Society, 2002.
Cited on pages 138 and 148.

[Mäk88] E. Mäkinen. On Circular Layouts. *International Journal of Computer Mathematics*, 1988.
Cited on pages 101 and 103.

[Mil67] Stanley Milgram. The small world problem. *Psychology Toda*, 1:61, 1967.
Cited on page 135.

[Mit79] Sandra L. Mitchell. Linear Algorithms to Recognize Outerplanar and Maximal Outerplanar Graphs. *Information Processing Letters*, 9(5):229–232, December 1979.
Cited on pages 103 and 109.

[MKNF87] Sumio Masuda, Toshinobu Kashiwabara, Kazuo Nakajima, and Toshio Fujisawa. On the $\mathcal{NP}$-completeness of a computer network layout problem. In *Proceedings of the 20th IEEE International Symposium on Circuits and Systems 1987*, pages 292–295. IEEE Computer Society, 1987.
Cited on pages 65, 82, 94, 101, 102, and 118.

[ML03] Rafael Martí and Manuel Laguna. Heuristics and Meta-Heuristics for 2-Layer Straight Line Crossing Minimization. *Discrete Applied Mathematics*, 127(3):665–678, May 2003.
Cited on page 89.

[MLMB01] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An Approach to Universal Topology Generation. In *Proceedings of the 9th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2001.
Cited on pages 138, 148, 149, and 155.

[MM65] John W. Moon and L. Moser. On Cliques in Graphs. *Israel Journal of Mathematics*, 3:23–28, 1965.
Cited on page 50.

[MMB00] Alberto Medina, Ibrahim Matta, and John Byers. On the Origin of Power Laws in Internet Topologies. *Computer Communication Review*, 30(2), April 2000.
Cited on page 150.

[MN99] Kurt Mehlhorn and Stefan Näher. *LEDA, A platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.
Cited on page 110.

[Moh91] Bojan Mohar. Eigenvalues, diameter and mean distance in graphs. *Graphs and Combinatorics*, 7:53–64, 1991.
Cited on page 72.

[MP02] Damien Magoni and Jean Jacques Pansiot. Analysis and Comparison of Internet Topology Generators. In *Proceedings of the 2nd International IFIP-TC6 Networking Conference*, pages 364–375. Springer, 2002.
Cited on page 148.

[MSM00] Christian Matuszewski, Robby Schönfeld, and Paul Molitor. Using Sifting for k -Layer Straightline Crossing Minimization. In *Proceedings of the 7th International Symposium on Graph Drawing (GD'99)*, volume 1731 of *Lecture Notes in Computer Science*, pages 217–224. Springer, January 2000.
Cited on pages 82, 94, 102, and 105.

[New05] Mark E. J. Newman. A Measure of Betweenness Centrality Based on Random Walks. *Social Networks*, 27(1):39–54, January 2005.
Cited on pages 39, 41, and 42.

[Nor40] Mary L. Northway. A Method for Depicting Social Relationships Obtained by Sociometric Testing. *Sociometry*, 3(2):144–150, April 1940.
Cited on pages 81 and 82.

[Nor54] Mary L. Northway. A Plan for Sociometric Studies in a Longitudinal Programme of Research in Child Development. *Sociometry*, 17(3):272–281, August 1954.
Cited on page 88.

[NSW02] Mark E. J. Newman, Steven H. Strogatz, and Duncan J. Watts. Random graph models of social networks. *Proceedings of the National Academy of Science of the United States of America*, 99:2566–2572, 2002.
Cited on page 47.

[PBM00] R. Poulin, M.-C. Boily, and B.R. Mâsse. Dynamical systems to define centrality in social networks. *Social Networks*, 22:187–220, 2000.
Cited on page 36.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Manuscript, 1999.
Cited on page 45.

[PL51] C.H. Proctor and C. P. Loomis. Analysis of sociometric data. In Marie Jahoda, Morton Deutsch, and Stuart W. Cook, editors, *Research Methods in Social Relations*, pages 561–586. Dryden Press, 1951.
Cited on pages 35 and 36.

[PS06] Adam Perer and Ben Shneiderman. Balancing Systematic and Flexible Exploration of Social Networks. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):693–700, 2006.
Cited on page 39.

[PSV04] Romualdo Pastor-Satorras and Alessandro Vespignani. *Evolution and Structure of the Internet: A Statistical Physics Approach.* Cambridge University Press, 2004.
Cited on page 138.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1992.
Cited on page 46.

[Pur98] Helen C. Purchase. Which Aesthetic has the Greatest Effect on Human Understanding? . In *Proceedings of the 5th International Symposium on Graph Drawing (GD'97)*, volume 1353 of *Lecture Notes in Computer Science*, pages 248–261. Springer, January 1998.
Cited on pages 82, 88, and 93.

[RM88] Marcello G. Reggiani and Franco E. Marchetti. A Proposed Method for Representing Hierarchies. *IEEE Transactions on Systems, Man*

and Cybernetics, 18(1), 1988.
Cited on page 82.

[Rou] University of Oregon Routeviews Project. `http://www.routeviews.org/`.
Cited on pages 126 and 147.

[Rud93] Richard Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, pages 42–47. IEEE Computer Society, 1993.
Cited on page 105.

[Sab66] Gert Sabidussi. The Centrality Index of a Graph. *Psychometrika*, 31:581–603, 1966.
Cited on page 37.

[Sch07] Thomas Schank. *Algorithmic Aspects of Triangle-Based Network Analysis*. PhD thesis, Universität Karlsruhe (TH), Fakultät für Informatik, 2007.
Cited on page 48.

[Sei80] Stephen B. Seidman. Clique-like structures in directed networks. *Journal of Social and Biological Structures*, 3:43–54, 1980.
Cited on page 51.

[Sei83] Stephen B. Seidman. Network Structure and Minimum Degree. *Social Networks*, 5:269–287, 1983.
Cited on pages 49, 124, and 133.

[SF78] Stephen B. Seidman and Brian L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6:139–154, 1978.
Cited on page 51.

[Shi53] Alfonso Shimbel. Structural Parameters of Communication Networks. *Bulletin of Mathematical Biophysics*, 15:501–507, 1953.
Cited on pages 38 and 39.

[SMO$^+$03] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S. Baliga, Jonathan T. Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks. *Genome Biology*, 13:2498–2504, 2003.
Cited on page 25.

[SSSV94] Farhad Shahrokhi, Ondrej Sýkora, László A. Székely, and Imrich Vrto. Book Embeddings and Crossing Numbers. In *Proceedings of the 19th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'93)*, volume 903 of *Lecture Notes in Computer Science*, pages 256–268. Springer, 1994.
Cited on page 101.

[SST02] Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster Graph Modification Problems. In *Proceedings of the 28th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'02)*, volume 2573 of *Lecture Notes in Computer Science*, pages 379–390. Springer, 2002.
Cited on page 55.

[ST99] Janet M. Six and Ioannis G. Tollis. Circular Drawings of Biconnected Graphs. In *Selected Papers from the 1st International Workshop on Algorithm Engineering and Experimentation (ALENEX'99)*, volume 1619 of *Lecture Notes in Computer Science*, pages 57–73. Springer, 1999.
Cited on pages 101, 103, and 109.

[ST04] Janet M. Six and Ioannis G. Tollis. A Framework for User-Grouped Circular Drawings. In Liotta [Lio04], pages 135–146.
Cited on pages 92, 98, and 118.

[Sto03] Maureen Stone. *A Field Guide to Digital Color*. CBMS Regional Conference Series in Mathematics. AK Peters, Ltd., 2003.
Cited on page 77.

[STT81] Kozo Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(2):109–125, 1981.
Cited on pages 82, 87, and 89.

[SW04] Gonen Sagie and Avishai Wool. A Clustering Approach for Exploring the Internet Structure. In *Proceedings of the 23rd IEEE Convention of Electrical and Electronics Engineers in Israel*, pages 149–152. IEEE Computer Society, 2004.
Cited on page 124.

[SZ89] Karen A. Stephenson and Marvin Zelen. Rethinking Centrality: Methods and Examples. *Social Networks*, 11:1–37, 1989.
Cited on page 43.

[Tar72] Robert E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, June 1972.
Cited on page 51.

[tea08] SONIVIS team. SONIVIS Tool Version 0.8, 2008.
Cited on page 25.

[TIAS77] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977.
Cited on page 50.

[Tor52] Warren S. Torgerson. Multidimensional Scaling: I. Theory and Method. *Psychometrika*, 17(4):401–419, December 1952.
Cited on page 70.

[Tuf90] Edward Tufte. *Envisioning Information*. Graphics Press, 1990.
Cited on page 77.

[Tuf07] Edward Tufte. *The Visual Display of Quantitative Information.*, volume 2nd ed. Graphics Press, 2007.
Cited on page 77.

[vD00] Stijn M. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.
Cited on pages 55 and 57.

[VF98] Thomas W. Valente and Robert K. Foreman. Integration and radiality: measuring the extent of an individual's connectedness and reachability in a network. *Social Networks*, 1:89–105, 1998.
Cited on page 38.

[vpt] visone project team. visone - software for the analysis and visualization of social networks. `http://visone.info`.
Cited on page 7.

[WA05] Stefan Wuchty and Eivind Almaas. Peeling the Yeast Protein Network. *Proteomics*, 5(2):444–449, 2005.
Cited on page 138.

[War07] Colin Ware. *Information Visualization, Second Edition: Perception for Design*. Morgan Kaufmann, 2007.
Cited on page 78.

[Why43] William F. Whyte. *Street Corner Society*. University of Chicago Press, 1943.
Cited on pages 87 and 88.

[Wil65] James H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, 1965.
Cited on page 46.

[WJ02] Jared Winick and Sugih Jamin. Inet-3.0: Internet Topology Generator. Technical Report CSE-TR-456, EECS, University of Michigan, 2002.
Cited on page 126.

[WM96] Xiaobo Wang and Isao Miyamoto. Generating Customized Layouts. In *Proceedings of the 3th International Symposium on Graph Drawing (GD'95)*, volume 1027 of *Lecture Notes in Computer Science*, pages 504–515. Springer, January 1996.
Cited on pages 67, 93, and 113.

[WS98] Duncan J. Watts and Steven H. Strogatz. Collective Dynamics of "Small-World" Networks. *Nature*, 393:440–442, 1998.
Cited on pages 47, 133, and 135.

[yWo08a] yWorks GmbH. yEd - Java Graph Editor, 2008.
Cited on page 25.

[yWo08b] yWorks GmbH. yFiles for Java, 2008.
Cited on page 68.

[ZB07] Hongyuan Zha and Christoph Buchheim. A New Exact Algorithm for the Two-Sided Crossing Minimization Problem. In *Proceedings of the 1st International Conference on Combinatorial Optimization and Applications*, volume 4616 of *Lecture Notes in Computer Science*, pages 301–310. Springer, 2007.
Cited on page 89.

Index

AS network, 123, 126, 147
attribute, 23
    network-level, 24

BFS, 20, 51
block, *see* component, biconnected
breadth-first search, *see* BFS

centrality, 35
    authorities, 45
    betweenness, 39
    closeness, 37
    current-flow betweenness, 42
    current-flow closeness, 43
    degree, 36
    eccentricity, 37
    edge betweenness, 40
    eigenvector, 44
    hubs, 45
    in-degree, 36
    information, 43
    out-degree, 36
    PageRank, 45
    radiality, 38
    random-walk betweenness, 42
    status, 44
    stress, 39, 40
    structural index, 35
centralization, 60
clique, 50
    k-clan, 51
    k-clique, 51
clustering, 52
    conductance cuts, 54
    coverage, 53
    modularity, 56
    performance, 55
    quality, 53
clustering coefficient, 47
cohesiveness, 49
component, 20, 34
    connected, 20, 51
    strongly connected, 20, 51
    weakly connected, 20, 51
confirmation, 22, 83, 89
connectedness, *see* component
core decomposition, *see* k-core
crossing reduction, 88, 103, 105, 119
cut, 20, 53, 54, 57
cut-edge, 51
cut-vertex, 51
cycle, 19

degree, 19
density, 19
depth-first search, *see* DFS
DFS, 20, 51
distance, 19
    average, 61
    characteristic, 61
dyad, 18

edge
    direction, 18, 34
    head, 18
    parallel, 18
    source, 18
    tail, 18
    target, 18
edge transformation, 29
ego-network, 28
eigenvalue, 21
eigenvector, 21

geometric transformation, 27
German company network, 64, 97

graph
  automorphism, 20
  complete, 19
  isomorphism, 20
  mixed, 18, 21
  multi, 21
  simple, 18
  size, 35
graph model
  core generator, 137
  preferential attachment, 136, 142
  random $\mathcal{G}(n, p)$, 133
  small world, 135

index, *see* weight
interface
  control panel, 26
  main view, 26
  main window, 25
  menu bar, 25
  status bar, 25
  toolbar, 25

k-core, 49

layout
  barycenter, 126
  circular, 65, 96, 102, 114
  force-directed, 66, 68, 93, 98, 126
  layered layout, 87, 108
  multi-circular, 93, 115
  multidimensional scaling, 69, 98
  radial levels, 114
  random, 65
  spectral, 72, 126
layout quality, 64
length, *see* attribute
loop, 18, 34

matrix
  adjacency, 20, 72
  incidence, 21
  Laplacian, 42, 72, 73
multi-edges, 34
multi-graph, *see* edge, parallel

network statistics, 59

normalization
  percentage, 36
  standardization, 36

partition, 20, 52, 112
partitioning, 93
path, 19
  shortest, *see* shortest path
problem
  SSSP, 20
problems
  all-pairs shortest-path, 38, 61

radius, 61
reciprocity, *see* confirmation

scale, 85, 90, 95
scatter plot, 95
score, *see* weight
shortest path, 19
sifting, 89, 105, 119
single-source shortest paths problem, *see* problem, SSSP
spectrum, 21
strength, *see* attribute
subgraph, 19

template, 27
transitivity, 48

visualization
  color, 77
  coordinate, 95
  grouping, 96
  label, 80
  multi-circular, 91, 98
  radial, 81, 98
  size, 76
  status, 87
  z-coordinate, 81

walk, 19
weight, 18, *see* attribute, 34

Appendix

Experimental Results of the Circular Layout Algorithm

Due to size constraints, figures start on the following page.

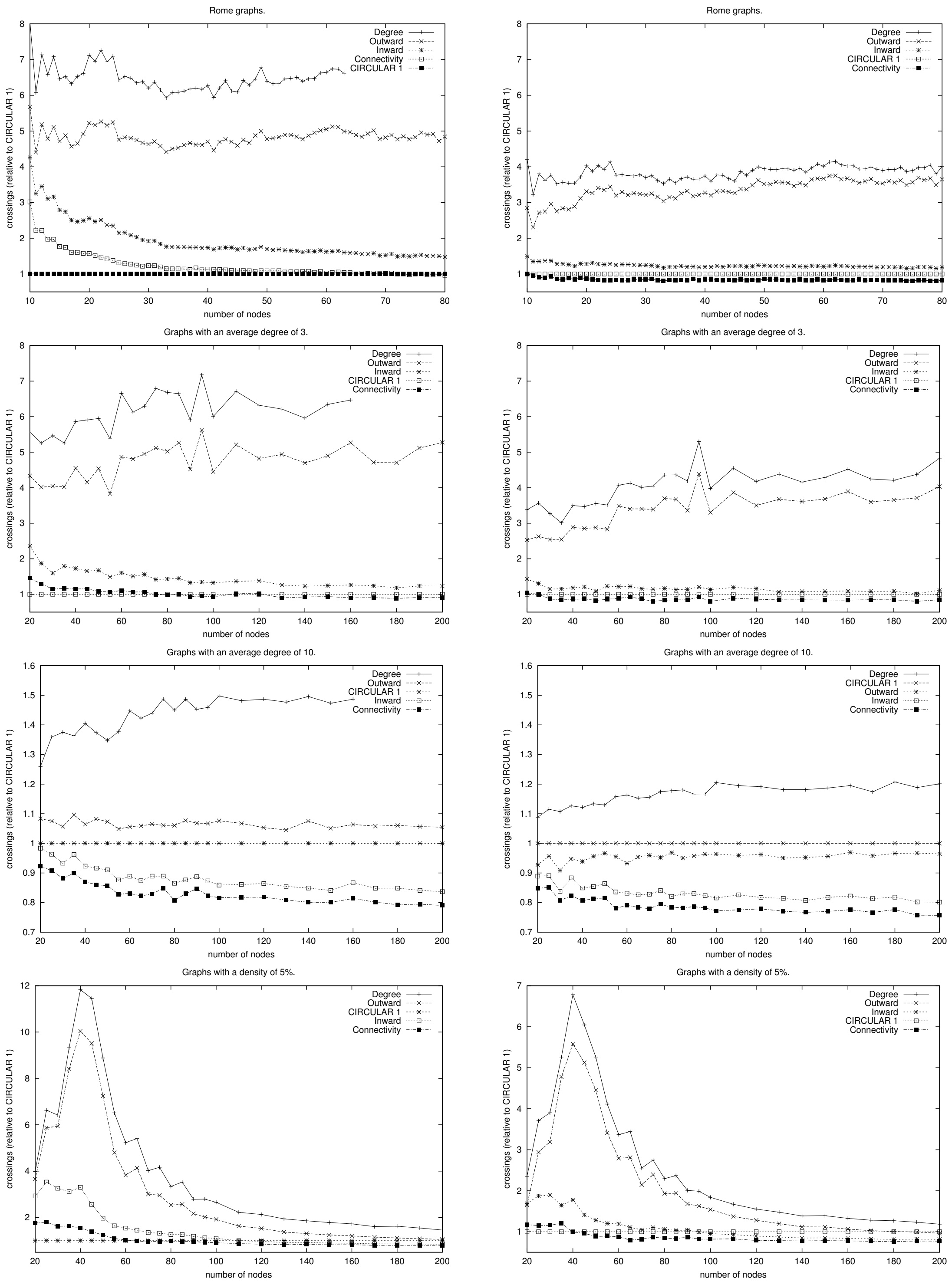


Figure 8.7: Greedy append: insertion orders combined with *Fixed* (left column) and *Crossings* (right column) placement rules

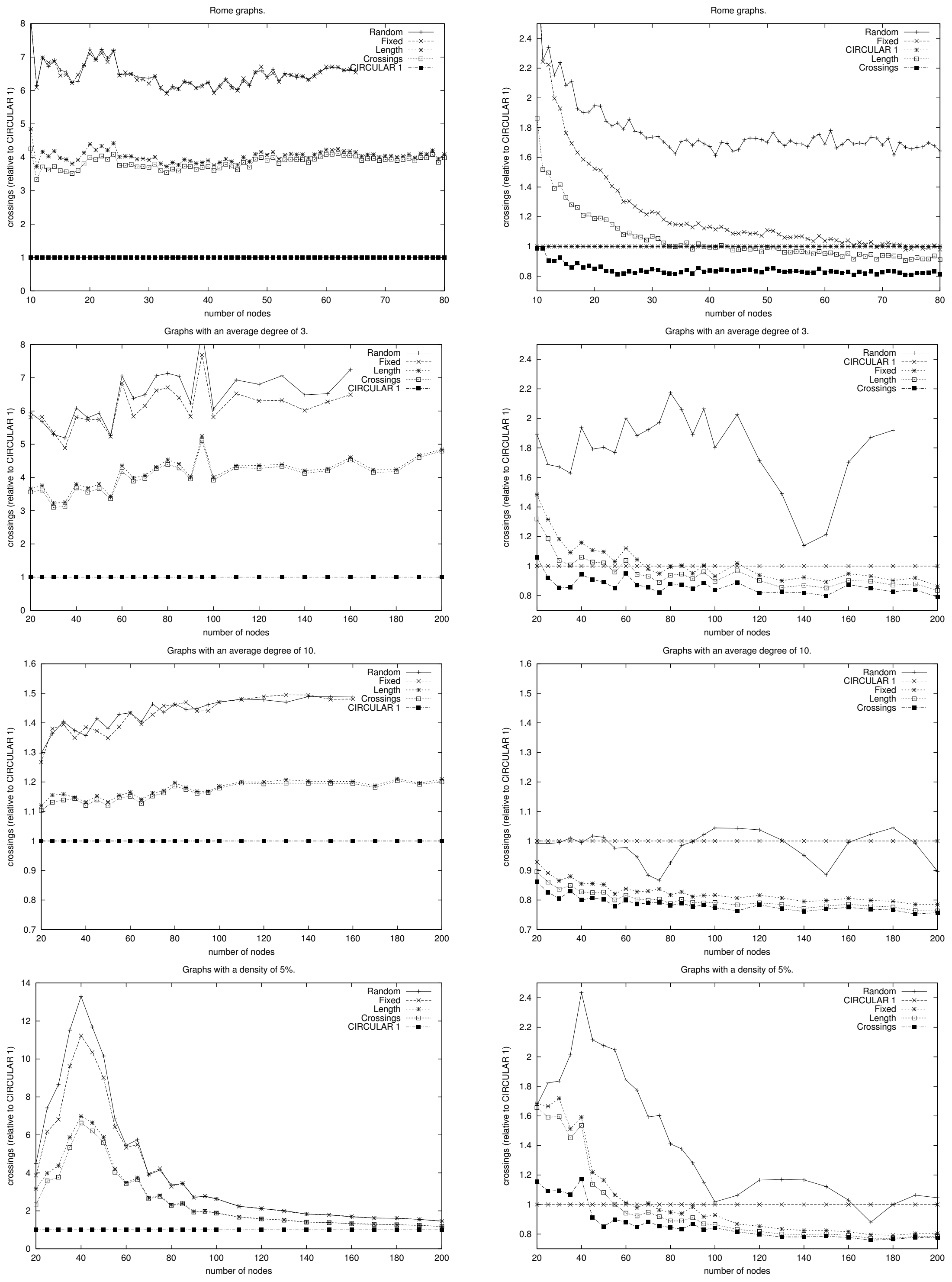
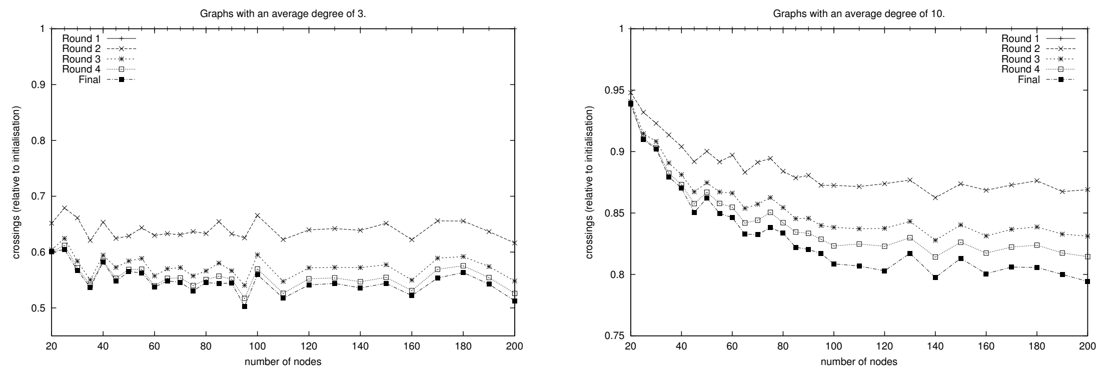
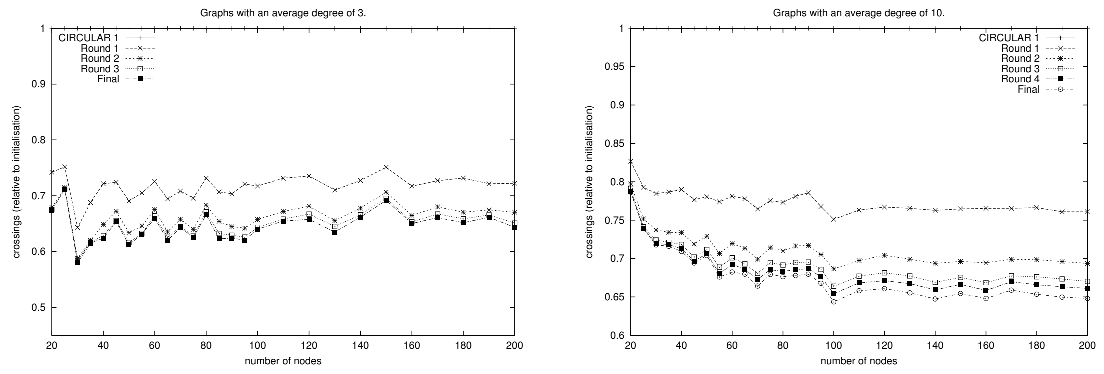


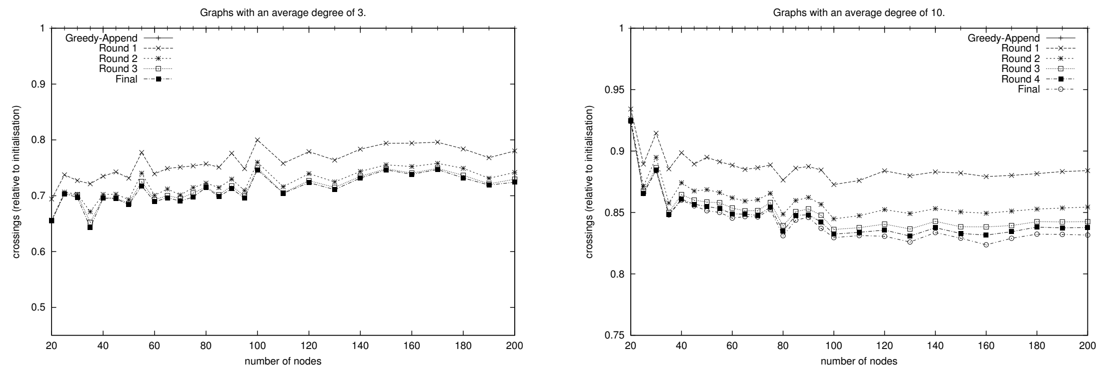
Figure 8.8: Greedy append: placement rules combined with *Degree* (left column) and *Connectivity* (right column) insertion orders



(a) Sifting without initialization.



(b) Sifting in combination with CIRCULAR 1.



(c) Sifting in combination with greedy append.

Figure 8.9: Circular sifting: improvement after various rounds

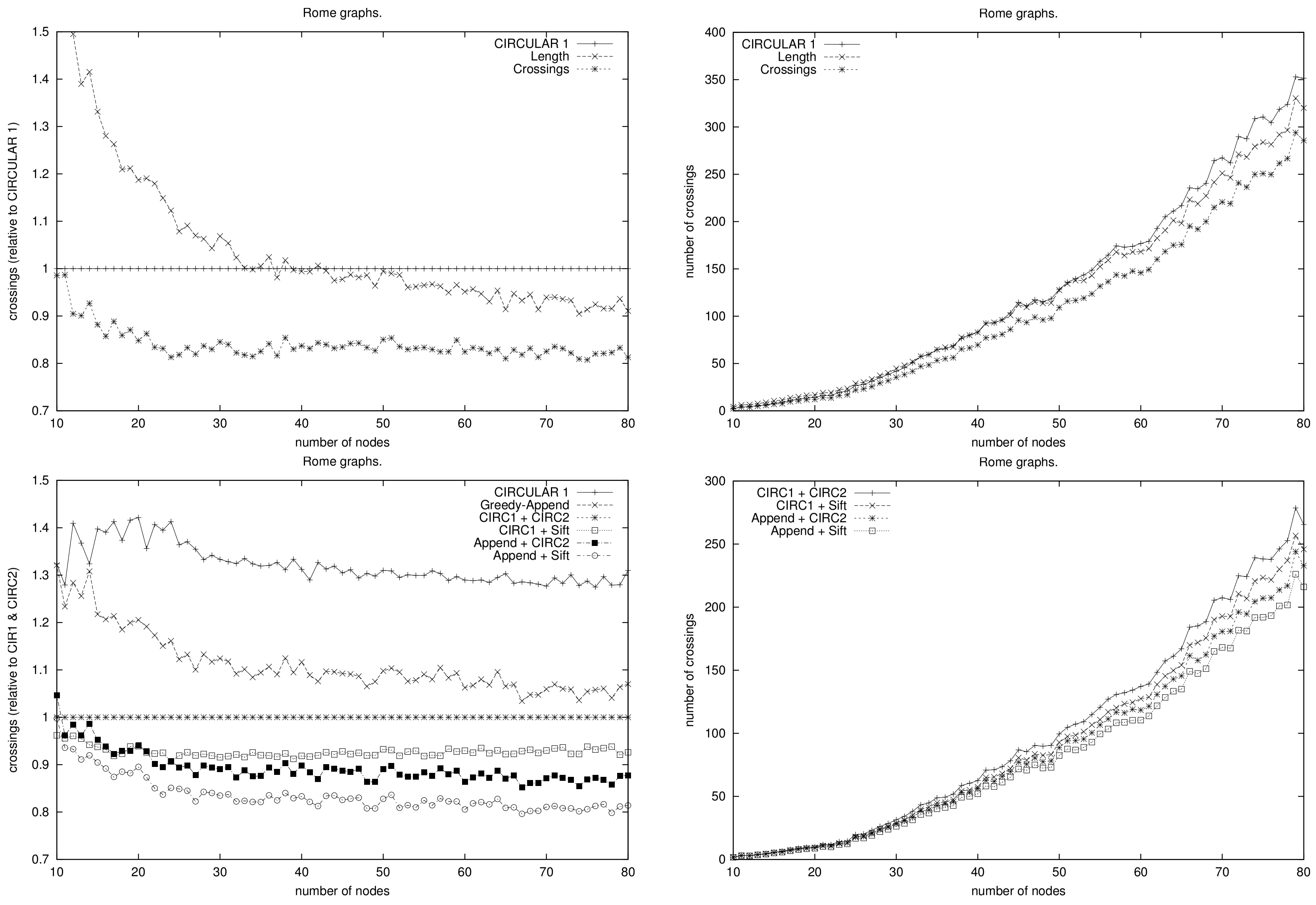


Figure 8.10: Results on the “Rome graphs”, a commonly used benchmark data set

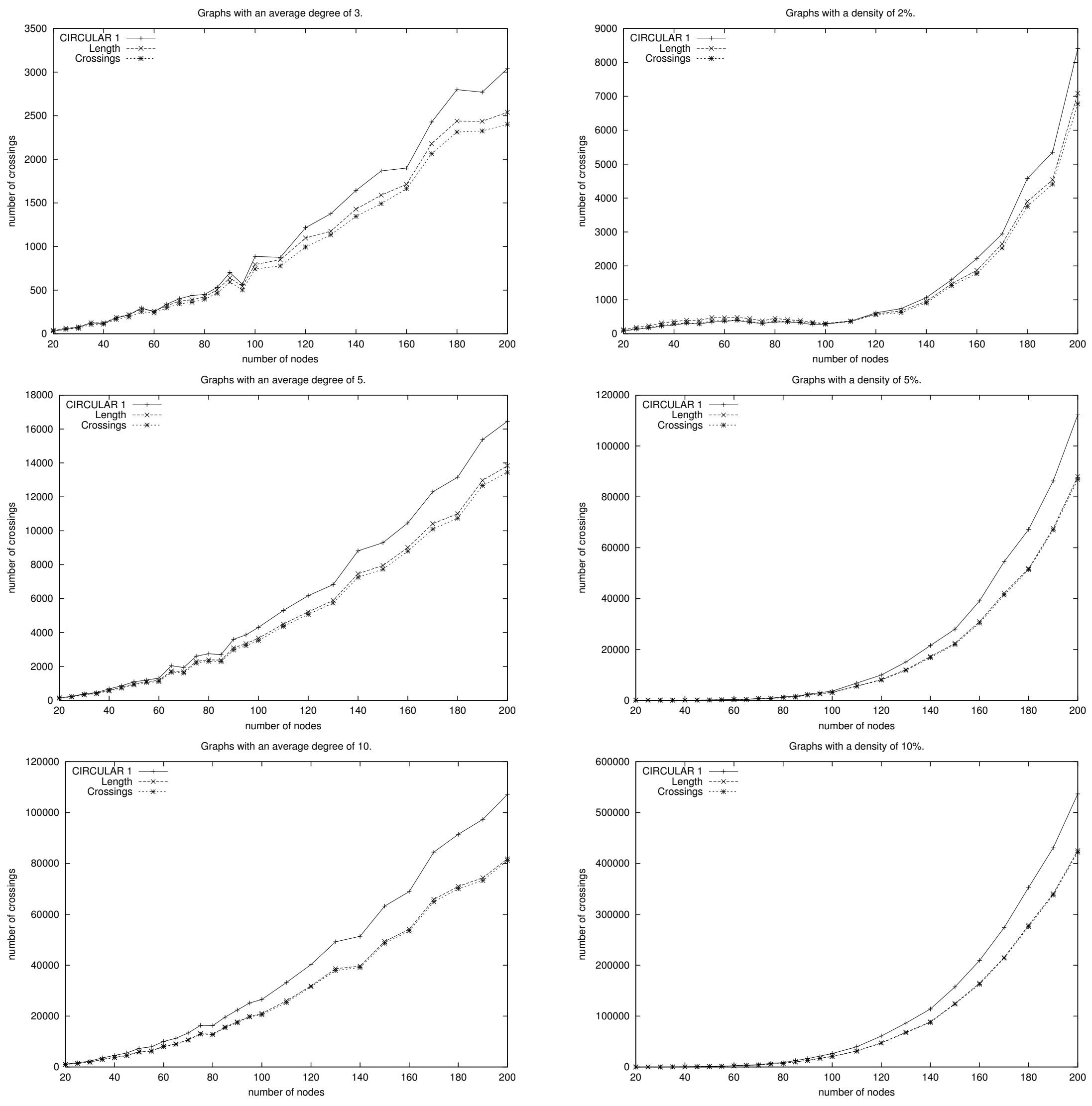


Figure 8.11: Initial layout: CIRCULAR 1, and Length and Crossings combined with Connectivity

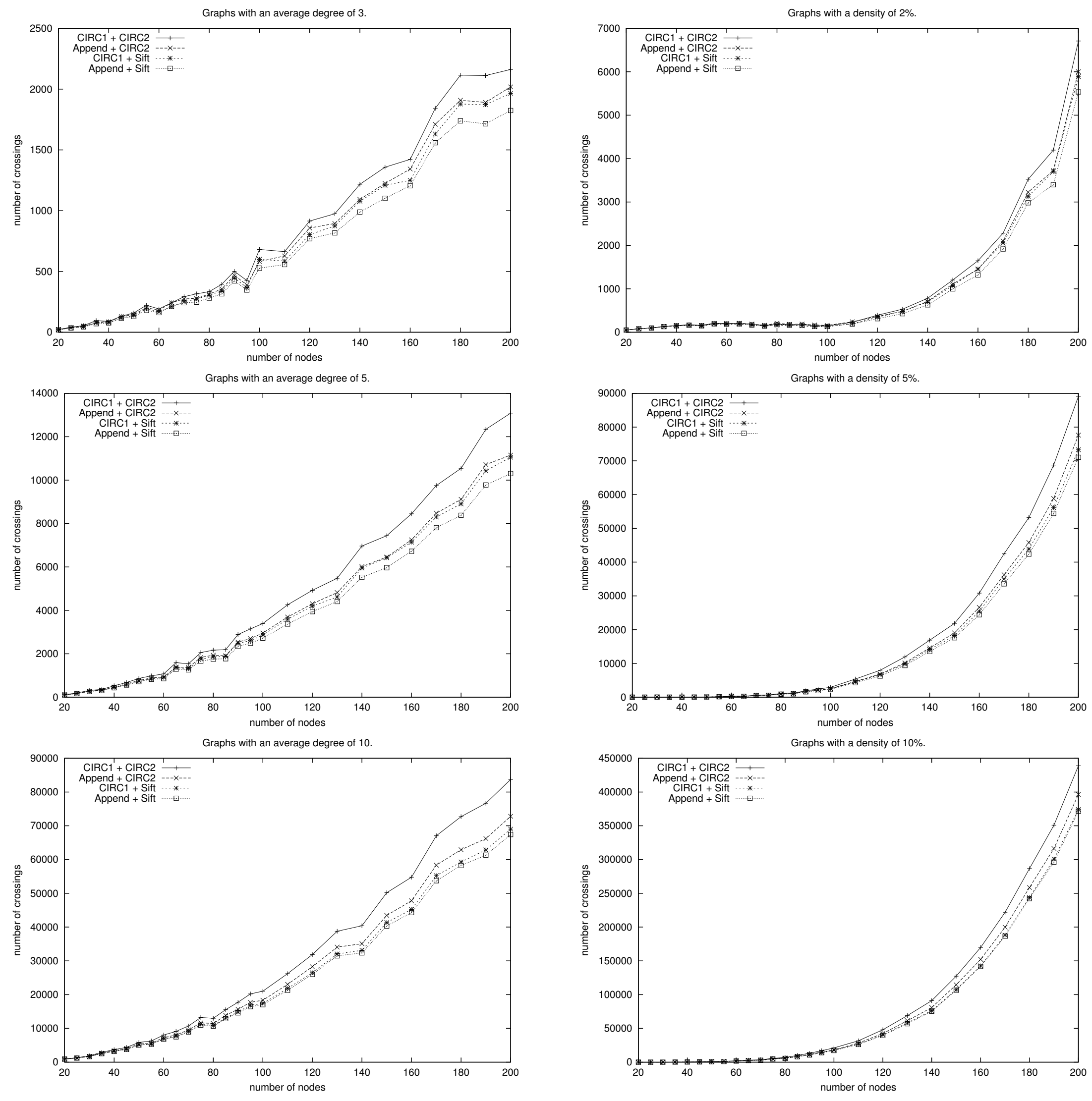


Figure 8.12: Improvement phase: various combinations of initial and improvement algorithms

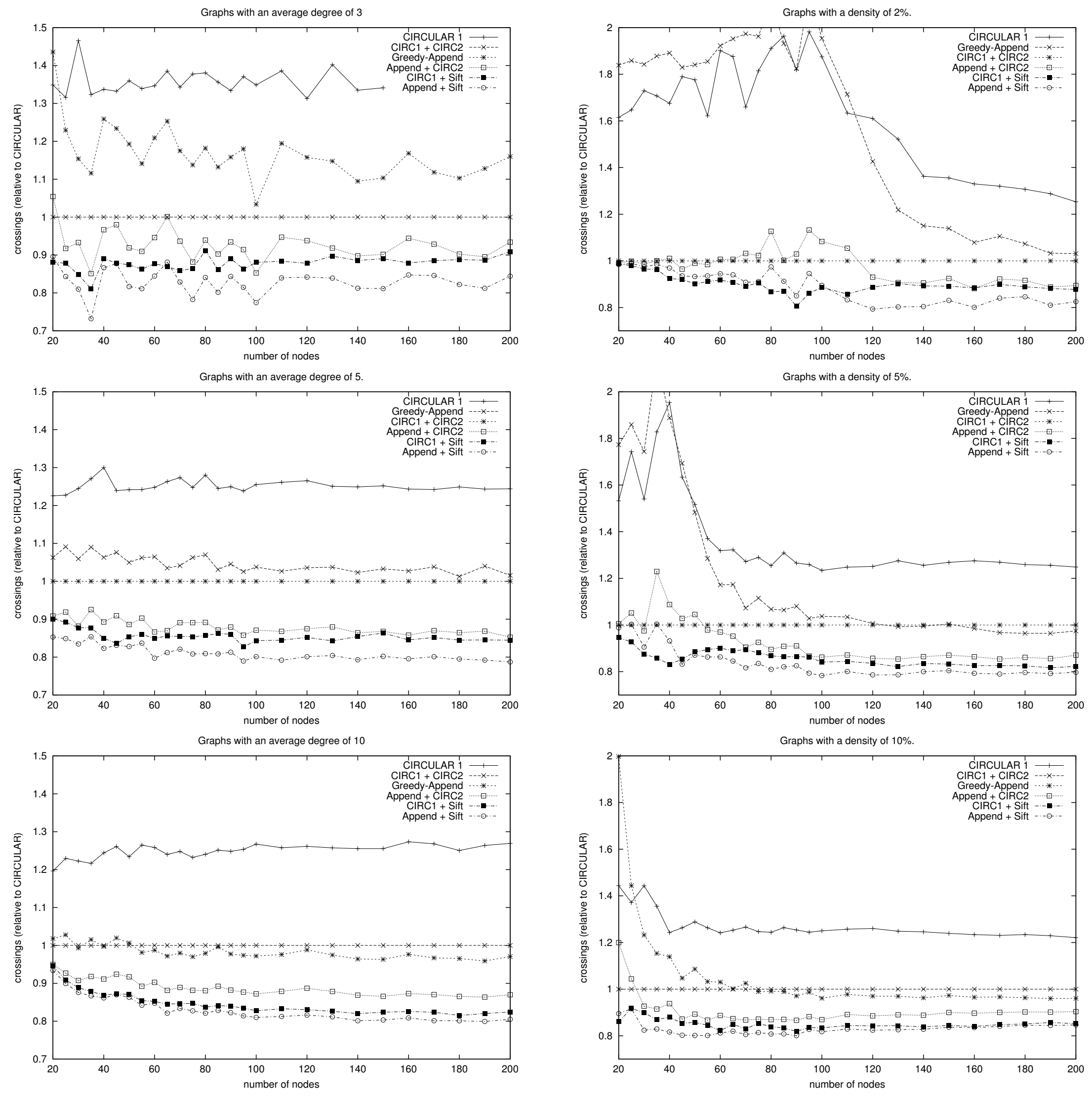
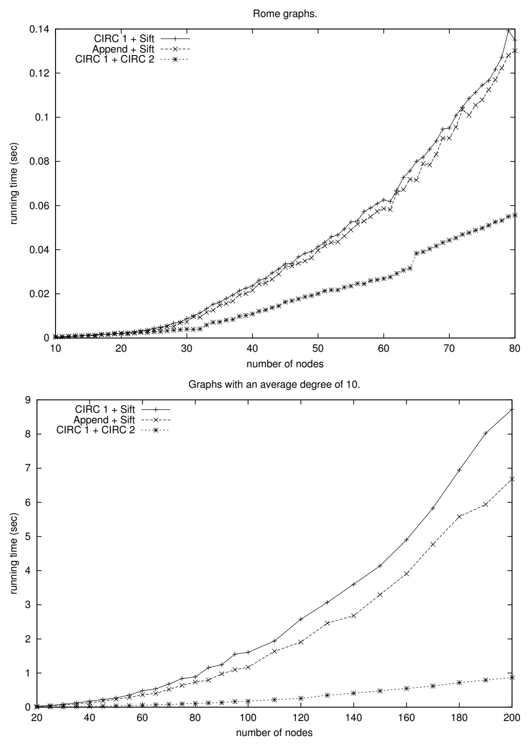


Figure 8.13: Results on random graphs relative to CIRCULAR



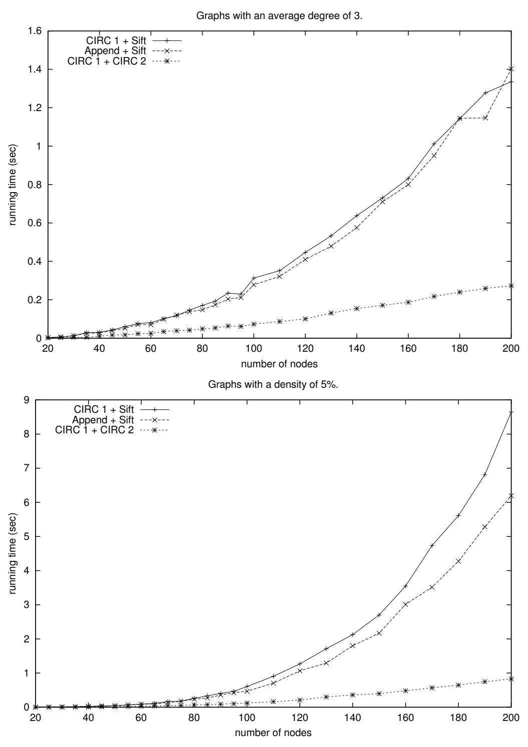
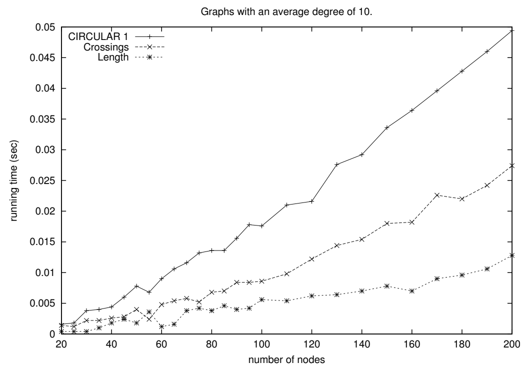


Figure 8.14: Running time: combinations of initial and improvement algorithms



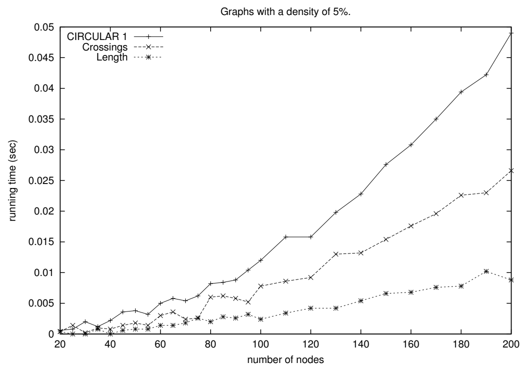


Figure 8.15: Running time: initial algorithms

Résumé

Michael Baur was born in Ravensburg in 1977. He obtained his diploma degree in Mathematics (majoring in Computer Science) from the University of Konstanz in 2003; the thesis's title is *"Kantenkreuzungen in Kreislayouts"* (*"Crossings in Circular Layouts"*). Already during his studies, in 2000, he started working as a student helper in the visone project. After his graduation, he continued work in this project as part of his Ph.D. studies as a research assistant in the group of Prof. Dr. Dorothea Wagner at Universität Karlsruhe (TH). These Ph.D. studies were financially supported by the priority programme 1126 *"Algorithmics of Large and Complex Networks"* of Deutsche Forschungsgemeinschaft (DFG).

Since 2005, Michael Baur has been part of the faculty of *"POLNET"*, an international summer school on the analysis of social networks, where he gave the lecture on mathematical foundations and supervised the practical course about UCINET and visone. In the same year, he established together with his colleague Alexander Paar the regular annual participation of students from Universität Karlsruhe (TH) in the ACM International Collegiate Programming Contest (ICPC).