Interactive Graph Drawing with Constraints

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Karsten Klein

Dortmund

2011

Tag der mündlichen Prüfung: **14.03.2011**

Dekan:
Prof. Dr. Peter Buchholz

Gutachter:
Prof. Dr. Petra Mutzel, Technische Universität Dortmund
Prof. Dr. Stephen G. Kobourov, University of Arizona

Abstract

This thesis investigates the requirements for graph drawing stemming from practical applications, and presents both theoretical as well as practical results and approaches to handle them.

Many approaches to compute graph layouts in various drawing styles exist, but the results are often not sufficient for use in practice. Drawing conventions, graphical notation standards, and user-defined requirements restrict the set of admissible drawings. These restrictions can be formalized as constraints for the layout computation. We investigate the requirements and give an overview and categorization of the corresponding constraints.

Of main importance for the readability of a graph drawing is the number of edge crossings. In case the graph is planar it should be drawn without crossings, otherwise we should aim to use the minimum number of crossings possible. However, several types of constraints may impose restrictions on the way the graph can be embedded in the plane. These restrictions may have a strong impact on crossing minimization. For two types of such constraints we present specific solutions how to consider them in layout computation:

We introduce the class of so-called *embedding constraints*, which restrict the order of the edges around a vertex. For embedding constraints we describe approaches for planarity testing, embedding, and edge insertion with the minimum number of crossings. These problems can be solved in linear time with our approaches.

The second constraint type that we tackle are clusters. Clusters describe a hierarchical grouping of the graph's vertices that has to be reflected in the drawing. The complexity of the corresponding *clustered planarity testing problem* for clustered graphs is unknown so far. We describe a technique to compute a maximum clustered planar subgraph of a clustered graph. Our solution is based on an Integer Linear Program (ILP) formulation and includes also the first practical clustered planarity test for general clustered graphs. The resulting subgraph can be used within the first step of the planarization approach for clustered graphs. In addition, we describe how to improve the performance for pure clustered planarity testing by implying a branch-and-price approach.

Large and complex graphs nowadays arise in many application domains. These graphs require interaction and navigation techniques to allow exploration of the underlying data. The corresponding concepts are presented and solutions for three practical applications are proposed: First, we describe Scaffold Hunter, a tool for the exploration of chemical space. We show how to use a hierarchical classification of molecules for the visual navigation in chemical space. The resulting visualization is embedded into an interactive environment that allows visual analysis of chemical compound databases. Finally, two interactive visualization approaches for two types of biological networks, protein-domain networks and residue interaction networks, are presented.

Kurzzusammenfassung

In zahlreichen Anwendungsgebieten werden Informationen als Graphen modelliert und mithilfe dieser Graphen visualisiert. Eine übersichtliche Darstellung hilft bei der Analyse und unterstützt das Verständnis bei der Präsentation von Informationen mittels graph-basierter Diagramme. Neben allgemeinen ästhetischen Kriterien bestehen für eine solche Darstellung Anforderungen, die sich aus der Charakteristik der Daten, etablierten Darstellungskonventionen und der konkreten Fragestellung ergeben. Zusätzlich ist häufig eine individuelle Anpassung der Darstellung durch den Anwender gewünscht. Diese Anforderungen können mithilfe von Nebenbedingungen für die Berechnung eines Layouts formuliert werden. In dieser Arbeit untersuchen wir die Anforderungen aus der Praxis und beschreiben eine Zuordnung zu Nebenbedingungen für die Layoutberechnung. Wir geben eine Übersicht über den aktuellen Stand der Behandlung von Nebenbedingungen beim Zeichnen von Graphen und kategorisieren diese nach grundlegenden Eigenschaften.

Von besonderer Wichtigkeit für die Qualität einer Darstellung ist die Anzahl der Kreuzungen. Planare Graphen sollten kreuzungsfrei gezeichnet werden, bei nicht-planaren Graphen sollte die minimale Anzahl Kreuzungen erreicht werden. Einige Nebenbedingungen beschränken jedoch die Möglichkeit, den Graph in die Ebene einzubetten. Dies kann starke Auswirkungen auf das Ergebnis der Kreuzungsminimierung haben. Zwei wichtige Typen solcher Nebenbedingungen werden in dieser Arbeit näher untersucht. Mit den *Embedding Constraints* führen wir eine Klasse von Nebenbedingungen ein, welche die mögliche Reihenfolge der Kanten um einen Knoten beschränken. Für diese Klasse präsentieren wir Linearzeitalgorithmen für das Testen der Planarität und das optimale Einfügen von Kanten unter Beachtung der Einbettungsbeschränkungen. Der zweite Typ von Nebenbedingungen sind *Cluster*, die eine hierarchische Gruppierung von Knoten vorgeben. Für das Testen der Cluster-Planarität unter solchen Nebenbedingungen ist die Komplexität bisher unbekannt. Wir beschreiben ein Verfahren, um einen maximalen Cluster-planaren Untergraphen zu berechnen. Wir nutzen dabei eine Formulierung als ganzzahliges lineares Programm sowie einen Branch-and-Cut Ansatz zur Lösung. Das Verfahren erlaubt auch die Bestimmung der Cluster-Planarität und stellt damit den ersten praktischen Ansatz zum Testen allgemeiner Clustergraphen dar. Zusätzlich beschreiben wir eine Verbesserung für den Fall, dass lediglich Cluster-Planarität getestet werden muss, der maximale Cluster-planare Untergraph aber nicht von Interesse ist. Für dieses Szenario geben wir eine vereinfachte Formulierung und präsentieren ein Lösungsverfahren, das auf einem Branch-and-Price Ansatz beruht.

In der Praxis müssen häufig sehr große oder komplexe Graphen untersucht werden. Dazu werden entsprechende Interaktions- und Navigationsmethoden benötigt. Wir beschreiben die entsprechenden Konzepte und stellen Lösungen für drei Anwendungsbereiche vor: Zunächst beschreiben wir Scaffold Hunter, eine Software zur Navigation im chemischen Strukturraum. Scaffold Hunter benutzt eine hierarchische Klassifikation von Molekülen als Grundlage für die visuelle Navigation. Die Visualisierung ist eingebettet in eine interaktive Oberfläche die eine visuelle Analyse von chemischen Strukturdatenbanken erlaubt. Für zwei Typen von biologischen Netzwerken, Protein-Domänen Netzwerke und Residue-Interaktionsnetzwerke, stellen wir Ansätze für die interaktive Visualisierung dar. Die entsprechenden Layoutverfahren unterliegen einer Reihe von Nebenbedingungen für eine sinnvolle Darstellung.

Acknowledgements

Many people supported me during the work on this thesis, and I would like to thank all of them. My special thanks go to Petra Mutzel and Gunnar Klau for introducing me to the field of Graph Drawing, working with them raised my interest in both the theoretical and practical aspects of this field. Without their enthusiasm and advice, this work would not have been possible. I would like to express my gratitude to Petra for the opportunity to pursue my research in her research group at the Technische Universität Dortmund. I thank my colleagues at the Algorithm Engineering chair, especially my roommate Carsten Gutwenger, for the good atmosphere and the interesting discussions. Collaboration of the following people on the various results in this thesis is also gratefully acknowledged: Stefan Wetzel for the work on Scaffold Hunter, and Mario Albrecht for the work on biological networks. I thank both of them for sharing their biochemical knowledge. I also thank Nils, Bernd, Carsten, Hoi-Ming, and Markus for proof reading parts of this work, and our secretary Gundel Jankord for doing all the administrative work and for help in so many things. I would like to thank Lars Hildebrand, Bernhard Steffen, Stephen Kobourov, and Petra Mutzel for agreeing to serve as members of my defense committee. A special thank you goes to Stephen for making it possible to attend my defense.

Thanks to all who gave permission to use their images or data in this thesis, they are acknowledged individually where appropriate.

Finally, a big thank you goes to Andrea, Carsten, Martin, Monika, and Stefan for being what they are - true friends.

CONTENTS

LIST OF FIGURES

Part I

BASICS

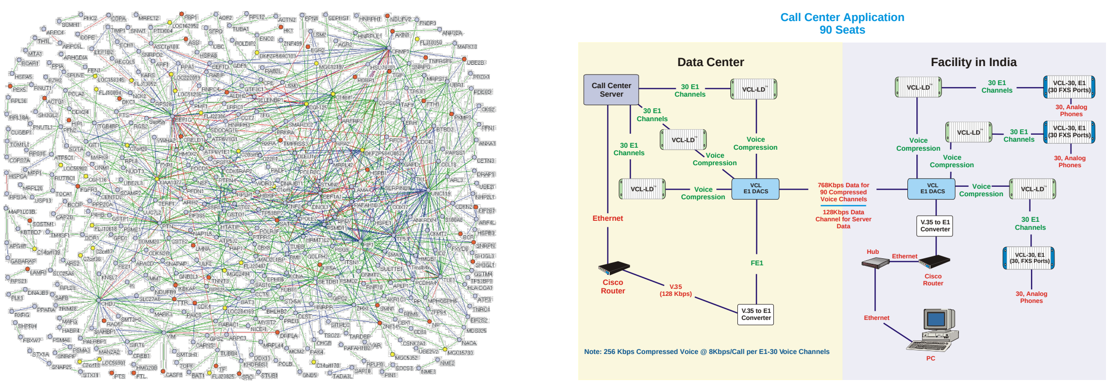
1. INTRODUCTION

In graph drawing, everything is said and done.

Falk Schreiber, 2005

In many application domains, lots of data is collected with only partial knowledge of the data's structure and the information it contains. Experimental data from biological processes or telecommunication data is nowadays relatively easy collected and stored, but hard to analyze and understand afterwards. Often, the data can be modeled in the form of a graph or more general structures, as, e.g., clustered graphs. The structure of such a graph might be given naturally by the inherent structure of the data, as in biological or social interaction networks, or by modeling processes, as, e.g., workflows, or it might be derived using an application-specific model.

Visualizing these graphs can greatly enhance the understanding of the underlying data, its inherent structures and patterns, and may therefore support the knowledge discovery process during analysis. As a consequence, graph-based diagrams are a widely used means for the graphical representation of complex data. Many information systems today do not only display diagrams in order to represent information, but allow the user to model the underlying data through interaction with the graphical interface. Modeling systems in such diverse areas like Software Engineering, Bioinformatics, Database Design, and Business Process Modeling using such interfaces have become standard tools for the development, maintenance and documentation of commercial as well as scientific projects. New notation standards and



(a) (b)

Figure 1.1: Graph layouts are used in a wide variety of applications. (a) shows a protein-protein interaction network as visualized in Stelzl et al. [2005], (b) shows the connection structure of a call center application [Orion].

methods in graphical modeling, such as BPMN [a] for business process modeling, UML [b] for software modeling, or SBGN [Novere et al., 2009] for modeling of biological networks, led to an increasing importance of graph representations.

Graph visualization can be done for three main goals:

- **Communication** to convey existing knowledge, as for example by static text book presentations.
- **Visual exploration** of unknown data, where interactive graph representations are often used in combination with analytical methods, e.g., from statistics, and linked with further visualizations like heat-maps, plots, etc.
- **Modeling** to create new data or modify data. The resulting visualizations can however also be used for communication.

Due to the growing demand for clear and concise visualizations of increasingly complex data models, automatic diagram layout is key to high quality representations and has become an integral part of many software systems. The scientific field *automatic graph drawing* deals with the computation of graph layouts in order to draw them in the form of diagrams in the plane in a comprehensible and meaningful fashion. Besides the optimization of a number of common aesthetic criteria, as for example the minimization of edge crossings, the visualization for real-world purposes poses additional challenges. In many application areas common representation conventions exist, and additional semantic annotations of the graph objects have to be considered for visualization. In addition, established work-flows and notation standards may impose requirements on the way a graph is laid out. Thus, constraints for the computation of graph layouts arise that need to be respected in order to achieve a meaningful and accepted visualization in the context of the application domain.

Whereas a large number of approaches exists to produce optimized graph drawings in various drawing styles with respect to a small number of standard optimization criteria, the incorporation of constraints is often done in a none-systematic fashion with ad-hoc approaches. Typically, existing layout methods are extended to cope with the most important drawing constraints. The choice of the algorithm is mainly influenced by the availability of code and the ease of adaption instead of the appropriateness and the quality of the final result. As a result, the handling of constraints has not yet found its way into common application software and is restricted mainly to research prototypes.

There are many general purpose diagramming software solutions, prominent examples are Microsoft Visio, Eclipse GMF, and Sybase PowerDesigner, and an even larger number of application specific tools that allow modeling with diagrams. Such tools often already include simple automatic layout facilities, but the results usually need to be improved manually to a rather large extent for presentation or publication purposes. Whereas mainstream software only provides simple automatic layout capabilities, state-of-the-art methods are mostly implemented for open-source projects that allow extension by simple plugin mechanisms, as, e.g., Cytoscape or Eclipse. Often, new methods are only implemented in tools that serve as an experimentation platform for research groups and medium-term projects, which neither have a significant number of active users nor are reliably maintained. Therefore the impact of such implementations is often small and there is a repeated effort to implement methods, as they are not available in easy-to-use graph drawing libraries.

An additional difficulty arises from the fact that in practice the graph sizes may range from just a few to millions of vertices, and due to the theoretical difficulty of the computational problems involved, efficient and suitable heuristics with acceptable quality have to be found for different practical purposes. Fortunately, depending on the goal of the visualization, the graph layout may not need to be concise for each part of the graph and display region. During a drill-down navigation for example, only significant parts of the graph near the navigation path have to be shown in detail, whereas the remaining parts can be represented as less accurate context information for navigation, or even be omitted. This raises several questions about how the user can interact with the system, how the abstraction of the data can be done to allow orientation, and how the change in the visualized data during navigation can be managed so that graph layouts are both clear and preserve the user's mental map.

Within this thesis, we will focus on graph layouts in the plane for human consumption. even though there are other important applications for automatic graph layout, as for example electronic circuit design.

Beyond graph layouts and the underlying algorithmic foundations, there are many additional aspects that need to be considered in order to provide adequate interactive graph visualization techniques, as for example human perception and human-computer interaction issues. Even though they are important, we will not address them here and focus on graph layout computation. We refer the interested reader to the corresponding literature [Ware, 2004, Chen, 2006].

1.1 Motivation

Everybody attached to graph drawing, especially people getting direct feedback from practitioners, knows, that there are two important problems for the practical application of theoretical results: First, there are often a number of requirements specified by the user and the application that do not always fit in the standard approach that optimizes a small set of aesthetic criteria. Second, automatic layout computation has to cope with the gap between the optimization of layouts for graphs changing over time and the users' wish for stability of the layout in order to keep familiar with it.

Above a common base of standard criteria for aesthetical visualizations, the three main reasons for graph drawing (as stated above) have their own requirements for graph representations and layout computation: Visual exploration requires fast responsive layout methods, and should allow to detect patterns and symmetry. As a graph cannot be used to model all aspects of a data set or system, graph representations often are embedded into a software environment that combines several types of views on the data. The requirements for such combinations may differ from a stand-alone graph drawing solution. Besides pure visualization techniques, we may also need to integrate analysis and data mining techniques for knowledge discovery processes.

In an interactive exploration scenario, due to the dynamic change in the data displayed and the short time the layout is used, the quality of a layout may still be acceptable even if clearly suboptimal, as long as the user's orientation is preserved and the layout computation is reasonably fast. In contrast, for presentation purposes a 'perfect' view on a typically small set of objects, that often has to fit a specific format or aspect ratio, is highly desirable.

As a modeling process can proceed over a longer period of time, where the graph model and representation changes over time and is stored persistently, modeling needs stability in subsequent applications of a layout algorithm, i.e., incremental layout capabilities. Established drawing conventions and notation standards need to be respected to create accepted visualizations that are readily understandable. In addition, the incorporation of user-defined constraints has to be allowed, as the user might want to arrange at least parts of the graph layout according to his preferences, and these changes have to be preserved.

In summary, depending on the application context, the task, and the data, quite different requirements may define what makes a good visualization. However, by an investigation of these requirements we can try to reduce them to a set of common standard constraints which allow to draw graphs for a variety of drawing conventions and application tasks. The purpose of this thesis is to collect and categorize requirements stemming from practical application areas, to formalize them as layout constraints, to review the current state of the art in interactive graph drawing with constraints, and to discuss solutions for selected problems from both theory and practice.

Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested and refined to the point where they can be usefully applied in practice. [...] to increase the impact of theory on key application areas.

AHO ET AL. [1997]

1.2 Organization of this Thesis

This thesis is organized as follows. After giving some basic definitions and results from graph theory and graph drawing in Chapter 2, Chapter 3 deals with the main application areas for graph drawing and collects requirements that arise from the use of graph visualizations in practice. In Chapter 4, we review graph drawing concepts with a focus on the evaluation of layout quality.

Chapter 5 gives an extensive overview and a classification of constraints in graph drawing and deals with their role and handling in automatic graph drawing. Chapter 6 introduces embedding constraints which impose restrictions on the order of edges around a vertex. We describe a linear time algorithm that tests if a graph given with a set of embedding constraints has a planar embedding that observes these constraints, and allows to compute such an embedding in the positive case. We also give a linear time algorithm for the edge insertion problem under embedding constraints. In Chapter 7, we examine a further important class of constraints. In clustered graph drawings, a hierarchical grouping of the vertices needs to be reflected. We review the state of the art and present approaches to tackle the clustered planarity problem, a longstanding open problem in graph drawing. The computational complexity of this problem is still unknown. The maximum c-planar subgraph problem is introduced, and a branch-and-cut approach to solve this problem is given. In addition, we describe how to adapt this approach to obtain a branch-and-price approach for clustered planarity testing. We then review a result by Dahlhaus, the first linear time clustered planarity testing algorithm for c-connected graphs, and give a new description based on SPQR-trees.

In Chapter 8 we review interaction and navigation concepts. Chapter 9 introduces three solutions for visual analysis of biological and chemical data. Scaffold Hunter is a tool for the interactive exploration of chemical space that uses a hierarchical classification of chemical molecules for navigation. RINalyzer allows the combined analysis of 2D and 3D protein structures. DomainGraph uses an interactive visualization of protein-domain networks to facilitate the investigation of the impact of alternative splicing events on protein function and structure.

Finally, Chapter 10 concludes by recapitulating the results presented in this thesis and their impact on current research, and gives an outlook on future work.

1.3 Corresponding Publications

The results in this thesis have partially been published in conference proceedings, journals, and technical reports. This section lists these publications with references to the corresponding chapters of this thesis.

- The results on graph embedding and edge insertion with embedding constraints (see Chapter 6) were first published in the conference proceedings of *Graph Drawing 2006* [Gutwenger et al., 2006] and an extended version then appeared in the *Journal of Graph Algorithms and Applications* (Special Issue on Selected Papers from GD 2006) [Gutwenger et al., 2008].

- The results on the maximum c-planar subgraph computation (see Section 7.2) appeared in the conference proceedings of *Graph Drawing 2008* [Chimani et al., 2008a].

- The new description of the linear time c-planarity test (see Section 7.4) is partially based on the publication of Dahlhaus [1998], and was published as a technical report [Dahlhaus et al., 2006].

- A description of the interactive exploration tool Scaffold Hunter (see Section 9.1), together with a proof-of-concept study, was published in the journal *Nature Chemical Biology* [Wetzel et al., 2009].

- The visualization approach for protein-domain networks (see Section 9.2) was first published in the conference proceedings of *MediVis08* [Emig et al., 2008b], an extended version appeared in the *Journal of Integrative Bioinformatics* [Emig et al., 2008a].

- The approach for the representation and analysis of residue interaction networks (see Section 9.3) appeared in the journal *Trends in Biochemical Sciences* [Doncheva et al., 2011].

1.4 Related Work

Two further published results are not described in detail here, but used in the sections on the requirements in application areas (see Section 3.2) and on Scaffold Hunter (see Section 9.1).

- In Albrecht et al. [2009], we characterize the problems that arise in the visualization of biological networks and formulate them as open graph drawing problems. In addition, the problems are discussed and possible solutions are suggested. Some of these results are used in the description of the requirements in application areas (see Section 3.2).
- In Klein et al. [2011], we describe an approach for efficient subgraph search in large databases. The approach is well suited especially for chemical compound databases and implemented as part of the Scaffold Hunter software (see Section 9.1).

2. PRELIMINARIES AND NOTATION

In this section we will review the notations and main graph theoretic and mathematical concepts that we will use in the remainder of this work. We use the mathematical model of a graph for the representation of the structural properties of a diagram. For a comprehensive overview on graph terminology and graph theory, see for example the textbooks by Jünger and Mutzel [2003] and Diestel [2010], as most of the notation here is derived from these sources. The following two sections cover basic definitions concerning graphs and graph drawings, Section 2.3 covers graph embeddings and planarity, Section 2.4 deals with concepts and data structures for graph decomposition, Section 2.5 shortly describes research areas related to the topic of this thesis.

2.1 Graphs

An *(undirected) graph* is a pair of two disjoint finite sets $G = (V, E)$, where V is the set of *vertices* and E is the set of *edges* of G. An edge $e \in E$ is an unordered pair u, v of vertices of G, its *end-vertices*. Vertices u and v then are said to be *adjacent*, and e and any of its end-vertices are said to be *incident*. The *degree* of a vertex is the number of its incident edges. If two or more edges have the same end-vertices, they are called *multiple edges*, and if $u = v$, e is called *self-loop*. The number of vertices $|V|$ is denoted by n in the following, the number of edges $|E|$ by m. In a *directed graph*, an edge e is an ordered pair (u, v) of vertices, e *leaves* u and *enters* v, it is directed from u to v and an *outgoing (incoming)* edge of u (v). All graphs in the following will be regarded as undirected if not mentioned otherwise. A *hypergraph* is a pair $H = (V, E)$ of disjoint sets, where the elements of E are non-empty subsets (of any cardinality) of V. A graph $G' = (V', E')$ is a *subgraph* of graph $G = (V, E)$, and G a *supergraph* of G', if $V' \subseteq V$ and $E' \subseteq E$. We call G' a *spanning subgraph* of G if $V' = V$. An induced subgraph is a subgraph such that $e = (v, w) \in E'$ for $v, w \in V' \iff e \in E$.

A *path* from vertex v to vertex u in a graph G is an alternating sequence of vertices and edges $v = v_0, e_1, v_1, e_2, \dots, e_k, v_k = u$ with $k \geq 0$ such that v_{i-1} and v_i are adjacent to e_i for $i = 1, \dots, k$. A path is *simple* if all vertices are distinct. A non-empty graph G is called *connected* if for each pair (u, v) of its vertices there is a path between u and v in G (*disconnected* otherwise). A maximal connected subgraph of a graph G is a *connected component* of G. If there are three distinct vertices u, v, w such that w lies on every path between u and v, then w is called a *cut vertex*. Obviously a *cut vertex* is a vertex whose removal disconnects the graph. An edge $e = (v, w)$ is called a *bridge*, if e lies on every path between v and w. For an integer $k \geq 0$, $G = (V, E)$ is called *k-connected* if $|V| > k$ and at least k vertices have to be removed to make the resulting induced subgraph disconnected. A graph is called *biconnected* (*triconnected*) in the special case $k = 2$ ($k = 3$).

Weight functions $w_V : V \to \mathbb{R}$, $w_E \to \mathbb{R}$ may assign weights to the vertices or edges, the respective graph then is called *weighted graph*.

The *density* $\delta(G)$ of a graph G is defined as

$$\delta(G) = \frac{|E|}{|V|}$$

Sometimes alternatively the definition

$$\frac{2|E|}{|V|(|V|-1)}$$

is used to cover the relation between the number of edges and the maximum number of edges possible.

A *clustered graph* imposes a hierarchical structure on the graph that allows to group vertices together, e.g., to model compartments in biological networks or departments for business processes. Feng et al. [1995a] introduced the following clustered graph model:

Definition 2.1 (Clustered Graph)**.** A *clustered graph* $C = (G, T)$ consists of an undirected graph G and a rooted tree T such that the leaves of T are exactly the vertices of G. Each node ν of T represents a $cluster$ $V(\nu)$ of the vertices of G that are leaves of the subtree rooted at ν. The tree T is called the $inclusion\ tree$ of C as it describes an inclusion relation between clusters. The graph G is called the *underlying graph* of C.

With *graph objects* we denote a graph's vertices, edges and clusters in the following, and if it is not ambiguous, also sometimes the corresponding graphical representations. We will call a graph that is annotated by additional data such as weights or semantic information for the graph objects a *network*.

2.2 Graph Drawings and Layouts

A *layout* or *drawing* of a graph $G = (V, E)$ is a mapping of each vertex $v \in V$ to a distinct point $p(v)$ in the plane and of each edge $(u, v) \in E$ to a simple open curve with endpoints $p(u)$ and $p(v)$ that does not pass through the image of any other vertex. If two curves share an interior point p, we say that they *cross each other* or *intersect* at p. Note that even though a graph and its drawing are fundamentally different things, and there may be infinitely many different drawings for a graph, often the same terminology for graph objects and their drawings are used. We may for example state that 'two edges intersect', where in fact the drawings of the edges intersect. Besides the position, in practice graph objects may have further geometric attributes, as for example size and shape. These are in practice often considered by using a simplified representation that fits into the layout definition described above, e.g. by replacing a 2-dimensional vertex by a rectangular representation made up of four 0-dimensional corner vertices and boundary edges, we can therefore stick to the simplified assumption that vertices are points. With *graph representation* or *visualization* we refer to the graphical representation that also includes additional graphical attributes like color or pattern.

In the literature several terms are used quite often to denote that a drawing exhibits certain properties, or is similar in appearance to established visualizations. These terms are sometimes used with conflicting meaning, we will therefore shortly explain what we mean when we use them.

A *drawing style* is a set of characteristics that a drawing has to exhibit, for example an orthogonal grid drawing restricts the vertices to integer positions and the routing of edges to a sequence of vertical and horizontal segments. A *drawing convention* is a set of restrictions emerging from widely accepted publications and typical practices in an application area or community. It gives a partially and often informally defined, incomplete impression on how a diagram for a certain type of graph should look like, derived from what is usual, e.g., by textbook traditions. A *drawing standard* is a clearly defined notation, with rules how to draw graphs or diagrams, often specified by an institution or consortium. Drawing standards may evolve from conventions to standardize the use of graph representations in an application area. Often standards are developed to better exchange and communicate information, and to improve the readability by familiar drawing notations, well known examples are the *Unified Modeling Language (UML)* and the *Systems Biology Graphical Notation (SBGN).*

2.3 Embeddings and Planarity

A drawing of a graph is *planar* if no two distinct edges intersect, and a graph is *planar* if it admits a planar drawing. A planar drawing of a graph partitions the plane into topologically connected regions called *faces*. The faces are bounded by the curves corresponding to the edges, only a single face is unbounded, it is called *external face*. A face f is uniquely described by the sequence of its boundary edges, and the *degree* deg(f) of f is defined as the number of its boundary edges, where each edge that occurs twice on the boundary of f is also counted twice. Two faces are adjacent if their boundaries share a common edge. A planar graph with maximum vertex degree 4 is called *4-planar* graph. A *combinatorial embedding* Γ of a planar graph G is defined as a clockwise ordering of the incident edges for each vertex of G with respect to a crossing-free drawing of G in the plane. When in addition the external face f_0 is fixed for Γ, (Γ, f_0) is called *planar embedding* of G.

Two planar drawings D_1 and D_2 of a planar graph G realize the same planar embedding of G, if and only if the two following conditions hold:

- The cycles of G that bound the faces in D_1 are the same cycles that bound the faces in D_2.
- The external face in D_1 is bounded by the same cycle as the external face in D_2.

The *dual graph* Γ^* represents the adjacency relation between faces in an embedding Γ of $G = (V, E)$ with face set F. $\Gamma^* = (V^*, E^*)$ is constructed as follows: $V^* = F$ and E^* contains an edge f_1, f_2 for each edge $e \in E$ with e on the boundary of both f_1 and f_2. In case edge e is a *bridge*, i.e., only on the boundary of a single face f, Γ^* contains a self-loop (f, f).

Given a pair of planar graphs G_1 and G_2 with $V(G_1) = V(G_2) = V$, a *simultaneous embedding* $\Gamma = (D_1, D_2)$ of G_1 and G_2 is a pair of crossing free drawings D_1 and D_2 of G_1

and G_2 respectively, such that for each vertex is mapped to the same point in the plane in D_1 and D_2. If each edge $e \in E(G_1) \cap E(G_2)$ is represented with the same simple open Jordan curve in D_1 and D_2, Γ is a *simultaneous embedding with fixed edges*. If the edges of G_1 and G_2 are represented with straight-line segments in D_1 and D_2, Γ is called *simultaneous geometric embedding*.

2.4 Graph Decomposition and SPQR-Trees

Let $G = (V, E)$ be a connected graph. A *block* of G is a maximal connected subgraph that does not contain a cut vertex. From the definition we have that each block is either a maximal biconnected subgraph, a bridge, or an isolated vertex. Different blocks of G share at most one vertex, which then has to be a cut vertex of G.

The relationship between blocks and cut vertices of a connected graph is given by the *block-cutvertex tree*, or *BC-tree* for short. It contains a vertex for each block and for each cut vertex of G, and these vertices are connected by an edge if the cut vertex is contained in the block.

A slight generalization of the BC-tree is the *block-vertex tree* $\mathcal{B}$ of a connected graph G (*block tree* for short). It represents the relation between the blocks and the vertices of G and contains a B-node for each block of G and a V-node for each vertex of G; a V-node v and a B-node B are connected by an edge if and only if $v \in B$. Two blocks that have a cut vertex in common are therefore also connected in the block tree by a path over the cut vertex. The *representative* of a vertex v of G in block B is either v itself if $v \in B$, or the first vertex on the unique path from B to v in $\mathcal{B}$. Whitney showed a helpful relation between the block structure and the planarity of a graph:

Theorem 2.1. *Whitney [1932] A graph G is planar if and only if each block of G is planar.*

We shortly describe a graph decomposition based on *SPQR-trees*, a data structure introduced by Di Battista and Tamassia [1996]. They comprise a decomposition tree $\mathcal{T}$ of a biconnected graph G according to its *split pairs*. A split pair of G is a pair of vertices that is either connected by an edge or has the property that its removal increases the number of connected components of G (a *separation pair*). Alternatively, $\mathcal{T}$ is often described as the decomposition of G into its triconnected components [Tutte, 1966] comprising serial, parallel, and triconnected structures.

SPQR-trees can be efficiently implemented in linear time [Gutwenger and Mutzel, 2001] and allow to represent all embeddings of a planar graph. This property can be used for optimization over all planar embeddings, a method that we will apply for the algorithm in Chapter 6. The decomposition allows us to compute a *normal form embedding* that will be used for the clustered planarity test in chapter 7.4. Our description follows the one given by Mutzel and Weiskircher [1999].

Construction of SPQR-trees A *split component* of a split pair $\{u, v\}$ is either an edge (u, v) or a maximal subgraph C of G such that $\{u, v\}$ is not a split pair of C. Let $\{s, t\}$ be a split pair of G. A *maximal split pair* $\{u, v\}$ of G with respect to $\{s, t\}$ is such that, for any other split pair $\{u', v'\}$, vertices u, v, s, and t are in the same split component.

Let G be a biconnected graph. The construction of an SPQR-tree $\mathcal{T}$ of G works recursively starting at an arbitrary edge e of G that is called the *reference* edge of $\mathcal{T}$. The SPQR-tree $\mathcal{T}$ of G with respect to e is a rooted ordered tree whose nodes are of four types: S, P, Q, and R. The end-nodes of e are used as the split pair associated with the first node (root) of $\mathcal{T}$. At every node μ of $\mathcal{T}$, the graph is split into the *split components* of the split pair p associated with that node, i.e., the maximal subgraphs of the original graph, for which p is not a split pair. To make sure that the split components are biconnected, we add an edge to them and continue by computing their SPQR-tree. The resulting trees are made subtrees of the node used for splitting. Each node μ of $\mathcal{T}$ has an associated st-graph, called the *skeleton* of μ (denoted with *skeleton*(μ)).

Concepts and Elements of SPQR-trees The two vertices of the split pair p associated with a node μ are called the *poles* of μ. The skeleton associated with a split pair p is a simplified version of the whole graph where some split components are replaced by single edges.

The decomposition tree with respect to reference edge e is a rooted ordered tree whose nodes are of four types that are defined by the structure and number of the split components of its poles:

Trivial Case: *Q-nodes* are the leaves of the tree, there is one Q-node for each edge e in the graph. The skeleton consists of the two poles that are connected by two edges, where one edge represents the edge e and the other edge the rest of the graph.

Parallel Case: If the split pair $\{s, t\}$ has at least three split components $G_1, \ldots, G_k$, the root of $\mathcal{T}$ is a *P-node* μ, whose skeleton consists of k parallel edges $e = e_1, \ldots, e_k$ between s and t.

Series Case: Otherwise, the split pair $\{s, t\}$ has exactly two split components, one of them is e, and the other one is denoted with G'. If G' contains cut-vertices $c_1, \ldots, c_{k-1}$ $(k \geq 2)$ that partition G into its blocks $G_1, \ldots, G_k$, in this order from s to t, the root of $\mathcal{T}$ is an *S-node* μ, whose skeleton is the cycle $e_0, e_1, \ldots, e_k$, where $e_0 = e$, $c_0 = s$, $c_k = t$, and $e_i = (c_{i-1}, c_i)$ $(i = 1, \ldots, k)$. The decomposition continues with the subgraphs G_i, where the poles are c_i and c_{i+1}.

Rigid Case: If none of the above cases applies, let $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ be the maximal split pairs of G with respect to $\{s, t\}$ $(k \geq 1)$, and, for $i = 1, \ldots, k$, let G_i be the union of all the split components of $\{s_i, t_i\}$ but the one containing e. The root of $\mathcal{T}$ is an *R-node*, whose skeleton is obtained from G by replacing each subgraph G_i with the edge $e_i = (s_i, t_i)$.

Except for the trivial case, μ has children $\mu_1, \ldots, \mu_k$, such that μ_i is the root of the SPQR-tree of $G_i \cup e_i$ with respect to e_i $(i = 1, \ldots, k)$. The endpoints of edge e_i are called the *poles* of node μ_i. Edge e_i is said to be the *virtual edge* of node μ_i in skeleton of μ and of node μ in skeleton of μ_i. We call node μ the *pertinent node* of e_i in skeleton of μ_i, and μ_i the *pertinent node* of e_i in skeleton of μ. The virtual edge of μ in skeleton of μ_i is called the reference edge of μ_i.

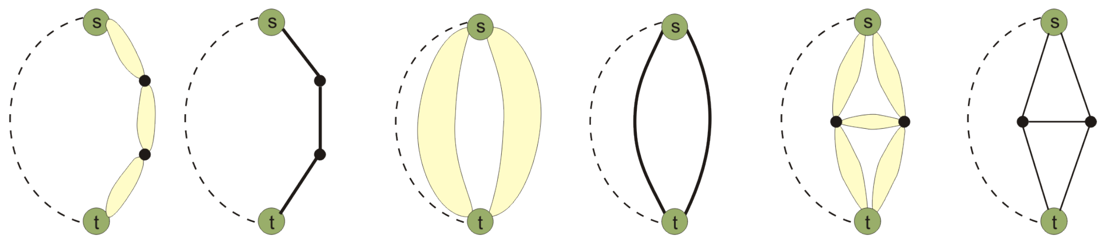
Each edge e in a skeleton represents a subgraph of the original graph:

Let e be an edge in *skeleton*(μ) and ν the pertinent node of e. Deleting edge (μ, ν) in $\mathcal{T}$ splits $\mathcal{T}$ into two connected components. Let $\mathcal{T}_\nu$ be the connected component containing ν. The *expansion graph* of e (denoted with *expansion*(e)) is the graph induced by the edges that are represented by the Q-nodes in $\mathcal{T}_\nu$. We further introduce the notation *expansion*$^+(e)$ for the graph *expansion*$(e) + e$.

Replacing a skeleton edge e by its expansion graph is called *expanding* e. The *pertinent graph* of a tree node μ is the subgraph of the original graph that is represented by the subtree rooted at μ; it results from expanding all edges in *skeleton*(μ) except for the reference edge of μ and is denoted with *pertinent*(μ). Hence, if e is a skeleton edge and ν its pertinent node, then *expansion*$^+(e)$ equals *pertinent*(ν). If v is a vertex in G, a node in $\mathcal{T}$ whose skeleton contains v is called an *allocation node* of v.

Let μ_r be the root of $\mathcal{T}$ in the decomposition given above. We add a Q-node representing the reference edge e and make it the parent of μ_r so that it becomes the new root.

Figure 2.1 shows the pertinent graph together with the corresponding skeleton for an S-, P-, and R-node. We will omit the discussion of Q nodes in the following because they are not needed to code the embedding of the graph G.



(a) S-node (b) P-node (c) R-node

Figure 2.1: Graph structure (left) and corresponding skeleton for S,P, and R case of the SPQR tree definition.

If G is biconnected and planar, its SPQR-tree $\mathcal{T}$ represents all combinatorial embeddings of G. In particular, a combinatorial embedding of G uniquely defines a combinatorial embedding of each skeleton in $\mathcal{T}$, and fixing the combinatorial embedding of each skeleton uniquely defines a combinatorial embedding of G.

2.5 Related Research Areas

Besides its mathematical foundations and the practical application areas, graph drawing plays a role in several related research areas concerned with visualization issues. We briefly describe terms from these areas that are used in this thesis. *Data Mining* is defined as the extraction of patterns or models from observed data with the help of statistical methods. It often is involved in a *Knowledge Discovery* process, which may be heavily supported by visual data exploration. *Information Visualization* is concerned with the communication of abstract data through the use of interactive visual interfaces [Ware, 2004, Keim et al., 2006]. Its goal is to display complex data in as much detail as possible without distracting the viewer. *Visual*

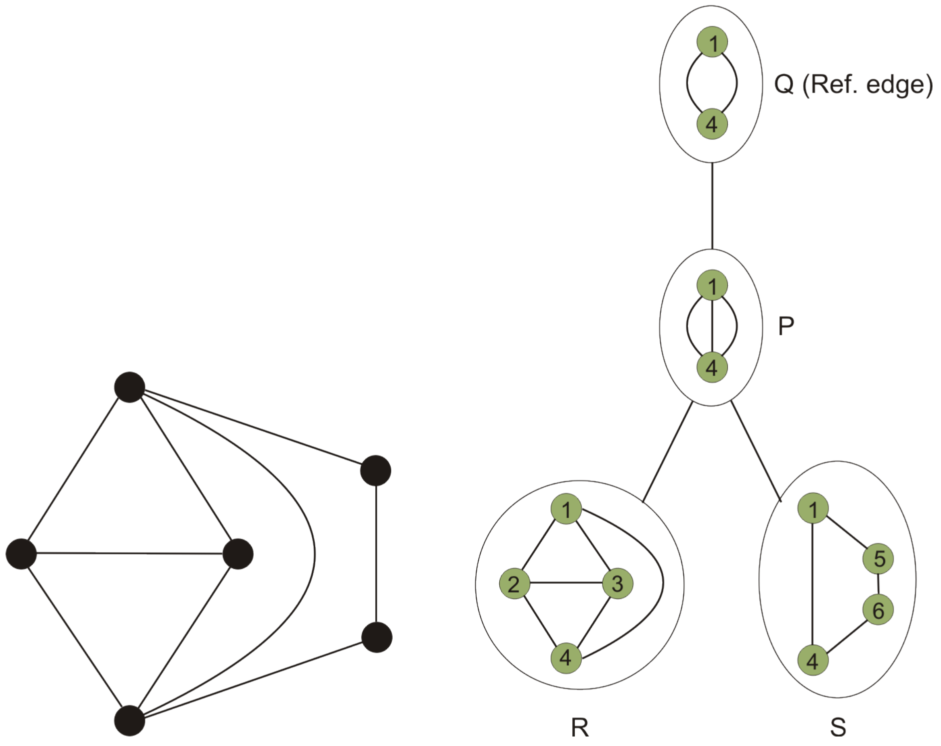


Figure 2.2: A graph G and its SPQR-tree (Q-nodes of R- and S-node omitted)

Analytics combines visualizations, data mining, and statistics by means of visual representations and interaction techniques in the analysis process of massive information volumes. The aim is to get a deeper insight into the data by appropriate visualizations that allow the human observer to 'detect the expected and discover the unexpected' [Thomas and Cook, 2005, Keim et al., 2008].

Perceptional organization concerns the question how the information detected by the sensory receptors are structured into the larger units of perceived objects and their interrelations, including contextual information [Palmer, 1999]. These processes are not yet completely understood, include the problem of subjective experience, and are still subject of ongoing basic research. Nonetheless, a couple of general principles and lots of laws have been defined, for example laws of grouping, e.g., similarity and proximity [Ellis, 1969]: Elements are perceived as aggregated into groups if they are placed close to each other, or move in a similar way during an animation. Though the theoretical treatment of such aspects seems to be far away from graph drawing research, at least the most fundamental findings should be taken into account for the evaluation of graph layouts and the development of new visualizations. Considering perceptional laws for the development of graph visualizations may improve the resulting layouts significantly. Instead of evaluating predefined layouts with respect to certain user tasks and predetermined aesthetical criteria, van Ham and Rogowitz [2008] therefore first asked users to rearrange vertices to best represent the structure, and then analyzed the layouts created. In comparison to their force-directed automatic layout, they found that human-created solutions contained significantly less edge crossings and did not value uniform edge length as much as the algorithm did. As a main result they concluded that besides minimization of edge crossings, human observers were very competent in identifying group structures and preferred corresponding layouts that emphasize these structures by confining them to edge-bounded regions, similar to classical clustered graph drawings.

3. GRAPHS, DATA AND DIMENSIONS

All models are wrong — some are useful.

GEORGE BOX

The interactive visualization of graphs with constraints raises many interesting theoretical questions, but it is mainly driven by requirements in practical applications. In order to develop solutions that are useful in practice, one has to examine the characteristics of the data and its semantics in application areas, the questions to be answered with the help of graph visualizations, and how the graphs are modeled from the raw data. Although the focus of this work is on the visualization of graphs that are given as input data, we will briefly discuss some of the related aspects in this chapter. In addition, we describe representative descriptions of the use of graph visualizations in important application areas.

3.1 Data Characteristics, Visualization Goals and their Influence on the Value of Representations

Multidimensional data play an important role in practical applications and the task to model a meaningful, perhaps also annotated, graph for visualization purposes is not trivial. The graph's topology can only represent a small part of the dimensions, often only a single dimension. For example, we can use the structural similarity of chemical molecules in visualizations of chemical compound databases to model a graph, where molecules are connected when their similarity value lies beyond a certain threshold. However, an entry on such a molecule in a typical biochemical database like PubChem or ChEMBL contains dozens to hundreds of properties, including chemical structure, biological activity, solubility, and many more. In addition to the question which of the data dimensions are modeled as relations or even vertices of the graph, selecting a meaningful visualization may be a difficult but crucial task for the visual data analysis process. It can be difficult to assess the importance or priority of the different dimensions for visualization, as the question to be asked during the knowledge discovery process may be unknown at the time of the modeling. Even the order in which the dimensions are arranged may play an important role for the understanding of the data [Ankerst et al., 1998].

In order to prepare multidimensional data for visual data analysis, which typically incorporates 2D or 3D techniques, the data needs to be projected into lower dimensions. This problem has been studied by many researchers and a large number of appropriate methods exist, including Multidimensional Scaling (MDS), Principal Component Analysis (PCA), and Self-Organizing Maps (SOM). An overview on data analysis and classification topics can be found in Batagelj et al. [2006].

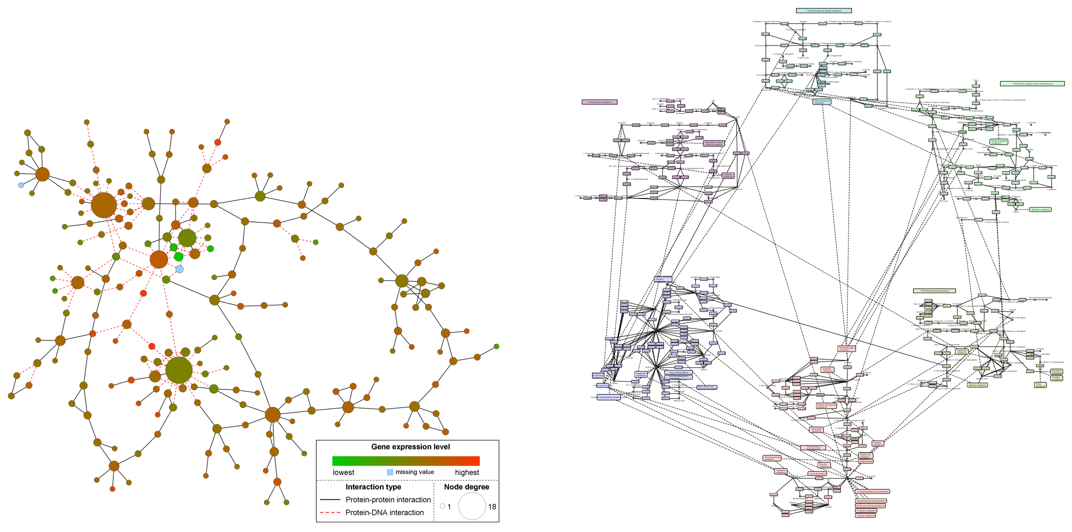
In some areas, however, there are inherent entities and relations that admit modeling of data as a graph, like social or biological network structures. As such a graph-based model only captures part of the data, additional information has to be represented by other means for visualization. These include graphical attributes like color or size and also supplementary properties of the resulting graph representation. These *secondary notation* elements can be modeled by constraints on the graph layout. Examples for such constraints are the alignment of vertices to show their equal status, a fixed order of the edges around a vertex to model a priority sorting, or an ordering of the vertices along the y-axis to represent a sequential flow. In Figure 3.1(a), vertex color and size are used to indicate additional information not covered by the graph structure. Clearly, the choice of the correct visual cue for representation is another challenge as some attributes are better suited than others depending on the type of data, like size or length for quantitative data.

In order to visualize supplemental information and aspects that cannot be integrated into the graph representation, graph drawing is often embedded into a context of information visualization, where the goal is not just to convey structure-related information but also to allow knowledge discovery based on additional data annotations. Graph-based visualizations in such a case are complemented and linked with other graphical data representations in an integrated fashion for visual analysis, as for example heatmaps, scatterplots, histograms, or dendrograms. An important task then is to link all resulting representations in a way that allows to perceive the data characteristics without significant loss of information due to the modeling and representation process. Requirements for graph drawing in such a scenario might differ from those for pure graph drawing representations, as the integrated graph view might only need to represent specific aspects in the overall visualization.

As a general goal, we would like to produce ‘nice’ drawings of graphs, i.e., drawings that are easy to read and convey well the represented information. Rules for a graph drawing to effectively communicate the underlying information depend on the application area and the graph instance, and are often difficult to define formally. When for example biologists have to specify which information needs to be conveyed by the drawing of a protein interaction network, the answer will vary from biologist to biologist and also differ strongly depending on the task to be performed. Even if a complete specification can be given, it does not automatically translate to a well-defined and clear visualization, and it is in general difficult to incorporate all resulting rules in an algorithmic approach. The user should then be enabled to adapt the optimization goals or the layout in an interactive environment.

Human postprocessing can help to improve the drawing and adapt it to additional requirements not considered by the automatic layout approach, as long as the automatic layout does not represent a local optimum far from an admissible drawing.

The dynamic nature of processes poses a further problem for a graph-based visualization. Often, series of networks have to be visualized for an interactive analysis, where the networks are either given as discrete states of a system or have to be aggregated, as an additional challenge, from continuous streaming data. In such cases, it is not sufficient to be able to create static representations that optimize the quality of each single drawing, as this could disrupt the user’s mental map when switching from one step to the next. The representations then need to be adjusted to allow a smooth transition that lets the user keep his orientation, and at the same time allow to highlight the dynamic changes. Another aspect that needs to be



(a) (b)

Figure 3.1: (a) Part of a human protein interaction network. The protein vertices are given a shade gradient according to their expression value; the vertex size corresponds to the number of interactions. The shades and styles of the edges represent different interaction types; solid lines indicate protein-protein, and dashed lines protein-DNA interactions. The graph was drawn with Cytoscape using its implementation of the spring-embedder algorithm. (b) A network that is partitioned according to the pathway concept. It consists of five pathways and their interrelations, drawn with KGML-ED [Klukas and Schreiber, 2007].

taken into account is the size of the data and the resulting graphs under investigation, both to consider efficiency aspects as well as perception issues when large numbers of objects have to be visualized. An important question in this context concerns the goal of the application workflow: The user might want to get an overview on the characteristics of the data set, to drill down large data sets to a small number of entities, or to gain deeper insight in the underlying relations, allowing for example hypotheses about formerly unknown processes. Depending on these goals, the approach and the techniques used may differ largely.

3.1.1 Large Graphs

One of the few resources increasing faster than the speed of computer hardware is the amount of data to be processed.

IEEE InfoVis 2003 Call-For-Papers

There is often a preeminent distinguishing feature between manually and automatically generated data sets – size. Given the computational power of today's computers and the automated processing capabilities in many areas, huge amounts of data can be generated quickly, but are difficult to handle. Huge data sets nowadays arise in many different research fields and are challenging not only because of the mere storage requirements, but more importantly because it is usually non-trivial to efficiently access, analyze, and process them. In bioinformatics, a single high through-put experiment can result in thousands of datapoints; since thousands of these experiments can be conducted per day, data sets with millions of entries can be quickly accumulated for further analysis. The Stanford Microarray Database, e.g., contains around 2.5 billion spot data from about 75,000 experiments [Demeter et al., 2007]. Other fields of increasing importance are, among others, the analysis of the dynamics in large telecommunication or computer networks, as well as social statistics and criminalistics.

Although we can improve both the theoretical and the practical performance of the graph drawing methods and data structures by means of *Algorithm Engineering* (for an overview on the topic see for example Mehlhorn and Sanders [2008], Chimani and Klein [2010]), most methods won't be applicable to huge graphs due to the complexity of the underlying theoretical problems. Many optimization problems in graph drawing are already hard to solve in their most basic form, and respecting further drawing constraints poses an additional challenge. This is not only a problem with regard to the algorithmic solution, but also the implementation effort. As even heuristic solutions, e.g., compaction methods for orthogonal graph drawing, are already quite involved, constraint handling has to be embedded into rather sophisticated implementations.

When we are able to handle these requirements also within interactive tools, the question how to visualize the data remains. Even though those graphs will typically show the characteristics of the respective application area, and the corresponding drawings need to adhere to appropriate drawing standards, large graphs pose specific problems that justify a separate treatment. The most simple question related to this topic is: What if the number of vertices is larger than the number of pixels that can be used for their visualization? Depending on the task, the visualization of more than a few hundred or thousand vertices renders insight into the structure infeasible in a single static visualization. Then additional *navigation* concepts are needed to allow the user to orient within the data. Often practitioners would like to see some kind of overall structure of the data under investigation, but at the same time be able to focus on specific regions of interest in detail. This encourages the *Overview-plus-Detail* paradigm, where for example subgraphs not in the focus can be aggregated to a single vertex while preserving the overall structure by appropriate edge connections, see Section 8.2 for a further discussion. As an example, in the biological network displayed in Figure 3.1(b), individual pathways could be collapsed to show the structure of the inter-pathway relations or to focus on a region of interest. Aggregation can be done based on a *clustering* of the graph or the data, and in some applications a natural cluster structure is given or known in advance, as, e.g., departments in a business process. In contrast, often first a clustering method has to be applied to identify groups of closely related objects, see, e.g., Fraley and Raftery [1998], Gaertler [2004]. However, an additional aspect of large data sets is that the graph or network structure might not be known in advance, as the modeling of the relations may be given only as a set of rules, e.g., using a similarity function. Therefore the characteristics of the overall graph cannot be used for visualization and navigation purposes, as only parts of the graph are dynamically created depending on the current focus, and it would be computationally infeasible to analyze the structure in advance.

In correspondence to the emerging importance of the specific aspects of large graph handling, several works in graph drawing and information visualization have been concerned with related concepts and problems, including scalability issues, see for example Kobourov [2000], Patrignani [2001], Munzner [2000]. We will discuss issues concerning the interaction with and the navigation in large graphs in Chapter 8.

3.2 Application Areas

Data from numerous application areas can be modeled as a graph for visualization. We present selected areas that either already use graph visualization or have the potential to benefit from tools that use graph drawing methods. We then shortly describe the corresponding use cases and data, the modeling of the data to achieve a graph representation, and the requirements for a meaningful drawing that supports the domain experts in analyzing the data. As the need for visualization in life sciences currently is a driving force for much theoretical and practical work in graph drawing, and the problems that occur cover a large part of the relevant research questions, we focus on that area and its description takes a major part of this section.

3.2.1 Life Sciences – Biology and Chemistry

A natural application of graph drawing can be found at the visualization of biological networks, where the relations of biological entities have to be understood and analyzed. The use of graphical visualization and analysis tools is therefore already widespread in the biology community. In stark contrast to that, the chemical community has only recently adopted the use of such tools, even though the tasks and data are often closely related (see also the discussion in Chapter 9).

In recent years, the development of high-throughput experimental techniques has led to the generation of huge data sets in the life sciences. Since manual analysis of this data is costly and time-consuming, biologists and chemists are turning more and more towards computational methods that support data analysis. The information in experimental data sets can often be either represented as networks or interpreted in the context of networks that serve as models of the biological system under investigation. These models are used, for example, to predict the behavior of the system and to guide further experiments.

Biology and chemistry often consider different aspects of the same goals and underlying processes, and the research is often intermingled in a way that makes a clear separation difficult and sometimes even unnecessary from a theoretical, modeling point of view. Nonetheless, the questions addressed, established representations and workflows may differ to a large extent, therefore the respective visualization approaches and user interfaces have to be especially tailored to allow an intuitive work on the data. Whereas in biology networks arise naturally, and biological processes are modeled and thought of as networks, a graph-based model of chemical data is much more uncommon. For the ultimate goal of creating an ‘ideal’ data representation and navigation approach, one has to realize that classical data representations for a long time influenced the workflows and established a certain view on the data. These representations cannot be simply replaced without compromising the user’s confidence on the reliability of the representation and the subsequent hypotheses drawn out of it. When the author visited a research department of a large chemical company, and tried to find out which information could be used to aggregate and remove large numbers of molecules early from a visual analysis process, a typical answer was ‘But I want to see all of them, how could I otherwise know what the computer did and what I am missing afterwards?'. In other words, introducing advanced visual analysis tools in chemistry is a long process where established data representations, as, e.g., spread sheets, need to be provided in addition to new ones to let users familiarize themselves with the new tools.

Biology

Biological research provides a broad range of data that can be visualized using graphs. Biologists aim at identifying and understanding complex interaction networks in a living organism to discover the underlying organizational principles, and try to gain deeper insight into the dynamic processes that actually take place. Such insight is needed to derive new hypotheses, both about the specific behavior of actors (e.g., proteins, genes,...) as well as the architecture and behavior of the overall system. The knowledge gain then can be used to propose ways to influence the processes in a precise way with minimum side effects, e.g., to cure a disease with a specifically designed drug. About a decade ago, the *systems biology* paradigm became popular. Systems biology tries to put local biological information, e.g., protein interactions in a cell, into the context of the entire system to understand how different levels work together. For a detailed discussion of the biological background see for example Koch et al. [2011].

Visual analytics of biological networks, i.e., the integration of interactive visualization with analysis techniques, is getting more and more popular in the biology and bioinformatics communities. It can be useful, e.g., to predict new functional substructures, like signaling pathways, to identify previously unknown members of known pathways, or to identify new network modules. Different network types occur in biology that lend themselves naturally to a graph representation, including metabolic networks, signal-transduction networks, (protein) interaction networks and many more, see the description below for an overview.

Sometimes, however, drawings are used only to show the size or complexity of the data. Biologists are therefore quite familiar with 'hairball' drawings of interaction networks; see Figure 3.2. Such static drawings do not help much in understanding and analyzing the data, which is reflected in the alternative term 'confusiogram' coined for them [Schwikowksi and Ideker, 2007]. Even at the time of writing, Lander [2010] called the 'hairball' an 'increasingly familiar image in the biology literature' and appoints it 'the dominant icon of the systems biology era'. In an online comment to that, this view is supported by Uetz, the creator of one of the first hairball representations (or at least of the most famous one) in biology literature [Schwikowski et al., 2000]. He states as a challenge for the interactive visualization in biology: 'What is still lacking today (to make hairballs more useful for biology) are computational tools that allow users to zoom into a hairball, visualize the details of the nodes and edges, and integrate all available information in a intuitive, user-friendly way. This would be the ultimate *understanding* machine.'

Wheelock et al. [2009] stress the importance of network-based analysis and the need to cope with the dynamics of the system: 'One important aspect of systems biology approaches is to identify the biological pathways or networks that connect the differing elements of a system, and examine how they evolve with temporal and environmental changes'. Aloy and Russell [2005] state that there is a need for the visualization of the information flow at physiological time scales to reflect the dynamic nature of the system and to allow the visualization

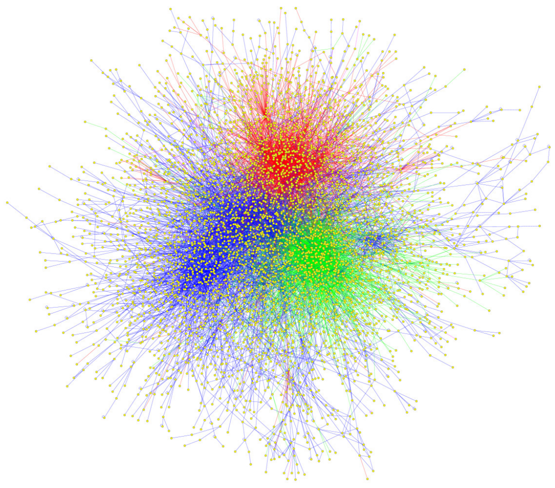


Figure 3.2: Human proteome and its binding interactions with a hairball visualization, as given in Lander [2010], there credited to Nicolas Simonis and Marc Vidal.

of simulations. As another main challenge for high-quality layouts of biological networks they identify the use of sub-cellular localization information, as for example information on the location of proteins with regard to cellular compartments like nucleus and cytoplasm. In other words, the identification of important subnetworks or patterns, their localization, and the dynamic evolution of the relation structure are primary analysis goals.

We can therefore see the bare visualization of the network structure still as an important task, but additional challenges occur due to the dynamics of such networks and the annotations like localization information that need to be taken into account. Even semantic annotations may be subject to change, e.g., the functional role and the interrelations of a protein may change over time, depending on the environmental conditions, the location, or the cell state. A further aspect currently often neglected in biological visualizations is that biological network models are subject to uncertainty due to the noise in the experimental data and the mathematical models used to generate network representations.

Biological networks are used to communicate many different types of data. This data can be encoded in the structure of the network, the network layout, or as graphical or textual annotations. The data itself may be primary data (i.e., directly measured), secondary data (i.e., derived, inferred, or predicted), or a mixture of both. There is a large variety of biochemical entities and values, e.g., gene sequences, transcripts, expression levels, proteins, metabolites, concentration, or fluxes (of mass or information), and even though not all of them may be in the focus of analysis at the same time, combining several of them may be desired to get the bigger picture of the processes in an organism. Corresponding to this variety of data and semantical information, biological visualizations are often enriched with multiple graphical features to depict this information, see for example Figure 3.3. Categorization information like the type of a biochemical entity or its functional annotation are often used to structure graph representations, e.g., by grouping entities of the same function, or by aligning entities of the same type.

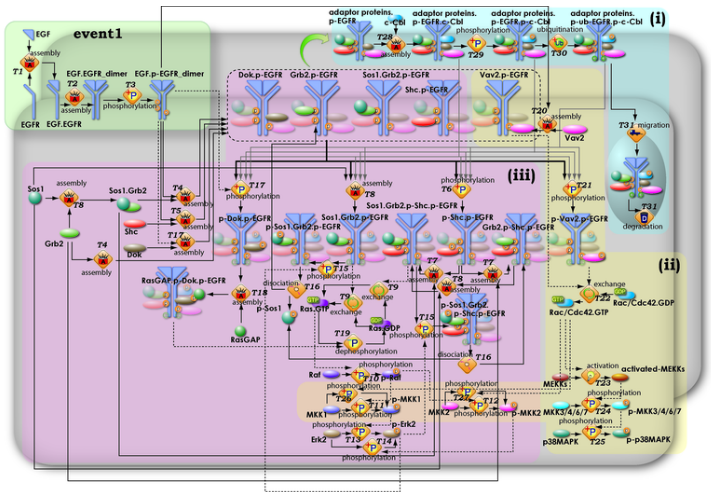


Figure 3.3: Example visualization of Cell Illustrator [Nagasaki et al., 2010], which shows the use of many different graphical features, like color, background color, vertex shape, additional icons, labels, and edge style.

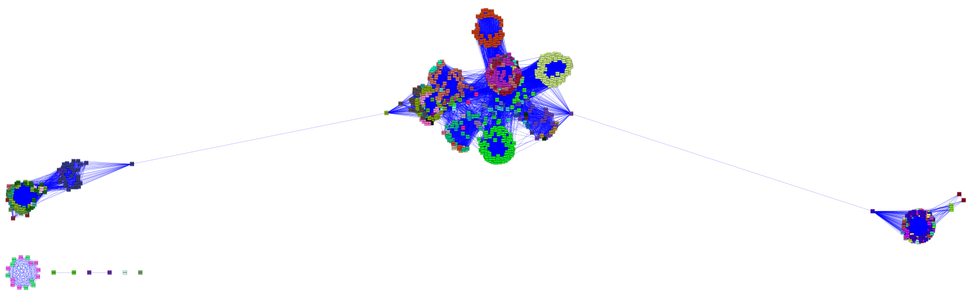


Figure 3.4: Visualization of the amidohydrolase protein superfamily (data from SUPERFAMILY, graph kindly provided by Scooter Morris), layout generated with OGDF's FM^3 implementation. Color coding is used to denote protein family annotations. Even though the algorithm does not consider semantic grouping information, the families are represented well due to the dense internal relations.

The biological networks do not only vary regarding the type of data that they represent, but also regarding their structure. Networks can range from very small, sparse graphs to huge and very dense graphs; the amidohydrolase protein superfamily network in Figure 3.4 for example contains about 1,000 vertices and over 50,000 edges. A further aspect that therefore needs to be handled is the development of techniques to cope with large and dense graphs. Databases like BioGRID or PDB contain up to millions of entries, IntAct [Aranda et al., 2010] contains over 250,000 binary interactions between over 50,000 proteins. Even though not all of these interactions might be of interest at the same time, an overview on the overall structure might be desired, complemented by a view for eventually large subsets with several thousand objects for closer investigation.

In contrast to that, for same sparse and rather small networks, e.g., metabolic pathways, drawing conventions have been established over time that help to express biological concepts (see discussion below). For these types of networks, results of automatic layout methods

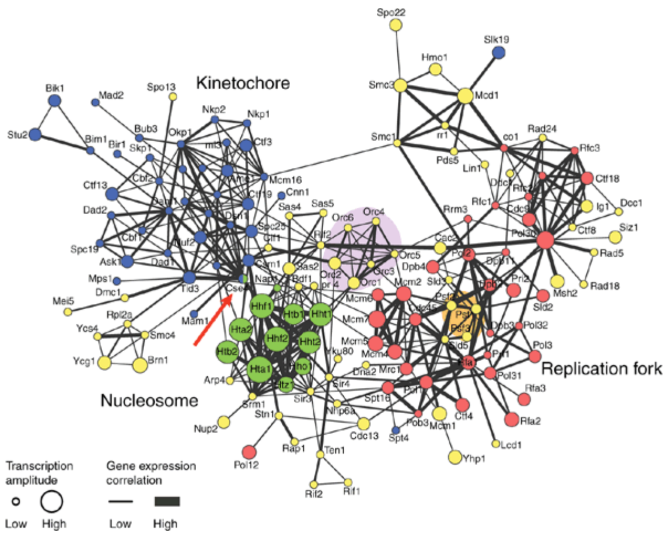


Figure 3.5: A protein interaction network with gene expression data annotations, laid out using a force directed algorithm in Cytoscape [Merico et al., 2009]. Vertices represent proteins that are annotated as being located on the chromosome by the Gene Ontology project [Ashburner, 2000], vertex color represents location categories, vertex size represents the amount of expression change during the cell cycle, and edge thickness models correlation between transcript profiles. The authors manually added visual cues to emphasize interesting regions (shading and arrows), and the labels have been placed manually.

need to resemble the main characteristics of these conventions in order to be accepted by the biology community. We will shortly review the important network types that occur.

Types of biological networks. *Protein interaction networks* represent physical interactions of proteins with each other or with other binding partners like DNA/RNA. The vertices in such networks represent proteins or sets of proteins. The interaction edges are normally undirected, but may be directed in case of heterogeneous networks (for example, protein-protein and protein-DNA/RNA interactions), resulting in mixed graphs. Each vertex and edge may be annotated with additional biological attributes like expression level, sub-cellular localization, and the number of interaction partners. For an example of a protein interaction network see Figure 3.5. The network paradigm can also be used to describe the topology and dynamics of protein structures; see Section 9.3.
Gene-regulatory and *signal-transduction networks* both use sets of directed edges to convey a flow of information. While gene regulation (regulation of gene expression) occurs exclusively within a cell and represents a regulatory mechanism for the creation of gene products (RNA or proteins), signal transduction refers to any process that transports external or internal stimuli via so-called signal cascades to specific cellular parts where a cell response (e.g., gene regulation) is triggered. While vertices in these networks represent molecular entities (genes, gene products, or other molecules), edges represent a flow of information (regulation or passing of a chemically encoded signal). See Figure 3.7 for signaling pathway diagrams.
Metabolic networks describe a network of chemical reactions, where substrate metabolites are converted into product metabolites, catalyzed by enzymes. Such a network is a hypergraph that is usually represented as a bipartite graph $G = (V_1 \cup V_2, E)$. The vertex set is

partitioned into the set V_1 of metabolite and enzyme vertices (enzymes catalyze the chemical reactions converting metabolites) and the set V_2 of reaction vertices.

The *pathway* concept tries to break the complexity of the reaction network in a living system to focus on a specific product or function. A pathway is a subnetwork that can be given through definition by established textbooks and databases (e.g., KEGG, BioCarta) or just as the subnetwork between two substances; see Figure 3.6. Even though the reduction to a pathway is helpful for the intuitive understanding, it represents an abstraction of the real, dynamic situation in a living cell. A biological process can be highly dependent on environmental conditions as temperature, concentration, and pressure, and during a process, a dynamic change of the involved entities' roles and of the network structure might occur.

There are no pathways, there is only one big network.

LEROY HOOD

Pathways may consist of a main linear path, but can also contain cycles, producing the chemicals that initiate them, or feedback loops that inhibit an earlier step. These components and patterns are often drawn in a way that facilitates easy identification. Pathway graphs are typically modest in size (< 100 vertices), sparse, directed, and acyclic. To avoid clutter and edge crossings, vertices are often *cloned*, i.e., several vertices exist for the same biological entity, and vertices may also represent whole other pathways.
Ontologies have become widely used in the life science community. Collaborative efforts such as the Gene Ontology (GO) or the KEGG Ontology (KO) have developed vocabularies to describe biological processes on different levels (e.g., participation of a gene product in signaling networks) and help biologists to identify available knowledge concerning a specific area of research. These categories are already well established for use in the analysis of biological processes. While the network structure of such ontologies can also be visualized using directed graphs, it is more important that visualization approaches for biological networks are able to integrate their information, preferably by graphical means, including integration in layout computation, e.g., by grouping vertices having the same annotation.

Established Visualizations. There are large posters (e.g., Nicholson's and Michal [2005]'s pathway maps) and several projects that have created static graphical representations of metabolic networks and offer access to these graphs via web pages (e.g., KEGG, BioCarta or BioCyc). Most of them provide clickable maps most of which are semi-automatically or manually created, and are enriched by highlighting enzymes ocurring in a specified organism. The open-source pathway database REACTOME [Croft et al.] provides interactive pathway maps, also linked with additional data views. It uses an automatic layout and supports the new systems biology standard for graphical notation SBGN. Interactive electronic molecular interaction maps (eMIMs) visualize molecular interaction networks based on a clearly defined notation [Kohn et al., 2006, MIM], and allow the user to navigate through them with links to molecular databases, references, and annotations. See Figure 3.7(b) for an example. Compared to the 'organic' looking style of manually drawn maps and the layouts automatically created by many graph drawing methods, these maps have a rather technical

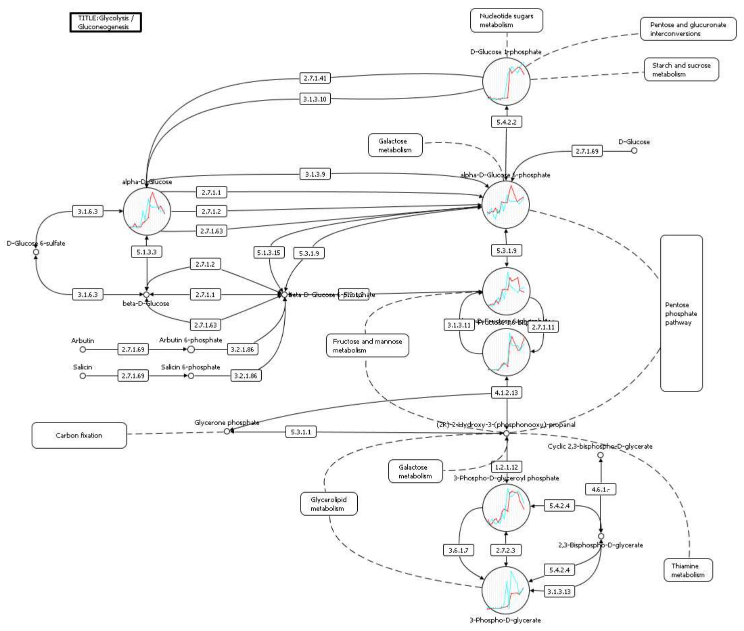


Figure 3.6: A part of the glycolysis/gluconeogenesis pathway with additional data mapped onto some vertices [Albrecht et al., 2009]. Circles encode metabolites, rectangles represent enzymes catalyzing the reaction, and rectangles with rounded corners denote other pathways. Time series data from two series (day: red, night: blue) were mapped on some vertices which have been enlarged. Solid and dashed lines represent reactions and connections to other pathways, respectively. The pathway data was derived from KEGG, and the graph was drawn with VANTED [Junker et al., 2006] in a style similar to the KEGG pathway picture.

appearance, close to a circuit diagram. This is both due to the consistent specification of symbols for entities and relationships that resemble symbols for electronic circuit diagrams, and the orthogonal edge routing common to such diagrams. The web-based tool iPath [Letunic et al., 2008] for the visualization and analysis of metabolic and regulatory pathways provides an interactive viewer that allows navigation through pathways based on a global map. This map is constructed using a large number of KEGG pathways in combination, and gives an overview of the complete metabolism in biological systems; see Figure 3.8. Even though the main drawing characteristics are quite similar among the online resources, they differ largely in the graphical notation used.

The availability of these representations has established a de facto standard for metabolic network drawings that features near-orthogonal drawings where important paths are aligned, relevant subgraphs are placed close to the center of the drawing, substances and products of a reaction are clearly separated, and co-substances are placed out of the main path close to the reaction. Manually generated metabolic pathway drawings often exhibit a small number of crossings and bends, and the orthogonal edge routing is replaced by straight-line routing if bends can be saved for short distance edges; see Figure 3.7(a). In the last years, many approaches considered the drawing of metabolic pathways, and a lot of layout algorithms try to at least partially obey established drawing styles (e.g., Karp and Paley [1994a], Becker and Rojas [2001], Goesmann et al. [2002], Schreiber [2002], Bourqui et al. [2006], Schreiber

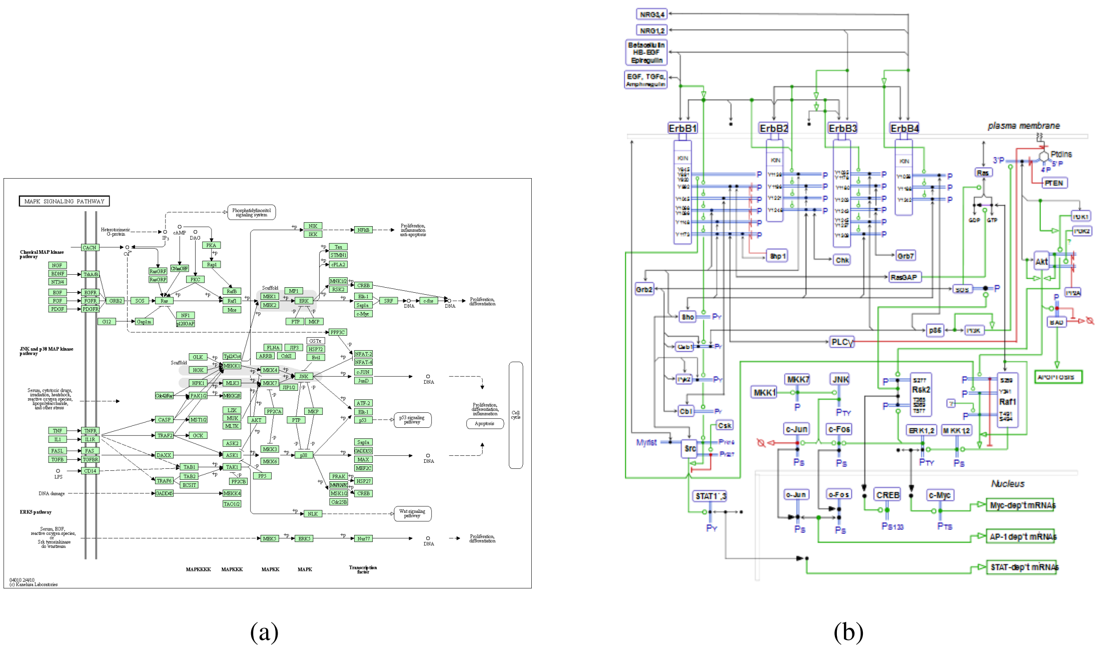


Figure 3.7: (a) MAP Kinase signaling pathway provided by KEGG and (b) network part of the signaling from EGF receptors in MIM style. Colors are used to distinguish interactions involved in different functions. Both figures show a clustering of vertices due to cell compartments, which have a fixed left-to-right (and top-down, respectively) order.

et al. [2009]).

Several modeling schemes have been suggested for the representation of cellular signaling networks. They can be divided into two classes, namely, the entity-relationship (ER) scheme and the state transition scheme. The ER scheme (used by KEGG, SPIKE, Kohn's map) focuses on the regulatory effects maintained between different proteins within a signaling network, while state transition schemes (used, e.g., in the tools CellDesigner, PATIKA, ProMoT) regard different post-translational modified versions of a protein as separate entities and seeks to trace transitions between these states. ER diagrams are much simpler, having all states represented by only one protein entity, but at the price of not accounting for possible temporal order constraints (such as, e.g., activation on A can only appear after phosphorylation on B).

As can be seen from the differing representation of networks given here, the rules for admissible drawings are not clearly defined and may differ at least slightly from user group to user group and from task to task. Standardizing the representations could remedy the situation and help biologists to familiarize with a unified notation, saving them the efforts to cope with several different notations. Recently, an attempt was made to unify the graphical representation of biological networks and processes by adopting drawing standards from other areas, leading to the SBGN (Systems Biology Graphical Notation) project [Novere et al., 2009]. See Figure 3.9 for example visualizations using the new notational standard. The authors of the first published standardization [Novere et al., 2009] compare their initiative with the standardization of electronic circuit schematics, which lead to a big breakthrough

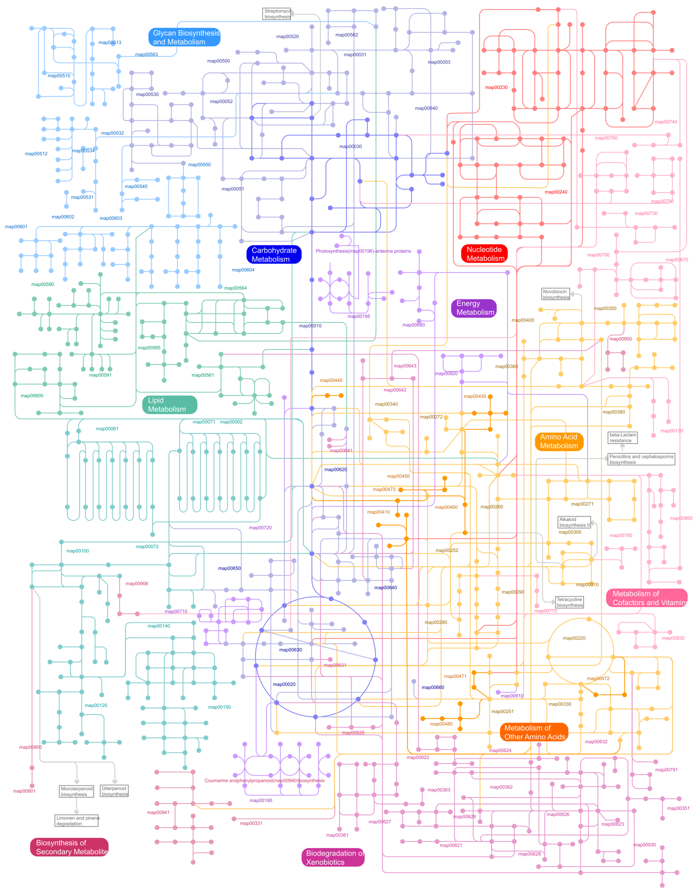
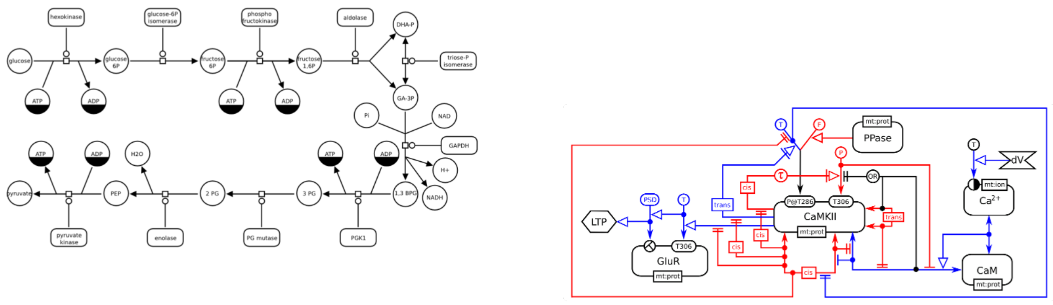

Glycan Biosynthesis and Metabolism
Carbohydrate Metabolism
Nucleotide Metabolism
Energy Metabolism
Lipid Metabolism
Amino Acid Metabolism
Metabolism of Cofactors and Vitamins
Metabolism of Other Amino Acids
Biosynthesis of Secondary Metabolites
Biodegradation of Xenobiotics

Figure 3.8: iPath global pathway map. Color coding is used to denote functional substructures.

in electronics. SBGN contains three orthogonal languages with their own set of symbols, semantics and interpretation rules: The process description language that shows the temporal courses of biochemical interactions, the entity relationship language that allows to see all the relationships in which a given entity participates (regardless of the temporal aspects), and the activity flow language that allows to see the flow of information between biochemical entities.

Visualization Goals. In the following, we describe typical use-cases that can be facilitated by visual graph representations. In Albrecht et al. [2009], sketches of the underlying modeling and drawing problems are given.

Visual comparison of biological networks. Conservation of biochemical function during evolution results in structurally similar molecular subnetworks across different organisms and species. Uncovering relevant similarities and differences or comparing networks in different states (e.g., diseased vs. healthy), at different time points, or under various environmental conditions (temperature, pressure, substrate concentrations, etc.) supports the biologist's understanding of the processes, e.g., to identify disease-specific patterns. Given a set of graphs



(a) (b)

Figure 3.9: SBGN representation of the glycolysis metabolic pathway, using a process description diagram (a), and of the effects of a depolarisation on the intracellular calcium, using an entity-relationship diagram (b).

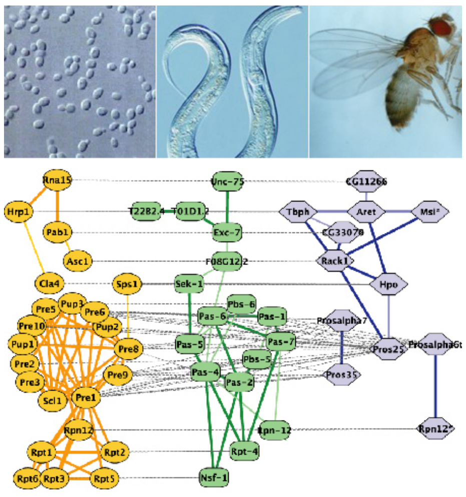
with a high degree of similarity between each other, the task is to compute a layout such that the differences (or the similarities) are highlighted. Approaches for different types of networks exist, including protein interaction [Sharan et al., 2005, Sharan and Ideker, 2006] and metabolic pathways [Forster et al., 2002, Schreiber, 2003]; see Figure 3.10.

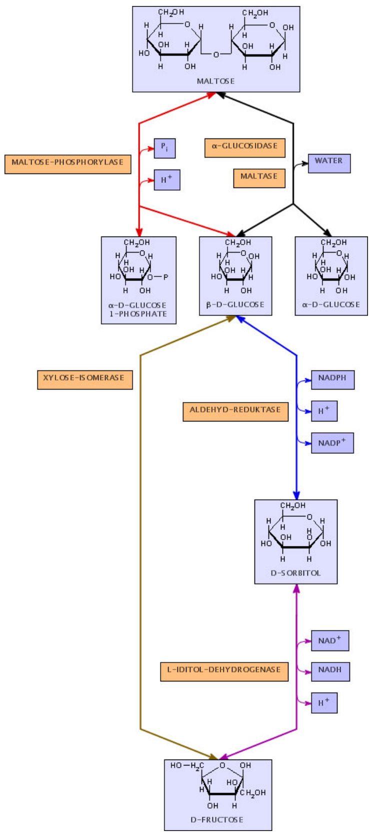
The visualization of larger graphs for comparison is a very challenging task, and probably not possible without interaction methods that allow to show only subsets of the data at a time and with abstraction of the data; see Figure 3.11.

Visual analysis of data correlation. Biologists frequently use correlation graphs as a means for visually expressing and exploring complex forms of correlation within their data. Normally, the information contained in the data is mapped as annotation onto a graph that represents the pathways characterized by experimental data. Biologists are then interested in a graphical representation that highlights the interrelation between the connectivity structure of pairs/subsets of vertices in the original network and their correlation. An example for an interesting correlation pattern would be a set of vertices that is closely connected within the underlying graph but exhibits only weak correlation in the data or vice versa.

Integrated representation of multiple overlapping networks. Different types of biological networks describe different functional aspects of the whole cell, tissue, or organism in question. To get a deeper, system-wide understanding, these networks need to be combined. The enzymes acting in metabolic networks for example are regulated and this regulation is described by a regulatory network. It is thus becoming increasingly common to integrate these different types of networks into joint networks. Figure 3.12 shows an example of integrating a gene-regulatory network and a metabolic network. A good joint layout of these networks should reveal the *interaction* between these networks, for example, how specific vertices of the gene regulatory network activate or inactivate whole subnetworks of the metabolic network. In order to simplify the identification of these subnetworks, mental map preservation on the level of the metabolic network is helpful.

Visualization of sub-cellular localization. Cells consist of distinct compartments, sub-cellular locations, separated from each other by membranes. Examples for these are the cytosol





(a) (b)

Figure 3.10: Visualizations of a protein interaction network (a) and a reaction network (b) that show the differences and similarities between several organisms. Visualization is done using two different techniques: separated and integrated layouts. The comparison of the protein interaction networks across multiple species allows to identify conserved complexes and pathways, picture as published in Sharan et al. [2005]. The BioPath tool provides a special pathway layout. Shown here is the reaction network between maltose and D-fructose as visualized in Forster et al. [2002]. A specific color scheme is used to distinguish reactions occurring in different organisms.

(the inner space of cell), the nucleus, the mitochondria, or chloroplasts in plants. The membranes enclosing a compartment separate parts of the biological networks as well. Different partitions of the network will be localized in different sub-cellular locations and hence cannot interact with each other directly. It is thus essential for an understanding of the network's function to integrate that spatial information into the layout of the network. The required localization data is either already contained in data sets derived from experiments, can be extracted from external sources, or can be predicted.

Visualization of multiple attributes. Often multiple attributes have to be considered when analyzing biological data. One example is time-series data which is frequently collected in order to better understand the dynamic behavior of a biological system. The combined representation of such time-series data and a corresponding network should allow biologists to gain new insights concerning the underlying system, for example, co-regulated sets and

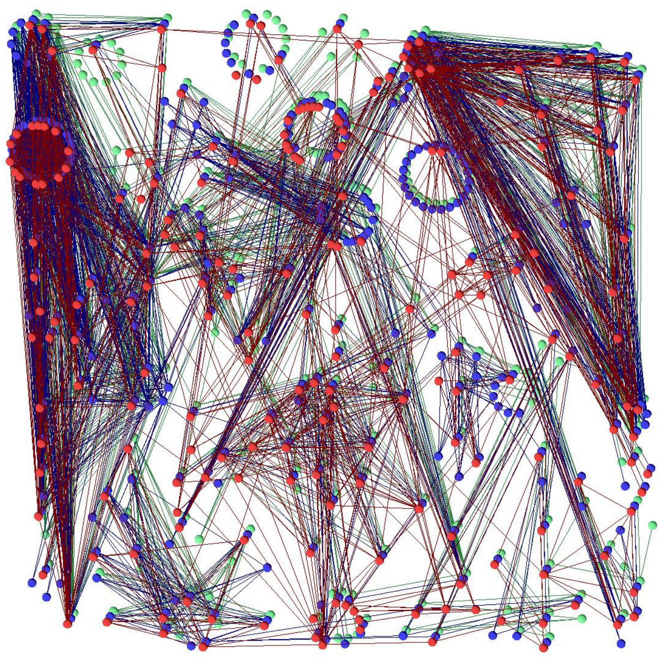


Figure 3.11: A large alignment (more than 200 proteins per species) of three protein interaction networks performed with the VANLO alignment tool [Brasch et al., 2009]. Simulated annealing was used to compute the layout, with a running time of less than 30 seconds.

their connection within the network. Such a combination can, for example, be achieved by mapping the data onto the vertices of a network; see Figure 3.6. However, also complex cases can occur, where multiple attributes of several hundred data sets need to be mapped, and significant structural patterns of sets with similar data annotations have to be detected. The simple use of small visualizations that replace the vertex representations is usually not sufficient, because a visual comparison of such small graphics in a large graph is impossible.

Visualization of flows and paths in networks. The qualitative and quantitative distribution of mass and signal flows (fluxes) within a network has to be analyzed under uncertainties of the data. The flow of certain paths may change over time (time-series of measurements) and the number of paths through the network is so big that not all of them can be displayed. Biologists are therefore only interested in the main paths through the network, i.e., those paths which possess a statistically significant flow and that transport a considerable percentage of the overall flow through the entire network. For directed graphs like metabolic networks, the layout must reflect the hierarchical nature of the flow, preserve layouts for subnetworks from textbook representations as closely as possibly, and at the same time emphasize the relevant parts and paths of the network. The main paths may be placed at the center of the layout and drawn as straight lines, and the distribution of the flow within the network has to be depicted, e.g., by using edge width or color. If the dynamic change of the main flow over time also needs to be visualized, smooth animations are required to preserve the user's mental map.

Exploration of hierarchical networks. Biological networks often comprise several thousand vertices and edges. To help exploring such large and complex structures the entire network

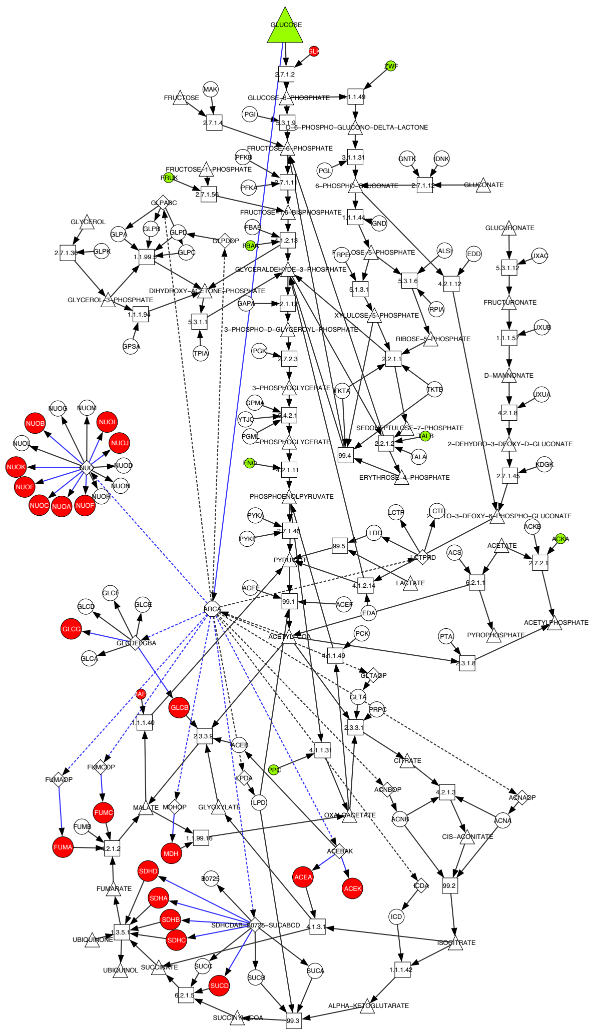


Figure 3.12: An example of an integrated network consisting of a part of the glycolysis network and a gene regulatory network [Yeang and Vingron, 2006]. Triangles represent metabolites, squares reaction flux, and circles genes and enzymes. Color coding denotes regulation, increase (green) or decrease (red). The edge style represents activation (solid) and inhibition (dashed), main paths are aligned.

is usually broken down in a hierarchical manner into pathways and sub-pathways. Biologists commonly focus on (sub-)pathways in a region of interest and explore their relation to other pathways. However, due to the many connections between different pathways often an abstract overview-like picture of all pathways and their connections, as well as an interactive navigation from a set of pathways to other connected or related pathways is desired. An example of such a set of pathways is shown in Figure 3.1(b). In order to allow the user to keep his orientation during exploration of the network, the layout changes resulting from a user interaction (e.g., selection of an additional pathway) should be small and also context information needs to be represented in an appropriate way. Expand-and-collapse mechanisms therefore need to be incorporated into layout algorithms such that drawing conventions and the mental map are preserved. These operations could be restricted to certain levels of abstraction, for example by only collapsing/expanding semantically meaningful substructures like pathways. One of the main challenges is that layouts for such subnetworks, as well as their relative position to each other, may be given. This layout information needs to be preserved as closely as possible. As these networks are too large to be laid out nicely as a whole, some overview graph or backbone could be defined by reduction or abstraction, covering the topologically or semantically relevant features of the network. The subsets, e.g., the pathways, do not need to be disjoint but may partially overlap. This poses an additional challenge for the visualization problem: Either the duplicates are merged, which complicates the task of mental map preservation, or it has to be clearly emphasized somehow that they represent the same biological entity.

Several requirements from biological visualizations, as the representation of localization information, are already handled by existing layout methods, e.g., using clustered drawings. Only a few approaches so far are addressing the problem of dense and large graphs. As the algorithms used for visualization are often not efficient enough, most approaches restrict the graph exploration in an interactive fashion to a small subgraph plus some context specific neighborhood (e.g., Huttenhower et al. [2009]).

The layout algorithms applied lag behind the state of the art, as findings from graph drawing, if not specifically developed with biological applications in mind, often do not make their way into bioinformatics tools. Although a number of specifically adapted algorithms exist [Li and Kurata, 2005, Kojima et al., 2010, Dogrusöz et al., 2006, Junker et al., 2006], the broad use is based on standard algorithms, as in the Cytoscape network visualization software [Killcoyne et al., 2009]. In a recent publication, even a random layout of a network was shown in comparison to a force-directed one to illustrate the importance of applying 'high quality' layout algorithms, and vertex labels had still to be placed manually [Merico et al., 2009].

There is huge number of special purpose tools in biology and bioinformatics for the graph-based visualization of data, far more than can be mentioned here. They range from fully-fledged stand-alone analysis tools over web-based applications to plugins for biological software platforms. However, only few of them have a significant influence and are reliably maintained and developed. The overwhelming number of projects that create new tools, and the often short life cycle of tools and projects make it difficult to keep track of all developments (however, this is also true for pure graph drawing projects, roughly half of the software projects presented in the survey of Jünger and Mutzel [2003] are not maintained anymore). For surveys on graph-based analysis and visualization of biological networks and corresponding tools see for example Aittokallio and Schwikowski [2006], Suderman and Hallett [2007], Pavlopoulos et al. [2008]. Suderman and Hallett [2007] give an excellent overview on existing tools and the features they provide, with regard to the requirements of biologists. As a main result, they state that single static network views are not sufficient. Instead, visualization tools should provide a multitude of views of different types and levels of detail, and allow dynamic navigation between linked views. As a further weakness of existing tools, they identify the weak support for visual comparison of similar networks, especially time-series. Since then a couple of tools integrated at least basic comparison functionality, e.g., REACTOME allows to compare pathways between species. Pavlopoulos et al. [2008] provide a comparative review of network visualization tools, highlighting individual strengths.

As most tools often focus on a specific task or type of data, there is no tool that can be recommended as 'the' outstanding exploration tool. Many tools also lack capabilities required for real-world use by practitioners (exchange formats, database connections, access to web resources, annotation interfaces, to name a few), and therefore do not have the necessary supporting user community.

For many tools however, both visualization of and layout computation for large graphs is a challenge. For the IntAct toolkit for example, a limitation to 300 binary interactions is implemented to 'avoid the complications related to data visualization of big networks' [Aranda et al., 2010]. Besides the problem that many open-source tools do not have an efficient memory handling architecture, this has two main reasons: First, there are often only standard layouts implemented which are not applicable for large graphs. Second, if a layout specification adapted to the biological requirements is done, it is either overly complex or the applied optimization technique is very slow in practice. Nonetheless, there are several important, established tools and platforms that include specifically developed state-of-the-art drawing algorithms and advanced visualization approaches. Many of these tools are created by groups from the biology and bioinformatics community (e.g., Cell Designer, Cell Illustrator, CADlive), others have an origin in the graph drawing or information visualization communities (e.g., VANTED, ProViz, Cerebral, Patika). VANTED (Visualization and Analysis of Networks containing Experimental Data) [Klukas and Schreiber, 2010] is of specific interest from a graph drawing point of view as it provides a number of sophisticated layout approaches for biological networks and was one of the first tools to support the SBGN. Also Cell Illustrator features an advanced layout method, the grid layout described in Kojima et al. [2010, 2008]. Cerebral (Cell Region-Based Rendering And Layout) facilitates layout of and interaction with biological interaction networks using sub-cellular localization annotation and vertex grouping according to functional annotation. Cerebral employs a simulated annealing approach for layout computation; according to the authors, it scales well to several thousand vertices; see Figure 3.13. Note that in Barsky et al. [2008] it was explicitly stated that collaborating practitioners rated layout capabilities of several tools (Osprey, Tulip, Cytoscape) as inappropriate for their needs because they did not allow to model both the localization information and the functional grouping.

Requirements. Due to the large variety of different network types, annotations, and scientific questions, a different handling of the networks and also differing layouts are preferable, and there cannot be a distinct optimal representation. Based on the data and use-cases, several key challenges for the graph representation can be identified:

- Integration of semantic and experimental data (and also derived analysis data), e.g., expression data, localization information, and (dynamic) flow information.
- Highlighting and handling of characteristic substructures, such as motifs and modules.
- Comparative visualization of networks, e.g., from different cell states, organisms, or experimental data sets.
- Visualization of time series to show network changes over time.

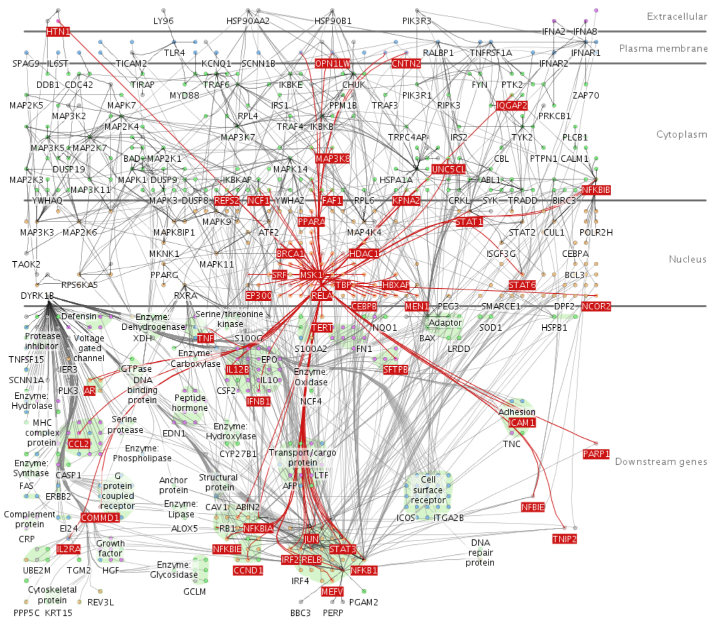


Figure 3.13: The layout of a signaling network in Cerebral, including sub-cellular localization information through horizontal layers that represent cell compartments [Barsky et al., 2008]. Highlighting is used to visually emphasize the neighborhood of a selected vertex.

- Resemble established drawing styles as close as possible.
- Visualization of and navigation in huge networks, as the data sets stemming from experiments and simulations are getting increasingly large.
- Consideration of uncertainty.

Each of those aspects, and the combination of several of them, are important and requested by the biology community. They all pose a challenge for current graph drawing research.

For interaction networks, classifications regarding type, similarity, and function of the involved actors have to be highlighted, e.g., by a grouping or a type-specific handling, to make identification and group affiliation possible. For most tasks also the visualization of localization or temporal information is a must. Clear separation of cell compartments and a directed, aligned drawing of reactants to reflect the temporal order are required. For networks that involve flow, e.g., signaling networks, the direction of the flow has to be respected, e.g., by a hierarchical drawing.

An extremely important task for which only a few methods exist is the comparison of multiple networks. These networks need to be visualized such that differences and similarities can be easily identified, e.g., by simultaneous drawings. However, existing approaches produce cluttered drawings for many real-world instances.

Even though biologists are used to cluttered drawings, interaction techniques that allow to visualize only relevant subsets of a graph at a time are needed. In a dynamic setting, collapsing and expanding of substructures can increase the clarity of the presentation and support the discovery process in large graphs. Specific patterns like motifs and functional complexes, which may be identified by an analysis of the graph and experimental data, should be clearly observable in visualizations for all types of biological networks, and similar subprocesses should be drawn in a similar way. Many tools exist to derive the corresponding information on modules and motifs from the network and the annotated data. Biologists are interested in identifying the role of the actors involved in biological processes, and visualization can support automated analysis, e.g., by highlighting key proteins or substructures. However, in contrast to areas like social sciences, there is no established visual paradigm how to do that, and as long as it is done, biologists don't seem to care much if a vertex is placed in the center or at the boundary of a drawing (however, important subgraph structures are frequently placed close to the center).

Networks for which established drawing conventions exist, such as metabolic and signaling pathways, have to be drawn in a way that adheres to these conventions. Conventions for pathway drawings include drawing important cascades as aligned paths, and cycles in a circular fashion. In addition, manual drawings often exhibit a certain symmetry, with a balanced distribution of angles around a vertex, and the edge routing is often nearly orthogonal. The placement of vertices and edges is restricted based on semantic information, e.g., the arrangement of participants of a reaction is done to reflect their role and temporal order in it: Substances and products of a reaction are clearly separated, and co-substances are placed out of the main path close to the reaction. Functional and type similarity is often reflected by aligning or grouping similar entities.

Chemistry

Although the use of graph structures in chemistry is not as obvious as in biology, they arise in several applications, ranging from modeling molecules as graphs for use in database indexing structures (see Willett et al. [1998], Klein et al. [2011]), over the representation of classification hierarchies (dendrograms), to network representations of property-based relations and similarities (see Section 9.1). The goal for graph based visualization approaches within the context of chemical projects is to enable the chemist to explore data, e.g., stemming from large compound databases, in an intuitive visual fashion by navigating along well-defined relations. Those relations can be based on structural identity or similarity of the chemical compounds, and are modeled into the navigation approach to allow the expert to either identify molecules with known specific properties or similar molecules that may be promising candidates for further investigation.

Independent of the different definitions of *chemical space* as either the space of all energetically stable, synthesizable molecules, all small organic (carbon-based) molecules, or simply all molecules that are potentially suitable for medical application, chemical space is big. Only a small fraction of the enclosed molecules, however, is biologically relevant and is therefore considered interesting enough for further investigation. Even though high throughput methods allow to process large numbers of compounds, still only a small part of the chemical space can be processed. Considering the effort in time and resources for experimental evaluation, navigation in chemical space should therefore reduce the complexity to a manageable level and at the same time allow to explore only the most interesting areas with respect to the task at hand. As a consequence, efficient and effective approaches for classification are needed. Depending on the goal of the analysis, rules may be applied to narrow down the set of compounds, e.g., by taking into account requirements for oral bioavailability. Nonetheless, identification of suitable candidates for further investigation, e.g. synthesis and experimental evaluation, involves expert knowledge and cannot be done in a fully automatic fashion.

Compared to other application areas, especially biology, the support of the analysis workflows in chemistry by integrated tools that combine both advanced visualization and interaction as well as analysis methods is rather weak even though the need for such tools has been formulated quite often. In addition, information needed for the purpose of data analysis is often spread across various data resources, making combination of the knowledge difficult. Recently there have been a few attempts to establish new software to remedy the situation, namely the tools Scaffold Hunter, Molwind, SARANEA, iPHACE and Scaffold Explorer which aim at facilitating the visual navigation within chemical compound databases.

The very recent development in visualization and analysis tools shows both the potential of and the need for more sophisticated integrated visual analytics software. Graph drawing methods can mainly be used to allow the intuitive navigation in the data, where no comparable methodologies existed so far.

Requirements. For network visualizations that are related to biochemical processes, the requirements are similar to the ones given above for biological networks. Regarding other graph-based visualizations, there are no established requirements so far, because their use is just in the beginning.

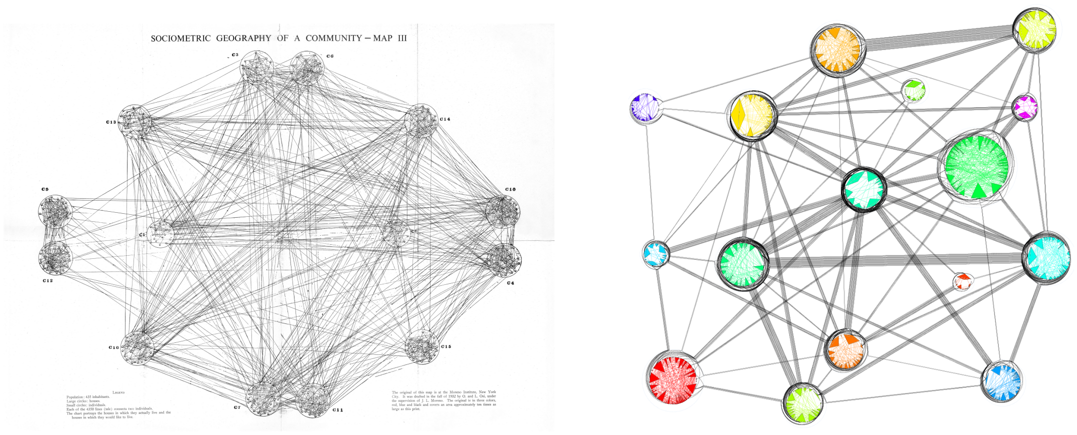
3.2.2 Social Sciences

Graphs are used in social sciences to model relations between individuals or communities and to represent the temporal change of those relations. Already in the 1930s Moreno [1953] introduced *sociograms* to model social networks, using graphs where *actors* are modeled as vertices and relations between them as edges or in the case of directed relations as arcs; see Figure 3.14(a). The resulting interaction graphs are then analyzed to detect structural and behavioral patterns and key players, to show flows, and to derive rules that allow to describe the dynamic development. An appropriate representation of the dynamics will help to detect and interpret structural changes over time. Regarding the layout quality, Moreno wrote 'the fewer the number of lines crossing, the better the sociogram'. The analysis of social networks is an important part in social sciences, as it can facilitate to identify the role of an individual or group within a larger entity, e.g., of a person in a peer group or an organization in society.

According to Freeman [2004], analysts try to uncover mainly two kinds of patterns: (1) those that reveal subsets of actors that are organized into cohesive social groups, and (2) those that reveal subsets of actors that occupy equivalent social positions, or roles. Automatic social network layout methods should therefore strive for layouts that highlight such structures. Typical further analysis tasks include the identification of hubs (important actors), common neighbors and neighborhood analysis, and path-finding. For some of these tasks, graph representations compete with matrix representations, but have also been used in a combined visualization approach [Henry and Fekete, 2007, Henry et al., 2007] to combine the strengths of graphs for path detection and the compact representation of dense subgraphs in matrices.

The identification of communities within social networks is often modeled as a graph clustering problem, and a visualization of the network then needs to reflect the corresponding clustering result; see, e.g., Figure 3.16. An interactive visualization approach has to allow aggregation of the detected communities, both to give a better structural overview and to reduce complexity. However, groups in social networks may not always be disjoint, making it difficult to apply the standard hierarchical clustering approaches. In order to identify important actors, a filtering of less important parts, e.g., actors with a small degree of connectivity, is also important and already realized in many analysis tools. An animated change in the layout for filtering is not wanted in each case, as the original positions and distances might be of interest for analysis.

Besides community detection, several network analysis measures and techniques are used to gain further knowledge. Centrality analysis aims at identifying the most important actors and is done by applying a centrality measure, like degree, betweenness and closeness centrality. Such measures can be used to derive distance or relative position information. In order to allow a visual analysis of these values, layouts should reflect such measures; see Figure 3.15. The need to represent distances has lead to the frequent use of radial and distance-based layout methods. Also circular drawing conventions that allow to map levels of organization have been proposed [Baur and Brandes, 2007, Baur et al., 2009]; see Figure 3.14(b). Baur [2008] distinguishes 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 of the network, and can be used to get an overview on the general graph structure. Analytic visualization refers to drawings which express the exact results of an automatic analysis, mapping them to corresponding vertex positions and edge routing.



(a) (b)

Figure 3.14: (a) Sociogram from Moreno's classical publication [Moreno, 1953] (b) a visualization of an email communication network that also takes institute affiliation into account to achieve a two level layout [Baur et al., 2009]. Both visualizations represent group structures, but the circular micro/macro graph layout in (b) gives a better overview on the structure and is less cluttered.

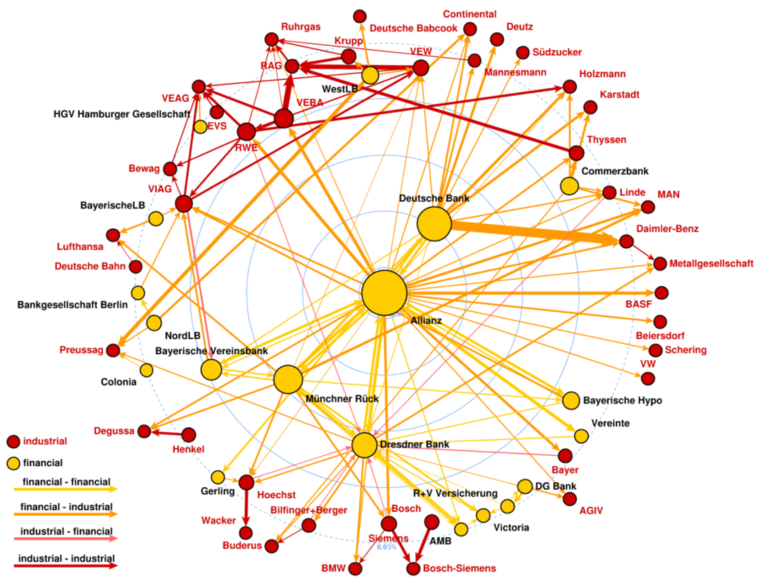


Figure 3.15: Visualization of a social network with a centrality-based layout, taken from the Visone web page. The network represents German capital ties in 1996. Typical for social interaction networks, central actors need to be placed in the center of the drawing, and vertex positions reflect centrality values. Also attributes such as vertex size, edge width, and color are used to reflect additional properties like importance and type of actors.

A common visualization feature is the mapping of data annotations to graphical attributes like size and color. Layout methods that do not take into account vertex sizes are therefore not well suited for social network visualization. In addition, also the edge width may be used to represent information; see Figure 3.19.

With the advent of modern telecommunication and internet technologies, there is an additional demand for the analysis of the corresponding information, and a wide variety of data is suitable for graph-based visualization. Prominent examples are social network communities on the internet, email and chat contacts, interaction graphs in criminalistics, citation relations in scientific communities, and the relations in popular movie and actor databases. The overwhelming number of online community services even lead to the term 'YASNS – yet another social networking service' [Shirky, 2003]. Whereas classical analysis typically had to cope with small sets of actors, the data sets that are collected in modern applications are huge with billions of entries, providing a new challenge for analysis and visualization methods. Social network analysis tools therefore have to be capable both to layout and handle small as well as huge networks properly, and need to provide navigation methods. Graph structures modeled from such data may also include dense substructures (e.g., closely related communities may form cliques), which need to be treated efficiently. There are three basic approaches to visualize large data sets: Just start with a view on the whole network (e.g., SocialAction), try to avoid clutter by reducing the view to a small focus set (e.g., Vizster [Heer and Boyd, 2005] and TreePlus [Lee et al., 2006a]), or allow an aggregated view on the data (e.g., PivotGraph [Wattenberg, 2006]). Without further methods to analyze and arrange a large network on screen, a 'whole network' view clearly makes detection of patterns extremely difficult.

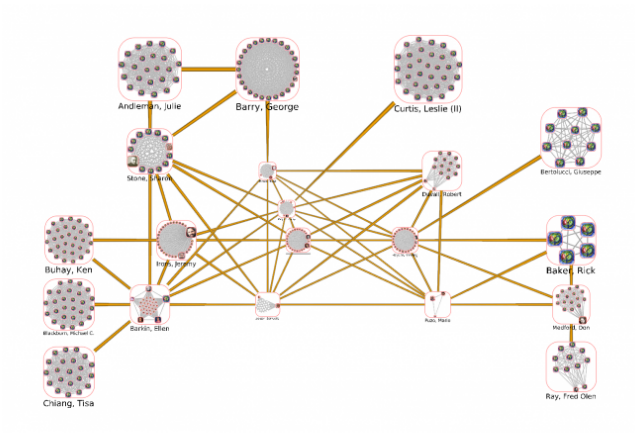


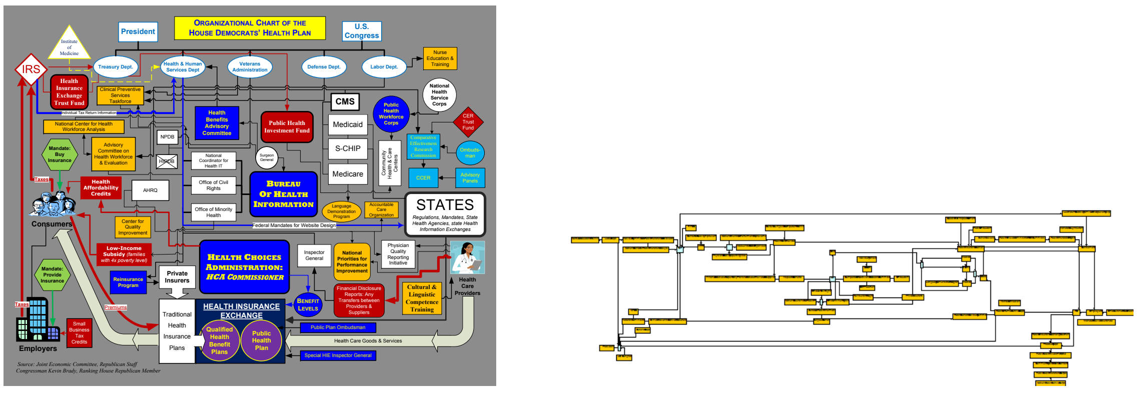
Figure 3.16: Visualization of a part of the IMDB movie database visualized with Tulip, taken from the project webpage [Tulip]. The force-directed layout of a clustering result was improved by manual alignment. In contrast to the visualizations in 3.18, the main focus is on an overview and the clear separation of subgroups.

In addition to the pure relational structures, the data is often enriched with temporal information, additional group structures (e.g., membership in organizations), and other supplementary information, e.g., demographic data which can be exploited for analysis. The dynamic behavior of social systems is an important aspect that needs to be represented, see, e.g., Brandes and Corman [2003] where a two-and-a-half dimensional stacking of subnetworks was used to unroll the network evolution.

As the modeling and visualization of social relations as networks is an established approach for analysis, there is already a decent amount of research done for customized graph layout methods, and also many interactive analysis tools have been implemented. Consequently, the term 'YASNAT – yet another social network analysis tool' was coined, showing that there doesn't seem to be a lack in tools, but none of them seems to be satisfactory for a large range of practical purposes, such that they are interesting and flexible enough to be maintained and extended over a longer period of time. Besides the support for graph analysis and statistics in general purpose tools and libraries, several graphical exploration tools tailored for social network analysis were developed, for example visone Baur [2008], SocialAction, and vizster. The visone software provides dynamic layout and animation facilities, vizster uses a spring-embedder with a Barnes-Hut implementation, and maps connectivity to spring strength.

Interactive web tools that allow to explore databases using featured static or dynamically created maps are getting more and more popular. The NNDB Mapper for example is a tool to visually track and explore the connections and activities between people from science, politics, business, and culture in the *Notable Names DataBase*, in order to document connections that may not be obvious, but helpful to explain a person's behavior. The implemented layout and navigation methods are however often rather poor, see Figure 3.18(b) for a visualization using the NNDB mapper.

An intentional use of a bad social network visualization can be seen in Figure 3.17. The network shows the relations of institutions involved to the U.S. health care reform plan,


Organizational Chart of the House Democrats' Health Plan
President
U.S. Congress
IRS
CMS
Medicaid
S-CHIP
Medicare
STATES
Consumers
Employers
Health Choices Administration: HCA Commissioner
Health Insurance Exchange
Bureau of Health Information
Private Insurers
Traditional Health Insurance Plans

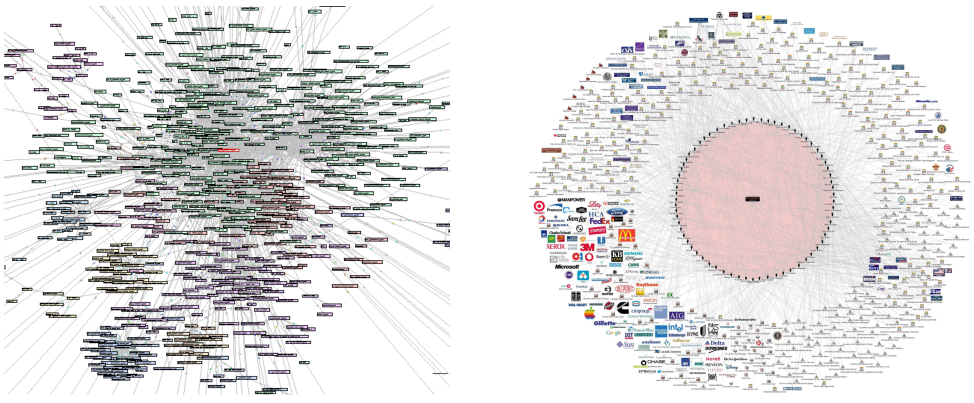
(a) Original (b) OGDF Orthogonal

Figure 3.17: Visualization of the Democratic Health Care Plan released by a republican politician, obviously published to make the plan look overly complex and confusing, and an orthogonal layout of the same graph structure (hyperedges modeled with dummy vertices (blue)) using OGDF.

and the cluttered layout is used to make the structures complex and difficult to understand. Obviously, some design rules or constraints were used, like the alignment of the 'president' and the 'U.S. Congress' vertices on top of the drawing. However, it is not the application of these constraints that makes the layout look so cluttered, as the intention was to create the impression of an overly complicated and confusing system. The orthogonal drawing derived using OGDF's topology-shape-metric implementation shows that the graph structure alone is rather simple. It needs only a single crossing, at the expense of more white space.

An interesting application of social network visualization is the analysis of *citation graphs*, i.e., graphs that link scientific papers by citation relations. The visualization of citation graphs can reveal interesting patterns and information about a research area. Citation graphs show the impact of a research community to other communities and also the influence received from other areas; see Figure 3.19(a). Properties like connectivity, diameter, degree distribution, cocitation, and community structure are useful indicators for the characterization of the research activity [An et al., 2004, Reid and Chen, 2007, White and McCain, 1998]. Also the evolution of those aspects over time can be of interest. The Graphael system allows the animated visualization of evolving graphs and was used for the interactive visualization of topic, collaboration, and citation graphs [Erten et al., 2003]. Its layout is computed with a force-directed approach, based on a modification of the GRIP algorithm [Gajer and Kobourov, 2002].

Requirements. As shown in McGrath et al. [1997], the layout of social networks can have a large impact on community detection and role perception, vertices closer to the center are assigned a higher importance, and vertices close to each other perceived as a group. Different layouts can highlight different features of a network [Blythe et al., 1995, McGrath et al., 1997], and may therefore be needed for visual analysis. Huang et al. [2007] summarize for their user study that the usefulness of a drawing convention for social networks does not only depend on the task, but also on the expertise of the viewer. With regard to the priority



(a) (b)

Figure 3.18: Social network visualizations with a high level of visual clutter which makes it difficult to derive any structural information. However, the right picture was created to display the density of the link structure, whereas the left picture was created for visual analysis purposes. (a) Visualization of the enron email communication, visualized using vizster. Colors indicate affiliation to closely related communities, based on a hierarchical clustering algorithm. Courtesy of Jeffrey Heer. (b) Network visualization created using the NNDB Mapper and showing 69 participants of the Bilderberg conference and their affiliations to social institutions, companies, and governments [PSR].

of aesthetic criteria, standard criteria like edge crossings, uniform direction of edges, vertex separation, and angular resolution showed relevant impact. However, while the number of edge crossings had an impact in user performance, spatial layout features were a main cause for misinterpretation of the data. Therefore the authors recommend not to distribute vertices evenly, in contrast to the popularity of force-based layout methods in social sciences. Drawing conventions preferred by the users did not always lead to improved task performance. Hierarchical layouts for example, even though perceived by users as high in usability, did not lead to high response accuracy, very likely due to angular resolution issues.

The positioning of actors in the center, or at a specific distance from it, and considering relative distance values of vertices to each other are typical requirements in social science visualizations. A requirement which is often neglected in graph drawing software is the need to place vertex and edge labels. As labels occur frequently in social network maps, and can often be of considerable lengths, bad placement can significantly deteriorate the quality and usefulness of the visualization. Also the vertex size and the edge width are used as visual cues and therefore need to be considered. Overlap removal techniques like the one of Gansner and Hu [2010] might help to cope with these requirements. Animation of the structural evolution is already a widely used instrument. Highlighting of key actors and clusters supports the analysis process. However, peripheral actors may also be of interest, as they could represent connections to external networks. For navigation purposes, expand-and-collapse techniques for substructures such as clusters can be used.

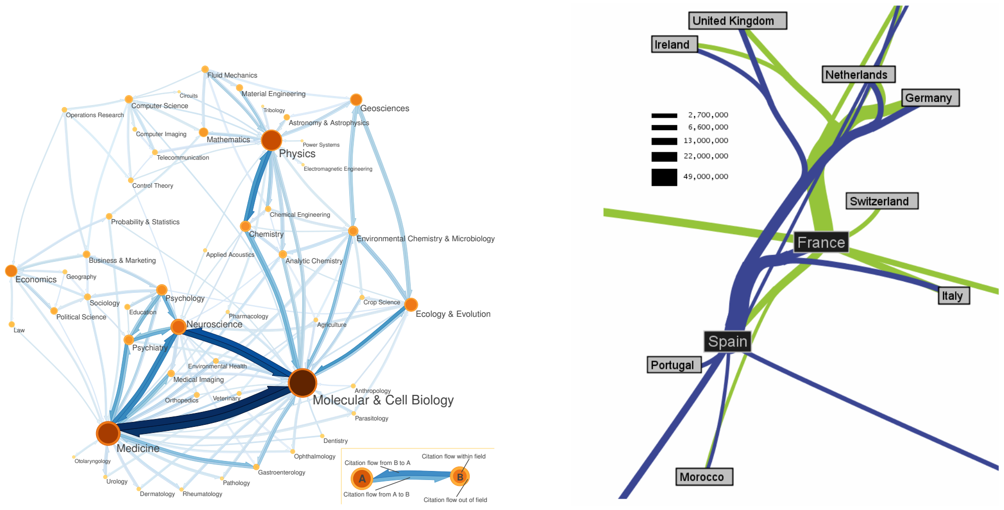


Figure 3.19: (a) Visualization of the citation flow between several scientific fields, where edge width represents the amount of flow [Eigenfactor]. (b) Flow Map Layout showing imports to Spain and France as visualized in Phan et al. [2005], the edge width represents the amount of flow. The layout tries to preserve the relative positions of the vertices to each other, minimizes edge crossings, enforces vertex separation by employing the force scan algorithm of Misue et al. [1995], and merges edge that share destinations by applying a hierarchical clustering based on Euclidean distance.

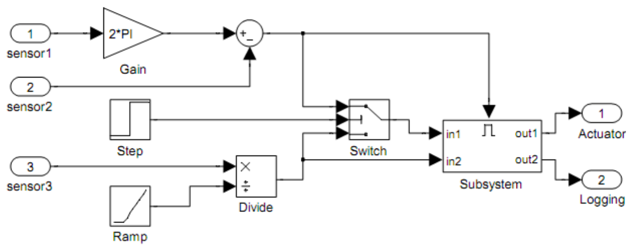


Figure 3.20: A data flow diagram from Simulink [Spönemann et al., 2010].

3.2.3 Software and Hardware Engineering

There is a large variety of graph applications in software engineering. Data flow and entity-relationship diagrams are used in CASE tools to depict the architecture of a software system, where relations between classes and modules in a program are modeled by a graph structure. Requirements comparable to the ones in software development can also be found in hardware development, as similar graphical modeling paradigms are used in modeling tools such as LabVIEW and Simulink. They allow a model-based design of systems over a graphical diagramming interface, employing state-charts and flow diagrams. See Figure 3.20 for an example data flow diagram.

In general, different aspects of software systems like structure, behavior, and evolution, can be covered by quite different types of diagrams. The Unified Modeling Language (UML)

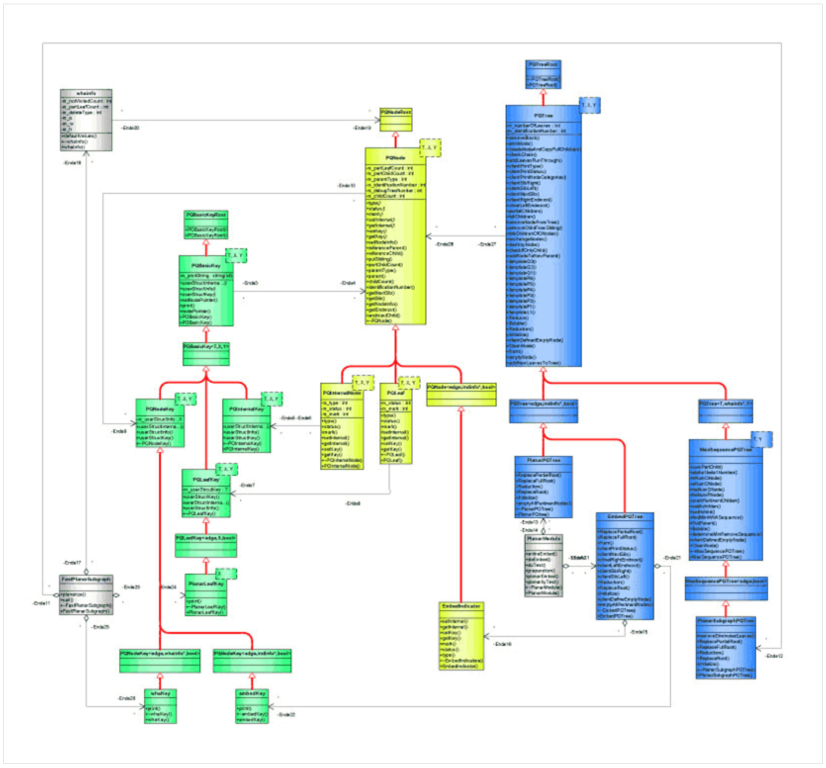
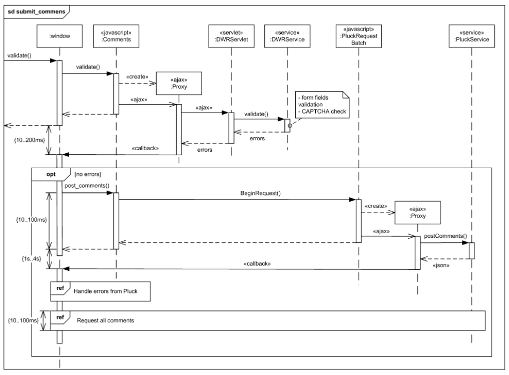


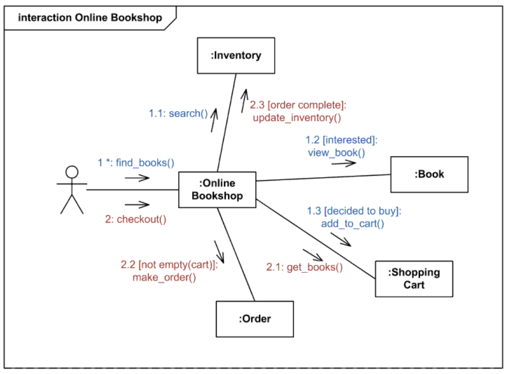
Figure 3.21: A UML class diagram representing a part of the OGDF library, drawn with OGDF's UML class diagram orthogonal layout. Structural subunits, in this case inheritance hierarchies, are clearly separated and drawn in a common direction. However, vertices are not aligned corresponding to the inheritance depth.

was developed to standardize the modeling and representation of these aspects, and comprises a set of corresponding diagram types. Structure diagrams, as, e.g., the UML class diagram, model a static structural view on the software and may contain abstract and implementation concepts; see Figure 3.21. Behavioral diagrams show the dynamic behavior of the objects in a system, including methods, collaborations, activities, and state histories; see Figure 3.22. The dynamic behavior of a system can be described as a series of changes to the system over time [UML, b]. The UML visualization standard allows to freely exchange software specifications and documentations and helps that these representations are readily understood by everyone involved in software engineering, decreasing design and maintenance time and effort. Going beyond the pure graphical notation, semantic information in UML allows to also verify and even execute models. Regarding visualization, UML's *Diagram Interchange* specification aims to provide an exchange format for diagram layouts.

Purchase et al. [2001] investigate the specific demands for the visualization of UML class diagrams, identifying the most important aesthetics for human comprehension. As most of the aesthetic criteria under investigation did not produce any significant results when considered alone, the authors assume that it is important to reflect semantical information, which is not considered by generic layout algorithms. In class diagrams, edges have a specified type: In the most basic case, they can be either associations or generalizations. Generalizations



(a)


interaction Online Bookshop
:Inventory
1.1: search()
2.3 [order complete]: update_inventory()
1.2 [interested]: view_book()
1 *: find_books()
:Online Bookshop
:Book
2: checkout()
1.3 [decided to buy]: add_to_cart()
2.2 [not empty(cart)]: make_order()
2.1: get_books()
:Shopping Cart
:Order

(b)

Figure 3.22: (a) A UML sequence diagram that shows the message interchange between lifelines. (b) A UML communication diagram [UML, a]. UML communication diagrams also show message flow between objects, but do not focus on the order of messages. The layout therefore has a less restricted form.

are directed and constitute inheritance hierarchies, which are the most important substructures for these diagrams. These substructures should be clearly identifiable in a diagram. There was only one criterion in the experiments of Purchase et al. [2001] that significantly influenced the task performance: the number of bends. Nonetheless the authors recommend the use of bends to support the semantic grouping of closely related objects, for example subclasses in an inheritance hierarchy.

Purchase et al. [2000] and Eichelberger [2003] state that there is no clear specification for the readability of UML diagrams. Wong and Sun [2006] suggest that corresponding rules can be derived from perceptual principles. They survey and classify related criteria from previous research and notation guidelines. Avoiding crossing lines is a rule often found in Best Practice guidelines for UML modeling (e.g., Ambler [2005]). Sometimes also diagonal and curved lines are noted to be unfavorable, whereas symmetry is promoted as it helps to understand the structure better and allows to detect recurring patterns. Vertices should be distributed well, with a lower bound on the distance to other vertices and edges. Poranen et al. [2003] discuss characteristics of good layouts for sequence diagrams and define three types of general constraints for sequence diagrams:

- Horizontal distance: Uniform horizontal distances between participants.
- Vertical distance: Uniform vertical distances between message arrows.
- Starting object: The object that starts communication is drawn to the left.

However they also give examples where these constraints should have smaller priority than other goals, e.g., when vertical distances should reflect exact message starting times.

Several tasks have quite different requirements: High-level modeling uses relatively small diagrams, which are manually edited in an incremental fashion, and often involves user-defined layout constraints. Due to the lack of support for mental map preserving automatic layout capabilities in today's graphical software modeling tools, an often used modeling guideline is 'focus on content first, appearance second'. If interpreted to mean that during the creation process, no time should be spent in rearranging the layout for readability reasons, the guideline ignores the facts that readability might also help in the creation process. During an iterative modeling process, readability is needed to understand and improve the model, as software projects may grow and be modified over a longer period of time, and are often developed by groups of developers at the same time. An incremental automatic layout that respects user-defined constraints would therefore be a real improvement. Such a diagram layout should be stable during the course of dynamic engineering or navigation processes.

Regarding the behavior of software systems, flow of control and data and also temporal aspects need to be visualized. For debugging and tuning purposes, hot spots and unused code should be clearly identifiable. Sequence and communication diagrams are examples for diagram types that allow to model the dynamic behavior of the software. They show the message flow between objects and differ clearly in the visual representation; see Figure 3.22. UML sequence diagrams describe the dynamics of a system as the interaction of participating objects with a temporally ordered sequence of exchanged messages. The vertical dimension in a sequence diagram represents the flow of time from top to bottom, vertical *lifelines* represent the interacting objects and are connected through horizontal message arrows. Jacobs and Musial [2003] link program execution to a UML class diagram. They modify the standard UML class diagram with a focus-and-context technique to provide access to both high level structural information and low level program details. Malloy and Power [2005] combine class and call graph diagrams in an interactive profiling tool that allows to show dynamic sequence and communication diagrams during program execution.

Besides specific semantic type information, *packages* are a main structural element in software engineering to group model elements. Packages may be nested and can be applied to model software packages and modules as in the JAVA and Modula languages, to distinguish namespaces as in C++, or just to represent the main parts in the organization of a software project. Packages should be integrated into a navigation approach for visualization, for example by using an expand-and-collapse mechanism. Use of packages is a major tool for abstraction of increasingly large and complex software systems. Other techniques involve a filtering based on a visibility status, and semantic zoom that hides object details with different levels of detail for different parts of the graph. Even in the absence of such package structures, both overview and detail visualizations for large software projects are needed.

Active software projects are subject to permanent dynamic changes, and the course of development can be modeled as a graph to track the evolution of a software [Collberg et al., 2003]. Frishman and Tal [2004] describe the use of a dynamic clustered graph representation for the visualization of mobile object frameworks, where objects are represented by vertices, and machines are represented as clusters containing the objects that are currently assigned to them. Edges between vertices represent logical relations between the objects, and connections between cluster representatives model physical connections between machines. As the objects might migrate to remote hosts during application execution, and machines (dis)connect to the network, the visualization needs to be dynamically updated. A

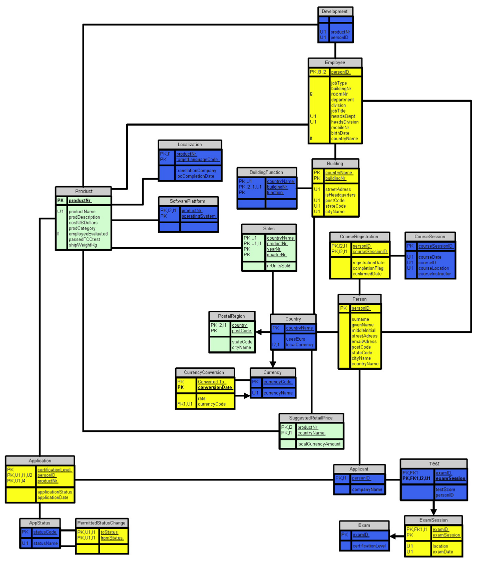


Figure 3.23: Visualization of a database schema, drawn in orthogonal style using OGDF.

nice example for an interactive graph-based software visualization is the web-based interactive linux kernel map [Shulyupin]. It allows to browse a top-down view of the linux kernel with a clustering into different layers, functional levels, and electronics.

Even though the UML is a widely adopted notation standard, entity-relationship diagrams (ERDs) have been used already for a long time, especially for relational database schema diagrams. Database schema diagrams show the structure and relationships of tables, views, and other database entities; see Figure 3.23. Such non-UML diagrams are still widely used and have specific drawing requirements. Orthogonal routing of the edges and a crossing minimization are preferred, but clear depiction of the schema structure is more important than a compact layout. Further layout rules include that directed edges are drawn in a common direction and tables are aligned in columns. This makes important (primary) tables better visible, which are typically not dependent on other tables. Edges should only attach at either the left or right side of a vertex. As vertices are in general large columns, automatic layout methods need to respect the size.

Electric circuit design and VLSI layout are examples for early uses of graph drawing methods, and several techniques used were also adopted for graph drawing purposes, as for example compaction heuristics. Layout diagrams do not have the goal of an 'aesthetical' graph representation, but represent the physical design. Constraints bound distances and lengths, e.g., to avoid signal interference and timing problems. Hence, the layout computation is highly constrained. In addition, instances can be very large, therefore often only fast heuristics can be used. Technical drawings like circuit diagrams are not meant to directly reflect a physical layout, but mainly represent the circuit logic for human understanding; see Figure 3.24. In a variety of application areas, such technical schematics are used for hardware design, and a fully developed and established graphical notation exists that is supported by a variety of software systems, e.g., CAD systems. Layouts for these drawings have

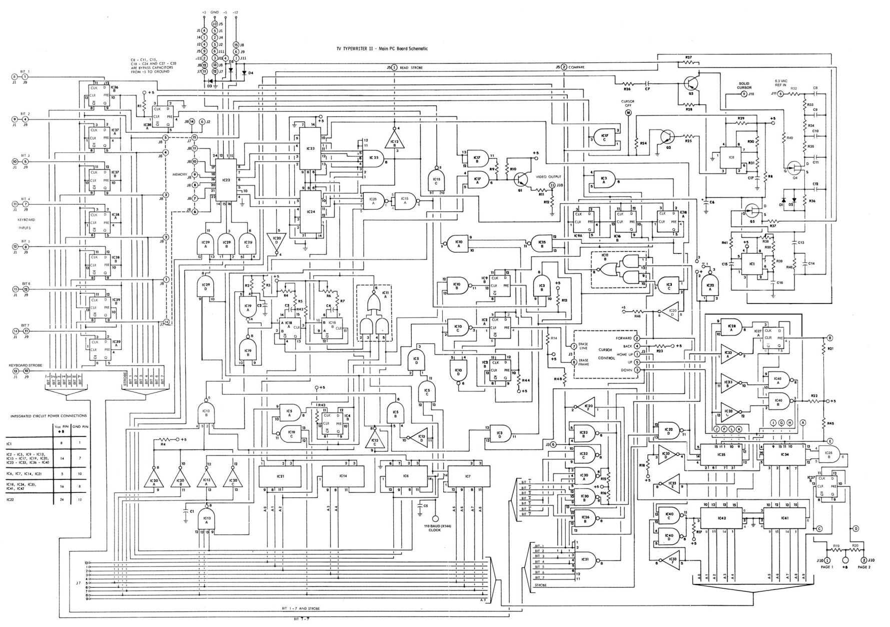


Figure 3.24: Schematic of a terminal system. Objects have fixed ports for edge attachment and edges are bundled to decrease clutter. Alignments of similar components and proximity of functionally related components support the understanding and guide the user's orientation even in a static visualization.

to reflect information like object size, minimum and maximum distances, allowed positions, ports, or relative order. The edge routing is done exclusively in orthogonal style. Drawings are also often organized in the direction of signal flow, e.g., from left to right.

Requirements. Several requirements stem from the nature of the depicted models: Vertices often have prescribed shapes that denote their function or type, edges are directed and possess fixed ports. Ports may either fix exact positions or only specify the vertex side at which to connect. A source port may be connected to multiple target ports, thus creating a *hyperedge*. These hyperedges then should not be visualized by several distinct edges, but combined as, e.g., the generalizations leading to the same parent classes in Figure 3.21.

The predominant drawing style for flow diagrams is orthogonal grid drawing; especially for hardware development, nearly all tools require edges to be drawn orthogonally. As bends in class diagrams may hamper readability, bend-minimizing drawing methods seem to be well-suited for layout computation.

Semantic information may specify the main structures that have to be highlighted: In order to clearly depict inheritance hierarchies in class diagrams, these have to be visually separated, i.e., generalizations should not cross and subtrees of different hierarchies should not be intermingled. In addition, all edges in inheritance hierarchies should point into the same direction. Signal and data flow direction has to be respected, e.g., by left-to-right drawings. The horizontal ordering of lifelines in sequence diagrams is not fixed and can therefore be chosen such that the length of the message arrows is minimized.

Vertices may also contain other vertices in a hierarchical composition, like in package structures, the visualization then has to draw the induced subgraph of the contained vertices within the composite vertex. For large graphs, the decomposition can also be used for abstraction and navigation purposes.

3.2.4 Further Application Areas

Out of the large number of application areas, we will shortly discuss a few further prominent examples which are either interesting because of their additional requirements or because the use of graph drawing is established with characteristic visualization conventions.

Security. Attack graphs model the security of a computer network and allow to identify paths that enable an attacker to penetrate a secured network. Depending on the model used and the analysis task, vertices in an attack graph represent states of a network, machines, attack goals, or attacks, and edges represent exploits, show the order of attacks, or model how the exploit of a condition can result in subsequent network conditions. Due to the directed nature of the edges and the importance of paths that lead to a specific attack goal, often hierarchical layouts are used to visualize attack graphs. O'Hare et al. [2008] emphasize the value of alternate layouts for the analytic benefits of graph visualization, as it fosters the recognition of fundamental underlying patterns that might otherwise be invisible. They also stress the fact that incremental layouts are highly preferable, both due to performance and orientation issues, to allow interactive navigation and exploration of attack graphs.

Attack graphs can be quite complex and therefore navigation methods, especially expand-and-collapse mechanism are needed to handle them; see Figure 3.25. Noel and Jajodia [2004] consider managing complexity in user interaction as the greatest challenge in making attack graph analysis practical for real networks. A key concept for this goal is the dynamic aggregation of subgraphs, for example according to protection domains (sets of machines with unrestricted access to each other).

Homer et al. [2008] propose techniques to group attack steps that share similar semantics, and also rules to trim the graph by removing steps that are rated as omissible for the understanding of the core security problems in a network. There are not only many tools to generate and manage attack graphs automatically, the security community is also well aware of the value of meaningful visualizations of these graphs. Several publications concerning specific visualization approaches for attack graphs exist, and also libraries and methods from the graph community are well known and used (e.g., Homer et al. [2008] uses Graphviz, O'Hare et al. [2008] uses yworks).

Business processes. Already for a long time, business processes have been visualized as graphs, and a huge number of modeling tools exist that allow a graph-based process modeling approach. Within business process management, workflow models represent real-world business processes. Within the corresponding process graphs, vertices can represent actors, entities, or activities and directed edges represent control flow. These models can be used to document and communicate processes, and workflow management systems may even allow their machine-aided execution. Compared to other application areas, the visualization is mostly used in a more static manner, but if automatic layout methods are used to improve the

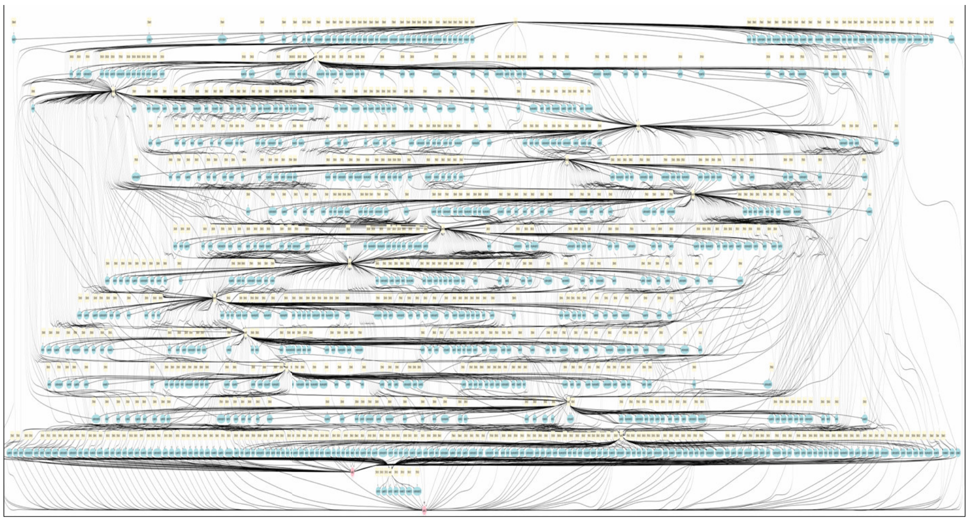


Figure 3.25: Complex attack graph visualization as visualized in Noel and Jajodia [2004]. A hierarchical layout style is used for this overview visualization. It is extremely difficult to track single paths or to identify relevant substructures.

visualization and subsequently the human understanding of the models, they need to allow at least incremental drawing during the creation of a process model. Business process models (BPM) are typically relatively small, such that they can be completely visualized on a single page or on screen.

A main concern is the visualization of control flow, i.e., the execution order of activities. This includes not only the order of the objects, but also an alignment of main flow paths. Vertices in BPM visualizations are never allowed to overlap, and manually created drawings nearly always show quite large and uniform vertex separation spacings. In addition, orthogonal edge routing is often preferred.

The Business Process Modeling Notation [BPMN, a] provides a widely adopted standard for the graphical notation of business processes; see Figure 3.26 for an example. Whereas BPMN is mainly concerned with the graphical representation for human understanding, the Business Process Execution Language [BPEL] allows execution and simulation. Models in BPEL therefore often contain more details, as they have to include most aspects of a process for simulation, including special cases. Another main visual language are Event-driven Process Chains (EPC) [Scheer et al., 2005]; see Figure 3.27 for an example.

Business process visualization is highly constrained, but often up to a point that together with the occurring classes of graphs makes the drawing easier. *Pools*, *lanes*, and *subprocesses* can be used to separate different aspects of a process, such as organizations, functions, or locations, e.g., to distinguish the internal part of a process from the customer part. A pool represents a participant (e.g., a company) in a collaboration and may contain a corresponding process (i.e., the process may not leave the pool). Message flow edges model the interaction between pools. The pool can be further partitioned by lanes to show the departments or roles responsible for the participants activities. The use of pools and lanes with a fixed geometric order of subgraphs, the hierarchical structure due to the flow, and the need to align main

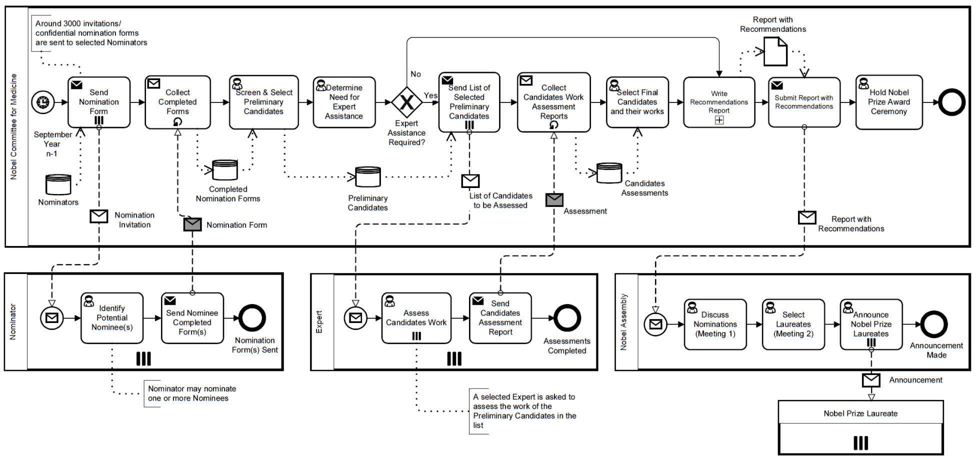

Nobel Committee for Medicine
Around 3000 invitations/ confidential nomination forms are sent to selected Nominators
Send Nomination Form
Collect Completed Forms
Screen & Select Preliminary Candidates
Determine Need for Expert Assistance
No
Yes
Expert Assistance Required?
Send List of Selected Preliminary Candidates
Collect Candidates Work Assessment Reports
Select Final Candidates and their works
Write Recommendations Report
Report with Recommendations
Submit Report with Recommendations
Hold Nobel Prize Award Ceremony
September Year n-1
Nominators
Nomination Invitation
Nomination Form
Completed Nomination Forms
Preliminary Candidates
List of Candidates to be Assessed
Assessment
Candidates Assessments
Report with Recommendations
Nominator
Identify Potential Nominee(s)
Send Nominee Completed Form(s)
Nomination Form(s) Sent
Nominator may nominate one or more Nominees
Expert
Assess Candidates Work
Send Candidates Assessment Report
Assessments Completed
A selected Expert is asked to assess the work of the Preliminary Candidates in the list
Nobel Assembly
Discuss Nominations (Meeting 1)
Select Laureates (Meeting 2)
Announce Nobel Prize Laureates
Announcement Made
Announcement
Nobel Prize Laureate

Figure 3.26: BPMN example, showing the Nobel prize process model. Diagram taken from the BPMN 2.0 example document [BPMN, b]. The diagram uses aggregation of activities by pools based on the groups involved, representation of control flow through ordering of vertices and edge direction, and also alignment of task flow.

characteristic paths, combined with the often very sparse graphs, sometimes do not leave much space for layout optimization. Semantics of the edges regarding different types of flows should also be taken into account. For example, horizontal sequence flows and vertical data and message flows with a small number of crossings for the message flows will improve the readability and understanding of the model.

Business processes can also be hierarchically decomposed in subprocesses, such that a top-level diagram only shows a rather abstract view on several processes that again can contain subprocesses down to the level of a single task. Decomposed processes have their own notational element, a '+' mark, to indicate that there are subprocesses involved. These can be visualized either by a separate diagram or by showing a thumbnail sketch in the parent vertex. Subprocesses and pools may also directly be connected by edges attaching at their boundaries.

Standard layout methods, even though applied in several tools, do not respect the combination of these requirements well. The clustered orthogonal drawing in Figure 3.28 is a nice representation of a process model with respect to the aggregation due to the department structure. It does, however, not represent the flow by maximizing paths with common edge direction. Depending on the task at hand, different visualizations of the same graph may therefore be appropriate to highlight different characteristics of the business process, either alternatively or simultaneously. Besides the use of standard graph layout methods in established business process modeling tools, also attempts have been made by the graph drawing community to provide interactive tools with layouts especially suited for the requirements, e.g., Effinger et al. [2009] present an interactive tool for BPMN.

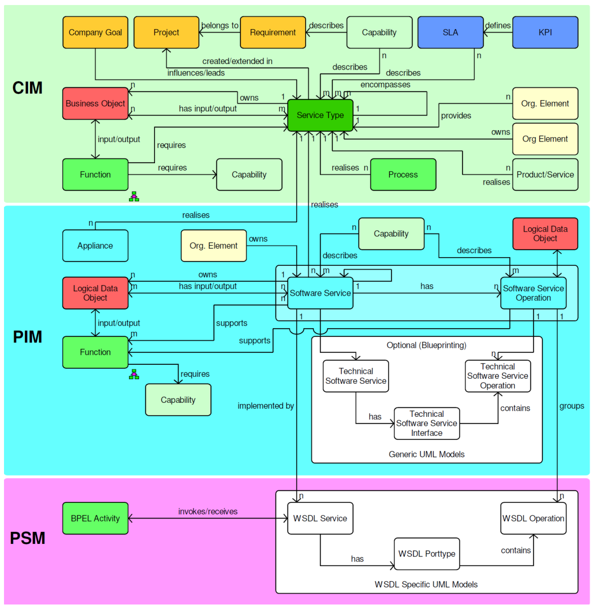


Figure 3.27: An event-driven process chain, using the typical orthogonal drawing style and clustering [Stein, 2008]. Three main layers can be identified, which are further subdivided by clusters.

3.2.5 General Requirements from Practical Applications

Although most practical applications have their own specific requirements for graph visualization, there are requirements that are important in multiple areas. There are, however, differences between tasks that cope mainly with empirical and experimental data, and tasks that involve modeling. Empirical data sets like the ones from biology, chemistry, and social sciences, are often dense, large, and require navigation techniques and support for analytical tasks like pattern finding or similarity search. Structures stemming from modeling processes, like business process models or software architectures are rather modest in size (although software projects can also grow over time to huge proportions). The application areas that use modeling also specified modeling languages and standardized rules for visualization early, which allow better communication of both structure and semantics. As an example, non-overlap is an often stated and much appreciated optimization goal in all application areas, but visualizations in biology still often exhibit overlap, whereas this is absolutely unacceptable in software engineering visualizations. A special role plays biology, as there are traditional drawing conventions from textbooks which are well-established for several diagram types (e.g., metabolic pathways), whereas for others there is no convention at all (e.g., protein interaction networks). One could argue that the existing drawing conventions in biology also stem from the manual creation of pathway drawings in the beginning, which somehow resembles a modeling process, whereas protein interaction networks were used to display raw experimental data.

The most frequent requirements are:

Grouping of Objects: The organization of objects in groups is a common pattern in many application areas, either due to strong structural relationships in such groups (e.g., modules in biological networks), semantic information (e.g., department structure in an organization),

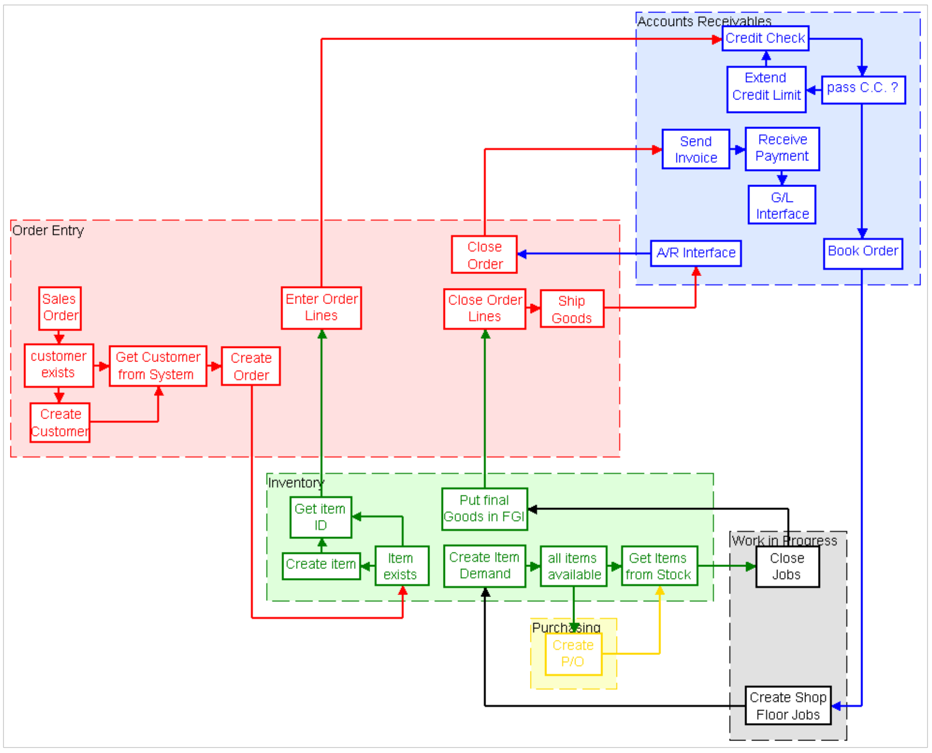


Figure 3.28: A business process model visualized as a clustered graph drawing. Clusters represent departments.

or information on the spatial distribution (e.g., cell compartments or sub-cellular location). An important requirement for the visualization is therefore that these groups are represented such that the resulting communities are easy to detect. This is typically modeled in graph drawing using hierarchical cluster structures, see Section 7. However, in practice several group structures may be of interest at the same time, e.g., localization information and functional categorization for proteins. In addition, organizational structures like the pools and lanes in BPMN diagrams have a more restricted structure with regard to nesting and at the same time impose constraints on the relative position. For example, the lanes of a pool are stacked in one dimension and edges between lanes may pass through in-between lanes. When the lanes are modeled by clusters, such crossings are not allowed in a clustered planar drawing. Additional techniques therefore need to be applied to handle such a partitioning of the drawing area well.

Reflection of hierarchies: A further requirement that occurs in many areas is that a (hierarchical) order of objects has to be reflected by the visualization, to represent the direction of a flow or chronological order, such that the sequence of events can be easily traced by users. This may also imply that either all or subsets of edges have to to be drawn monotonically in the same direction. Several levels of flow may be present at the same time, e.g., data and message flow.

Stability: Even though graph visualization can either be done for a static representation, as for publication purposes, or for dynamic exploration, in both cases the graph changes dynamically in an iterative creation or exploration process. A requirement that is therefore nearly always stated by practitioners is the stability of the drawing after reapplication of the layout algorithm, following a small change in the data or the layout algorithm parameters. The author had many discussions with practitioners from several application areas, including biology, chemistry, telecommunication, criminalistics, and software engineering, and also observed discussions on this topic in online resources. Judging from the feedback given, lack of stability is one of the most important reasons hindering the application of automatic graph layout in application software. Users tend to apply small manual changes to an automatically generated layout, and they demand from the system to preserve these changes, even when it is not directly clear how to formally specify such a change with respect to the new layout.

Alignment: A very simple, yet effective way to organize objects in a diagram is alignment. Alignment of objects is either a mandatory requirement as in the visualization of business process models, a strongly recommended practice as in software diagrams, or just a helpful tool to arrange objects based on user preference. Alignments are used to increase the readability, to highlight specific processes and flows, or to show membership to the same substructure or organizational unit, as, e.g., activity regions (swim lanes) in UML activity diagrams.

Navigation in large data sets: In application areas with a need to understand and communicate large amounts of data, grouping structures can also be used for navigation purposes. An expand-and-collapse mechanism based on the inherent structure of the data may both help to gain a deeper understanding of the data and to keep the orientation.

Fast layouts for large graphs: Whereas static visualizations for presentation purposes are no time-critical tasks, in many application areas interactive data exploration systems need quick or even guaranteed response times. Efficient layout computation methods for real time navigation in huge data-sets are therefore critical for the success of graph drawing in such applications.

Highlighting: In addition to the standard task in graph drawing, the computation of a layout to allow the understanding of the overall graph structure, many application areas require that important players can be detected. Importance here may be based on structural properties, e.g., large degree or centrality, but can also be derived purely by semantical annotations. As the center of the drawing is a focus area, several application areas adopt the convention to place important vertices there. However, several types of diagrams from graphical notations have already fixed semantics for areas or dimensions of the drawing. In sequence diagrams that use the vertical dimension to represent time an important message might not be simply placed in the center. Besides single vertices, important substructures like biological motifs or inheritance hierarchies also need to be clearly visualized.

Vertex sizes and non-overlap: Vertices may stand for different types of entities and need to be represented accordingly. Images, labels, or data may be mapped onto them, therefore vertices often need to be drawn with non-uniform sizes; see Figures 3.6 and 3.21. As overlap is often unacceptable in these cases, automatic graph drawing solutions have to take into account shapes and sizes. These properties can also be exploited for layout improvement, for example in orthogonal drawings bends may be saved at large vertices, where enough space is available to route edges to adjacent vertices in parallel. Many current drawing methods, especially the most popular force-directed methods, do not support such requirements well.

Attributes like vertex size or color may either be prescribed, e.g., to permit to inscribe a text label of a vertex, or be used to model additional information, as, e.g., in Figures 3.1 and 3.5. Use of these attributes can also be the result of established visualization conventions, narrowing the space for free mapping of data onto visual attributes, and making specifically tailored software applications necessary.
Restricted embeddings: The order of the edges around a vertex might be restricted, e.g., to separate incoming and outgoing edges, or to model fixed sides or ports for edge attachment (e.g., in data flow diagrams). In addition, subgraphs may have to be separated, i.e., they are not allowed to be nested in each other (e.g., inheritance hierarchies).
Visualization of dynamic processes: Due to the inherent temporal component of processes in many applications, dynamic drawings are specifically important. Such applications include the visualization of biological processes, monitoring and analysis of telecommunication networks, and also the analysis of changes over time for social networks.
Labeling: Nice placement of vertex and edge labels, or respecting them already for layout computation, can improve the visualization quality significantly. This is especially imported for social and biological network representations, however, only quite simple heuristics are typically implemented, if any.
Important Notations and Languages: A number of drawing standards and notations from application areas are important that need to be supported by automatic graph drawing tools to be successful in the corresponding area. The Unified Modeling Language (UML) is in widespread use in software engineering and business process modeling. The Business Process Modeling Notation (BPMN) is the predominant notation for business process modeling. The Systems Biology Graphical Notation (SBGN) is a relatively new graphical notation standard for the modeling of biological processes [SBGN]. For the first time, the SBGN seems to be a widely accepted approach for the standardization of biological network visualizations, and despite the early status of standardization, the SBGN is already supported at least partially by several of the important visualization tools. The notations described are mainly used in software engineering, biology, and business process modeling. However, there is a widespread use of further established modeling or notation languages, like Petri nets, in many application areas. Even though notations help in standardizing the way in which diagrams are visualized, they still leave much freedom in how to arrange objects in a diagram. Several results indicate that the ability to create *and* to read diagrams differs largely between novices and experts [Petre, 1995, 2010]. Experts seem to make much better use of ‘secondary notation’ (i.e., information that is not part of the formal syntax) for layout and perceptual cues to improve reading and comprehension. Examples for secondary notation elements are clustering and adjacency of functionally similar or related objects, use of white space, labeling, etc. These elements are used to improve the layout quality by emphasizing important information. High-quality automatic layouts therefore allow to make the value of graphics use independent on the individual style and skill of the diagram creator. This however includes the challenge to avoid misleading use of visual cues that might suggest relations that are not present in the data, e.g., by incorrectly showing symmetry or grouping.

Respecting the semantics of the graph objects is an aspect that is wished by practitioners, but often only partially achieved by graph layout solutions. This is often due to the fact that simple standard layout algorithms are implemented by non-experts in graph drawing as simple extensions to existing applications over and over again. The situation is similar to the demand for GUI development tools, where several established toolkits exist (Tcl/Tk, Qt) and components are even part of programming languages (Java). Only in a few cases, users need to develop GUI components from scratch on their own. For graph layout however, the first attempt made is often to implement a simple spring embedder variant instead of using state-of-the-art implementations. If systems for use in application areas are implemented by graph drawing experts, they provide sophisticated graph drawing solutions, but lack support from a sufficient user community. They are often only maintained adequately for a short time, and do not provide the necessary interfaces, exchange formats, and interactions for use in the respective application areas. One could recommend to provide implementations for large open-source projects that are maintained and extended over longer periods. However, instead of providing implementations of state-of-the-art methods for a specific software, a much better solution would be to provide stable and easy-to-use interfaces to well-documented graph drawing libraries, such that practitioners might start to employ graph drawing software in their own projects. A difficulty arises in the fact that the specifications of network models, notations, and drawing requirements are often not formalized and therefore both practitioners and the graph drawing community are not able to easily define a useful mapping to drawing styles, methods, parameters, and constraints. Maybe this can only be remedied by adopting clearly defined and documented notations and exchange formats and by promoting them in application areas. A mapping of graphical features and requirements from notation standards like UML or SBGN must then be given such that practitioners are able to implement their own solutions, using graph drawing libraries as a black box.

Another important aspect for the acceptance of graph drawing software in practice is the evaluation of systems and solutions, e.g., by user studies. Such studies can be of great help to make sure that the solutions are well designed for practical purposes. Nonetheless, such evaluations are rarely done (or at least published), perhaps because it is difficult to design and conduct reasonable experiments.

4. AUTOMATIC GRAPH DRAWING

Graph Drawing is the art to produce a picture of a graph.

WWW.GRAPHDRAWING.DE

As we have already seen, graph drawing is an important technique when information has to be visualized for human users. The field of automatic graph drawing is concerned with the automatic, i.e., computer-based, generation of graph representations. It investigates suitable and efficient optimization models, the necessary mathematical methods and data structures, and the corresponding algorithmic aspects.

In this section we will give an overview of the current state of the art in automatic graphic drawing, including the evaluation of the quality of a drawing, optimization goals, and established drawing styles. Even though it is not the focus of our work, we will also shortly discuss existing general approaches and computational methods, as their characteristics are important in the context of both interactivity and the incorporation of constraints.

Due to the broad range of applications, and the variety of methods applied in automatic graph drawing, graph drawing research has impact on many different disciplines and at the same time is influenced by both problems and results stemming from application and research areas. The topics handled include the basic graph theoretical and algorithmic problems, algorithm engineering to ensure applicability of the concepts in practice by providing adequate data structures and implementations, and also adaptions with respect to different fields of application. In addition, human-computer interaction issues have to be considered when it comes to the development of appropriate representation concepts and their evaluation. Even though the handling of constraints for graph drawing and especially the aspects of interactive graph drawing in end-user software system are also related to fields outside computer science, as for example psychology, we will focus here on the algorithmic and computational aspects of automatic graph drawing. Therefore we will not elaborate on the cognitive or perceptional aspects, and only describe them briefly when needed, for example as a motivation for optimization goals and evaluation criteria. These aspects are treated in the field of human-computer interaction that is concerned with the design, evaluation and implementation of interactive computer systems for human use. For an overview, see, e.g., Shneiderman and Plaisant [2009], for a discussion of perception principles in the context of network visualization; see, e.g., Nesbitt and Friedrich [2002].

Layout computation can be thought of as some kind of optimization process that tries to achieve a layout that is as close as possible to the optimum with regard to predefined quality criteria. Many of the problems that have to be solved for that purpose are difficult in a mathematical sense, i.e., NP-complete, and some of the criteria are contradicting or at least difficult to solve in conjunction. On the one hand, appropriate (heuristic) mathematical and combinatorial methods have to be applied, while on the other hand, approaches that achieve a balanced optimization of the quality criteria are needed. In common graph drawing approaches, a wide variety of methods is used, including flow-based methods, (integer) linear programming, but also heuristics like local search methods.

There are two main approaches to compute a graph layout. The first one directly defines a cost function that is minimized in an iterative process and combines all optimization goals. This approach is taken for example in *force-directed algorithms*, which model the drawing problem as a system of interacting physical objects, and then apply an algorithm to approximately compute an equilibrium state of the system with low energy. The general assumption is that low energy states correspond to readable layouts. The second approach defines a priority order of optimization criteria and then optimizes them one by one, either exactly or heuristically, where the output of one optimization step is used as input for the next. This approach is taken for example for orthogonal drawings within the *topology-shape-metrics* paradigm, where first crossings are minimized and a planarized embedded representation is computed, then the bends are minimized and finally a minimization of edge lengths or area is performed. The complexity of realizing further optimization goals, e.g., user-defined constraints, depends on the optimization approach taken. It is conceptually simple to add constraints to the cost function of the first approach, but it may be difficult to specify a computation method that allows to efficiently compute satisfying results, e.g., due to convergence issues. On the other hand it is not always straightforward how to add additional requirements to the second approach. The alignment of vertices to the topology-shape-metrics approach, for example, has to be expressed as a goal for the first two phases, such that the resulting shape allows to compute edge lengths in the third phase that guarantee alignment.

In order to assess the usefulness of graph drawing methods, the quality of the resulting layouts need to be evaluated. Central questions therefore are: Can you model layout quality by an optimization function? And how can this function be algorithmically optimized? Some criteria that are easy to state may be difficult to cover with a single value or number, as for example the symmetry, and even if it is possible, as for the number of crossings, they may be either be difficult to compute or to optimize. We discuss the question of layout quality in the next section, and give a short overview on drawing styles and algorithmic approaches in Section 4.2.

4.1 Evaluating the Quality of a Drawing

As the goal of graph drawing is to present graphs as lucid as possible to allow to understand the structure of the underlying data, the question arises how to define a good drawing. In their classical paper on the visualization of Entity-Relationship (ER) diagrams, Batini et al. [1984] state

> It is a hopeless matter to define formally what is a pleasant ER diagram and what is not.

Moreover, Sugiyama and Misue [1991] already state that ‘readability depends upon the problems being studied and, more intrinsically, on the drawing’s audience’. In addition, graphs can vary in a variety of structural properties, and also semantic information might have to be considered for the quality evaluation. Obviously it is therefore not an easy task to determine suitable criteria that allow to conduct a useful evaluation. Fortunately, we do not need to give up, as there are still ways to distinguish good and bad layouts. Looking at Figures 4.1 and 4.5, each of which shows two different drawings of a graph, most observers will identify the drawings to the right as the clearly better ones. As can be seen in the following discussion, we can also identify formal criteria that can be measured to judge the quality of a drawing, i.e., how well it allows to transport the underlying information. In addition, at least some aesthetics seem to be not just based on subjective judgment, but also human perception follows some general rules based on the structure of the visual communication channel. There is a lot of research done in multiple disciplines, including graph drawing, human-computer interaction, and psychology, that is concerned with this question, and there is an agreement on several criteria that are considered to be important for the quality of a drawing. The most prominent of those criteria are the number of edge crossings, the number of edge bends, vertex and vertex-edge occlusion (sometimes also called *edge tunneling*), and also the angular resolution. Purchase et al. [1995] performed empirical experiments that validate the justification of criteria like bend number, crossing number, and symmetry for the readability and understanding of graph drawings.

Clearly, the importance of these criteria for the insight gained by the human observer again depends on the specific graph instance, the individual observer, the application area, the drawing style, and the task at hand. The value of a drawing in a sequence of drawings stemming from a time-series animation might be optimal to reflect the change and at the same time be sufficiently readable, but might not be acceptable in a static context. Similarly, in interactive systems the user might gain a mental map of the graph structure, and the conservation of this map during navigation or animation will be more important for orientation than the optimization of fixed aesthetic criteria. Therefore there probably is no global quality function that fits all uses.

The goals of revealing information and creating aesthetically pleasing layouts at the same time may lead to contradicting optimization criteria. As a simple example, representing community structures by grouping vertices and adapting distances to reflect group connectivity is opposed to the goal of uniform edge lengths. Kamada and Kawai [1989] state that the number of edge crossings is an important requirement in drawing graphs, but may not be a good quality criterion because it interferes with the goal of a uniform distribution of the vertices and edges in many cases. As an alternative, they proposed to focus on the *balance* of the drawing instead, i.e., to achieve a uniform distribution of the vertices and edges and to display the symmetric structures within the given graph also by symmetric pictures. In a recent publication, also Huang et al. [2010] suggest that is often better to make compromises between aesthetics, instead of trying to satisfy one or two of them to the fullest.

Coleman [1993] asks for a 'fair tradeoff' between the criteria such that the decline in the value of one criterion can be traded for a reasonable improvement of the remaining criteria with respect to the perceived quality of the drawings: 'Care needs to be taken to choose aesthetic functions of comparable strength so that none is washed out by the others'. A combination of such aesthetic functions then can be used to compare two different drawings produced by different algorithms (or algorithm settings) to decide which algorithm's results better conform to the chosen aesthetics, or to guide the optimization process of an algorithm, e.g., to select the next candidate in an iterative local search approach.

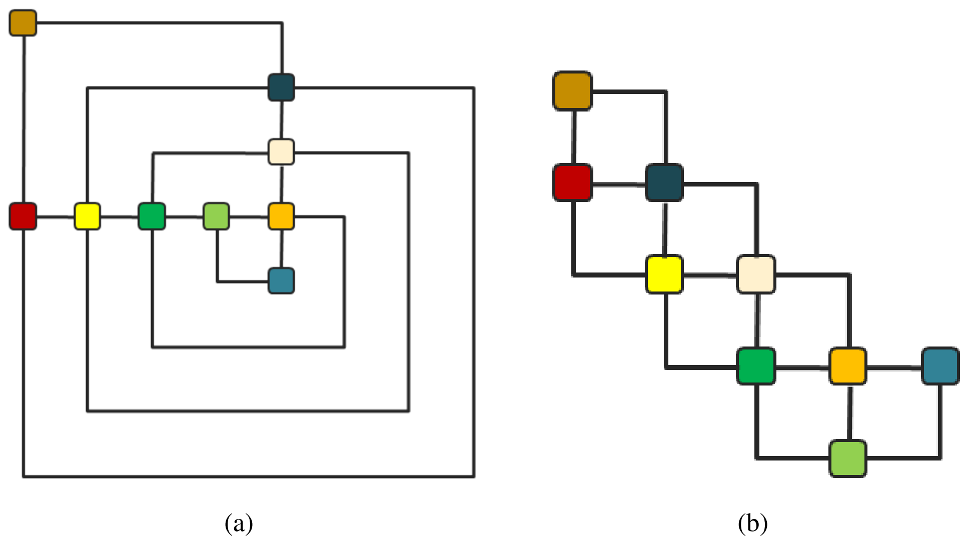


Figure 4.1: Embedding and external face selection can have an extreme effect on the clarity of a graph visualization. In this case, it is relatively easy to derive the criteria that lead to the improved representation in (b), minimum depth and maximum external face.

When we cannot combine aesthetics criteria in a single value, but have same kind of pareto frontier of maximal elements with respect to the set of criteria, the user may be allowed to guide the optimization and to select the 'best', i.e., most suitable result [do Nascimento and Eades, 2002, Biedl et al., 1998].

4.1.1 Layout Criteria

In their seminal paper on methods for layered drawing of hierarchical structures, Sugiyama et al. [1981] identify a couple of what they call 'readability elements'. They state that, even though readability may also substantially depend on the audience and on the problem studied, some common aspects of readability may be captured by such elements. In Sugiyama and Misue [1991] this concept is extended by classifying those elements into *drawing conventions* and *drawing rules*, where the former comprise the fundamental constraints that need to be strictly observed, whereas the latter contain the objectives that only need to be satisfied as much as possible. As they discuss hierarchical structures and compounds, the drawing conventions are mainly concerned with specific corresponding aspects. The most simple one demands that vertices have to be drawn as rectangles, other demand the hierarchical layout of vertices, the downward drawing of adjacency edges, and the representation of inclusion relations between vertices by the inclusion of the corresponding rectangles. As structural drawing rules Sugiyama and Misue define elements that address the minimization of edge crossings, 'straightness' of lines (minimization of bends), and closeness of adjacent nodes, which should support the traceability of paths. They also specify an order on the rules to express a priority, where closeness is most important, followed by edge crossings, edge-vertex crossings, and line straightness. As a minor geometric feature, they demand that ingoing and outgoing edges at a vertex are arranged in a balanced way. Also in the ground-breaking paper describing the topology-shape-metrics approach [Batini et al., 1986], a taxonomy of aesthetics is given, including as general criteria a small number of edge crossings, a small area, small total edge length, and a small number of bends. Coleman and Parker [1996] propose to model graph layout as a multi-objective optimization problem for their Aesthetic Graph Layout approach (AGLO), and give a list of 19 standard aesthetics, including reasonable vertex distances, edge length, angular resolution, and also rules for the visualization of (ordered) trees. For the combination of those aesthetics, they state that they 'rely on intuition and experimentation' and propose an additive model.

Dunne and Shneiderman [2009] propose the use of the term *readability metric* instead of *aesthetics* to underline the fact that they are mainly interested in how well a drawing communicates the underlying data, and not in how visually pleasing it is. They admit, however, that some of the most informative visualizations also are the most beautiful. Even though adherence to a set of readability metrics alone does not guarantee that the resulting drawing is also understandable, as the metrics may not be well suited for the task, or fail to support the intended impact, it should at least improve the information communication. As requirements for readable drawings, Dunne and Shneiderman propose what they call *NetViz Nirvana*:

(a) Every vertex is visible.

(b) For each vertex you can count its degree.

(c) For each edge you can follow it from source to destination.

(d) Clusters and outliers are identifiable.

Obviously, these requirements resemble some of the well-known aesthetic criteria, as vertex overlap minimization, maximization of the angular resolution, and grouping of strongly related sets of vertices. Based on the ideas of Purchase [2002], Dunne and Shneiderman propose readability metrics that are scaled to a continuous scale from $[0, 1]$ to allow the assignment of clearly defined readability requirements. In addition, they distinguish between global metrics and, similar to Herman et al. [2000], individual metrics for vertices and edges. Their main metrics are:

- Vertex Occlusion: Measures the uniquely distinguishable items in the drawing (an item is either a vertex or a connected mass of overlapping vertices), where a value of 1 indicates that every vertex is uniquely distinguishable from its neighbors, and a value of 0 that all vertices create one connected mass. An individual vertex metric is proportional to the ratio of the vertex' representation area obscured by other vertices.

- Edge Crossings: Measures a scaling of the number of crossings against an approximation of the upper bound of the number of possible crossings. A vertex metric only considers the incident edges, an edge metric a single edge.

- Edge Crossing Angle: Measures the average deviation of edge crossing angles from the ideal angle of 70 degrees [Huang et al., 2008].

- Edge Tunneling: Measures the number edge occlusions by vertices compared to an upper bound.

In addition, they briefly mention a couple of further metrics, including edge bends, angular resolution, and path continuity, which was backed as an important quality criterion by experimental studies [Huang et al., 2008, Ware et al., 2002] (continuity is the deviation of the angles between incoming and outgoing edges at the vertices on a path).
Summarizing the discussion above, we can derive three basic principles for readability in graph drawing:

- Vertices connected by an edge should be drawn near to each other to show the relation between them.
- Vertex separation: Vertices should not be drawn too close to each other, in particular they should not overlap. Putting vertices on a grid, or at least maintaining a minimum separation distance, avoids misinterpretations that come from a visual clustering of vertices, perceived as groups of related vertices. As this may conflict with another aesthetic, the grouping of vertices that are indeed related, Bennett et al. [2007] suggest that the distance between vertices in a cluster should be equal, and the number of different distance levels should be minimized.
- The number of edge crossings should be minimized.

A couple of criteria are either derived from that or added to further improve the readability:

- Minimize the number of bends.
- Minimize the total and the maximum edge length.
- Minimize the area.
- Achieve good angular resolution: The maximization of the smallest angle between incident edges at each vertex helps to visually separate the edges.
- Achieve uniform edge length.
- Distribute vertices evenly.
- Reflect symmetry.

Clearly even if each criterion is reasonable, they may conflict with each other and an improvement in one of them does not automatically translate to an improvement in the overall drawing quality. Minimizing the area for example can be done by drawing vertices close together, making the diagram harder to read.

Although several criteria seem to be good quality indicators in general, this is not necessarily true for any application area or graph instance. The evaluation criteria stated above are somehow generic and do not assess how well the layout reflects the conventions from an application area. When we have semantical or structural information in addition to the pure graph structure, as for example a hierarchy or a clustering on the graph's vertices, this information has to be reflected in the drawing of the graph. An evaluation of the drawing then also has to take into account how well the drawing represents the additional information. When trying to cover the quality of a drawing with a combination of quality criteria, it is therefore important to again assess the usefulness of those criteria and the way of combination by empirical evaluation. Unfortunately, as such evaluations are relatively difficult to conduct, the impact of automatic graph drawing results in applications is only rarely assessed. Purchase et al. [2001] test the influence of several aesthetic criteria on the task performance for UML class diagrams. They consider the bend number, vertex distribution, edge length variation, direction of flow, orthogonality, edge lengths, and symmetry. A low and a high effect version of several diagrams was created for each aesthetic criterion, and the difference in task performance was measured. Besides bend number none of the criteria had a significant impact, although all of them are regarded as valuable and established aesthetic criteria for layout quality. Note that the pure task performance in such experiments might be different to the cognitive load associated with it, an aspect that was investigated by Huang et al. [2009]. Another experimental evaluation of layout aesthetics for UML diagrams Purchase et al. [2000] shows that edge crossing reduction is a very important aesthetic criterion, and that orthogonality is highly preferred. However, this is only the case as long as the number of bends is relatively small.

When quality criteria are integrated into an algorithmic approach, there often is still much freedom in the way they are handled. Such an approach therefore may be parameterized, e.g., to influence the weighting factors of different criteria within an objective function. This gives the user some influence to adapt the layout results to his personal preferences, or to tune the algorithm with regard to specific classes of graphs or instances. As a drawback, it leaves the user with the task of optimizing the settings within parameter space (which possibly might by huge). Especially for end-user applications, it might therefore be helpful to reduce the possible settings, either by only allowing a few discrete steps, or by providing some predefined settings that proved to be reasonable for large classes of graphs.

Put together, a nearly undisputed criterion is the minimization of edge crossings, which is not only stated very often, but also thoroughly validated by experiments. A further important criterion is the visibility of vertices, i.e. the avoidance of vertex overlap. Most other criteria seem to be more or less important for layout quality, depending on several factors like data, application, task, and user. However none of these criteria should be considered to have a value independent of the others, and even edge crossings might be acceptable if several other criteria are thereby improved considerably. Ware et al. [2002] even argue that crossing minimization may sometimes violate the perceptional law of good continuity, which is related to the perception of paths. Judging from their experiments, they propose that crossings should not be avoided if this introduces increases 'bendiness' of the drawing, and that crossing minimization should have higher priority for 'relevant' edges. However, crossing minimization is backed as the main quality metric for UML diagrams and sociograms in user studies [Huang et al., 2006, Purchase et al., 2000].

4.1.2 Drawing Styles

There is a large number of requirements on how a diagram should look like in different application areas, and also different algorithmic solutions lead to drawings with specific characteristics, therefore various drawing styles have emerged up till now. In applications and publications often method and style names are used interchangeably, even though there

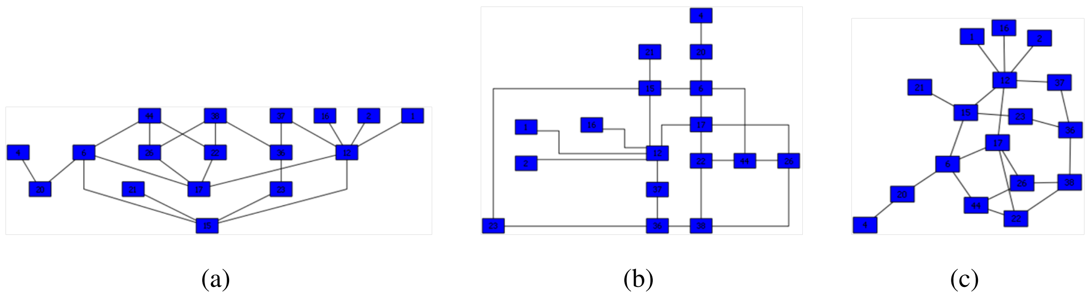


Figure 4.2: (a) Hierarchical, (b) orthogonal, and (c) straightline drawing styles.

is no one-to-one correspondence. We briefly describe the most common layout styles in the following. Each of these established layout styles has a couple of application areas where it is preferred as the layout standard over the others.

Hierarchical drawings: Hierarchical drawings model a (partial) ordering of the vertices given by the edge directions. The order of the vertex representations in the drawing with respect to the y-coordinate then has to correspond to this ordering. See Figure 4.2(a). Such a layout can be used to reflect a sequence of process steps or the flow of a commodity. Sugiyama et al., who pioneered the most popular hierarchical drawing approach, motivate the use of a hierarchical drawing style with two arguments: They claim that hierarchical drawings attain effective regularities which help humans to grasp structures, and that the drawing algorithm can be made simpler than others. Clearly, the second argument is a bit elastic, as non-hierarchical drawings can be achieved with relatively simple algorithms, and on the other hand the intrinsic problems for high-quality hierarchical drawing algorithms can be quite involved. For example, the minimization of crossings for layered hierarchical drawings (see below) is NP-hard.

Orthogonal Drawings: Orthogonal graph drawings restrict an edge representation to a sequence of horizontal and vertical segments. See Figure 4.2(b). This is often favorable for technical drawings like electronic circuit schematics. Each of the main optimization goals for orthogonal drawings, crossing minimization, bend minimization, and total edge length minimization, is NP-complete. Therefore, they are often optimized heuristically and also separately, as in the topology-shape-metrics approach (TSM) [Tamassia, 1987]. First, in a planarization step an embedded planar graph is constructed by replacing crossings with dummy vertices, making the subsequent bend minimization polynomial-time feasible. For 4-planar graphs, an orthogonal drawing guarantees good lower bounds for the aesthetic criterion of angular resolution. Due to the time requirements of the steps, the TSM approach is not suited for huge graphs. In addition, the resulting layouts for large graphs are sometimes rather poor with respect to the quality, especially regarding the efficient use of drawing space. Often, large unused areas are enclosed by paths, and edges are routed close together in channels, making it difficult to derive the overall structure of the graph.

Straight-line Drawings: In straight-line drawings, the layout is determined solely by the position of the vertices. See Figure 4.2(c). Wagner [1936] was the first to show that every planar graph has a straight-line embedding. Tutte [1963] showed that every triconnected

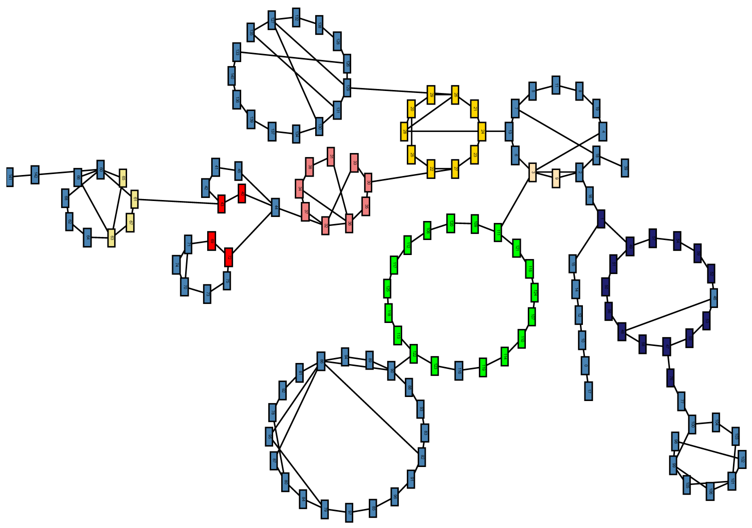


Figure 4.3: A drawing in circular style, with an assignment to the circles based on the biconnected components of the graph.

planar graph has a convex embedding. Planar straight-line drawings may need $\Omega(n^2)$ area and the minimum angle between adjacent edges may be very small in straight-line drawings. There are several techniques that can lead to quite different straight-line drawings. Examples include *convex representations*, where planar graphs are drawn such that faces are drawn as convex polygons. *Force-directed* methods in contrast try to achieve a good distribution of the vertices on the drawing region and nearly uniform edge lengths.

Circular: In circular drawings the vertices are placed on one or more circles, see Figure 4.3. Assignment to a circle may be used to emphasize group structures, for example to visualize network topologies or interaction networks. Care has be taken to minimize the crossings in such a drawing by assigning an appropriate vertex order. Also the exact positions of the vertices can be chosen such that the overall edge length is improved [Gansner and Koren, 2007], and also the appropriate placement of the circles can help to reduce clutter in the drawing.

Radial Layout: In a radial graph layout one vertex has the focus and is placed at the center of the layout. All other vertices are laid out on concentric circles around it depending on their distance from the focus vertex in a spanning tree of the graph. A first practical radial drawing approach was proposed in Eades [1992]. Typically, the angular width that is assigned to a vertex v is given by the ratio of the number of leaves in the subtree rooted at v to the total number of leaves in the tree. This is only a heuristic assignment because it does not respect the level on which the leaves reside. Inner level leaves may allocate more space than outer level leaves. Also, the size of the vertices is not used in the width assignment. Vertices within the subtrees should be placed within a convex wedge of the assigned angular width to avoid edge-circle crossings. The constant distance between adjacent radii may lead to problems if the numbers of objects on each level differ significantly. See Section 9.1 for an application using radial layouts.

Combinations: As some layout styles are clearly better than others for the visualization of specific types of graph structures, several approaches try to combine several styles within the same drawing. Bertault and Miller described a basic version of this approach within their Compound Graph drawing approach [Bertault and Miller, 1999]. There, for each internal node v of the compound inclusion tree, a suitable algorithm could be chosen by a so-called *mode* function to draw the subgraph induced by the children of v. TopoLayout [Archambault et al., 2007] uses a recursive decomposition of the graph and applies special drawing methods to detected subgraphs of a certain type. As such an approach directly exploits the corresponding characteristics of those subgraphs, these can be reflected in the drawing and allow to improve the display of symmetries and to highlight isomorphic subgraphs.

4.2 Approaches

In this section we overview the most common state-of-the-art approaches for graph drawing. With approaches we denote general drawing techniques which constitute the framework of an algorithmic graph drawing solution. Individual steps of these approaches may be realized by implementing different methods and algorithms.

Coleman and Parker [1996] propose a list of goals that a good graph layout algorithm in their opinion should strive for:

- Generality – be able to lay out different classes of graphs.
- Flexibility – be able to use different layout styles.
- Transparency – it should be clear what the algorithm sets out to achieve.
- Competence – be able to produce satisfying results for the class of graphs it was designed for.
- Speed – be sufficiently fast for the intended purpose.

Whereas the requests for speed and competence are undisputable, it is unclear why a generic layout method should be preferred over a high-quality method for a specific class of graphs. When an algorithm designed for a specific class of graphs achieves better results for such graphs than a general 'all purpose' approach, users might prefer the restricted algorithm, and drop the generality and flexibility requests. And even if we want to provide flexibility for the user, we could try to hide the corresponding methods underneath a uniform user interface. A basic question in this context is: Does the user need to understand the workings of the layout method? In a generic and flexible approach, the user has control over the behavior of the algorithm by adding or deleting aesthetics and changing the way they are combined and optimized. However, he should not be forced to experiment, and as many users will not be graph drawing experts, they will have difficulties in tuning a graph drawing method to give satisfying results.

As a consequence, dedicated layout computation approaches for specific drawing styles or graph classes have been developed that achieve high-quality drawings in reasonable time, including the classical approaches for hierarchical [Sugiyama et al., 1981] and orthogonal drawings [Tamassia et al., 1988], force-directed methods, and algorithms for trees [Buchheim et al., 2002]. However, recently several algorithmic solutions where presented that allow to incorporate drawing style constraints in a flexible yet efficient approach [Schreiber et al., 2009, Brandes and Pich, 2009b].

Often drawing approaches have preconditions that need to be satisfied by the input graph. A large number of drawing algorithms only accept planar graphs and would therefore exclude a huge class of graphs from being processed. The *planarization approach* [Batini et al., 1984] transfers non-planar graphs into planar representations. First, a planar subgraph with as many edges as possible is computed, then the remaining edges are inserted with the least number of crossings possible (the *edge insertion problem*). Each crossing that is created is replaced by a dummy vertex, such that a planarized representation of the input graph is obtained. The resulting planar graph can then be drawn using planar drawing algorithms, and the final drawing can be interpreted as a drawing of the original graph. The decision which edges cross clearly restricts the potential drawings, and the virtual vertices both may lead to an increased running time and a result far from the optimum. In addition, the minimization of crossings is an NP-hard problem [Garey and Johnson, 1983], and at least for larger instances, heuristics have to be applied. Nonetheless, both crossing minimization heuristics and methods based on planarization approaches have proven to be successful in practice [Gutwenger and Mutzel, 2004].

In general, *declarative* and *algorithmic* (sometimes called imperative or constructive) approaches can be distinguished. A declarative approach declares a relation, i.e., tells us ‘what is’, for example by declaration of specific geometric constraints. In contrast, an algorithmic approach states ‘how to’ do something, for example by giving the exact specification of steps that lead to a drawing with prescribed characteristics, modeled by a cost function to be optimized. Declarative approaches in Graph Drawing often resort to general solution techniques, and are therefore slower than algorithmic approaches, but allow to directly express aesthetic criteria. Algorithmic approaches are by far the dominating type of approaches in graph drawing, at least partially because of the lack of efficient constraint solvers available for a long time. Declarative approaches, however, seem to gain more and more attention in recent years. As both approaches lack certain qualities, there have been attempts to combine both in an integrated approach [Lin and Eades, 1995, Frick et al., 1996]. Typically either the flexibility and ease of declarative approaches is searched, applying, e.g., constraint-solving systems, or particularly efficient solutions are needed and therefore pure algorithmic approaches are used. Constraint-based methods seem to be suited well for easy addition of additional drawing constraints without large implementation and algorithmic difficulties, but the main concerns here are the efficiency of the resulting approach and a reasonable conflict-solving to guarantee a certain quality of the computed drawings.

4.2.1 Topology-Shape-Metrics

For orthogonal graph drawings, there is one dominating computation paradigm, the so-called *topology-shape-metrics (TSM)* approach. This approach separates the optimization of the main optimization criteria crossings, bends, and total edge length (or area), by using separate steps for each. A priority is imposed by first minimizing the number of crossings, then the number of bends, and finally the edge length. Each step leads to a representation of the

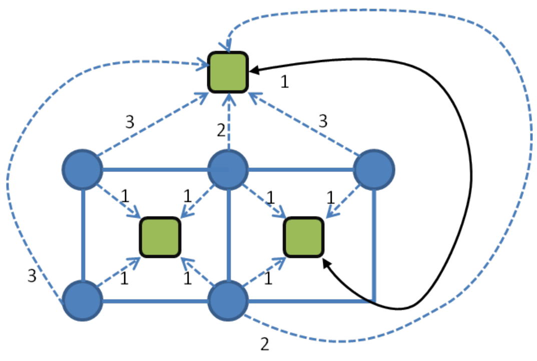


Figure 4.4: A flow network for the orthogonal shape computation step. Network nodes are depicted as green, rounded rectangles, black arrows depict network arcs for edge bends, blue, stippled arrows depict network arcs for angles between edges. Only arcs with positive flow are shown.

graph that is taken as input for the subsequent step. The topology step computes a planar embedding of G, for which the shape step computes an orthogonal representation, i.e., a specification of angles and bends for the edges of G. Finally, this representation is used to compute edge lengths in the metrics step, leading to an orthogonal drawing of G. Compared to other approaches for orthogonal drawings, this approach in general leads to much better results due to a much smaller number of edge crossings. The implementation of the TSM approach involves some non-trivial issues. In case the input graph is not planar, it first has to be planarized. While bend minimization is NP-complete in the general case, the fixed embedding of this representation allows bend minimization in polynomial time in the following shape step. This is typically achieved by applying flow-based methods as described in Tamassia [1987]. Each vertex is represented by a source node n_v with an outgoing flow of 4 units, where each unit represents an angle of $\pi/2$, and each face is represented by a sink node n_f. For each edge e incident to a vertex, the network contains an arc from n_v to n_f, the node representing the face to the right of e in the given embedding. Face nodes of faces adjacent to a common edge are connected by bidirectional arcs for each such edge. The sink nodes have a demand of $2 \cdot size(f) - 4$ for all inner faces, where $size(f)$ is the number of angles in the face, and of $2 \cdot size(f) + 4$ for the fixed outer face. Flow on arcs between faces models bends on the edges, and flow from vertex nodes to faces models angles of edges at the vertices. See Figure 4.4 for an example flow network. Flow based methods are well-established techniques for graph drawing and can also be used to efficiently compute compact drawings within the shape step.

Even though TSM is especially designed for the computation of orthogonal layouts, the resulting layout characteristics may differ significantly depending on the choice of the methods for the three steps, allowing different orthogonal drawing models to be realized. Models that allow large vertices in order to minimize the bend number can be realized with rather basic implementation effort, whereas models that guarantee prescribed or uniform vertex size, are much harder to realize [Fößmeier and Kaufmann, 1996, Eiglsperger et al., 2004]. The Kandinsky model [Fößmeier and Kaufmann, 1996], for example, needs a sophisticated treatment of the shape step in order to guarantee uniform vertex sizes in a consistent orthogonal drawing. It requires all but one outgoing edge per vertex side to bend. A restriction to the case where all but one edge bend to the same direction, and from at least $deg(v)$ sides edges emanate, the so-called *simple Kandinsky* model [Bertolazzi et al., 2000], simplifies the treatment and allows to use standard techniques for computation.

Respecting prescribed vertex sizes, on the other hand, requires a specific treatment in the metrics step, as the standard compaction algorithms suitable for the metrics computation typically only work on 4-degree graphs with 0-dimensional vertices. A considerable amount of work has been published concerned with the integration of constraints within the topology-shape-metrics approach. As edge bends and edge lengths are optimization goals for orthogonal drawings, most methods for the realization of the corresponding steps allow to integrate constraints on them quite naturally. Fixing of angles in the shape step or bounding the number of bends on an edge can be easily specified in the flow techniques applied for shape calculation. It is however difficult to satisfy additional constraints stemming from the semantics or established drawing conventions. The three steps are processed independently in a given order, but constraints may have to be considered in each of them, requiring a transfer of constraint information across the different step implementations.

4.2.2 Energy-Based Models and Local Search

Energy-based drawing methods constitute the most common drawing approach for undirected graphs. They are often preferred over alternative methods because they are reasonably fast, allow straight-forward extensions for a large number of drawing constraints, and are relatively easy to implement. The basic underlying idea of energy-based methods is that the graph is modeled as a system of objects that contribute to the overall energy of the system, and an energy-minimized state of the system corresponds to a nice drawing of the graph. In order to achieve such an optimum, an energy function is minimized. There are various models and realizations for this approach, and the flexibility in the definition of the energy model and objective function allows a wide range of both optimization methods and applications. An important advantage of energy-based methods, based on the iterative nature of the numerical methods to compute the layout, is that they allow an animation of the change from a given layout to a new one, as they permit to use a given drawing as input. A disadvantage is that the computation may end in a local energy minimum far from the optimum; see Figure 4.5(a). There is a wealth of publications concerning energy-based layout methods; see Di Battista et al. [1999b], Kaufmann and Wagner [2001] for an overview.

A special case are force-directed models, where the graph objects are modeled as physical objects that mutually exert forces on each other. A force equilibrium should represent an acceptable drawing with respect to the modeled aesthetic criteria, uniform edge length and uniform vertex distribution. In the most simple model, unconnected vertices repel each other, and vertices linked by edges attract each other as if they were connected by a physical spring, explaining the alternative term ‘spring embedder algorithm’. Force-directed methods are most suitable for undirected, sparse graphs such as free trees. Already Tutte [1963] used such an approach in one of the earliest graph drawing methods, based on barycentric representations that are obtained by solving a system of linear equations.

Force-directed drawing methods have a long history; the first mentioning of force-directed methods is commonly credited to the area of printed circuit board design, where a system of elastic leads and repulsive forces was described for the construction of circuit board drawings [Fisk and Isett, 1965]. The much better known classical spring embedder approach was introduced by Eades in 1984. It models vertices as steel rings and edges as springs, such that the mechanical forces exerted by the springs in a given layout define the energy of the system. A minimization of the overall system energy is associated with a layout that optimizes the Euclidean distances between the vertices with respect to the ideal distance. Several variations where proposed that tried to improve the practical running time [Frick et al., 1995, Fruchterman and Reingold, 1991, Tunkelang, 1994]. Another important concept for the practical improvement of energy-based methods is the approximation of the forces to speed up the force calculation. Typically, the repulsive forces are computed approximately whereas the attraction forces are computed exactly [Hachul and Jünger, 2004, Quigley and Eades, 2000]. This involves the application of space decomposition structures, as for example quad trees, for geometric clustering, as well as efficient approximation schemes, as for example the multipole method.

The energy-based approach by Kamada and Kawai [1989] uses the shortest graph-theoretic paths as ideal pairwise distance values and subsequently tries to obtain a drawing that minimizes the overall difference between ideal and current distances in an iterative process. Kamada and Kawai propose to use a two-dimensional Newton-Raphson method to solve the resulting system of non-linear equations in a process that moves one vertex at a time to achieve a local energy minimum. As the cost function involves the all-pairs shortest-path values, the complexity is at least $\mathcal{O}(V^2 logV + VE)$ or V^3 for weighted graphs, depending on the algorithm used, and $\mathcal{O}(V^2)$ for the unweighted variant. Later on, stress majorization was introduced as an alternative and improved solution method [Gansner et al., 2004].

4.2.3 Multilevel Paradigm

The multilevel paradigm is a generic approach to handle large datasets by reducing the complexity over a number of hierarchically ordered levels. It is well suited for graph algorithms and can be used to improve the layout computation especially for energy-based layout methods, regarding both the layout quality and computation time. The first use of the multilevel approach is commonly credited to Barnard and Simon [1994], where it was used to speedup the recursive spectral bisection (RSB) algorithm. In the context of graph partitioning, Karypis and Kumar showed that the quality of the multilevel approach can also be theoretically analyzed and verified [Karypis and Kumar, 1995].

The main idea is to construct a sequence of increasingly smaller graph representations ('coarsening levels') that still conserve the global structure of the input graph G approximately, and then compute a sequence of approximate solutions, starting with the smallest representation. Intermediate results can then be used on the subsequent level to speed up the computation and to guarantee a certain quality. The graph representations are typically created by series of graph contractions, where a set of vertices is collapsed to a single representative on the next, smaller level.

Walshaw [2003] and, independently, Harel and Koren [2002] and Gajer et al. [2000], introduced the multilevel paradigm in the field of graph drawing, after a closely related concept, the *multi-scale* method, was proposed by Hadany and Harel [2001]. In the FADE paradigm Quigley and Eades [2000], a geometric clustering (typically by recursive space decomposition) of the vertex locations is performed. This process, along with implied edge cre-

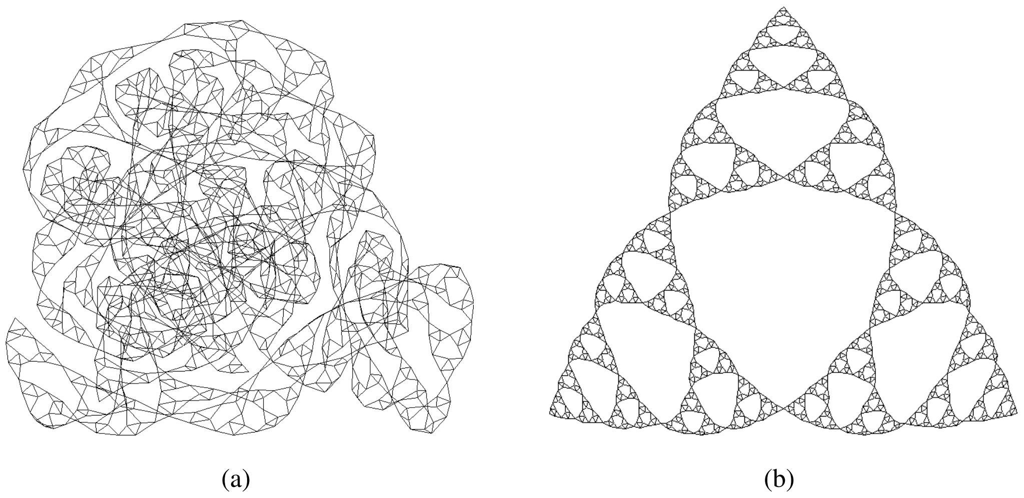


Figure 4.5: Bad layout for graph Sierpinski6 resulting from random placement and random coarsening in a multilevel approach (a) and alternative layout achieved by using solar placer and edge cover merger (b).

ation, constructs a hierarchical compound graph. The hierarchical compound graph, which includes the decomposition tree, is used to approximate the non-edge forces in the force directed graph drawing algorithm. After computing the forces, the underlying graph layout is updated, and this in turn requires the recomputation of the hierarchical compound graph, which was constructed on the previous vertex locations. This iterative process improves the quality of the graph drawing and the hierarchical compound graph, and can also be used for multilevel viewing of large graphs at different levels of abstraction.

Multilevel approaches can help to overcome local minima and slow convergence problems by improving the unfolding process due to a good coarsening and subsequent placement. As the input layout is not reused, its quality does not influence the result. Multilevel methods can cope even with very large graphs. However, it may still happen that the resulting multilevel layout represents a local minimum far from the optimum; see Figure 4.5.

Hachul and Jünger [2007] presented an experimental study of multilevel layout algorithms for large graphs, including energy-based and algebraic approaches. Frishman and Tal [2007] and Godiyal et al. [2009] investigated the use of the GPU for multilevel layout computation, and Archambault et al. [2007] described a multilevel algorithm that draws graphs based on the topological features they contain. Bartel et al. [2010] presented an experimental comparison of multilevel layout methods within a modular multilevel framework.

4.2.4 Layered Layout

Sugiyama et al. [1981] proposed a method to draw a graph in a hierarchical style by assigning a layer to each vertex based on a topological numbering. The vertices are then drawn on horizontal parallel lines representing the layers. The topological numbering guarantees that for each directed edge (u, v) the y-coordinate of u is bigger than the one of v. As this is only possible for acyclic digraphs, in case of cycles a preprocessing is needed that reverses a minimal number of edges to make the graph acyclic. This feedback arc set problem is known to be NP-hard, therefore heuristic are applied in most approaches. In a subsequent step, the *crossing minimization*, the vertices on each layer are ordered to minimize the edge crossings, which is also an NP-hard problem. In a final step geometric coordinates are assigned according to layer assignment and vertex order. Coordinates are computed to draw edges that span more than one layer close to vertical lines. These steps can be realized in several ways to optimize different drawing properties, as the height and width of the drawing, and vertical extent of the edges.

4.2.5 Projection Techniques

A projection p works as a function from a higher m-dimensional space A to a d-dimensional space B, where $d \in \{1, 2, 3\}$. The goal is to preserve the distances as well as possible, i.e., to minimize the difference between the dissimilarity δ of two data points $x, y \in A$ and the distance of their projection points $p(x), p(v) \in B$

$$\|\delta(x, y) - d(p(x), p(y))\| \quad \forall x, y \in A$$

where d is the Euclidean distance. Projection techniques can be used to visualize dissimilarities of objects by producing a graphical representation from given data annotations. Given for example distances between geographical locations by travel time tables, one could try to reconstruct the positions of the locations on the map up to rotation and translation.

Multidimensional Scaling

Multidimensional scaling (MDS) is a technique that allows to map high-dimensional data into a low-dimensional space by a non-linear projection, where the distances are aimed to correspond to the dissimilarity of the data points. In the realm of graph drawing it can be used for the visualization of datasets in two- or three-dimensional space. The distances of the data points in the drawing are calculated such that they approximate given dissimilarity values between pairs of data points as close as possible. The dissimilarity may be given as the geometric relationship between the n data points in the m-dimensional space A defined by the m-dimensional data vectors. For example, in classical MDS a Euclidean distance matrix can be given as dissimilarity measure and when graphs have to be visualized, the graph-theoretic distance, i.e., shortest path values, might be of interest. MDS can be helpful to reveal relationships between the input data points. There are a number of methods that can be used to carry out multidimensional scaling. An example is the so-called 'Sammon's Mapping' that uses an iterative nonlinear optimization (gradient) to minimize the error function defined by the distance differences. Elements of a cluster in the input space are mapped to positions close to each other. This minimization can be done by using the steepest descent procedure as proposed in the original publication. A much better optimization performance is achieved by using *stress majorization*. *Stress majorization* is a method that was already known for a long time for multidimensional scaling. In graph drawing, *stress* is a measure how far the distances in a given drawing differ from the ideal distances, such as graph-theoretic distances. While Kamada and Kawai already used such an energy function in their drawing approach [Kamada and Kawai, 1989], Gansner et al. [2004] formally introduced the stress majorization approach for graph drawing. The majorization approach minimizes the stress function by iteratively minimizing a simple majorizing function, i.e., a function that approximates and bounds the stress function from above. Due to some favorable properties as guaranteed monotonic decrease of stress, stress majorization shows a better rate of convergence compared to other optimization methods like gradient descent.

This method allows to incorporate additional constraints into the optimization process, and consequently, a number of extensions to the basic approach were presented; see Section 5.1.2.

The stress function for a position vector p is the sum of squared residuals

$$\sigma(p) = \sum_{i<j} w_{ij}(\|p_i - p_j\| - d_{ij})^2$$

where w_{ij} is a *normalization constant*. For use in graph drawing and multidimensional scaling, this constant is set to $d_{ij}^{-\alpha}$ with $\alpha = 2$ in the approach of Kamada and Kawai, and $\alpha = 0$ or $\alpha = 1$ in MDS approaches. It can however also be used as a weight related to the input dissimilarity when set to d_{ij}.

A problem that arises here is that several classes of constraints are more difficult to model within this approach as compared to the classical energy-based methods, as only the distances are used for optimization. Brandes and Pich try to overcome this problem for radial constraints, which can be expressed by the distance to a center, by using an iterative process that uses a convex combination of standard distance and radial distance. Increasing the influence of the radial distance in a linear fashion during the process, the vertices are projected on concentric circles in the final stages. Brandes and Pich [2009a] presented a study on distance-based graph drawing, comparing multidimensional scaling methods with algebraic and multilevel approaches. Their experiments show that for the representation of graph-theoretic distances by Euclidean distances, stress minimizing approaches perform better than force-directed placement, but achieve poor results for certain classes, such like small-world graphs.

As non-linear methods like MDS derive their results in an iterative optimization process, the process can be animated and a stop criterion can be used to abort the optimization when the solution quality seems to be satisfactory.

4.3 Graph Drawing Tools and Libraries

There is a wide variety of software tools and libraries dedicated for graph visualization. They differ in the range of layout methods provided, the intended audience and the application areas they focus on. At one end of the user spectrum there are end users that are interested purely in knowledge discovery or presentation regarding the underlying data. The other end consists of experts that also would like to learn about the algorithms used and to play around with the features and parameters of the methods used. Libraries of the latter category contain state-of-the-art layout methods and often allow to specify additional drawing constraints, amongst others GDToolkit, Graphviz, OGDF, Tulip, and Pajek. We refer the reader to an excellent online resource on existing graph drawing tools and libraries that allows to select an appropriate software based on the required features: The Graph Visualization Software

References [GVSR] is a collection of existing graph layout and visualization software, its web interface allows to comfortably browse and compare libraries, viewers, and editors.

We will shortly describe OGDF in Section 5.2.2, as some of the solution in this work are implemented within OGDF.

Part II

CONSTRAINTS

5. OVERVIEW AND CLASSIFICATION

In constraints lies the key: 'good' secondary notation involves disciplined and appropriate application of constraints to the available freedoms of presentation.

MARIAN PETRE

In order to improve the readability of a graph drawing, or to adjust it to personal preferences or drawing conventions, constraints that restrict the solution space to admissible drawings can be defined that need to be observed by an automatic layout algorithm. Aspects that have to be considered for constraint handling include the specification of constraints, their visual representation in user interfaces, the complexity of the underlying computational problems, and their algorithmic treatment. A huge number of publications in several scientific areas study graph drawing constraints with regard to their theoretical and algorithmic treatment, their handling in software libraries and end-user applications, and their influence on the human perception and the readability of the drawing.

This chapter gives an extensive overview and a classification of drawing constraints and deals with their role and handling in automatic graph drawing. In Chapters 6 and 7, theoretical results regarding two important constraints, cluster and embedding constraints, are given, and a description of practical approaches for the visualization of biological and chemical data using several drawing constraints follows in Chapter 9.

5.1 Constraints in Graph Drawing

Typical user requests and application-inherent constraints for drawings in real life applications cannot be adequately modeled solely by graph structures, and thus also cannot be directly taken into account by graph drawing methods without a special treatment.

Graph drawing first was mainly concerned with how to produce *nice* or *aesthetically* pleasing drawings of graphs. Graph drawing methods then were used as the basis for automatic diagram layout computation, and different drawing styles were developed to cover the most basic aesthetical requirements from different application areas. Requirements in diagram layout computation include however not only purely aesthetical criteria, but also user preferences, semantic requirements, and rules for the layout arising from the conventions in the respective application area. When for example flow is modeled by a directed and acyclic graph, a drawing that reflects those properties by drawing all edges oriented in the same direction is preferable. In addition, restrictions may simply represent technical necessities, as for example the visual resolution that can be handled by the display device, resulting in a lower bound on the angle between adjacent edges (*angular resolution*) or on the distance between vertices and non-incident edges. *Constraints* specify or restrict the drawing of a graph to improve readability and comparability of drawings, to adhere to an established drawing convention, or to keep the user's orientation by preserving the mental map in an interactive drawing environment. In this most simplistic definition, a constraint restricts the layout properties of one or more graph objects. We could define a constraint of type *alignment* to restrict a subset of the vertices of a graph to lie on a specified line, or restrict each edge of a graph to be drawn as a sequence of horizontal and vertical segments. However, the term constraint is often used just to specify *additional* restrictions with respect to a certain drawing style, or even within a computational approach for a drawing style. For practical approaches we will therefore always have some kind of restriction in the sense of constraints either added within a specific drawing approach or to a method. Tamassia et al. [1988] consider constraints in this sense as additional input to the drawing problem, which represent semantic requirements associated with the specific graph and application domain. This however implies a definition of 'constraint' that is dependent on a drawing approach and the application context, which is not adequate for a general discussion of constraints. General requirements stemming from drawing styles are constraints even if they are not handled individually within a specific drawing approach. Therefore it seems to be justified to see every restriction of drawing properties as a constraint. When it comes to practical solution approaches, however, we often have some assumption on a basic drawing style besides specific instance-dependent constraints. It is therefore reasonable to speak of 'adding constraint x to drawing approach y' instead of talking about a combination of constraints x and y. Thus we will also categorize existing constraint satisfaction methods depending on either established drawing styles or approaches. With such a model, we have constraints, which we can classify according to the type of restriction that they specify, and results and methods that treat such constraint in a specific context, e.g., a given drawing style or computational approach. We will therefore stick to the definition of a constraint as a restriction of the drawing for the general discussion, but nonetheless for the discussion of practical issues only consider constraints within a context of a given application or drawing approach.

5.1.1 Aspects of Constraint Handling and Constraint Characteristics

We use the term constraint to denote restrictions on the set of admissible layouts, whereas aesthetics are connected to the quality of the layout as described in Section 4.1. Clearly, constraints and aesthetic criteria can be integrated and optimized in a computational approach in a similar way, as long as they can be defined and specified in a formal sense. An example for such a formal specification is the minimization of the number of crossings, compared to the informal requirement that a drawing should not be cluttered, which is also an aesthetical requirement, but cannot directly be transferred into an optimization goal for layout computation. Similar to the situation for aesthetics, often a combination of several constraints has to respected, and appropriate priorities have to be chosen, as some of the constraints may also be conflicting. As there may be different approaches to the understanding of data by visualization, and also because personal preferences can greatly differ, it it difficult to design a constrained drawing approach that fits the demands of all users.

With our general definition of constraints, a drawing convention from a graph drawing viewpoint is just a combination of a number of constraints. From a practical point of view, a drawing convention might also include additional rules for the graphical representation of graph objects, as for example a specific style for edges of a certain type etc. In this work, the term constraints does not denote drawing conventions that determine the graphical representation of the graph objects with respect to attributes like color, line style etc., as they are not significant for the layout of the graph. Note, however, that some graphical properties may have an impact on the admissible layouts. Flow in technical drawings or biological networks, e.g., might be represented using the thickness of the corresponding edge, and in order to provide the space needed we have to specify a corresponding constraint, as most drawing algorithms only consider the edge representations as hairlines.

In discussions concerned with the way graph drawings should look like, several terms are used to describe characteristics of drawing constraints, with sometimes overlapping meanings. These terms are either used to classify the goals associated with constraints, to specify the motivation behind their use, or to describe which properties they influence. We will first give examples of these characterizations that show the different facets of constraints and their handling.

A basic distinction that can be made, especially for interactive drawing systems, is to distinguish between system- and user-defined constraints. Even though this distinction does not directly affect the computational treatment, it may be of importance for the representation and handling of constraints. In order to allow a user to adapt a drawing to his own preferences, there must be a way to specify and modify constraints, optionally also with a graphical representation that helps the user to keep track of the specified constraints. User-defined constraints are also needed to address the problem that it is often impossible to formally specify in advance which characteristics define a suitable drawing for several applications. Especially in a data exploration scenario, where new hypotheses are generated on the fly and have to be reflected by the layout, an accordingly tailored drawing method cannot be derived in advance. Clearly, for a theoretical treatment it is of no interest if the constraint was derived automatically, pre-specified, or given by the user.

A second aspect concerns the goal of the constraint: *Aesthetic constraints* are constraints that are applied only to improve the readability of a drawing and may be dropped in exchange for other constraints if those help to achieve a better drawing quality. Also concerned with the quality of the information transfer are *semantic* constraints, but these are required due to the application-specific role of the associated objects and cannot be dropped without affecting the quality of the drawing. Examples for such a constraint are that generalization edges in UML class diagrams should not cross each other, and an assignment of specific vertex sides to ingoing and outgoing flow edges, where an attachment of an output edge at an input side then clearly violates the semantic requirements. From an algorithmic viewpoint, the semantic information can be analyzed in a preprocessing step, such that the semantic nature of the constraint is not relevant for the layout computation, but they are intuitive and helpful also for the user's constraint management. In most cases, a drawing constraint defined by the user is meant to be satisfied permanently, not just at the time of definition. On the other hand, a constraint may only be reasonable in a specific state, e.g., for a fixed layout, and therefore does not need to be persistently satisfied. This is often the case for purely aesthetic constraints, whereas semantics based constraints need to be respected permanently.

The extent of the affected part of the graph is a further criterion for distinction of constraints. A *global constraint* may restrict properties of the whole drawing, as, e.g., a given aspect ratio, and a *local constraint* just restricts graph objects, e.g., by fixing a vertex position. The latter case is also an example for *absolute* constraints, whereas *relative* constraints specify a requirement concerning the drawing of a part of the graph only in relation to the rest of the graph G_R (or a part of it). Cluster constraints for example specify that the drawing of a cluster lies within a region r that does not intersect with the region of another cluster, i.e., the drawing of G_R is outside r. Badros [2000] gives as a corresponding definition 'A constraint is a relation that we would like to maintain'. Note that in the case of a local constraint, we can still ask for the constraint to hold, e.g., for each graph object, which may lead to conflicts even though each single constraint may be feasible alone. We cannot, for example, require each edge of a triangle to be bend-free in an orthogonal drawing without a conflict.

An additional issue is the question if a constraint is valid for all layouts or depending on the input instance [Poranen et al., 2003]. An example for the former type is the placement of the starting object in a sequence diagram to the left of the drawing, for the latter a placement of a selected vertex subset close to the center. Note that this global constraint is also a semantic constraint, as it implies that a special role is assigned to graph objects.

A constraint may have an impact on different characteristics of the drawing, as for example the geometry or the topology, and these characteristics can also be used to classify constraints. Closely related to this classification is the question what has to be done during an algorithmic approach to achieve these characteristics, we can for example specify constraints on the embedding to achieve a certain topology of the resulting drawing.

The final characteristic of a constraint we mention here is concerned with the question whether it has to be satisfied exactly or can be weakened to allow the satisfaction of other constraints or aesthetical requirements. Constraints in the former class are called *strict constraints*, constraints in the latter class are called *soft constraints*. Obviously, this is not a fixed property of the constraint itself, but dependent on the application context and the user's preferences. *Priority* values for the constraints can be used to solve conflicts, and to decide during computation to which extent soft constraints have to be satisfied.

We might ask how local constraints like 'do not let this edge cross any other edge' and global optimization goals like 'crossing minimization' are related. Both consider edge crossings, and if we treat the former constraint as a soft constraint, we could also see it as a crossing minimization on a single edge. We can always avoid crossings on a single edge, but if we demand this for all edges, the constraints are in conflict for non-planar graphs. In this context, crossing minimization then gives us a global prioritization criterion for a set of crossing constraints, one for each edge. We allow a crossing on an edge if it helps us to achieve the minimum number of overall crossings. We could therefore also understand crossing minimization as a weak global constraint that we try to respect the best we can.

Note that due to the overlap of the characteristics given above no categorization can include all those aspects with a strict separation and be at the same time unambiguous. A categorization of constraints therefore will always be somewhat subjective and represent only an abstraction of the real constraint topology.

5.1.2 History and Preliminary Work

A large amount of work concerning constraints has been published in the scientific literature, including surveys [Tamassia, 1998], general frameworks [He and Marriott, 1998, Kamada and Kawai, 1991], and a huge number of approaches to integrate constraints into existing drawing methods. However, the term constraint is not always used consistently. This section gives an overview on the history, the main concepts, and most relevant results.

The use of constraints in graphical systems was already introduced in the early 1960s with the drawing editor SketchPad, an interactive drawing system with a graphical user interface (partially operated using a light-pen). It allowed to specify geometric constraints and can be seen as a predecessor of today's CAD systems [Sutherland, 1964]. In the seminal paper by Batini et al. [1986] that describes a layout approach for data flow diagrams, the authors distinguish between aesthetics and constraints. Constraints are defined as additional semantic information provided to a drawing algorithm. Sugiyama and Misue [1991] define what they call drawing rules, drawing conventions, and readability elements that are modeled as constraints and objectives for their multi-stage multi-objective problem for layered graph drawing (see Section 4.1.1).

While these approaches can be seen as applications of constraints for the definition of, or within, a drawing style, Böhringer and Paulisch [1990] were among the first to explicitly address the problem of user-specified constraints and give a general mechanism to extend layout methods with constraints. Their approach is restricted to linear equations and the dimensions have to be treated independently of each other. Within their approach, the set of constraints is maintained by a *constraint manager* that also keeps the constraint set consistent by dropping inconsistent constraints. The approach allows absolute positioning within a fixed coordinate system, relative positioning of vertices in relation to other vertices ('A is on top of B'), and cluster constraints to group vertices together. They provide an exemplary description of the integration into Sugiyama's layout algorithm, whose drawing rules are quite close to the structural constraints allowed in their approach. Even though they promise a general mechanism, this description shows that already for such a related algorithm considerable work has to be done to adapt the system, and no general interface or approach is given to do this for other layout algorithms. Instead of integrating constraints, this approach is closer to a constraint solving system that allows to simulate drawing styles which can be expressed by the linear constraints.

Kamada and Kawai [1991] proposed a model for a constraint-based layout framework already in 1991, and developed a number of prototype systems (TRIP, IMAGE) based on this model. In constraint-based systems, a declarative approach is taken where relations between variables are stated in the form of constraints, specifying the properties of a solution to be found. The constraint set is then solved by a value assignment such that the solution is consistent with the maximum number of constraints. He and Marriott [1998] describe the *constrained graph layout model*, a general framework that allows to implement constraint-respecting layout modules for different layout styles.

Di Battista et al. [1999b] distinguish drawing conventions, aesthetics, and constraints, where drawing conventions are defined as rules that a drawing must satisfy to be admissible, for example orthogonal edge routing. In their terminology, aesthetics specify graphics properties that only have to be applied as much as possible. Constraints are restricted in Di

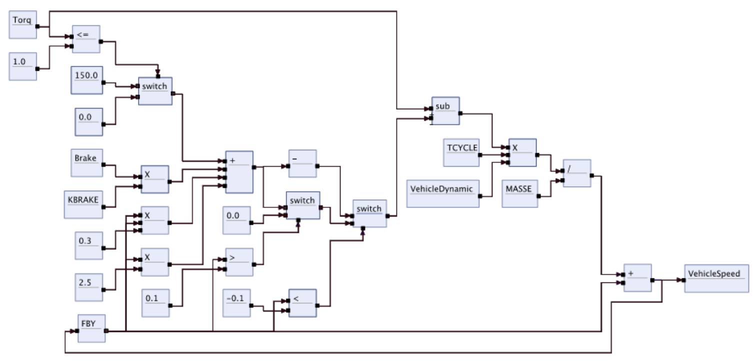


Figure 5.1: Data flow diagram drawn with the KIELER platform [Spönemann et al., 2010]. Upward constraints are realized by adapting the hierarchical drawing approach of Sugiyama et al. [1981] to depict the flow direction. The adaption takes special care of the port constraints which have to be respected to conform to the drawing conventions for data flow diagrams.

Battista et al. [1999b] to subgraphs or subdrawings, as for example highlighting of an important path in a drawing of a process diagram by aligning the edges. If the number of bends in the drawing should be minimized, this would be an aesthetical requirement, whereas the condition that a specific edge should not be bent would be a constraint.

Many practical applications have constraints on the drawings that restrict the set of admissible planar embeddings. We call *topological constraints* drawing constraints that impose a restriction on the embedding of the graph by definition, as opposed to constraints that influence the embedding only indirectly due to the restriction of other layout parameters. Several quite different approaches have been taken concerning topological constraints.

The most simple case to restrict the order of edges around a vertex is bimodality: For an embedding of a directed graph, the circular list of edges around each vertex has to be partitioned into incoming and outgoing edges. A planar directed graph is *bimodally planar* if it has a bimodal embedding that is planar. This property can be tested in linear time [Bertolazzi et al., 2002]. Buchheim et al. [2006] adapt the planarization approach and the exact crossing minimization to consider bimodality. A typical example how topological constraints can be motivated from practice are drawings of database schemata [Di Battista et al., 2004, 2002]. Table attributes are arranged from top to bottom within a rectangular vertex representing a table, and links connecting attributes may attach either at the right or the left side of the corresponding graphical table representation, but not at the top or the bottom. Another area of application are data flow diagrams, which represent data flow models that arise for example in software and hardware development tools. Each edge in such a diagram represents a data path with specified source and target ports. Ports restrict the embedding but may also influence and complicate further aspects of layout generation, as for example bend minimization and edge routing for orthogonal drawings. Spönemann et al. [2010] proposed an approach based on the Sugiyama framework to support port constraints in hierarchical drawings of data flow diagrams. See Figure 5.1 for a drawing computed by their implementation, which gives a clear improvement upon previously known results.

We present a representation for a large class of topological constraints together with an algorithm for constrained planarity testing and edge insertion in Section 6.

The integer linear programming approach of Eiglsperger et al. [2000] considers side and port constraints in the shape computation phase of orthogonal graph drawing. Although such an ILP-based solution is really flexible, it is only practicable for a very small number of constraints, as it is also extremely slow. Bertault [2000] presents an approach to preserve the topology of a drawing within a force-directed method. By restricting the vertex movement to wedge-shaped zones around a vertex the edge-crossing properties of the initial drawing are preserved. Dornheim [2002] studies the problem of computing embeddings satisfying a special class of topological constraints, These constraints consist of a cycle together with two sets of edges that have to be embedded inside or outside the cycle, respectively. See Figure 5.8 for an example. Buchheim et al. [2006] describe how to adapt the planarization approach for directed graphs when incoming and outgoing edges have to appear consecutively around each vertex. Eiglsperger et al. [2003] and Gutwenger et al. [2001] consider *mixed-upward* drawings where only a subset of the edges is directed and builds a forest. Directed edges should not cross each other and the embedding has to assure that directed trees are not intermingled. Gutwenger et al. [2008] describe a class of embedding constraints that allow to restrict the order of incident edges around a vertex based on a constraint hierarchy, see Section 6. Harrigan and Healy [2007] investigate the planarity testing problem for leveled graphs, i.e., where vertices have prescribed y coordinates, under these embedding constraints. In the last years, the stress majorization methods has gained increased attraction for the integration of constraints. The basic method has been extended to allow hierarchical drawings [Dwyer and Koren, 2005], radial drawings [Brandes and Pich, 2009b], vertex overlap removal [Gansner and Hu, 2010], and also topology preservation [Dwyer et al., 2009]. Dwyer et al. [2009] use topology preservation to achieve stability, meaning that for a horizontal and vertical sweepline scan the order of the graph objects does not change. Recently, Angelini et al. [2010a] presented a linear-time algorithm to test if a given embedding of a subgraph of G can be extended to an embedding of the entire graph G. Preservation of the embedding is easy to maintain for planarization-based methods, nonetheless the quality of the resulting drawing after a few insertion operations may quickly deteriorate. As energy-based methods typically do not care about an embedding, but mainly consider the vertex positions, it is more difficult to add a corresponding functionality.

Eiglsperger et al. [2004] show how to incorporate constraints that stem from drawing conventions for UML class diagrams into orthogonal drawing algorithms. These constraints affect each of the three steps of the TSM approach. First, the planarization has to take into account the inheritance hierarchies. Crossings between generalization edges are not allowed, and an embedding needs to be chosen that avoids that inheritance hierarchies are intermingled or nested. Second, the shape computation is restricted as bends on the generalizations are not allowed and hierarchies should be oriented in the same directions. Even though these restriction seem to be easily satisfiable, the implementation is quite complex. In addition, the constraints may lead to worst-case drawings if not additional measures are taken to avoid them: The inheritance hierarchies are rigid structures and as the main priority is on crossing reduction, edges might be routed around the whole drawing to avoid crossings, leading to very long edges with a large number of bends. Siebenhaller [2009] tackles a combination of constraints, including a specific type of port constraints, within the TSM approach.

Simultaneous embedding problems cope with the layout of several graphs on a shared vertex set. Variants include simultaneous embedding with fixed edges (where common edges between graphs share the same Jordan curve in the simultaneous drawing), simultaneous geometric embedding (where shared vertices need to be drawn on the same position), and colored simultaneous embedding (where the restriction is relaxed to allow a vertex to be mapped to a subset of points). Obviously, this is not a problem that is only concerned with the topology of the layout, but also with the geometry. There is a broad range of important applications, including the visualization of biological networks to compare them across species or cell states, or to highlight temporal changes. Due to the increasing need for such drawings in practice, research on this topic has been enforced in recent years. Unfortunately, several related problems have been found to be computationally hard. Estrella-Balderrama et al. [2007] proved that determining whether two planar graphs admit a geometric simultaneous embedding is an NP-hard problem. Chimani et al. [2008b] extend the crossing number problem to simultaneous graphs and show NP-completeness for this problem. There are however also positive results: Erten and Kobourov [2005] present a linear time algorithm to embed any pair of planar graphs on the grid with at most three bends per edge, and at most one bend ore edge in the case of trees. Braß et al. [2007] show the existence of simultaneous embeddings for pairs of paths, cycles, and caterpillars, and give counter-examples for general planar graphs, outer-planar graphs, and triples of paths. Fowler et al. [2008] show how to decide in linear time whether a pair of biconnected outer-planar graphs has a simultaneous embedding with fixed edges. Brandes et al. [2007] study colored simultaneous embeddings and show positive and negative results for several basic cases. Frati et al. [2009] investigate constrained simultaneous embeddings, where the combinatorial embedding of the graphs is fixed. Estrella-Balderrama et al. [2010] describe GraphSET, a tool especially designed for simultaneous graph drawing that allows to both study theoretical problems as well as to produce simultaneous drawings.

In order to allow control over the orientation of the edges in the drawing, Sugiyama and Misue [1995] propose an extension to Eades' spring model that uses magnetic springs and magnetic fields. One of the goals is to draw edges of different type with a different pre-specified orientation. By combining three standard magnetic fields – polar, parallel, and concentric – they model a rotational force on the edges and for example manage to draw graphs with nearly orthogonal edge routings. In contrast to other force-directed methods, the magnetic force approach is therefore also able to cope with edge directions, where, e.g., an upward parallel magnetic force field can lead to drawings where edges tend to point upward.

Often, prescribed vertex sizes and edge lengths need to be considered. When all distances are given exactly, distance-based graph drawing can be applied [Brandes and Pich, 2009a]. Force-directed algorithms often allow to set a zero energy length of the edge, such that the algorithm tries to achieve this length as close as possible, but often have problems avoiding vertex overlaps. However, a near-optimal reflection of pairwise distances in a graph-based map may not necessarily be the best way to transport the information, as empirical evidence suggests that there is a 'route effect' which leads to an underestimation of distances for connected vertices — there is some kind of spatial distortion in perception and memory [Tversky, 1992, Klippel et al., 2004]. Di Battista et al. [1999a] describe how to respect prescribed vertex sizes in the *Simple-Kandinsky* drawing style, Friedrich and Schreiber [2004] show

how to do this in the hierarchical drawing approach, and Hong and Mader [2008] give a corresponding approach for straight-line drawings.

Approaches for several drawing styles aim at the representation of hierarchical clustering structures: Wang and Miyamoto [1995] extend the force-directed layout approach to allow a clustering of vertices. They construct a *meta-graph* by collapsing clusters, *meta-vertices* that represent clusters then are assigned a rectangular region to accommodate the cluster subgraph. During the iterative force calculation, weights are shifted from forces between vertices of different clusters towards forces between met-vertices, such that the relative positions between vertices in the same cluster are more and more preserved. [Eades and Huang, 2000] use a virtual vertex for each cluster within a force-directed approach. Cluster vertices are connected to the corresponding virtual vertex to keep them close to the cluster center, and repelling forces between virtual vertices and vertices of other clusters are used to keep clusters separated. Genc and Dogrusöz [2003] and Dogrusöz et al. [2004, 2009] use a force-directed compound graph layout (i.e., with a multi-level nesting of subgraphs) for biological pathway visualization that takes into account localization information. They apply a gravitational force to restrict vertices to compartments in the drawing. Their approach takes into account vertex sizes and and arranges vertices with respect to their role within a reaction. See Figure 5.2 for an example layout. Eades and Feng [1997] propose a method to compute orthogonal grid drawings for clustered graphs, and Di Battista et al. [2001] show how to extend the planarization approach for orthogonal graph drawing to integrate cluster structures. Bourqui et al. [2007] are the first to attack the clustered drawing problem with a multilevel drawing approach. See Chapter 7 for further results and an in-depth discussion of cluster-related topics.

Already in 1994 Karp and Paley discussed the representation of metabolic pathways and proposed a drawing approach that uses different layout methods for different pathway topologies [Karp and Paley, 1994a,b]. Biologically relevant topologies such as linear, cyclic and branching pathways are identified and used as the backbone of the layout. Becker and Rojas [2001] proposed an approach that allows to draw pathways in a way that highlights two main properties of established pathway visualizations: Cycles are detected and drawn in a cyclic fashion, and main paths are drawn in an aligned fashion to capture the flow of reactions. See Figure 5.3(a). Also Bourqui et al. [2006] try to respect drawing conventions for metabolic pathways by highlighting cycle and cascade substructures. They perform a decomposition based on a clustering approach, which leads to what they call a 'quotient graph'. This structure is then drawn using the mixed model layout and the substructures are drawn using hierarchical and circular methods. Brandes et al. [2004] propose a method for visualizing a set of related metabolic pathways across organisms using two-and-a-half dimensional graph visualization with the aim to visually analyze the (dis)similarities. Two-dimensional graph visualizations of each pathway are stacked on top of each other in an ordering based on the Hamming distances of the underlying graphs. The layouts are determined by a global layout of the union of all pathway graphs using a variant of the Sugiyama algorithm. See Figure 5.3(b). A similar approach is also used to visualize the dynamics of social discourse networks [Brandes and Corman, 2003].

Nöllenburg and Wolff [2005] investigate the metro map layout problem, a problem that at first seems to be quite easy to solve as already a large part of the layout is fixed: Given a

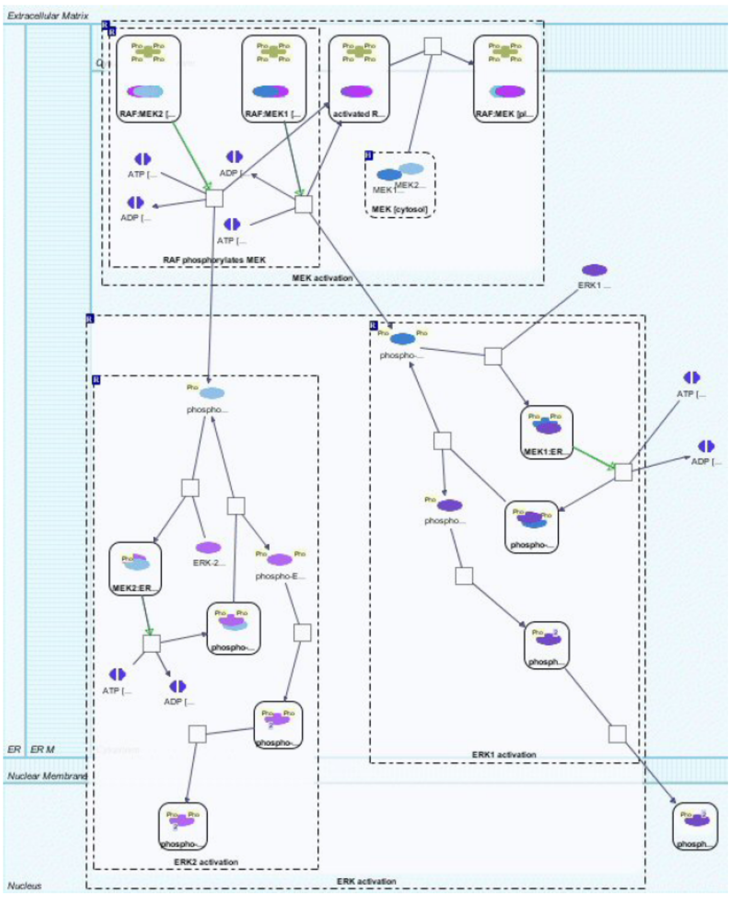
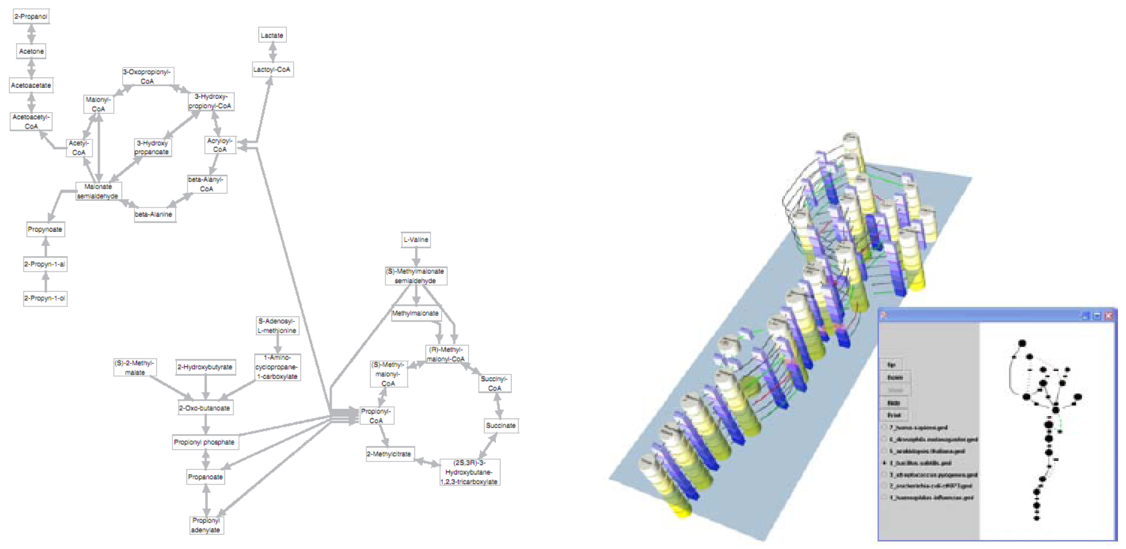


Figure 5.2: A compound graph layout obtained by the force-directed approach of Dogrusöz et al. [2009].

planar graph G of maximum degree 8, a fixed embedding of G, and a fixed location for each vertex, and a *line cover*, i.e., a set of paths and cycles of G that covers the edges of G (the metro lines), find a nice drawing of G. In order to achieve 'metro map' like drawings, they specify 'nice' using the following constraints:

- Respect the given topology ($\star$).
- All edges must be octilinear line segments ($\star$).
- Each edge has a minimum length ($\star$).
- Each edge has a minimum distance from each non-incident edge ($\star$).
- Each metro line should have few bends.
- The total edge length should be small.
- The relative position of a pair of vertices (u, v) should be similar to the one defined by the input positions. The relative position here is defined as the angle between the x-axis in positive direction and a line through the two layout positions.



(a) (b)

Figure 5.3: (a) A pathway drawn using the algorithm of Becker and Rojas. Circular drawing and alignment is used to highlight the main structures. (b) 2 1/2 D drawing of related pathways in WilmaScope using the algorithm of Brandes et al. (picture from Brandes et al. [2004]).

Constraints marked with a ($\star$) are strict constraints, that guarantee a planar, octilinear and embedding preserving drawing, whereas the remaining constraints are soft constraints and can be seen as aesthetic optimization goals. For example, path-continuity is preferred for metro maps, as metro lines should not bend too often. Nöllenburg and Wolff [2005, 2010] present a mixed-integer linear program (MILP) that finds a drawing which satisfies the strict constraints, if one exists and optimizes a weighted sum of costs corresponding to the soft constraints. The approach achieves better results than previous approaches with the drawback of significantly higher running time. See Figure 5.4 for an example layout. However, as metro map layouts are static and do not need to be computed very often, the running time of several minutes is still reasonable. We can not hope to find a general efficient solution for the problem, as Nöllenburg [2005] shows that the metro map problem is NP-complete.

When incorporating constraints into an existing layout approach, several problems might occur. First of all, the optimization process might be well suited to solve the original problem, but the adaption can lead to results that are far from acceptable layouts. That is, local minima of the objective function may represent ‘in between’ solutions that are not satisfactory in any of the aspects. In addition, adding constraints in such a way often clutters even elegant algorithmic solutions and the corresponding implementations. The alternative approach of constraint-based layout in the original realization had several drawbacks regarding flexibility and performance, but a series of results in the previous years showed significant improvements due to customized constraint solving techniques. We will describe advances for this approach and an alternative approach for constraint integration, the iterative compu-

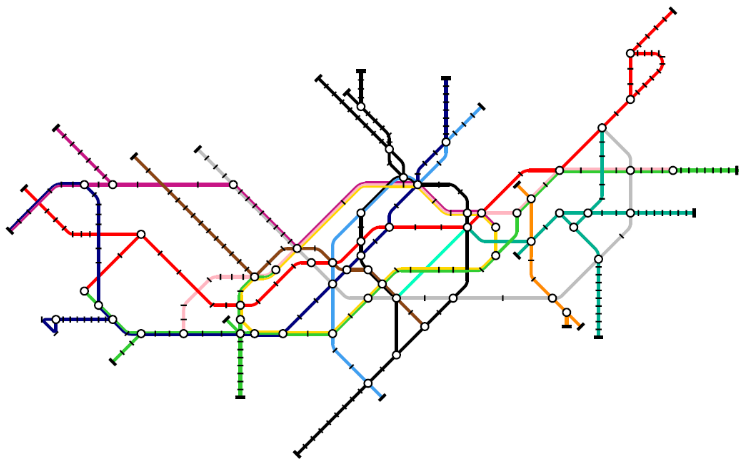


Figure 5.4: A metro map layout of the London underground network obtained by the mixed-integer programming approach of Nöllenburg and Wolff [2010].

tation of grid-based layouts, in more detail.

Constraint-Based Layout. Kamada and Kawai [1991] introduced a constraint-based object layout system called COOL that includes a linear constraint solver which is able to solve geometric constraints. These constraints are divided into rigid constraints, which have to be satisfied exactly, and pliable constraints, which can be satisfied approximately. COOL applies a least squares approximation for this purpose. Within the TRIP prototype system [Takahashi et al., 1998], constrained layout generation is done in two steps. First, the layout algorithm of Kamada and Kawai [1989] is used to generate an initial, unconstrained layout, which is translated into linear constraints for the graph objects' positions. These constraints then are solved together with the geometric constraints using COOL's constraint solver. He and Marriott [1998] introduce the *constrained graph layout model* aiming at the use in interactive applications. The two main drawbacks of the classical graph layout model that they try to overcome are: a) Lack of stability in a dynamic drawing scenario, b) Restrictions on the graph layout due to fixed layout aesthetics, hindering the integration of constraints which express the underlying semantics of the graph objects. The input for the graph layout module in constrained graph layout consists of the graph, a set of constraints over the x and y positions of the vertices, and a partial assignment of suggested values for the vertex coordinates. The graph layout module computes an assignment of vertex coordinates which is feasible, i.e., satisfies the constraints, gives a *good* layout, and assigns values which are as close as possible to the suggested values. Whereas the constraints model additional semantic information about the graph, the specification of suggested values allows the layout module to preserve the previous layout of the graph. Different graph layout modules may allow different classes of constraints and also embody different layout algorithms and aesthetic criteria. He and Marriott present several exemplary constrained layout modules, all of which allow general linear arithmetic constraints, with modifications of the cost function from Kamada and Kawai [1989] as objective function. They apply an active set method for optimization, and quadratic programming for their tree layout modules, and test their approach on several small graphs.

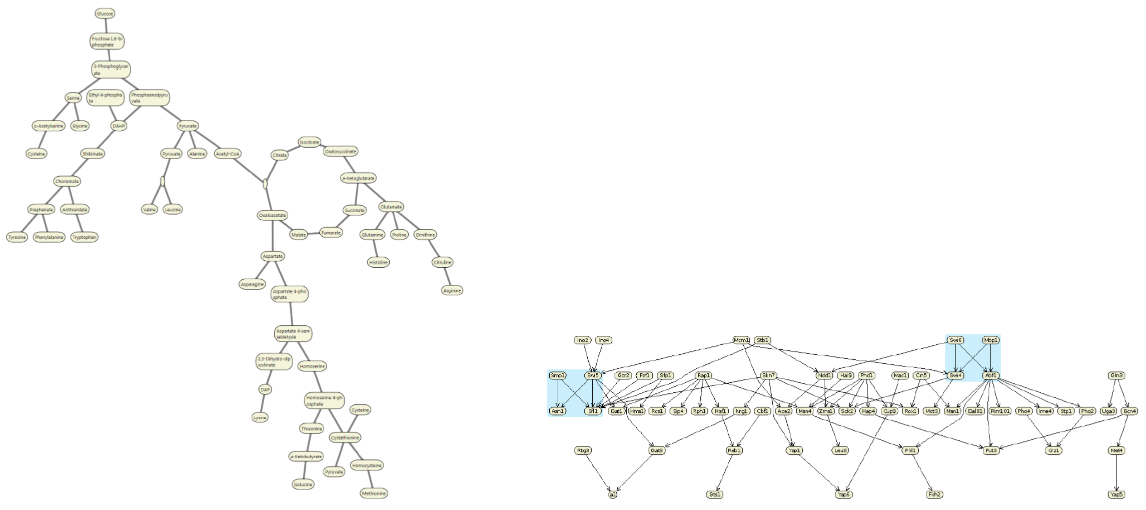
In Dwyer and Koren [2005] constraint programming techniques are combined with force-directed placement to draw directed graphs in a hierarchical fashion. The resulting DIG-COLA approach aims at representing hierarchies while preserving the aesthetic properties achieved by force-directed placement, including uniformity of edge lengths, better representation of cycles and symmetry, and balanced aspect ratio. The linear separation constraints imposed by the given hierarchy information are integrated using convex quadratic programming in a stress majorization approach, which can be solved efficiently using any standard solver. As an application they show how to apply their approach for directed multidimensional scaling. DIG-COLA is implemented for the Graphviz library. As a generic standard solver was used to solve the quadratic problem, the approach was relatively slow in practice.

In an attempt to speed up the solving for the special case of stress majorization with orthogonal ordering constraints, Dwyer et al. [2005] propose an iterative gradient-projection algorithm especially adapted for their problem formulation: In each step, a projection is performed that projects the result of the move onto the feasible region to satisfy the ordering constraints. As this projection can be performed in $O(mn + n \log n)$, with m the number of ordering levels and n the number of variables, this can be done fast in practice, as the running time in each iteration is dominated by the $O(n^2)$ complexity of the stress function.

Dwyer et al. [2006a] extend force-directed layout with separation constraints (IPsep-CoLa). A gradient projection method is used to solve a one-dimensional quadratic objective subject to the separation constraints for that dimension. In order to overcome convergence issues with this solution, and to achieve an speed-up, Dwyer and Marriott [2007] propose a scaling technique. Dwyer et al. [2009] show how to preserve the topology of a drawing during an improvement of path lengths, based on a gradient projection approach. Further constraints like vertex overlap, alignment and edge direction remain satisfied.

Dwyer [2009] describes a technique for constrained graph layout that improves on previous approaches in terms of performance and the flexibility of the supported constraints. The technique combines a force-directed layout approach with a simple constraint relaxation scheme, supporting separation constraints over Euclidean distance or distance projected on an arbitrary direction vector. Dwyer [2009] shows how to model three interesting constraint types with his approach, linear alignment with arbitrary orientation, circular constraints, and non-overlap for vertices or even clusters with arbitrary convex hulls. See Figure 5.5(a). The underlying force-directed layout approach uses a multipole-based force approximation, leading to an overall running time of $O(n \log n + m + c)$ per iteration, where c is the number of constraints. Dwyer and Robertson [2009] show how also non-linear constraints, e.g., circular drawing constraints, can be satisfied within this approach using a combination of several projection techniques.

Schreiber et al. [2009] describe a very elegant solution to biological network visualization problems. They propose a generic layout algorithm that aims at the visualization of different biological network types. The algorithm is based on the constrained graph layout approach from Dwyer et al. [2006a], where placement constraints are part of the optimization. In Schreiber et al. [2009] the constraints are further restricted to separation constraints



(a) (b)

Figure 5.5: (a) A metabolic pathway drawn by the constrained layout approach of Dwyer [2009]. Downward edge direction is achieved by oriented distance constraints, and the circular style of the cycle is achieved by a system of rigid distance constraints for the cycle's vertices. (b) Drawing of a gene regulatory network generated by the constrained layout approach of Schreiber et al. [2009]. The layout resembles a layered Sugiyama style drawing. Two *bi-fan* motifs are drawn similarly using equality separation constraints that fix a precomputed layout (highlighted by background color).

of the form $u + g \leq (=)v$, enforcing a gap g between u and v in either x or y direction. This restriction preserves computational efficiency while allowing enough flexibility for a wide variety of layout styles for different biological networks, see Figure 5.5(b). The separation constraints allow to handle vertex alignment and orthogonal ordering, non-overlap, enforcement of edge directions, and draw occurrences of network motifs in a similar way.

Even though the constraint-based approach seems to be flexible and efficient, it is not yet used frequently in practical implementations. Wybrow et al. [2008] see the lack of usability studies that investigate the value of constraint-based systems as a main obstacle for the widespread use.

Grid-based layout. In grid layout algorithms, vertices constituting the graph are mapped to grid points which satisfy their biological localization information, and are arranged to minimize a cost function defined over all possible mappings to penalize edge-edge crossings, vertex-edge crossings, and distance between vertices. Since finding the layout with the minimal cost is NP-hard, grid layout algorithms employ heuristics that move the vertices iteratively to a new position, applying the change that creates the largest cost reduction.

In order to overcome what they identify as the drawbacks related to force-directed methods, Li and Kurata [2005] propose a grid layout method for biochemical networks. They identify as major problems that

- little information about the network topological structure can be gained,

- cluster structures cannot be displayed clearly, and
- distances between vertices are not well proportioned, as dominant repulsive interactions make the distribution of vertices scattered, while the minimal distance between vertices is often too short compared with the drawing area size. This makes it difficult to obtain compact layouts.

Their algorithm, too, treats vertices as interacting particles, but places them on a square grid. A cost function is used based on the topological network structure, such that closely related vertices attract each other and less related vertices repel each other, and a local search method is employed to optimize the layout. Due to the grid placement, vertices cannot be placed too close to each other and each vertex has a finite number of potential positions on the grid points. A Manhattan distance metric is used to decrease the computational demand, and the cost function is the sum over all weighted pairwise distances. The weights derived from the number of short paths between a pair of vertices – the more short paths exist, the stronger is the attraction between these vertices. For the global optimization they use a simulated annealing approach, where first a neighboring layout is computed by perturbing the current layout, i.e., placing some vertices based on given perturbation rate on vacant grid positions. Afterwards, this layout is improved by applying a steepest descent local minimization that checks for each vertex and each vacant position if moving the vertex would improve the current objective value, and repositions the vertex in that case. This process is repeated until no improvement is possible with that method. The computation could be done in a straightforward fashion with a huge computational effort of $n(m - n)$ times the objective function evaluation ($O(n^2)$), where m is the number of grid points. The authors therefore suggest to keep a cost change matrix in memory, as the largest part will stay constant during a single vertex move, requiring $O(mn)$ additional space.

As a result, the grid layout algorithm by Li and Kurata [2005] focuses on the display of cluster structures, defined by strong interrelations between vertices, and on a uniform vertex distribution in a straightline drawing. The algorithm is implemented for the tool CADLIVE. Even though the layout already gives an acceptable overview of the overall structure with respect to the clusters, a closeup shows the problems with vertex-edge crossings and small angular resolution, see Figure 5.6.

The main limitations of their algorithm that they state themselves are edge crossings and slow computation time. The performance does not allow to layout mid-sizes graphs in reasonable time or to use the method in an interactive fashion. In addition, it is not well suited for metabolic pathways that need specific vertex sizes, edge labels, and highlighting of the main paths.

The main use of cluster structures in Li and Kurata [2005] is the grouping of closely related vertices, as these often form a functional module in the biochemical networks under consideration. Even though their main criticism of standard force-directed methods is based on the fact that non-adjacent vertices are not drawn closely together, they base cluster attraction on the adjacency relations. However, already in Dogrusöz et al. [2004] ideas like gravitational forces have been used successfully to model such clusters in force-directed methods, and the quadratic running time proved sufficient for interactive use with medium-sized graphs. Also vertex distances can be directly incorporated into such approaches, e.g.,

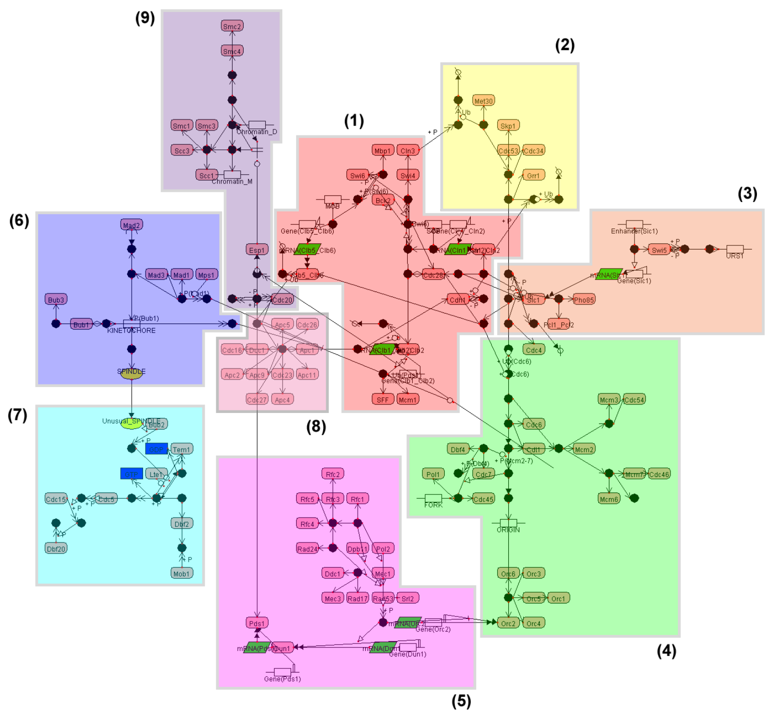


Figure 5.6: Regulatory cell cycle, picture taken from the CADlive homepage, and originally published in Li and Kurata [2005].

by using distance-based methods.

Kato et al. [2005] adapt the cost function of Li and Kurata [2005] to consider edge crossings, vertex-edge overlap, and cellular compartments, clearly with a trade-off in running time. However, their modeling of cellular compartments reduces the number of potential new positions for a vertex and therefore speeds up the computation in practice. Their algorithm is implemented in the tool Cell Illustrator.

Resorting to the argument of Li and Kurata, also Kojima et al. [2007] claim that force-directed algorithms may not be suitable for compact layouts of complex biological pathways. They add that torus-shaped regions like the plasma membrane cannot be handled well as regional constraints in these force-directed algorithms, and therefore also apply a grid-based layout for the visualization of biological pathways. In a series of algorithmic improvements, Kojima et al. [2007, 2008, 2010] presented a grid layout method that takes into account localization information and tries to reduce crossings. They define a cost function that includes the attraction and repulsion forces known from the spring embedder approach, extended by edge-edge and edge-vertex crossing costs. Their iterative approach moves a vertex in each step in a greedy manner to a new grid position, with calculation time $O(|E|^2 \cdot min(h, w) + hw)$, where h and w denote the grid dimensions. This is clearly much too slow for interactive visualizations. Furthermore they note that it is difficult to select parameters for their algorithm correctly and that it is not possible to capture the reaction flow, as, e.g., in signaling networks. Besides localization information, the grid layout approach can also model alignment constraints into the costs such that vertices with the same biological attributes are aligned

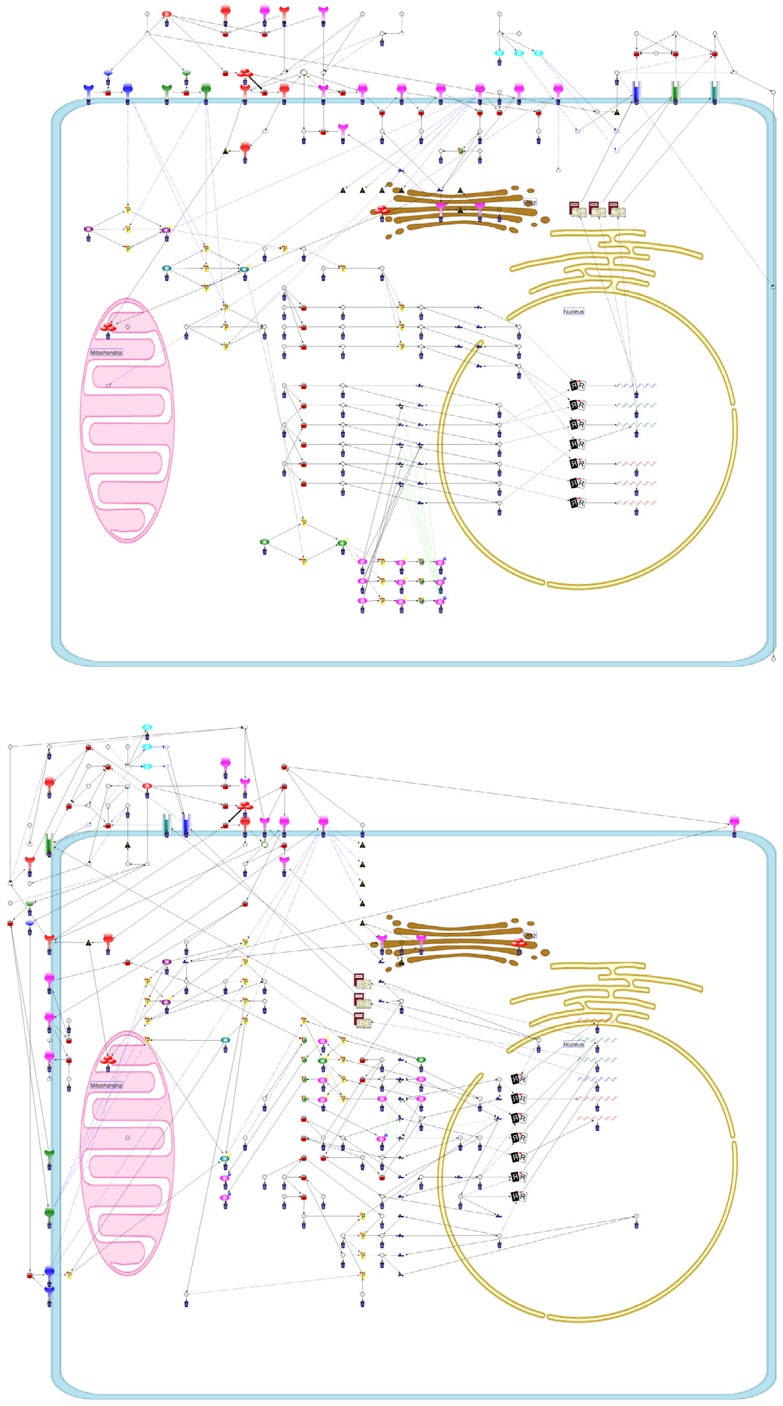


Figure 5.7: Visualization of an endothelial signal transduction pathway as given in Kojima et al. [2007], manually created (upper) and using the grid layout algorithm (lower). The manual layout facilitates the recognition of patterns and substructures better as it draws similar substructures in a similar way. It also exhibits less crossings, both between edges and of the mitochondria representation (pink). Alignment in two dimensions and grouping of vertices is used, also on a level below the compartment level, to show relations between similar or functionally related vertices.

vertically with a higher probability. Figure 5.7 shows a layout obtained with this approach compared to a manually created layout of the same graph.

5.1.3 Classification of Constraints

As there are many aspects associated with graph drawing constraints, there are different ways to classify constraints and the resulting categories do not always allow unambiguous assignment of specific constraints to a single category. Depending on which aspect is in the focus of research, some categories might be more important then others.

From the discussion above, we can distinguish two main groups of constraint characteristics: Characteristics that are dependent on the application, task, or user preference, and characteristics that are independent constraint properties. The first group includes strictness, priority, the classification into aesthetic or semantic constraints, the motivation for a constraint, and the general validity for all instances. Independent characteristics are

- *Type:* the drawing properties restricted by the constraint.
- *Extent:* the distinction of global and local constraints.
- *Quality:* the distinction of absolute and relative constraints.

These three characteristics answer the questions *what* is constrained, *where* it is constrained, and *how* it is constrained. Each one has consequences regarding the algorithmic processing of the constraint, its computational complexity, the resulting drawings, and may also influence the way the constraint can be represented in the user interface. We will classify the constraints based on these characteristics.

For the constraint type, we use as the three main classes geometry, topology, and routing. Clearly, more than one drawing property may be restricted at the same time by a constraint, e.g., the constraint may determine both the topology and the geometry. Constraints that only determine a single property of the drawing may also have side effects that will influence another property. For example, in order to satisfy a specified geometric ordering of the vertices in x-direction, a corresponding embedding might be needed to allow a planar drawing, even though not explicitly stated by the constraint.

We state the most relevant constraints in the following and categorize them according to the criteria above. The list of constraints is partitioned according to the type, and extent and quality are given in parentheses (L=local, G=global, A=absolute, R=relative).

Geometric constraints. Since a graph layout specifies the geometric position of graph objects, geometric constraints are a natural class of restrictions. These include for example the fixing of a vertex to a specific position, but also a geometric relation of graph objects to each other, like ‘A is below C’ or ‘A has distance r to B’. A classical area where geometric constraints are necessary to guarantee correctness of the final layout is VLSI Design, where for example minimum and maximum distances between objects are defined to avoid signal interference and timing issues. Note that in the CAD domain these constraints are sometimes called ‘numerical constraints’, whereas geometric constraints are restricted to relations like parallelism, concentricity, collinearity, tangency, or perpendicularity. The latter constraints can be used to derive a drawing that conforms to the requirements from a given sketch that is only topologically correct.

Geometric constraints include:

Position (L;A,R)*:* A constraint that restricts a vertex to stay in a given topologically connected region, including point, line, or circle. An absolute version assigns a vertex to a fixed region, whereas relative versions may require vertices to lie in the same, but not in a fixed, region. A special variant is the point set embedding, where a planar graph G together with n points in the plane is given and a planar drawing of G is required that maps each of the vertices of G to a distinct given point. The problem might be further restricted by giving the mapping of the vertices to the points as input, or by requiring a straight-line drawing, see, e.g., Giacomo et al. [2008], Binucci et al. [2010]. The position may also be required to be relative to the drawing area, e.g., to place a vertex in the center of the drawing.

Ordering (L;R): A constraint that specifies an ordering of the positions of vertices or vertex sets in a given direction, e.g., an orthogonal ordering in x-direction.

Partition (L;A)*:* The drawing area (or parts of the drawing area) is partitioned into regions of prescribed shape (e.g. rectangles, such as lanes in process diagrams) and relative position, and vertices can be assigned to one of the partition cells.

Distance (L;R): Restricts the distance between vertices or sets of vertices. We may specify exact, minimum, or maximum distances, and also require the same relative distances between pairs of vertices to distribute vertices evenly.

Alignment (L;A,R) Restricts a set of vertices to lie on a common line. It has two parameters, the slope s and the y-intercept i, and several variations: In case both s and i are fixed, this is just the positioning constraint from above. Free s or i values lead to a relative constraint. Often, only horizontal or vertical lines are required. We can also model parallel lines by requiring the same (unspecified) slope for these lines with a relative constraint on the corresponding vertex sets. A soft variant of this constraint allows to aim for a layout that preserves similar relative positions of vertex pairs, modeled by the angle between x-axis and a line through these vertices.

Vertex size (L;A): Size values for the vertices are prescribed and need to be preserved in the drawing. Even though the main goal here is non-overlap, which might simply be achieved by sufficient scaling, the vertex sizes should be taken into account during layout computation to avoid large drawing area with unused spaces.

Shapes (L;R): Requires a set of vertices to be drawn in a specific shape, e.g., a circle as in [Dwyer and Robertson, 2009]. Isomorphic parts can be drawn in the same way to show symmetries or allow to recognize similar parts, e.g., specific subnetworks with known functionality in biological networks. Note that this is not a positioning constraint as the circle is not given, the vertices only need to have the same, but arbitrary, distance to an unspecified common center position.

Proximity (L;R): Put (similar) objects close to another. In contrast to distance constraints, this requires that the remaining (dissimilar) objects are far apart, otherwise the distance may be misleading.

Area and aspect ratio (G;A): A constraint that confines the drawing to be contained in a page or screen format.

Topological constraints. Topological constraints restrict the topology of the drawing, e.g., by fixing the order of edges around a vertex. For planar graphs, the topology is given by a planar embedding of the graph. Main topological constraints are:

(Adjacent) edge order (L;A): Restricts the order of the edges around a vertex (clockwise or up to mirroring), e.g., due to assigned ports or attachment sides.

Partial embedding (G,L;A,R): Fixes the embedding of a subgraph, as, e.g., discussed in Angelini et al. [2010a]. The global variant fixes the topology, as, e.g., in Bertault [2000], Chimani et al. [2005], Dwyer et al. [2009]. Several isomorphic subgraphs might be assigned the same embedding.

Restrict crossings (L;A): Forbid crossings on an edge (or in subsets of edges), or bound their number.

Embedding depth (L;A): Place a vertex on the outer boundary or at any other embedding depth.

Cluster (L;R): Place vertices in common, disjoint regions. A constraint that restricts both the geometry as well as the topology of the drawing. In a drawing of a graph with a given clustering structure, the clusters should be drawn as simple regions, for instance bounded by a rectangle that only contains vertices of the respective cluster. In order to facilitate understanding of such a drawing, the number of edges crossing these regions or other edges should be minimized. An important combinatorial problem then is clustered planarity of a clustered graph, i.e., to decide if a clustered graph can be drawn without any edge-edge or edge-region crossings, see Section 7 for a discussion. Cluster constraints are integrated in a number of drawing approaches, approaches based on force-directed methods typically only satisfy the region confining aspect of cluster constraints. There are also attempts to allow a more flexible way to represent the clusters, as for example polygonal-shaped clusters [BioGRID].

Routing constraints. Routing constraints cover aspects of edge routing, as, e.g., restrictions regarding bends or the edge orientation.

Restrict bends (L;A): Forbid bends on an edge or bound their number.

Edge orientation (G,L;A,R): Sets of edges (or all edges) have to be oriented in a prescribed direction, i.e., drawn as a curve that is monotonically nondecreasing in the orientation direction. Note that the edge orientation then implies the ordering of start and end vertex in the orientation direction, but the converse is only true for straight-line drawings. *Upward drawings* restrict all edges to be drawn monotonically in the same direction.

Angle (L;A): Specify the angle between two consecutive edges incident to a vertex, or between two edge segments at a crossing. This constraint also has geometric character.

Attachment (L,A): Specify an attachment point for an edge at one of the end-vertices. This constraint also has geometric character and is also a part of port constraints.

5.1.4 Constraints for Requirements from Practical Applications

We come back here to the requirements from practical applications that were given in Section 3.2.5 and briefly describe a mapping of those requirements to the constraints described

Drawing Style	Proposed by	Constraint types		
		geometric	topological	routing
Orthogonal	[Di Battista et al., 1999a]	vertex size		
	[Eades and Feng, 1997]		cluster	
	[Eiglsperger et al., 2000]	upward, edge direction, side and port constraints		
	[Eiglsperger et al., 2004]		restrict crossings	restrict bends
	[Siebenhaller, 2009]		port constraints	
	[Wiese and Kaufmann, 1998]	vertex size		
Hierarchical	[Böhringer and Paulisch, 1990]	linear eq.		
	[Friedrich and Schreiber, 2004]	vertex size		
	[Spönemann et al., 2010]		adjacent edge order	attachment
	[Harrigan and Healy, 2007]		edge order	
Radial Straight-line	[Brandes and Pich, 2009b]	position, distance		
	[Li and Kurata, 2005], [Kojima et al., 2007, 2008, 2010], [Kato et al., 2005]		cluster	
	[Brandes and Pich, 2009b]	position (radial), distance		
	[Brandes and Schlieper, 2006]	distance		angle
	[Brandes et al., 2000b]			angle
	[Gansner and Hu, 2010]	distance		
	[Fruchterman and Reingold, 1991]	area		
	[Dwyer and Koren, 2005]	orient., vertex size		
	[Dwyer et al., 2005]	ordering, orient., vertex size		
	[Dwyer and Robertson, 2009]	align., orient., shape, vertex size		
	[Dwyer et al., 2006b]	align., distance, vertex size		
	[Dwyer et al., 2009]	vertex size	partial emb.	
	[Schreiber et al., 2009]	align., distance, ordering, shapes		
	[Hong and Mader, 2008]	vertex size		
	[Takahashi et al., 1998]	linear eq.		
	[Wang and Miyamoto, 1995]	cluster, linear eq., vertex size		
	[Dogrusöz et al., 2004]	cluster		
	[Bertault, 2000]		partial emb.	
General	[Dornheim, 2002], [Angelini et al., 2010a]		partial emb.	
	[Gutwenger et al., 2008], [Harrigan and Healy, 2007]	edge order		
	[Di Battista et al., 2001], [Feng et al., 1995a]		cluster	

Tab. 5.1: A selection of representative solutions concerning constraint integration for specific drawing styles. The row labeled with 'General' denotes planarity testing and embedding approaches. References in brown color denote approaches using the constraint-based approach.

in this section. This is not in each case a 1-to-1 mapping, as requirements from practice may involve the combination of multiple basic constraints.

Grouping of objects: In several application domains it is necessary to draw graphs so that vertices are organized into groups. Lanes, compartments, and packages for example confine objects to closed disjoint regions in the drawing. Grouping of vertices can be modeled using clustering, and visualized using clustered drawings. Even though it does not directly make any difference for graph drawing if the clusters were computed based on a similarity measure or the graph structure, or were given as predefined groups, it may influence the performance of some layout algorithms (e.g., force-directed methods), as they may assume or rely on strong connectivity within a cluster compared to the connectivity with the exterior. Methods for clustered orthogonal drawings, while perfectly placing clusters in disjoint regions, may fail to place the vertices of a cluster in close proximity, see Figure 7.1. Sometimes the regions or their relative position may be specified in advance, as for the lanes in BPMN pools. Then additional partitioning, (relative) position, and distance constraints are needed to achieve a satisfying layout. Clustered drawing methods are well studied and implemented in practical tools. They however do not always work well when additional constraints have to be integrated.

Reflection of hierarchies: Restricting the edge direction was one of the earliest constraints handled in automatic graph drawing, and for directed graph drawing, the approach of Sugiyama et al. [1981] is the most widely used and accepted method. Recent advances in constrained based layout show that this constraint can also easily be satisfied in these more flexible approaches.

Stability: Depending on the application and the type of diagram, quite different requirements for stability occur, and several approaches exist to express them with appropriate metrics. Typically no single constraint type is sufficient to cover the corresponding issues, we discuss this topic in more depth in Chapter 8.1.

Alignment: An alignment can simply be expressed by a geometric constraint that restricts the involved graph objects to the same x or y coordinate, or more general to a line with a fixed slope. Alignments can occur as absolute constraints, where the coordinate is fixed or the common line runs through a fixed point. In the relative version, the line may move as long as the slope is preserved. Although this constraint is a trivial task for constrained based methods, it needs a bit more work in energy-based methods, as long as the coordinate is not fixed. One way to realize an alignment for energy-based methods is to use an additional force that represents the projection onto a common line and forces the vertices to move in that direction. For layout methods that follow the topology-shape-metrics approach, an alignment is much more difficult to achieve as long as the vertices are not lying consecutively on a path.

Highlighting: For this requirement, quite different approaches are needed. Placing important players in specific areas may require position and embedding depth constraints. These constraints can help to place a vertex in the center of the drawing, or at the deepest level or outer boundary of the embedding. When substructures need to be highlighted, this can be done by shape, partial embedding, and alignment constraints. As these constraints do not necessarily avoid overlap or nesting between substructures, they can be further supported by clustering and crossing restrictions. Even then the result might not be as expected: For the

main path of a pathway for example, it is not enough to align the vertices and avoid crossings of the connecting path. We also need to assure that the edges are routed accordingly, that is we have to apply appropriate angle and bend restriction constraints.
Vertex sizes: This requirement directly matches the corresponding type of constraint.
Restricted embeddings: Restricted embeddings like bimodal embeddings or embeddings that need to respect port or side constraints can be modeled by topological constraints such as edge order and partial embedding.

5.2 Constraint Handling

Even though over-constrained, but consistent, systems may lead to a reduced number of solutions, the addition of constraints to layout computation will in general require additional computational effort for the solution of the layout problem. Depending on the types of constraints and the layout methods used, combining several constraints may render the optimization process for practical graph sizes infeasible. Nonetheless, as we have seen above, successful implementations of layout methods that respect combinations of constraints for specific drawing standards in application areas exist and are applicable for practical work.

When judging the potential of supporting a constraint within a drawing approach, both the effort to include it in the model and implementation as well as the computational effort have to be considered. Several graph drawing approaches quite naturally support constraints that are either concerned with parameters that are part of the optimization goals, or can easily be modeled into the established computational solution methods used. Thus, a limited constraint-satisfaction capability is achievable also within the algorithmic drawing approach. Examples include the restriction of bend and angle values and also vertex sizes in the topology-shape-metrics approach. Non-uniform vertex sizes can be modeled naturally with a non-overlap guarantee in the topology-shape-metrics approach due to the compaction phase which computes a grid drawing. Due to the bend and angle computation in the shape phase, corresponding constraints can be integrated quite easily. Differing vertex sizes and fixed angles may pose a problem for other approaches, for example energy-based methods, where often specific overlap removal routines are used in a post-processing step [Gansner and Hu, 2010]. Such an overlap effect is even exploited in the Graphael system [Erten et al., 2003] for the visualization of temporal changes, where corresponding vertices of different time slices are connected by an edge and the repulsion forces between these vertices are omitted, such that these vertices stay close to an initial position throughout the evolution of the graph. What makes the use of energy-based models so attractive in many different applications, is that they can easily be extended to include many common drawing constraints. Some classes of constraints can be added just by adding a force or by extending the objective function with a term. The major disadvantage is that the constraints may only be satisfied approximately, especially when several constraints have to be satisfied at the same time. Depending on the method used to compute the drawing, priorities may be modeled by defining cost factors in the objective function.

For some constraint-based approaches the use of generic constraint solvers was proposed, for others efficient solving techniques were presented, e.g., Dwyer and Robertson [2009]. As constraint solving is a research field on its own, a lot of general constraint solving techniques

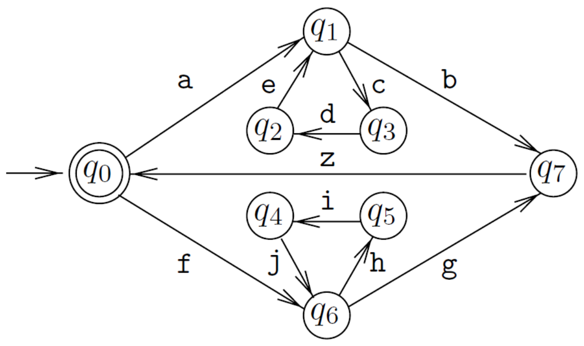


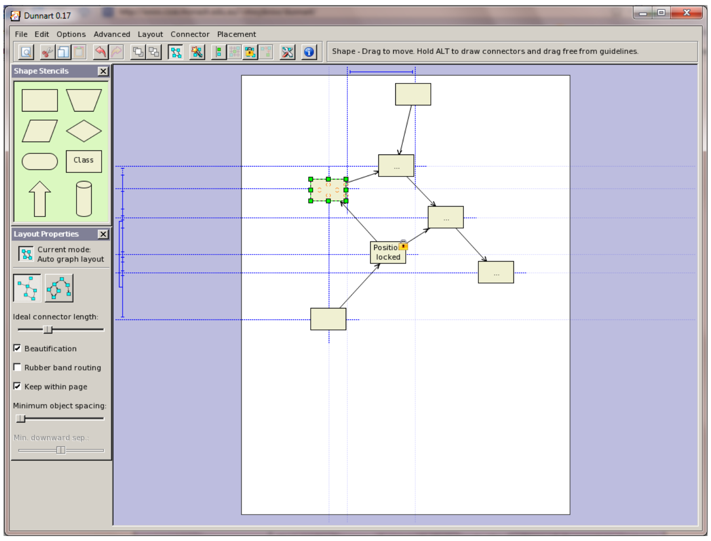
Figure 5.8: Transition diagram of a finite automaton as given in Dornheim [2002]. The automaton accepts the language $a(cde)^*bz + f(hij)^*gz)*$ and the embedding is chosen to reflect the structure of the expression. Such constraints on the embedding can be handled with the embedding constraints of Dornheim [2002] and Gutwenger et al. [2006].

and solvers exist, and we will not elaborate on this topic here.

Even when a single type of constraint can be respected efficiently, respecting combinations of several constraints may be difficult. Moreover, for some constraint types already negative results are known: Dornheim [2002] considers topological constraints given as triples (C_i, In_i, Out_i), where C_i is a cycle of the input graph and In_i and Out_i are disjoint sets of edges that have to be drawn inside and outside of C_i, respectively. He shows that the corresponding T-planarity problem, which asks if a graph has a planar embedding respecting these constraints, is NP-complete. Brandes and Pampel [2008] show that it is NP-hard to preserve the orthogonal ordering for the rectilinear and the equal-edge-length drawing models. This implies that bend-minimum orthogonal layout is hard under ordering constraints. The ordering preservation is of interest in practice, e.g., when an initial, partially manually generated drawing (sketch) should be simplified or beautified, or when dynamic graph changes require redrawing.

When the angular resolution has to be optimized, one could suppose that the efficient computation of orthogonal shapes, as, e.g., by using the prevalent flow network technique [Tamassia, 1987], can be extended to other resolution models, where angles between edges are multiples of $180/k$, e.g., octi-linearity for $k = 4$. However Bodlaender and Tel [2004] show that the realizability of orthogonal drawings, given a precomputed shape, is an exception. Instead, a k-gonal representation does not suffice for the existence of a corresponding planar layout, as it may require crossings. Brandes and Schlieper [2006] show that planar straightline realizability for trees with prescribed angles can be done in linear time, whereas in general the problem is NP-hard [Garg, 1998].

There are two additional important aspects when constraints need to be handled in graph drawing: the user interface and the internal implementation issues. Without an easy-to-use interface for the specification and representation of constraints, constraint handling will not help much in supporting the user in accomplishing his visualization goal. This includes both the issues with constraint handling over a graphical user interface, as in graph drawing editors, and also the mechanisms to provide a programmatic interface. In the former



Shape - Drag to move. Hold ALT to draw connectors and drag free from guidelines.

Current mode: Auto graph layout

Ideal connector length:

☑ Beautification

☐ Rubber band routing

☑ Keep within page

Minimum object spacing:

Min. downward sep.:

Figure 5.9: Screenhot of the Dunnart constraint-based diagram editor. The user gets a visual feedback on the defined constraints. In this case fixing of a vertex position is indicated by a lock, vertical alignment by a blue dashed line through the two involved vertices, and vertex separation by distance markers connected to each involved vertex by a blue dashed line.

case, a graphical representation and a constraint specification interface must be given, in the latter a system that supports definition of constraints, e.g., by parsing a constraint rule XML database, must be implemented. However, graphical representations for active constraints that help the user to keep track of the constraint status can also easily lead to a cluttered drawing. Such representations were implemented in the experimental constraint layout software GLIDE [Ryall, 2001] and also in the constraint-based network authoring tool Dunnart [Dwyer et al., 2008], see Figure 5.9. There are however no systematic studies so far that investigate the value of representation concepts, and on their influence on user performance (including the resulting layout quality). Wybrow et al. [2008] provide first results in this direction. They investigate user behavior with the Dunnart tool, and also give a discussion of issues regarding both the constraint representation and the feedback during direct manipulation by the user. An interesting question in the context of multiple possible solutions is which one should be chosen. Biedl et al. [1998] redirect this question to the user and propose several layout solutions in parallel.

Overall, there needs to be a thorough concept to handle constraints both on an implementation and an algorithmic level. In graph drawing libraries for example, appropriate interfaces and process definitions are needed to allow a consistent and easily understandable modeling of constraints and the respective transformation into code. In the following, we will first give an overview on current graph drawing libraries that facilitate the specifications constraints and then describe a concept for constraint handling for OGDF.

5.2.1 Layout Libraries and Tools

Handling constraints has been part of the graph drawing research for a long time, and many tools and systems have been built. Several tools also include interactive capabilities to allow the user to define and manage the drawing constraints. Although there has already been an impressive amount of work for the integration of constraint both in theory and practice, most of the developed approaches led to prototypical implementations that are no longer available or maintained. We shortly describe important concepts of older systems and summarize the current state-of-the-art.

Already in 1993, Dengler et al. proposed a system with a constraint-driven layout. The constraints are inferred automatically to satisfy good layout design rules, and their algorithm tries to satisfy as many of them as possible.

Kosak et al. [1994] proposed a straight-line drawing approach based on so-called *visual organizatorial features* (VOFs). Kosak et al. distinguish between syntactic validity, perceptual organization, and aesthetic optimality as categories for layout consideration. As a means to describe a diagram's desired topology and perceptual organization they propose VOFs, a set of basic rules to arrange vertices with respect to several constraints:

- horizontal and vertical alignment
- axial and radial symmetries
- various shape motifs (T, hub)
- left-to-right or top-to-bottom sequential placement
- simple vertex proximity
- horizontal and vertical alignment: Draw a set of vertices at the same x or at the same y coordinate.

The layout task then was formulated as a constrained optimization problem, using a symbolic description of the desired topology and organization. Their rule-based approach leads to problems due to the local nature of the rules, inconsistencies, and interacting VOFs, therefore also an alternative genetic algorithm is given. Even though they claim that this approach works well and is easily extendable to further layout rules, it is much too slow for interactive layout generation. The concept reflected the assumption that human layout design could be captured using a set of sublayout pattern rules, and that perceptional organization should have priority over syntactic aesthetic considerations.

In case a tool allows the user to define his own constraints, the user needs some guidance during the diagram editing process. Active constraints and also defined, but violated, constraints need to be represented in a way that allows the user to control the overall view with respect to valid drawing results. Otherwise, the user would have to start an evaluation by an explicit request, and as a result the given drawing may change significantly in order to conform to given diagram style rules.

A layout approached based on VOFs was later realized in the GLIDE diagram editor [Ryall, 2001]. GLIDE's layout computation is based on the spring-embedder model, and did not apply constraint solving techniques, but introduced additional springs to satisfy simple linear constraints approximately. There are also many graph drawing systems that are capable of handling clustered structures, for example Polyphemus, a system for exploring and visualizing Computer Networks, using the GDToolkit [GDT].

The GDToolKit allows user-defined drawing constraints to be specified over particular API methods for the implemented layout algorithms. A user could for example avoid crossings on an edge e of graph ug by calling $ug.new\_constraint\_uncrossable\_edge(e)$ before a planarization step. GDToolkit allows to restrict embeddings, the angles and bends in orthogonal drawings, and also the edge direction (upward drawings).

Adaptagrams [Dwyer and Wybrow] is a library of tools and reusable code for adaptive diagramming applications, for example graph drawing and chart layout tools. The Adaptagrams repository includes a solver for the quadratic programming problem in which the squared differences between a placement vector and some ideal placement are minimized subject to a set of separation constraints. In addition, a library for constraint graph layout is included that provides force-directed layout using the stress-majorization method subject to separation constraints. Non-overlapping vertex and cluster requirements as well as directed graph layout and topology preservation can be handled. Adaptagram's libavoid library provides and object-avoiding connector routing for use in interactive diagram editors. Adaptagrams currently contains the most flexible and advanced methods for constraint graph layout. Dunnart [Dwyer et al., 2008] is a constraint-based diagram editor that dynamically maintains the user-defined constraints during diagram editing, including the features provided by the Adaptagrams libary. Active constraints are graphically represented to support the user's editing.

5.2.2 Constraints in OGDF

The Open Graph Drawing Framework (OGDF) [OGDF] is a open-source software library for the automatic layout of graphs. It contains various sophisticated algorithms and data structures for drawing graphs which cannot be found in any other software. OGDF provides a wide range of graph drawing algorithms that allow to reuse and replace particular algorithm phases by using a dedicated module mechanism. The focus is on planarization methods and orthogonal layout, but also many classical layout algorithms like hierarchical and force-directed methods are contained in the library. Clustered graphs and hypergraphs as well as further drawing constraints motivated by practical applications are supported. The library is available under the GNU General Public License. The OGDF also provides an XML-based graph storage format called OGML. OGML is developed to allow both a structured human readable, and also easy to parse and validate format, combined with the flexibility to add application specific data.

The concept for constraint integration in OGDF tries to achieve maximum flexibility in constraint definition and at the same time independence from specific algorithm implementations. We tried to keep constraint handling and the respective interfaces simple, but also ensured that the set of supported constraints can easily be extended. Whereas often specification of constraints is part of the interface of a layout algorithm's implementation, we follow an approach that allows to specify and store constraints independent of a specific layout algorithm. Parts of the concept were developed and implemented by a stu-

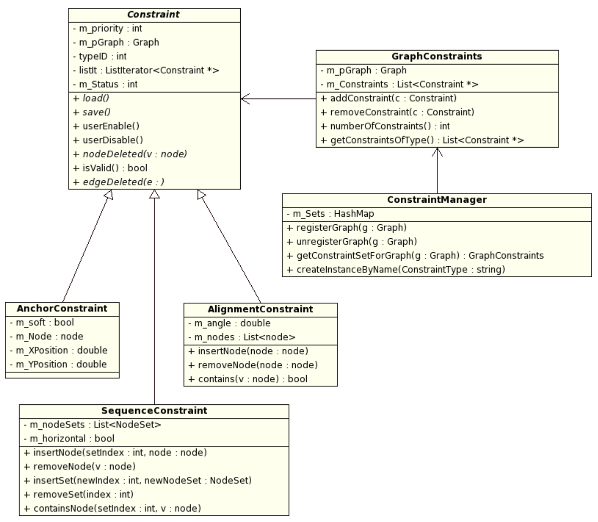

Constraint
- m_priority : int
- m_pGraph : Graph
- typeID : int
- listIt : ListIterator<Constraint *>
- m_Status : int
+ load()
+ save()
+ userEnable()
+ userDisable()
+ nodeDeleted(v : node)
+ isValid() : bool
+ edgeDeleted(e :)
GraphConstraints
- m_pGraph : Graph
- m_Constraints : List<Constraint *>
+ addConstraint(c : Constraint)
+ removeConstraint(c : Constraint)
+ numberOfConstraints() : int
+ getConstraintsOfType() : List<Constraint *>
ConstraintManager
- m_Sets : HashMap
+ registerGraph(g : Graph)
+ unregisterGraph(g : Graph)
+ getConstraintSetForGraph(g : Graph) : GraphConstraints
+ createInstanceByName(ConstraintType : string)
AnchorConstraint
- m_soft : bool
- m_Node : node
- m_XPosition : double
- m_YPosition : double
AlignmentConstraint
- m_angle : double
- m_nodes : List<node>
+ insertNode(node : node)
+ removeNode(node : node)
+ contains(v : node) : bool
SequenceConstraint
- m_nodeSets : List<NodeSet>
- m_horizontal : bool
+ insertNode(setIndex : int, node : node)
+ removeNode(v : node)
+ insertSet(newIndex : int, newNodeSet : NodeSet)
+ removeSet(index : int)
+ containsNode(setIndex : int, v : node)

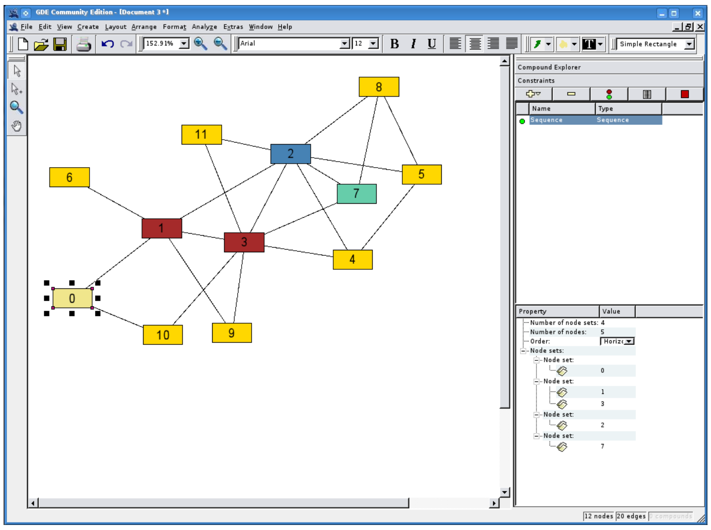
Figure 5.10: Class diagram depicting the implementation of the constraint handling in OGDF

.

dent project group under the joint supervision of Markus Chimani, Carsten Gutwenger, and Karsten Klein [PG478]. Adding constraint handling to OGDF includes the development of a constraint handling framework to specify, represent, and store constraints, as well as the implementation of algorithms that can cope with them.

Architecture. Figure 5.10 shows OGDF's main constraint handling classes. Class `Constraint` is the base class for all implementations of constraint types. It stores a priority that defines how important the constraint is such that for example in case of contradicting constraints that cannot be satisfied simultaneously, less important constraints can be ignored. The validity status of a constraint may also be checked, as the constraint may be temporarily disabled by the user or the system (e.g., as a result of a consistency check). Even though the validity is returned using a boolean value, internally an integer value is used to store the reason for invalidity on a finer level, to indicate, e.g., syntactical incorrectness or just inactivity. Class `ConstraintManager` is the central class that manages all defined constraints for all existing graphs. It creates constraint instances during the load process and stores a set of `GraphConstraints`. An instance of class `GraphConstraints` stores a list of all constraints defined for a single graph instance, and allows to modify it by adding or deleting or adding constraints.

The interface for a layout algorithm now simply is extended as follows: In addition to an instance of class `GraphAttributes`, which keeps a graph attributes, as the vertex coordinates, an instance of class `GraphConstraints` is passed that contains the constraints defined for the graph. As an algorithm might not support all possible constraints, it can filter the constraints by calling the method `getConstraintsOfType(int ConstraintType)` to obtain the supported constraints.


GDE Community Edition - [Document 3 *]
File Edit View Create Layout Arrange Format Analyze Extras Window Help
152.91%
Arial
12
Simple Rectangle
Compound Explorer
Constraints
8
11
2
6
5
7
1
3
4
0
10
9
12 nodes
20 edges
0 compounds

Name	Type
Sequence	Sequence

Property	Value
Number of node sets:	4
Number of nodes:	5
Order:	Horiz
Node sets:	
Node set:	
	0
Node set:	
	1
	3
Node set:	
	2
Node set:	
	7

Figure 5.11: Use of a sequence constraint in the GDE editor. Four vertex sets have to be placed with fixed ordering in horizontal direction (colors reflect set affiliation).

Storing Constraints. Constraints specified within the OGDF can be stored using the OGML. Even though a solution that defines specific constraints as fixed elements of the OGML XML schema seems to provide maximum readability and allows to validate against the schema, we decided to implemented a different approach. First of all, with such a schema-based definition the OGML schema and the parser would have to be adapted often for new or changed constraint definitions, and a dependency of the constraints on the OGML would be created. Instead, we decided to keep maximum flexibility and independence by introducing only a generic *constraint* element. Constraint properties and affected graph objects are defined using a toolkit of basic data types together with a *composed* element that allows complex compositions of basic data elements. This concept allows to easily represent objects, object sets, and also inclusion relations, as in hierarchical cluster structures. See listing 9 for an example of an alignment constraint. The data that has to be stored may vary largely from constraint type to constraint type, which would make a common serialization mechanism quite complex and difficult to maintain and extend. Therefore each constraint class in OGDF has to implement its own serialization method.

The basic data types consist of the following types:

- *int* for integer values
- *num* for floating point values
- *bool* for boolean values
- *string* for string values
- *nodeRef* for node references
- *edgeRef* for edge references

- *labelRef* for label references

```
<constraint id=''c03'' name=''Alignment1'' type=''Alignment''>
<num name=''angle'' value=''45''/>
<composed name=''NodeSet''>
  <nodeRef idRef=''n01''/>
  ...
  <nodeRef idRef=''n23''/>
<composed/>
</constraint>
```

Listing 5.1: Example of an alignment constraint representation in OGML, consisting of a angle specification and a *composed* element that contains the affected vertices.

As *node* elements in OGML can contain other *node* elements, compound and cluster constraints can simply be represented by inclusion, see listing 12.

```
...
<node id=''n01''>
<node id=''n01\_1''>
  <label>
    <content>
      Keep it short and
    </content>
  </label>
</node>
</node>
...
```

Listing 5.2: Example of *node* inclusion to represent compound hierarchies.

When an OGML file is loaded that contains constraint definitions, the constraint manager creates an instance for the defined constraint type and passes it the corresponding part of the parse tree. The constraint instance then loads the related information, including the information on the affected graph objects.

Algorithms. As a proof of concept, a force-directed algorithm (based on the spring embedder approach) was extended to respect Alignment, Anchor, and Sequence constraints. See Figure 5.11 for an example layout with a sequence constraint applied that forces vertex sets to a fixed placement order with respect to the horizontal axis.

6. EMBEDDING CONSTRAINTS

In many application domains where information visualization is based on graph representations, graph drawings for specific diagram types are only valid if they adhere to certain restrictions of the graph's embedding. In database diagrams, for example, links between attributes should enter the tables only at the left or right side of the corresponding attributes, the placement of reactants in chemical reactions or biological pathways should reflect their role within the displayed reactions, and in UML class diagrams, generalization edges should leave a class object at the top and enter a base class object at the bottom. Even more important is the possibility to use such drawing restrictions in order to express the user's preferences and to guide the layout phase.

In this chapter, we consider restrictions on the allowed order of incident edges around a vertex, for example, to specify groups of edges that have to appear consecutively around the vertex or that have a fixed clockwise order in any admissible embedding. Such constraints may occur in the form of *side constraints*, where incident edges are assigned to the four sides of a rectangular vertex, or *port constraints* where edges have prescribed attachment points at a vertex. Our first contribution is the introduction of an embedding constraint model in Section 6.1. In particular, we introduce three types of constraints which may be arbitrarily nested: *grouping*, *oriented* (prescribed clockwise order), and *mirror* constraints (prescribed reversible order). We call a planar embedding that fulfills the given set of constraints an *ec-planar* embedding.

An important optimization goal for the computation of graph layouts is the minimization of crossings. The problem of minimizing the number of crossings in a drawing is NP-hard [Garey and Johnson, 1983] and no practically efficient method exists so far. In practice, the problem is attacked via the *planarization* approach [Batini et al., 1984] which first deletes a number of edges until the remaining graph is planar and then carefully reinserts them (iteratively) so that the number of crossings is minimized, see for example Gutwenger and Mutzel [2004]. This chapter deals with the integration of the embedding constraint concept into the planarization approach. The first step of this approach can be solved by successive ec-planarity testing. Our second contribution therefore is a linear time algorithm for testing if a graph with a set of embedding constraints is ec-planar, see Section 6.3. The main challenge is to incorporate oriented constraints, where a given clockwise order of (groups of) incident edges needs to be satisfied. Furthermore, we characterize all possible ec-planar embeddings using BC- and SPQR-trees, which also yields a linear time algorithm for computing an ec-planar embedding.

The second step of the planarization approach can be tackled by repeatedly solving the *optimal edge insertion problem*: This problem asks for inserting an edge $e = (v, w)$ into a planar graph so that all crossings involve e and their number is minimized. Alternatively, the problem can be stated as finding an embedding of a planar graph G where the given edge can be inserted with the minimum number of crossings. This problem has been solved in linear time using the SPQR-tree data structure [Gutwenger et al., 2005]. The algorithm essentially computes a shortest path Ψ between those nodes in the SPQR-tree $\mathcal{T}$ of G whose skeletons contain v and w, respectively. The optimal insertion path is then constructed by simply concatenating locally optimal insertion paths of the tree nodes on Ψ.

However, if embedding constraints have to be observed, i.e., restrictions on the order of the edges around the vertices of G are given, locally optimal solutions need not lead to globally optimal solutions and the greedy approach cannot be applied anymore. The best local decision now depends on the decisions for other parts of the edge insertion path. Our third contribution is thus a new linear time algorithm to solve the *optimal ec-edge insertion problem* (see Section 6.4): Given an ec-planar graph G with an additional edge e and a set of embedding constraints C for the graph $G + e$, our algorithm computes an *ec-planar* embedding of G together with a crossing minimal edge insertion path for e that observes C.

There is not much previous work concerning constraints on the admissible embeddings of a graph, see Section 5.1.2 for a description. On the other hand, linear time algorithms for planarity testing and embedding are long since known; see Hopcroft and Tarjan [1974], Booth and Lueker [1976], Chiba et al. [1985], Mehlhorn and Mutzel [1996], Boyer and Myrvold [2004].

This chapter is organized as follows. Section 6.1 formally defines the embedding constraint model. The first part of the ec-planarity test consists of transforming the input graph into an *ec-expansion* which is described in Section 6.2; the characterization of ec-planar embeddings and the ec-planarity test itself is then presented in Section 6.3. Section 6.4 covers the linear time algorithm for solving the ec-edge insertion problem. Finally, we conclude the chapter with remarks on open problems.

6.1 ec-Constraints and ec-Planarity

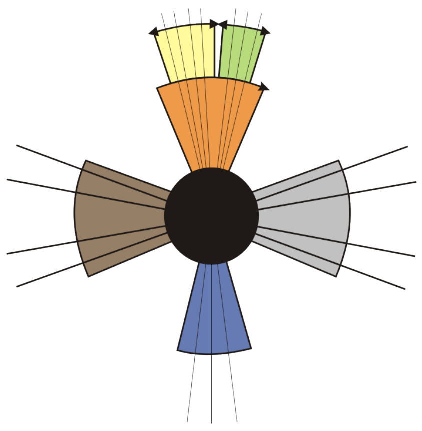
Let $G = (V, E)$ be a graph. An embedding constraint specifies the admissible clockwise order of the edges incident to a vertex in a combinatorial embedding of G. In this chapter, we consider the case where a vertex has at most one embedding constraint and either all or none of the edges incident to a vertex are subject to embedding constraints.

An *embedding constraint* at a vertex $v \in V$ is a rooted, ordered tree T_v such that its leaves are exactly the edges incident to v. The inner nodes of T_v, also called *constraint-nodes* or *c-nodes* for short, are of three types: *oc-nodes* (oriented constraint-nodes), *mc-nodes* (mirror constraint-nodes), and *gc-nodes* (grouping constraint-nodes). Since T_v is an ordered tree, it imposes an order on its leaves and thus on the edges incident to v. We consider this order as a cyclic order and represent all *admissible* cyclic, clockwise orders of the edges incident to v by defining how the order of the children of c-nodes in T_v can be changed:

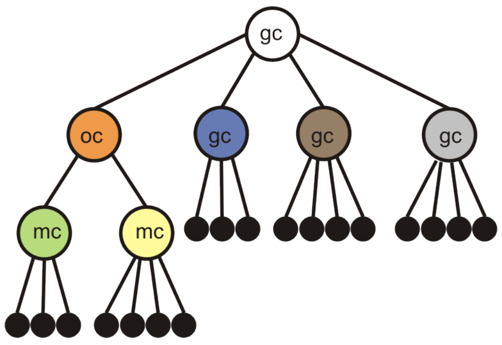
gc-node: The order of children may be arbitrarily permuted.

mc-node: The order of children may be reversed.

oc-node: The order of children is fixed.



(a) Partitioning of edges.



(b) Constraint tree.

Figure 6.1: The hierarchical partitioning of edges imposed by an embedding constraint (a) and the corresponding constraint tree (b).

Figure 6.1 shows an example for an embedding constraint. A c-node with a single child is obviously redundant, therefore we demand that each c-node has at least two children. While gc- and mc-nodes alone resemble the concept of PQ-trees Booth and Lueker [1976], the additional concept of oc-nodes is necessary to model constraints that arise in many practical applications, and that complicate planarity testing.

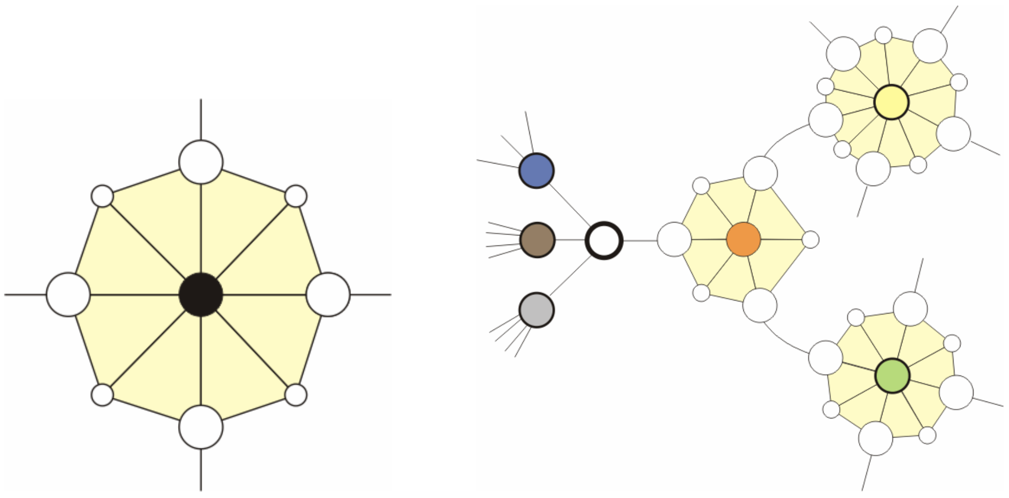
Let C be a set of embedding constraints at distinct vertices of G. A combinatorial embedding Γ of G *observes* the embedding constraints in C, if for each embedding constraint $T_v \in C$, the cyclic clockwise order of the edges around v in Γ is admissible with respect to T_v. A planar embedding observing the embedding constraints in C is an *ec-planar embedding* with respect to C, and (G, C) is *ec-planar*, if there exists an ec-planar embedding of G with respect to C.

6.2 ec-Expansion

A basic building block of the ec-planarity test is a structural transformation applied to a given graph G with embedding constraints C. For each embedding constraint T_v at vertex v, this transformation expands v according to the structure of T_v. We call the resulting graph the *ec-expansion* $E(G, C)$ of G with respect to C. The details of this transformation are given below.

6.2.1 Construction of the ec-Expansion

The *ec-expansion* $E(G, C)$ of G with respect to C is constructed as follows. Let $T_v \in C$ be an embedding constraint and T'_v the subgraph obtained from T_v by omitting its leaves. Recall that the leaves of T_v are exactly the edges incident to v. We replace v in G by the tree T'_v and connect the edges incident with v with the parents of the corresponding leaves. This transformation introduces a vertex in G for every c-node in T_v. Each vertex u corresponding



(a) Wheel gadget. (b) Vertex expansion.

Figure 6.2: Expansion gadgets: (a) a wheel gadget replacing a vertex with degree 4; (b) vertex expansion according to the constraint tree in Figure 6.1(b) (the thick hollow vertex is the root).

to an oc- or mc-node is further replaced by a *wheel gadget* which is a wheel graph with $2d$ spokes, were $e_1, \dots, e_d$ are the edges incident to u. Then, the respective wheel gadget consists of a cycle $x_1, y_1, \dots, x_d, y_d$ of length $2d$ and a vertex, called *hub*, incident to every vertex on the cycle; see Figure 6.2(a). The vertex u is replaced by this wheel gadget, such that e_i is connected to x_i for $1 \leq i \leq d$. According to the type of the expanded c-node, we distinguish between *O-hubs* (oc-nodes) and *M-hubs* (mc-nodes). We refer to the edges introduced during the ec-expansion as *expansion edges*. Figure 6.2(b) shows the expansion of a vertex according to the constraint tree shown in Fig 6.1(b).

The purpose of the wheel gadgets is to model the fixed order of the children of the corresponding c-node. Since a wheel gadget is a triconnected graph, it admits only two combinatorial embeddings that are mirror images of each other. The order in which non-gadget edges are attached to the wheel cycle is either the order given by the corresponding c-node, or the reverse order. Every face adjacent to the hub is a triangle. We call these faces *inner wheel gadget faces*.

Lemma 6.1. *Let $G = (V, E)$ be a graph with embedding constraints C. Then, its ec-expansion $E(G, C)$ has size O$(|V| + |E|)$ and can be constructed in time O$(|V| + |E|)$.*

Proof. Consider an embedding constraint $T_v \in C$. Since the leaves of T_v are in one-to-one correspondence to the edges incident to v and each c-node has at least two children, the size of T_v is linear in $\deg(v)$. We replace each oc- and mc-node μ by a wheel gadget with $4\deg(\mu)$ edges. Thus, the expansion of vertex v creates $\mathrm{O}(\deg(v))$ edges, and the total number of additional edges in $E(G, C)$ is bounded by $\sum_{v \in V} \mathrm{O}(\deg(v)) = \mathrm{O}(|E|)$. Therefore, the size of the expansion graph is $\mathrm{O}(|V| + |E|)$, and the expansion can obviously be computed in $\mathrm{O}(|E(G, C)|) = \mathrm{O}(|V| + |E|)$ time. □

6.2.2 ec-Expansion and ec-Planar Embeddings

In this section we discuss the relationship between planar embeddings of the ec-expansion $E(G, C)$ and ec-planar embeddings of (G, C). Though the ec-expansion serves as a tool for modeling the embedding constraints in C, a planar embedding of $E(G, C)$ needs to fulfill certain conditions in order to induce an ec-planar embedding of G with respect to C. We call a planar embedding Γ of $E(G, C)$ *ec-planar* if

(a) the external face of Γ does not contain a hub;

(b) every face incident to a hub is a triangle consisting solely of edges of the corresponding wheel gadget; and

(c) each O-hub h is *oriented correctly*, i.e., the cyclic, clockwise order of the edges around h in Γ corresponds to the order specified by the corresponding oc-node.

Let Γ be an ec-planar embedding of $E(G, C)$. We obtain an ec-planar embedding of (G, C) as follows. For each vertex v with corresponding embedding constraint in C, there is a connected subgraph G_v in $E(G, C)$ resulting from expanding v. Let $\bar{G}_v \subset E(G, C)$ be the graph induced by the vertices not contained in G_v. The conditions above assure that the planar embedding Γ_v of G_v induced by Γ is such that $\bar{G}_v$ lies in the external face of Γ_v. The edges that connect G_v to $\bar{G}_v$ correspond to the edges incident to v in G. Their cyclic clockwise order around G_v is admissible with respect to T_v, since the wheel gadgets fix the order of the edges specified by oc- and mc-nodes, and O-hubs are oriented correctly. We shrink G_v to a single vertex by contracting all edges in G_v while preserving the embedding, thus resulting in an admissible order of the edges around v.

If we have an ec-planar embedding of (G, C), then the edges around each vertex v are ordered such that the constraints in T_v are fulfilled. It is easy to see that we can replace each such vertex v by the expansion graph corresponding to T_v in such a way that we obtain an ec-planar embedding of $E(G, C)$. Thus, we get the following result:

Lemma 6.2. *Let G be a graph with embedding constraints C. Then, (G, C) is ec-planar if and only if $E(G, C)$ is ec-planar. Moreover, every ec-planar embedding of $E(G, C)$ induces an ec-planar embedding of (G, C).*

6.3 ec-Planarity Testing

It is well-known that planarity testing can be reduced to biconnected graphs, i.e., it is sufficient to test the blocks of a graph independently. However, adding embedding constraints complicates this task. Let G be a graph with embedding constraints C. Consider a cut vertex c in G that connects two blocks BC_1 and BC_2 via the edge sets S_1 and S_2, respectively; see Figure 6.3(a). If these edge sets are subject to embedding constraints that force the edges in S_1 and S_2 to be intermixed as in Figure 6.3(a), then the given graph is not ec-planar even if its blocks are ec-planar. We solve this problem by first applying the ec-expansion to the graph. This replaces the cut vertex c by a wheel gadget so that c does not separate BC_1 and BC_2 anymore; see Figure 6.3(b).

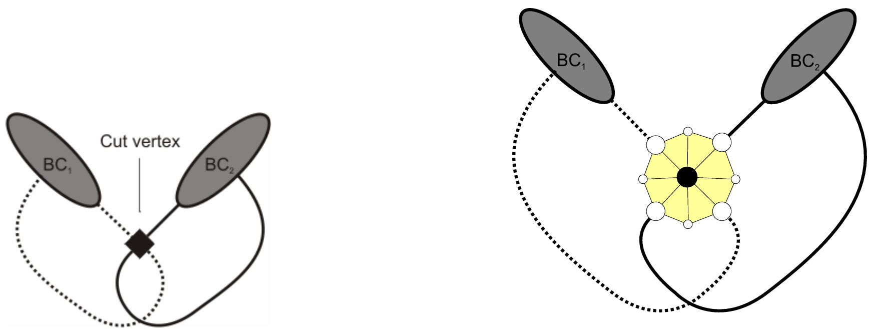


Figure 6.3: A crossing is needed between edges of two blocks due to embedding constraints (a). The expansion using a wheel gadget merges the two blocks into a single one (b).

By Lemma 6.2, we know that it is sufficient to test the ec-expansion $E(G, C)$ for ec-planarity. In contrast to the graph G itself, the following lemma shows that we can test the blocks of $E(G, C)$ separately.

Lemma 6.3. *$E(G, C)$ is ec-planar if and only if every block of $E(G, C)$ is ec-planar.*

Proof. If $E(G, C)$ is ec-planar, then there is an ec-planar embedding of $E(G, C)$, and this embedding implies an ec-planar embedding for each block of $E(G, C)$.

Suppose now that each block of $E(G, C)$ is ec-planar. Consider a wheel gadget $\mathscr{G}$ in $E(G, C)$. Since $\mathscr{G}$ is triconnected, $\mathscr{G}$ is completely contained in a single block B of $E(G, C)$. For each edge $(u, v) \in \mathscr{G}$, the pair $\{u, v\}$ is not a separation pair in B by construction, hence every inner wheel face of $\mathscr{G}$ is also a face in every planar embedding of B. Moreover, the hub of $\mathscr{G}$ is not a cut vertex of $E(G, C)$, since all its incident edges are in B.

We construct an ec-planar embedding of $E(G, C)$ as follows. We start with an arbitrary block B of $E(G, C)$. Let Π be an ec-planar embedding of B. In particular, the external face of Π is not an inner wheel face of a wheel gadget. We add the remaining blocks successively to Π. Let B' be another block of $E(G, C)$ that shares a vertex c with B, and let Π' be an ec-embedding of B'. We pick faces $f \in \Pi$ and $f' \in \Pi'$ that are adjacent to c and not inner wheel faces of a wheel gadget. This is possible, since the only vertices adjacent solely to inner wheel faces are the O- and M-hubs. Then, we insert Π' with f' as external face into the face f of Π. This results in an ec-planar embedding of $B \cup B'$. We can add the remaining blocks (if any) in the same way, resulting in an ec-planar embedding of $E(G, C)$. □

If we can characterize all ec-planar embeddings of the blocks of $E(G, C)$, the construction in the proof of Lemma 6.3 also shows us how to enumerate all ec-planar embeddings of $E(G, C)$ by traversing its BC-tree. In the following, we devise such a characterization. Let B be a block of $E(G, C)$ and $\mathcal{T}$ its SPQR-tree.

Observation 6.1. *Every wheel gadget $\mathscr{G}$ is completely contained within the skeleton of an R-node. In particular, the hub of $\mathscr{G}$ occurs only in the skeleton of a single R-node.*

Proof. $\mathscr{G}$ is triconnected, and for each edge $(u, v) \in \mathscr{G}$, the pair $\{u, v\}$ is not a separation pair in B by construction. Therefore, all edges of $\mathscr{G}$ occur in the same skeleton graph, which must be the skeleton of an R-node μ. The hub h of $\mathscr{G}$ is only incident to edges of $\mathscr{G}$ and no other edge of B, hence h occurs only in *skeleton*(μ). □

If B is planar, then the skeleton of an R-node is a triconnected planar graph, thus having exactly two planar embeddings which are mirror images of each other. We call two O-hubs contained in the same skeleton S *conflicting* if none of the two planar embeddings of S orients both O-hubs correctly. The following theorem gives us an easy to check condition for ec-planarity and characterizes all possible ec-planar embeddings:

Theorem 6.1. *Let G be a graph with embedding constraints C. Let B be a block of $E(G, C)$ and $\mathcal{T}$ its SPQR-tree. Then, the following holds:*

(a) B is ec-planar if and only if B is planar and no skeleton of an R-node of $\mathcal{T}$ contains conflicting O-hubs.

(b) If B is ec-planar, then the embeddings of the skeletons of $\mathcal{T}$ induce an ec-planar embedding of B if and only if each O-hub in the skeleton of an R-node is oriented correctly.

Proof. If B admits an ec-planar embedding, then this embedding induces embeddings of the skeletons of $\mathcal{T}$ such that every O-hub in the skeleton of an R-node is oriented correctly. In particular, no R-node skeleton contains conflicting O-hubs.

Suppose now that B is planar and no R-node skeleton contains conflicting O-hubs. For each R-node skeleton containing at least one O-hub, we can chose planar embeddings such that all O-hubs are oriented correctly within the skeletons. We have to show that the embeddings of the skeletons induce an ec-planar embedding of B, even if we chose arbitrary embeddings for the remaining skeletons. This holds, since every such embedding Π has the property that each O-hub is oriented correctly because wheel gadgets are completely contained within R-node skeletons by Observation 6.1, and inner wheel faces are preserved. We can pick any face of Π as external face which is not an inner wheel face (such a face always exists) and obtain an ec-planar embedding of B. □

Function ISECPLANAR depicted in Algorithm 6.1 applies Theorem 6.1 and devises a linear time ec-planarity test, which can easily be extended so that it computes an ec-planar embedding as well.

Theorem 6.2. *Let $G = (V, E)$ be a graph with embedding constraints C. Then, the function* ISECPLANAR *tests (G, C) for ec-planarity in time $O(|V| + |E|)$. Moreover, if (G, C) is ec-planar, an ec-planar embedding of (G, C) can also be computed in time $O(|V| + |E|)$.*

Proof. By Lemma 6.2 and 6.3, it is sufficient to test every block of $E(G, C)$ for ec-planarity. Hence, the correctness of ISECPLANAR follows from Theorem 6.1.

Constructing the ec-expansion (Lemma 6.1) and testing planarity [Hopcroft and Tarjan, 1974] can be done in linear time. For each block B of $E(G, C)$, we construct its SPQR-tree, which requires linear time in the size of B; see Gutwenger and Mutzel [2001]. The check for conflicting O-hubs is easy to implement: For each R-node skeleton S, we compute a planar embedding of S. If this embedding contains both correctly as well as incorrectly oriented

```
function IsEcPlanar(Graph $G$, Constraints $C$) : bool
    Construct ec-expansion $E$ of $(G, C)$.
    if $E$ is not planar then return false
    for each block $B$ of $E$ do
        Construct SPQR-tree $\mathcal{T}$ of $B$.
        for each R-node $\mu \in \mathcal{T}$ do
            if skeleton$(\mu)$ contains two conflicting O-hubs then
                return false
            end if
        end for
    end for
    return true
end function
```

Algorithm 6.1: Ec-planarity testing.

O-hubs, then there is a conflict, otherwise not. Since the total size of skeleton graphs is linear in the size of B and a planar embedding can be found in linear time (see, e.g., Chiba et al. [1985]), we need linear running time for each block. Hence, the total running time is linear in the size of $E(G, C)$ which is $O(|V| + |E|)$ by Lemma 6.1.

In order to find an ec-planar embedding of G, we just have to compute embeddings of the skeleton graphs for each block as described in Theorem 6.1 and combine the embeddings as described in the proof of Lemma 6.3. □

6.4 ec-Edge Insertion

We now solve the following ec-edge insertion problem:

EC-EDGE INSERTION PROBLEM	
Instance:	an ec-planar graph $G = (V, E)$, two distinct vertices $u, v \in V$, a set of embedding constraints C for $G + (u, v)$
Solution:	an ec-planar embedding Π of (G, C) and an ec-edge insertion path p for (u, v)
Minimize:	the length of p

6.4.1 ec-Edge Insertion Paths and ec-Traversing Costs

We first generalize the terms insertion path and traversing costs introduced in Gutwenger et al. [2005]. Intuitively, the edges in an insertion path are the edges we need to cross when inserting an edge (x, y) into an embedding. Let $G + (x, y)$ be a graph with embedding constraints C. An *ec-edge insertion path* for (x, y) in an ec-planar embedding Π of G is a sequence of edges $e_1, \ldots, e_k$ of G satisfying the following conditions:

(1) There is a face $f_x \in \Pi$ with $x, e_1 \in f_x$, a face $f_y \in \Pi$ with $e_k, y \in f_y$, and faces $f_i \in \Pi$ with $e_i, e_{i+1} \in f_i$ for $1 \leq i < k$.

(2) The edge order around x and y is admissible with respect to C if (x, y) leaves x via face f_x and enters y via face f_y.

Finding a shortest ec-insertion path in a fixed embedding Π is easy: We only need to identify the set of faces F_x incident to x where the insertion path may start, and F_y incident to y where it may end, and then find a shortest path in the dual graph of Π connecting a face in F_x with a face in F_y.

We are interested in the shortest possible ec-insertion path among all ec-planar embeddings of G, which we also call an *optimal ec-insertion path* in G. In particular, we need to identify the required ec-planar embedding of G. In order to represent all ec-planar embeddings of G, we apply Lemma 6.2 and use its ec-expansion instead. More precisely, we use the subgraph $K = E(G + (x, y), C) \setminus e$, where $e = (v, w)$ is the edge of $E(G + (x, y), C)$ connecting the expansion of x with the expansion of y. An ec-insertion path in an ec-planar embedding of K is defined as before with the only difference that we replace the second condition with

$(2')$. $e_1, \ldots, e_k$ contains no expansion edge of K.

It is easy to see that we can also use this definition for a subgraph B of K and two distinct vertices of B that are not hubs.

We adapt the notion of traversing costs defined in Gutwenger et al. [2005] to ec-planarity. Let e be a skeleton edge, and let Π be an arbitrary ec-embedding of the graph *expansion*$^+(e)$ with dual graph Π^*, in which all edges corresponding to gadget edges have length ∞ and the other edges have length 1. Let f_1 and f_2 be the two faces in Π separated by e. We denote with $P(\Pi^*, e)$ the length of the shortest path in Π^* that connects f_1 and f_2 and does not use the dual edge of e. Hence, we have $P(\Pi^*, e) \in \mathbb{N} \cup \{\infty\}$.

The following lemma follows analogously to the result shown in Gutwenger et al. [2005].

Lemma 6.4. *Let μ be a node in $\mathcal{T}$ and e an edge in skeleton(μ). Then, $P(\Pi^*, e)$ is independent of the ec-embedding Π of expansion$^+(e)$.*

Proof. Let m be the number of edges in $G_e :=$ *expansion*$^+(e)$ and G'_e be the graph obtained from G_e by replacing each gadget edge with $m + 1$ parallel edges. Then, each embedding Π of G_e corresponds to an embedding Π' of G'_e, and $P(\Pi^*, e)$ is ∞ if and only if the corresponding path in Π' is longer than m. Lemma 1 in Gutwenger et al. [2005] shows that for the general case, i.e., without embedding constraints, $P(\Pi^*, e)$ is independent of the embedding Π. Applying this lemma and observing that the ec-embeddings of G_e are a non-empty subset of the embeddings of G_e yields the lemma. $\square$

Thus, we define the *ec-traversing costs* $c(e)$ of a skeleton edge e as $P(\Pi^*, e)$ for an arbitrary ec-embedding Π of *expansion*$^+(e)$.

6.4.2 The Algorithm for Biconnected Graphs

The hard part is to find an ec-insertion path in a block B of K. Our task is to compute an optimal ec-insertion path between two nodes v, w of B. The function OPTIMALECBLOCKINSERTER shown in Algorithm 6.2 and 6.3 solves this problem. In this algorithms, we use the notation $\min_{i,n} \Lambda$ which returns a tuple in the set Λ of n-tuples whose i-th component is minimal among all tuples in Λ.

The function OPTIMALECBLOCKINSERTER is called with a block B of an ec-planar ec-expansion and two distinct vertices v and w of B. Since we assume that B contains all gadget edges, we do not need to pass further constraint information for the edge (v, w). In particular, using any insertion path in any ec-planar embedding of B that connects v and w and does not cross a gadget edge yields an ec-embedded planarization of $B \cup (v, w)$. Hence, we look for an ec-embedding of B that allows the insertion of the edge (v, w) with the minimum number of crossings.

First, we compute the SPQR-tree $\mathcal{T}$ of B and embed the skeletons such that they imply an ec-embedding of B, i.e, the R-node skeletons are embedded correctly. Then, the shortest path $\Upsilon := \mu_1, \dots, \mu_k$ between an allocation node μ_1 of v and μ_k of w is identified. In order to achieve a consistent orientation, we root $\mathcal{T}$ such that Υ is a descending path in the tree, i.e., μ_i is the parent of μ_{i-1} for $i = 2, \dots, k$. Note that the rooting of the SPQR-tree implies a direction of the skeleton edges: the edges in a skeleton with reference edge $e_r = (s, t)$ are directed such that the skeleton is a planar st-graph; see, e.g., Di Battista and Tamassia [1996]. This direction is necessary in order to identify the left and the right face of an edge.

The algorithm traverses the path Υ from μ_1 to μ_{k-1} and iteratively computes the lengths of the shortest ec-insertion paths that start from v and leave the pertinent graph P_i of μ_i to the left or to the right, respectively, where all ec-embeddings of P_i are considered. Here, left and right refer to the direction of the reference edge of μ_i. These lengths are maintained in the variables λ_ℓ and λ_r. Finally, when node μ_k is considered, this information is used to determine a shortest insertion path ending at w.

For each node μ_i, the following information is computed:

- ϕ_ℓ^i (resp. ϕ_r^i) indicates if the shortest ec-insertion path leaving P_i to the left (right) uses the shortest ec-insertion path that leaves P_{i-1} to the left (in this case the value is ℓ) or to the right (the value is r).

- Δ_ℓ^i (resp. Δ_r^i) is the subpath that is appended to the path leaving P_{i-1} when leaving P_i to the left (right).

These values are used solely for the purpose of creating the optimal ec-insertion path at the end of the function. If $s \in \{\ell, r\}$ denotes a side, we denote with $\bar{s}$ the other side, i.e., $\bar{\ell} = r$ and vice versa.

The body of the for-loop starts by expanding all edges of the skeleton S_i of μ_i except for edges representing v or w. The resulting graph is called G_i. If $1 < i < k$, then G_i will contain two virtual edges e_v (representing v) and e_w (representing w). Note that we obtain P_i (plus reference edge) by replacing e_v with P_{i-1}.

We distinguish according to the type of μ_i. If μ_i is a P-node, then the optimal ec-insertion path leaving P_{i-1} to the left (right) is also an optimal ec-insertion path leaving P_i to the left

```
function OptimalEcBlockInserter(Block $B$ of $K$, vertex $v$, vertex $w$)
    Construct SPQR-tree $\mathcal{T}$ of $B$ such that the embeddings of the
    skeletons imply a feasible embedding of $B$.

    Find the shortest path $\mu_1, \dots, \mu_k$ in $\mathcal{T}$ between an allocation node
    $\mu_1$ of $v$ and $\mu_k$ of $w$.
    Root $\mathcal{T}$ such that $\mu_k$ becomes the parent of $\mu_{k-1}$ (if $k > 1$).

    $\lambda_\ell := \lambda_r := 0$                ▷ length of shortest insertion path leaving to the left/right

    for $i = 1, \dots, k$ do
        let $S_i$ = skeleton($\mu_i$)

        let $G_i$ be the graph obtained from $S_i$ by replacing each edge not
        representing $v$ or $w$ with its expansion graph, and let $\Pi_i$ be the
        embedding of $G_i$ induced by the embeddings of the skeletons of
        $\mathcal{T}$.

        ▷ $\phi^i_{l/r}$ indicates which insertion path of $\mu_{i-1}$ is chosen;
        ▷ $\Delta^i_{l/r}$ denotes the subpath within $S_i$ when leaving left/right.
        if $\mu_i$ is a P-node then
            $(\phi^i_\ell, \Delta^i_\ell) := (\ell, \epsilon)$; $(\phi^i_r, \Delta^i_r) := (r, \epsilon)$          ▷ No crossings required.
        else                                                          ▷ S- or R-node.
            if $i = 1$ then
                $L_v := R_v :=$ the set of adjacent faces of the copy of $v$ in $S_i$
            else
                let $e_v$ be the representative of $v$ in $S_i$
                $L_v := \{$ the left face of $e_v\}$
                $R_v := \{$ the right face of $e_v\}$
            end if

            if $i = k$ then
                $L_w := R_w :=$ the set of adjacent faces of the copy of $w$ in $S_i$
            else
                let $e_w$ be the representative of $w$ in $S_i$
                $L_w := \{$ the left face of $e_w\}$
                $R_w := \{$ the right face of $e_w\}$
            end if

            ▷ Compute shortest ec-insertion paths (from l/r to l/r) within $G_i$.
            ▷ Note: $p_{\ell r} = p_{\ell\ell}$ and $p_{rr} = p_{r\ell}$ if $i \in \{1, k\}$.
            $p_{\ell r} :=$ ShortestEcInsPath($\Pi_i, L_v, L_w$)
            $p_{\ell\ell} :=$ ShortestEcInsPath($\Pi_i, L_v, R_w$)
            $p_{rr} :=$ ShortestEcInsPath($\Pi_i, R_v, L_w$)
            $p_{r\ell} :=$ ShortestEcInsPath($\Pi_i, R_v, R_w$)

                                                    ▷ Continued on next page...
```

Algorithm 6.2: Computation of an optimal ec-insertion path (biconnected case).

```
                ▷ Collect possible solutions.
                $\Lambda_\ell := \{\, (\lambda_\ell + |p_{\ell\ell}|, \ell, p_{\ell\ell}),\ (\lambda_r + |p_{r\ell}|, r, p_{r\ell}) \,\}$
                $\Lambda_r := \{\, (\lambda_\ell + |p_{\ell r}|, \ell, p_{\ell r}),\ (\lambda_r + |p_{rr}|, r, p_{rr}) \,\}$
                if $\mu_i$ is an R-node that can be mirrored then
                    $\Lambda_\ell := \Lambda_\ell \cup \{\, (\lambda_\ell + |p_{rr}|, \ell, p^*_{rr}),\ (\lambda_r + |p_{\ell r}|, r, p^*_{\ell r}) \,\}$
                    $\Lambda_r := \Lambda_r \cup \{\, (\lambda_\ell + |p_{r\ell}|, \ell, p^*_{r\ell}),\ (\lambda_r + |p_{\ell\ell}|, r, p^*_{\ell\ell}) \,\}$
                end if
                ▷ Pick best solution.
                $(\lambda_\ell, \phi^i_\ell, \Delta^i_\ell) := \min_{1,3} \Lambda_\ell$
                $(\lambda_r, \phi^i_r, \Delta^i_r) := \min_{1,3} \Lambda_r$
            end if
        end for
        ▷ Build final ec-insertion path. Note: $\lambda_\ell = \lambda_r$ always holds here!
        $s_k := \ell$                                          ▷ Start with empty path.
        for $i := k$ downto $1$ do                             ▷ Collect path backward.
            $p_i := \Delta^i_{s_i}$; $s_{i-1} := \phi^i_{s_i}$
        end for
        return $p_1 + \cdots + p_k$
    end function
```

Algorithm 6.3: Function OPTIMALECBLOCKINSERTER (part 2).

(right); we just need to permute the parallel edges in S_i such that e_v is the leftmost (rightmost) edge. Otherwise, we have four possibilities for extending an ec-insertion path leaving P_i. Such a path may start in a face left or right of e_v, and may end in a face left or right of e_w. In addition, we have to consider two special cases: if $i = 1$ then G_i contains v and the ec-insertion path may start in any face adjacent to v; if $i = k$ then G_i contains w and the ec-insertion path may end in any face adjacent to w. We compute the (at most) four possible shortest ec-insertion paths using the function SHORTESTECINSPATH(Π, F_s, F_t). Here Π is an ec-embedding of an ec-expansion, F_s are the faces where the insertion path may start, and F_t are the faces where it may end. The ec-insertion path is found using a breadth-first search (BFS) in the dual graph of Π, where edges corresponding to gadget edges are removed (which means that it is forbidden to cross their primal counterparts). We call these shortest ec-insertion paths $p_{\ell\ell}, p_{\ell r}, p_{r\ell}, p_{rr}$, where $p_{\ell\ell}$ stands for the path starting in a face in L_v and ending in a face in R_w etc. We have two choices for a shortest ec-insertion path leaving P_i to the left if we consider only the given embedding of the skeleton of μ_i:

- We leave P_{i-1} to the left (or start at v if $i = 1$) and end in a face in R_w (i.e., we enter e_w from right). This path has length $\lambda_\ell + |p_{\ell\ell}|$.
- We leave P_{i-1} to the right (or start at v if $i = 1$) and end in a face in R_w (i.e., we enter e_w from left). This path has length $\lambda_r + |p_{r\ell}|$.

For the shortest ec-insertion path leaving P_i to the right, we have two similar cases. Further choices are possible if μ_i is an R-node that can be mirrored. We could mirror the embedding

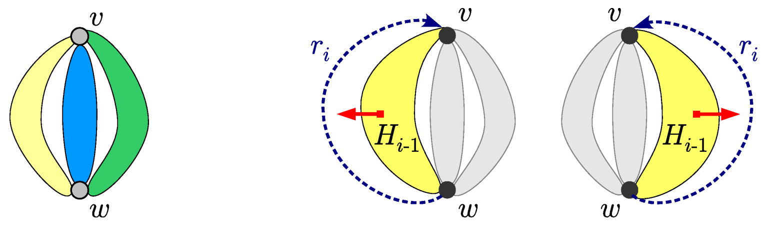


Figure 6.4: Proof of Lemma 6.5; $k = 1$ and μ_1 is a P-node (a), and μ_i is a P-node (b).

of S_i, expand the skeleton edges as before such that we obtain an embedding $\tilde{\Pi}_i$, and compute the four paths in $\tilde{\Pi}_i$ again. Notice that $\tilde{\Pi}_i$ is not simply the mirror image of Π_i. However, this is not necessary. We observe that, e.g., the path $\tilde{p}_{\ell\ell}$ is obtained from p_{rr} by reversing the subsequences of edges that have been created by expanding a common skeleton edge of S_i. We call this path p^*_{rr}. A similar argumentation holds for $\tilde{p}_{\ell r}, \tilde{p}_{r\ell}, \tilde{p}_{rr}$. It follows that we have at most four possible choices for leaving P_i to the left and to the right, respectively. Among all possible choices, we pick the shortest one.

After processing all nodes μ_i, it is easy to reconstruct the best ec-insertion path from v to w using $\phi^i_{\ell/r}$ and $\Delta^i_{\ell/r}$. Notice that $\lambda_\ell = \lambda_r$ holds at the end, since $L^k_w = R^k_w$.

6.4.3 Correctness and Optimality

Lemma 6.5. *There exists an ec-embedding* Π *of* B *such that* $p_1 + \cdots + p_k$ *is an ec-insertion path for* v *and* w *in* B *with respect to* Π.

Proof. Consider the path $\Upsilon = \mu_1, \dots, \mu_k$ computed by the algorithm. By construction of Υ, the skeleton of μ_1 contains v, the skeleton of μ_k contains w, and, for each $j = 2, \dots, k-1$, the skeleton of μ_j contains neither v nor w. Moreover, Υ does not contain a Q-node.

First, we prove the lemma for the case where Υ consists of a single node μ_1. In this case, the skeleton of μ_1 contains both v and w. We distinguish two cases according to the type of μ_1:

(1) μ_1 **is a P-node.** Let Π be an arbitrary ec-embedding of B. Since v and w share a common face in Π, the empty path returned by the algorithm is an ec-insertion path for v and w in B with respect to Π; see Figure 6.4(a).

(2) μ_1 **is an S- or an R-node.** In this case the graph G_1 constructed by the algorithm is the original block B, since all skeleton edges are expanded. Moreover, Π_1 is an ec-embedding of B, and $p_{\ell r} = p_{\ell\ell}$ and $p_{rr} = p_{r\ell}$ are ec-insertion paths in B with respect to Π_1. We do not need to consider the case where the embedding of the skeleton can be mirrored, since this will not yield a shorter path. Hence, p_1 is either $p_{\ell r}$ or p_{rr} and thus an ec-insertion path in B with respect to Π_1.

Assume now that $k > 1$. For $i = 1, \dots, k$, we denote with H_i the pertinent graph of μ_i, with r_i the reference edge of μ_i in H_i, and, for $1 < i$, with e_i the edge in *skeleton*(μ_i) whose pertinent node is μ_{i-1}. Recall that $s_i \in \{\ell, r\}$ is the side of H_i where the computed insertion

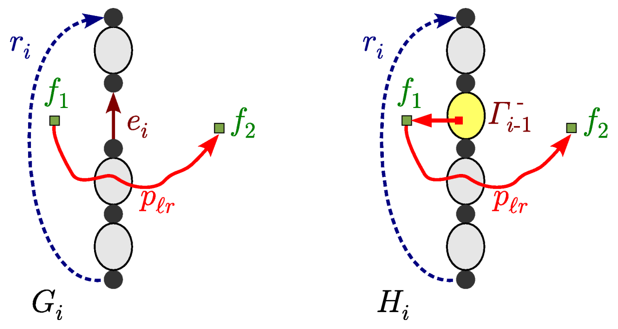


Figure 6.5: Proof of Lemma 6.5; μ_i is an S-node, $L_v = R_w = \{f_1\}$, $R_v = L_w = \{f_2\}$.

path shall leave. We show by induction over i that, for $1 \leq i < k$, there is an embedding Γ_i of H_i such that $p_1 + \cdots + p_i$ is an ec-insertion path leaving H_i at side s_i. The embeddings $\Gamma_1, \ldots, \Gamma_{k-1}$ are iteratively constructed during the proof. For our convenience, we denote with Γ_i^- the embedding of $H_i - r_i$ induced by Γ_i.

$i = 1$. Consider the different types for node μ_1:

(1) μ_1 **is a P-node.** This case does not apply, since μ_2 is not an allocation node of v.

(2) μ_1 **is an S- or an R-node.** In this case, $G_1 = H_1$, and p_1 is a path leaving either Π_1 or the mirror image of Π_1 to side s_1. Hence, we set Γ_1 to Π_1 or its mirror image, respectively.

$1 < i < k$. We distinguish again between the types of μ_i.

(1) μ_i **is a P-node.** In this case, $p_i = \epsilon$, i.e., no further edges need to be crossed. The embedding Γ_i is obtained as follows. If $s_i = \ell$, we permute the edges in *skeleton*(μ_i) such that e_i is to the right of r_i; otherwise, we permute the edges such that e_i is to the left of r_i. Then, we replace e_i by Γ^-_{i-1}, and the remaining edges $e \neq r_i$ in *skeleton*(μ_i) by an arbitrary embedding of *expansion*(e); see Figure 6.4(b).

(2) μ_i **is an S- or an R-node.** In this case, p_i is either $p_{s_{i-1}s_i}$ or $p_{\bar{s}_{i-1}\bar{s}_i}$; the latter case corresponds to mirroring the embedding of *skeleton*(μ_i) before.

We first consider the case in which p_i is set to $p_{s_{i-1}s_i}$, i.e., an ec-insertion path in the embedding Π_i that starts in a face at side s_{i-1} of e_i and ends in a face at side $\bar{s}_i$ of edge r_i. We obtain Γ_i by replacing e_i by Γ_i^- in Π_i; see Figure 6.5. Since the ec-insertion path $p_1 + \cdots + p_{i-1}$ leaves Γ_i^- to the side s_{i-1}, $p_1 + \cdots + p_i$ is an ec-insertion path leaving Γ_i to the side s_i.

Finally, assume that $p_i = p_{\bar{s}_{i-1}\bar{s}_i}$. Let $\tilde{\Pi}_i$ be the embedding that we obtain by first mirroring the embedding of *skeleton*(μ_i) and then expanding and embedding each skeleton edge not representing v or w as before. We observe that p_i is an ec-insertion path in $\tilde{\Pi}_i$ that starts in a face at side s_{i-1} of e_i and ends in a face at side $\bar{s}_i$ of edge r_i; see Figure 6.6. With the same argumentation as above, we obtain Γ_i by replacing e_i with Γ^-_{i-1} in $\tilde{\Pi}_i$.

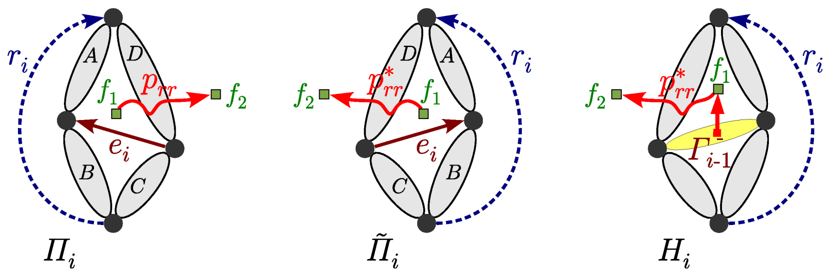


Figure 6.6: Proof of Lemma 6.5; μ_i is an R-node, $R_v = \{f_1\}$, $L_w = \{f_2\}$.

To conclude the proof, we consider the node μ_k. We know that μ_k is either an S- or an R-node, and we may assume that $p_k = p_{s_{i-1}s_i}$, since $p_{\ell r} = p_{\ell\ell}$ and $p_{rr} = p_{r\ell}$ holds for $i = k$. Hence, p_k is an ec-insertion path in Π_k that starts in a face at side s_{i-1} of e_i and ends in a face adjacent to the copy of w in G_k. We obtain Π by replacing e_k with Γ^-_{k-1} in Π_k, and thus $p_1 + \cdots + p_k$ is an ec-insertion path for v and w in Π. □

Lemma 6.6. *Let Π' be an arbitrary ec-embedding of B and let p' be a shortest ec-insertion path for v and w in B with respect to Π'. Then $|p'| \geq |p_1 + \cdots + p_k|$.*

Proof. Let G_i, S_i, and s_i be as defined in OPTIMALECBLOCKINSERTER, and let λ^i_ℓ and λ^i_r be the value of λ_ℓ and λ_r, respectively, after the i-th iteration of the for-loop. For $i = 1, \ldots, k$, we denote with H_i the pertinent graph of μ_i. Observe, that Π' induces embeddings of G_i and S_i. Accordingly, we denote the induced embedding of G_i with Π'_i, and of S_i with Σ'_i.

Since p' is a shortest ec-insertion path, it does not visit a face twice. Therefore, we can subdivide p' into $p' = p'_1 + \cdots + p'_k$ such that p'_i contains exactly the edges of p' that are in G_i, for $1 \leq i \leq k$. This follows from the fact that H_i shares only two vertices with the rest of the graph and p' does not visit a face twice. For $1 \leq i < k$, we denote with $s'_i \in \{\ell, r\}$ the side at which the ec-insertion path $p'_1 + \cdots + p'_i$ leaves H_i in Π'.

We show by induction over i that $\lambda^i_{s'_i} \leq |p'_1 + \cdots + p'_i|$.

$i = 1$. If $k = 1$, then $G_1 = B$ and the proposition follows immediately, so assume $k > 1$. If μ_1 is not an R-node, then $\lambda_{s'_1} = 0$ and the proposition follows immediately. Otherwise, the algorithm also computes the shortest ec-insertion path leaving at side s'_1 in Σ'_1, where the costs of the edges are their traversing costs. Since the traversing costs are independent of the embedding by Lemma 6.4, we get $\lambda^1_{s'_1} \leq |p'_1|$.

$1 < i < k$. Assume now that $\lambda^j_{s'_j} \leq |p'_1 + \cdots + p'_j|$ for $1 \leq j < i$. We distinguish two cases:

(1) μ_i **is a P-node.** In this case, we have $s'_{i-1} = s'_i$, since $p_i + \cdots + p_k$ does not contain an edge of H_{i-1}. This yields

$$\lambda^i_{s'_i} = \lambda^{i-1}_{s'_{i-1}} \leq |p'_1 + \cdots + p'_{i-1}| \leq |p'_1 + \cdots + p'_i|.$$

(2) μ_i **is an S- or an R-node.** Observe that p'_i is an ec-insertion path in Π'_i starting in the face at side s'_i of the edge representing v and ending in a face at side $\bar{s}'_{i+1}$ of the edge representing w if $i < k$, or a face adjacent to w otherwise. This implies

```
function OptimalEcInserter(ec-expansion $G$, vertex $v$, vertex $w$)
    Compute the block-vertex tree $\mathcal{B}$ of $G$
    Find the path $v, B_1, c_1, \dots, B_{k-1}, c_{k-1}, B_k, w$ from $v$ to $w$ in $\mathcal{B}$.
    for $i := 1, \dots, k$ do
        let $x_i$ and $y_i$ be the representatives of $v$ and $w$ in $B_i$
        $p_i :=$ OptimalEcBlockInserter$(B_i, x_i, y_i)$
    end for
    return $p_1 + \cdots + p_k$
end function
```

Algorithm 6.4: Computation of an optimal ec-insertion path.

an ec-insertion path in Σ_i', where the costs of a skeleton edge are its traversing costs. On the other hand, the algorithm computes a shortest ec-insertion path in Σ_i', since the traversing costs of a skeleton edge are independent of the embedding by Lemma 6.4. Thus, we get $\lambda_{s_i'} - \lambda_{s_{i-1}'} \leq |p_i'|$, and hence

$$\lambda_{s_i'} \leq \lambda_{s_{i-1}'} + |p_i| \leq |p_1' + \cdots + p_i'|.$$

Finally, we get $|p_1 + \cdots + p_k| = \lambda^i_{s_i'} \leq |p'|$ and the lemma holds. $\square$

Theorem 6.3. *Let $B = (V, E)$ be a block of K and let v and w be two distinct vertices of B. Then, function* OptimalEcBlockInserter *computes an optimal ec-insertion path for v and w in B in time O$(|E|)$.*

Proof. The correctness and optimality of the algorithm follows from Lemma 6.5 and Lemma 6.6. Constructing the SPQR-tree and embedding the skeleton graphs takes time O$(|E|)$; see Gutwenger and Mutzel [2001], Hopcroft and Tarjan [1973a], Chiba et al. [1985]. Let $G_i = (V_i, E_i)$ be the graph considered in each iteration of the for-loop. Then, each iteration takes time $O(|E_i|)$, since ShortestEcInsPath takes only time linear in the size of G_i by applying BFS. Moreover, the set E_i consists of some edges E_i' of G plus at most two virtual edges (the representatives of v and w). Thus, $|E_1| + \cdots + |E_k| = $ O$(|E|)$, and hence we get a total running time of O$(|E|)$. $\square$

6.4.4 Generalization to Connected Graphs

The edge insertion algorithm can easily be generalized to connected graphs by using the same technique as in Gutwenger et al. [2005] for the unconstrained case; see Algorithm 6.4. For each block B_i on the path from v to w in the block-vertex tree $\mathcal{B}$ of G, we compute the optimal ec-edge insertion path p_i between the representatives of v and w with a corresponding ec-planar embedding Π_i. Then, we concatenate these ec-edge insertion paths building the optimal ec-edge insertion path for v and w.

The correctness proof in Gutwenger et al. [2005] uses induction over the number of blocks on the path from v to w in $\mathcal{B}$. We briefly recall this proof. Let $B_1, \dots, B_k$ be the blocks on this path and let H_i be the union of the blocks B_1 to B_i. Let Π_i be an embedding of B_i such that p_i is an optimal edge insertion path for the representatives x_i and y_i in B_i with respect to Π_i. Let Ψ_i denote the concatenation $p_1 + \cdots + p_i$.

An embedding Γ_i for H_i with an optimal edge insertion path Ψ_i can be iteratively constructed by combining the embedding Γ_{i-1} for H_{i-1} and the embedding Π_i for block B_i. Both y_{i-1} and x_i denote the same vertex in G and there exist optimal edge insertion paths Ψ_{i-1} for v_1 and y_{i-1} as well as p_i for x_i and y_i. Therefore there is a face $f \in \Gamma_{i-1}$ that contains y_{i-1} and either v_1 if Ψ_{i-1} is empty or the last edge in Ψ_{i-1}. Similarly, there is a face $f' \in \Pi_i$ that contains x_i and either y_i if p_i is empty or the first edge in p_i. We can directly concatenate the two paths if both faces coincide. This can be achieved by choosing f as the external face of Γ_{i-1} and placing this embedding of H_{i-1} into face f' of Π_i. Then Ψ_i is an optimal ec-insertion path for v_1 and y_i in H_i with respect to Γ_i.

We need to show that—under the presence of embedding constraints—ec-planarity is preserved, i.e., Γ_k is still an ec-planar embedding. The only critical aspect in each step is the selection of f as the external face; but this does not change the clockwise order of the edges around the vertices of G. Furthermore, we ensure in the computation of the ec-edge insertion paths p_i that we do not cross any expansion edges. Hence, we know that the paths Ψ_{i-1} and p_i do not start or end in a face containing a hub. Therefore, the ec-planarity conditions are still fulfilled and Γ_k is ec-planar.

It is obvious that $p_1 + \cdots + p_k$ is an ec-edge insertion path for v and w with respect to an embedding Π that results from inserting the remaining blocks not contained in H_k (as shown in the proof of Lemma 6.3) into Γ_k. The length of the computed ec-edge insertion path is obviously minimal, since a shorter path would imply that there exists a shorter path within a block. The block-vertex tree of a graph can be constructed in linear time and the running time of OPTIMALECBLOCKINSERTER(B_i, x_i, y_i) is linear in the size of the block B_i (Theorem 6.3), thus yielding linear running time for OPTIMALECINSERTER.

Together with Lemma 6.1, we obtain the following result:

Theorem 6.4. *Let $G = (V, E)$ be a graph with embedding constraints C and $e = (v, w) \in E$ such that $G - e$ is ec-planar. Then, we can compute an optimal ec-edge insertion path for (v, w) in $G - e$ in $O(|V| + |E|)$ time.*

6.5 Conclusion and Future Work

We introduced a flexible concept of embedding constraints which allows us to model a wide range of constraints on the order of edges incident to a vertex. We presented a linear time algorithm for testing ec-planarity, as well as a characterization of all possible ec-embeddings. The latter is particularly important for developing algorithms that optimize over the set of all ec-planar embeddings. We showed that optimal edge insertion can still be performed in linear time when embedding constraints have to be respected. In order to devise practically successful graph drawing algorithms, the following problems should be considered in the future:

- Develop faster algorithms for finding ec-planar subgraphs.
- Solve the so-called *orientation problem* for orthogonal graph drawing, e.g., allow us to fix some edges to attach only at the top side of a rectangular vertex. The problem arises

when angles are assigned at each vertex between adjacent edges to fix the assignment to the vertex's sides, e.g., in network flow based drawing approaches. The vertices then need to be oriented such that the edges that are assigned to the same sides at different vertices are aligned.

- In some applications, only a subset of the edges is subject to embedding constraints at a vertex v, i.e., some edges can attach at arbitrary positions. Hence, we wish to extend the concept of embedding constraints for so-called *free edges* that are not contained in the tree T_v.

7. CLUSTERED GRAPH DRAWINGS

The Milky Way is nothing else but a mass of innumerable stars planted together in clusters.

GALILEO GALILEI

Clusters are a means to organize data in groups of similar objects, where the organization structure is typically given by a hierarchical inclusion relation. A suitable clustering fosters the understanding of the underlying data and helps to gain better insight in the inherent structure. As we have seen in Section 3.2, use of clustering is common in several application domains, were it is used to allow to identify families of closely related objects, e.g., to classify proteins. Graphs that represent the corresponding data can also be organized in such a way by a clustering of the vertices, where the inclusion hierarchy is described by a tree whose leaves are the graph's vertices. The resulting structure can be visualized by grouping the vertices of a cluster in a common, usually rectangular, region. In addition, the cluster structure can also be used for navigation purposes, e.g., by only visualizing representatives for each cluster, such that also large graphs can be handled. Navigation within this structure then has to be realized transparently such that the user keeps his orientation, see Section 9.1 for a corresponding application. Drawing clustered graphs is not only a prevalent problem in practical applications of graph drawing, but also in graph theory, since the occurring graph theoretical problems are in particular challenging, even in simplified special cases.

There are several matters associated with clustered graphs that are interesting from a graph drawing point of view. A major topic is the algorithmic treatment of planarity issues, as the grouping of vertices in a common region might lead to a large number of unnecessary crossings when no special care is taken. A further issue is the navigation in clustered data and the interaction with the respective representations, including the representation and handling of the clusters and the cluster structure in practical tools. Besides the use of a corresponding layout to highlight cluster affiliation, also graphical attributes can be used to distinguish clusters. In this chapter, we will focus on clusters as constraints in the layout restricting sense, and discuss the resulting aspects regarding planarity issues. Section 7.1 introduces the basic concepts of clustered drawings, and presents the main existing approaches and results. In Section 7.2 we describe an ILP-based approach to compute the maximum c-planar subgraph, which includes the first practical c-planarity test for non-c-connected graphs. Section 7.3 describes a branch-and-price approach for c-planarity testing. A major result in clustered planarity research was the development of a linear time c-planarity testing algorithm for c-connected clustered graphs by Dahlhaus [1998]. The description from the original publication is difficult to read, as it omits some explanations and proofs that are helpful for the understanding, and also uses the unusual characterization of a graph decomposition by graph grammars. We give a new, hopefully clearer, description that uses

SPQR-trees for decomposition in Section 7.4.

7.1 Problem Definition, Notations, and Previous Work

Feng et al. [1995a] introduced the clustered graph model as follows:

Definition 7.1 (Clustered Graph)**.** A *clustered graph* $C = (G, T, r)$ – or *c-graph* for short – consists of

- an (undirected) graph $G = (V, E)$,
- a rooted tree T, and
- an inner vertex r of T (the root) such that
- the set of leaves of T is exactly V.

Each vertex ν of T represents a *cluster* of the vertices $V(\nu)$ of G that are leaves of the subtree rooted at ν, and $V(\nu)$ induces a subgraph $G(\nu)$ of G. We call the inner tree vertices of T *nodes* to distinguish inner tree vertices from the vertices of the underlying graph. As the ancestor relation of the nodes in T describes an inclusion relation between the clusters, T is called *inclusion tree* of C, G is called the *underlying graph* of C. A *sub-cluster* of node ν is a node in the subtree rooted at ν. An edge between a vertex of $V(\nu)$ and a vertex of $V - V(\nu)$ is called *outgoing* edge and is *incident* to ν.

Suppose that $C_1 = (G_1, T_1)$ and $C_2 = (G_2, T_2)$ are two clustered graphs, T_1 is a subtree of T_2, and for each node ν of T_1, $G_1(\nu)$ is a subgraph of $G_2(\nu)$. Then we say C_1 is a *sub-clustered graph* of C_2, and C_2 is a *super-clustered graph* of C_1.

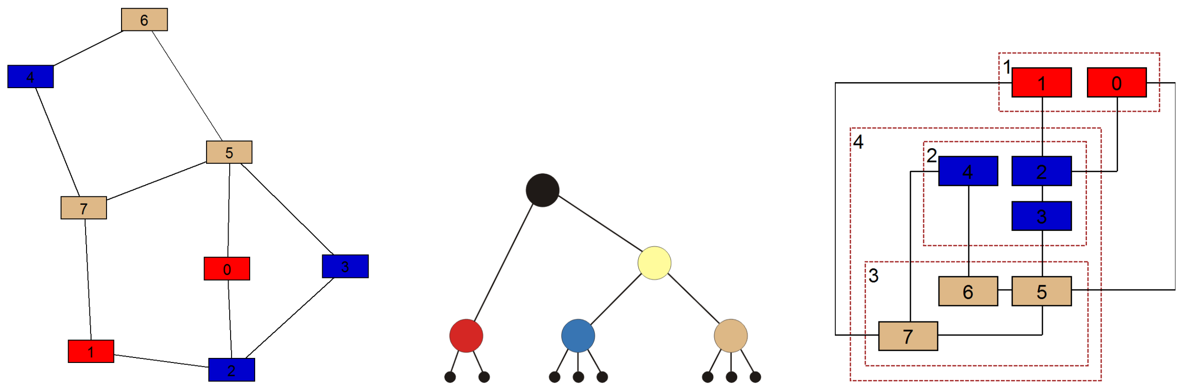
A drawing of a clustered graph should reflect the inclusion relation such that the subgraph induced by a cluster ν is drawn inside a simple closed region that does not contain parts of clusters that are not descendants of ν. Feng et al. [1995a] define a clustered graph drawing as follows:

Definition 7.2 (Clustered Graph Drawing)**.** A $drawing$ of a clustered graph $C = (G, T)$ is a representation of the clustered graph in the plane. Each vertex of G is represented by a point. Each edge of G is represented by a simple curve between the drawings of its endpoints. For each node ν of T, the cluster $V(\nu)$ is drawn as a simple closed region $R(\nu)$ defined by a simple closed curve in the plane such that:

(1) the regions for all sub-clusters of $R(\nu)$ are completely contained in the interior of $R(\nu)$;

(2) the regions for all other clusters are completely contained in the exterior of $R(\nu)$;

(3) if there is an edge e between two vertices of $V(\nu)$ then the drawing of e is completely contained in $R(\nu)$.



(a) Force-directed Drawing (b) Cluster Tree (c) Clustered Orthogonal Drawing

Figure 7.1: A force-directed layout of the underlying graph of a clustered graph C, the cluster tree, and a corresponding clustered orthogonal drawing. The blue and the red cluster are not connected, therefore C is not c-connected.

Cornelsen and Wagner [2006] decouple the pure representation of a clustered graph C, which might just be given by a drawing of the underlying graph, and the fact that the inclusion relation is reflected by the drawing by defining an *inclusion representation* of the inclusion tree T as follows: Each node of T is represented by a simple closed region bounded by a simple closed curve. The drawing of a node or leaf ν of T is contained in the interior of the region representing a node μ if and only if μ is contained in the path from ν to r in T. The drawings of two nodes ν and μ are disjoint if neither μ is contained in the path from ν to r nor ν is contained in the path from μ to r in T.

Defining the inclusion representation has the advantage that we can call a drawing of the underlying graph a drawing of the c-graph, and denote specific clustered drawings (reflecting the inclusion relation) by saying that the drawing is an inclusion representation. Otherwise, we would have to distinguish between a drawing of C and a representation of C in the plane which does not need to be a drawing of the clustered graph. We therefore adopt this terminology in the following. Obviously, the definitions only cover the layout aspects of the representation, and in practical applications also graphical attributes like shape and color might be important for the distinction of clusters. Even though it is sufficient for the theoretical treatment of clusters to restrict a vertex representation to a point, in most practical applications vertices have graphical representations that are more complex than a single point. Practical cluster drawing approaches therefore need to treat the vertices accordingly.

Several algorithms for drawing c-graphs have been proposed, including force-directed methods [Eades and Huang, 2000], planar straight-line 'convex drawings' where each cluster is represented by a convex polygon [Eades et al., 1999a, Nagamochi and Kuroya, 2007, Hong and Nagamochi, 2010], and orthogonal grid style drawings [Eades et al., 1999b, Di Battista et al., 2001]. In Feng et al. [1995b] an exponential lower area bound for straight-line convex planar drawings of c-graphs is shown.

A cluster ν is called a *connected cluster* if $G(\nu)$ is connected. A c-graph C is *c-connected* if each cluster ν of C is connected. Note that authors dealing with clustered graphs sometimes call such a c-graph *connected*, whereas we will reserve the term *connected* to the underlying graph. A *chunk* is the set of vertices of a connected component of a cluster ν, i.e., in case ν is connected, it has a single chunk, otherwise it consists of a set of chunks $\nu_1, \dots, \nu_k, k > 1$.

A c-graph $C = (G, T)$ is *completely connected* if and only if for each inner node ν of T both $G(\nu)$ and $G(V \setminus V(\nu))$ are connected. See Figure 7.10.

Cornelsen and Wagner state the equivalence of the following characterizations, showing the relation of complete connectivity with the selection of the inclusion tree root:

When (G, T, r) is a clustered graph, the following statements are equivalent:

- (G, T, r) is completely connected
- (G, T, ν) is c-connected for every inner node ν of T
- (G, T, ν) is completely connected for every inner node ν of T.

These statements can easily be seen as follows: Clearly, when we change the root of the inclusion tree from r to ν, we change the situation in the nodes on the path from ν to r. When (G, T, r) is completely connected, for each node μ along the path its former parent cluster now contains the former complement of $G(\mu)$ as induced subgraph. The old root node, which did not have a parent node, also has only the empty graph as complement. For all other nodes the induced subgraph stays the same.

When (G, T, r) was completely connected before the root change, all the complements and therefore now all induced subgraphs have to be connected, and the unchanged nodes also still have connected induced subgraphs. For the reverse direction, if (G, T, ν) is c-connected for every inner node ν of T, we can see that by changing the root for each ν, also each complement in the original setting with r as root needs to be connected to give c-connectivity, therefore (G, T, r) is completely connected. With a similar argument we can see that (G, T, ν) has to be completely connected for each ν.

We can observe that there are several aspects of a layout that allow to identify groups of vertices that belong together: Vertices of a distinct group should be drawn 'close' together, and parts of a cluster should not be separated by parts of other clusters, e.g., enclosed in cycles. Note that the definition of a clustered drawing assures the latter condition, but vertices do need to be drawn close together in such a drawing, see Figure 7.2. In Figure 7.1(a), the blue vertices are visually separated by the brown and red vertices, such that only the vertex color allows to identify vertices that belong together.

7.1.1 Clustered Planarity

When dealing with general graphs, the concept of planarity plays a central role and the minimization of crossings in a drawing of a (possibly non-planar) graph is an important task. Similar to this, clustered graphs should also be drawn with the smallest number of edge crossings possible. Due to the characteristics of clustered graph drawings, there are additional issues to consider and the planarity concept needs to be extended accordingly.

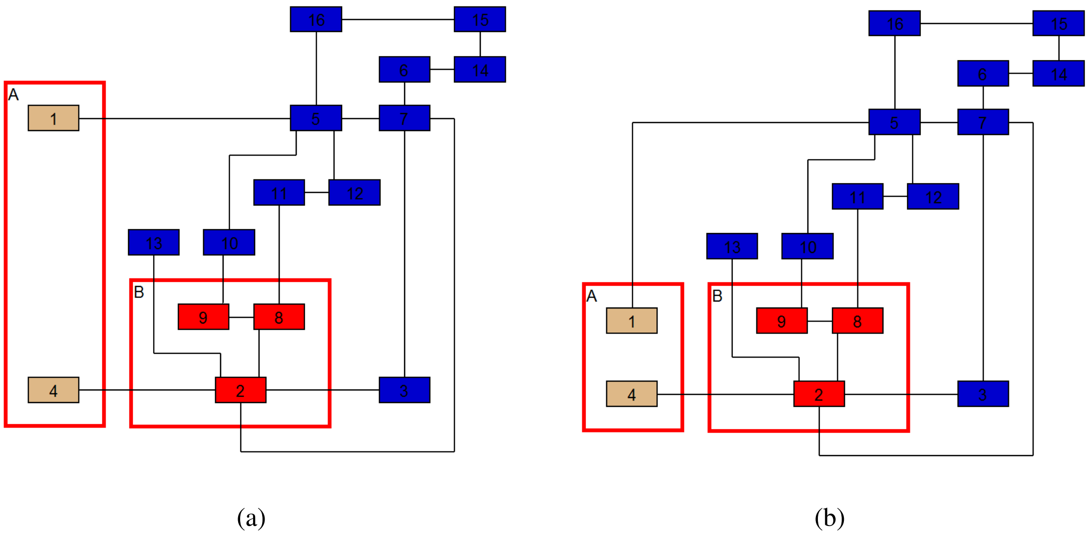


Figure 7.2: (a) Clustered orthogonal drawing showing large inner-cluster distance between vertices in cluster A. Clearly the large inner-cluster distance is not in conformance with grouping perception rules. With a simple update, we can improve upon the automatic layout by inserting an additional bend (b).

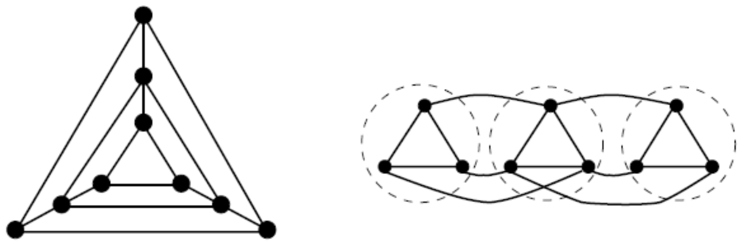


Figure 7.3: Classical example by Feng et al. [1995a] of a planar, but non-c-planar c-graph, where clusters are defined by the triangle structures. In the left drawing, regions of the clusters are not disjoint, in the right drawing the regions are drawn disjoint, but then an edge crossing has to occur.

The drawings of an edge e and a region $R(\nu)$ in a drawing of a clustered graph have an *edge-region crossing* if the drawing of e crosses the boundary of R more than once. Even when no edge crossings occur, a drawing might contain such edge-region crossings either because an edge might cross through a non-connected cluster, or because an edge might leave the cluster region first and then enter it again.

The following definition extends the standard planarity concept accordingly:

Definition 7.3 (C-Planar Drawing (Cornelsen and Wagner [2006]))**.** A *c-planar drawing* of a clustered graph (G, T, r) consists of

- a planar drawing of the underlying graph G
- an inclusion representation of the rooted tree (T, r) such that

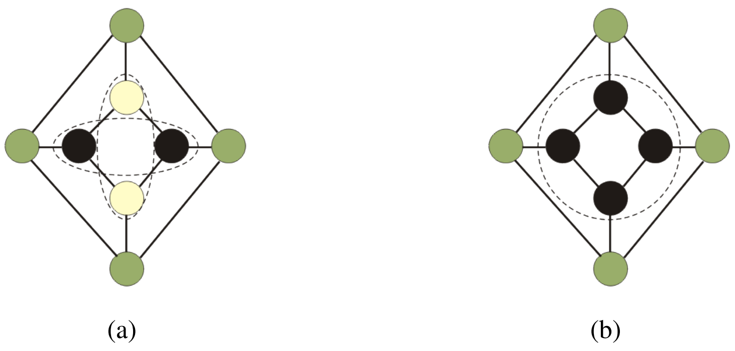


Figure 7.4: Non-c-planar drawings of two clustered graphs. Colors and boundaries represent clusters. (a) Green vertices are children of the root cluster and two cluster regions cross, but none of them is enclosing the other. As the regions are not disjoint, the drawing is not an inclusion representation. Making both clusters connected by adding an edge results in a non-planar graph, i.e., the c-graph is non-c-planar. (b) Green and black vertices mark sibling cluster in the cluster tree, and the black cluster is completely enclosed by the green one. The c-graph, however, is c-planar.

- each edge crosses the boundary of the drawing of a node of T at most once.

The three conditions forbid crossings between edges, between edges and regions, as well as between regions. A clustered graph is called *c-planar*, if it admits a c-planar drawing. Figure 7.1(c) shows an example of a c-planar drawing, note that the embedding of the graph in 7.1(a) also allows a c-planar drawing although the blue cluster seems to be split.

See Figure 7.4 for non-c-planar drawings of clustered graphs even though they are planar and do not contain edge-region crossings.

The complexity of recognizing c-planarity for general clustered graphs is a long-standing open problem in graph drawing. Though it is yet unknown if this problem is solvable in polynomial time, several special classes of clustered graphs can be recognized in polynomial time. We can distinguish the following main cases:

Completely connected. If the graph is completely connected, then c-planarity is equal to simple planarity of the underlying graph [Cornelsen and Wagner, 2006].

C-connected. When each cluster is connected, c-planarity can be tested in linear time [Dahlhaus, 1998]. See also Section 7.4.

Non-c-connected. There are several problem cases known where c-graphs can be tested in polynomial time. They restrict the graph structure, the clustering structure, or its interplay with the structure of the underlying graph. Note that also the underlying graph does not need to be connected in this case. Some problems concerning the concepts of 'face' and 'embedding' occur when clusters do not have to be connected. If G is planar and connected, we can speak of the embedding of G, giving us corresponding faces. But the embedding of G still may allow some freedom in the order of the outgoing edges around a cluster. Figure 7.5 shows an example where alternative routings of an edge are given that cross a cluster boundary. Even though the c-graph is c-planar, given a fixed embedding of the underlying graph that allows a c-planar drawing, c-planarity depends on the edge order at the cluster boundary.

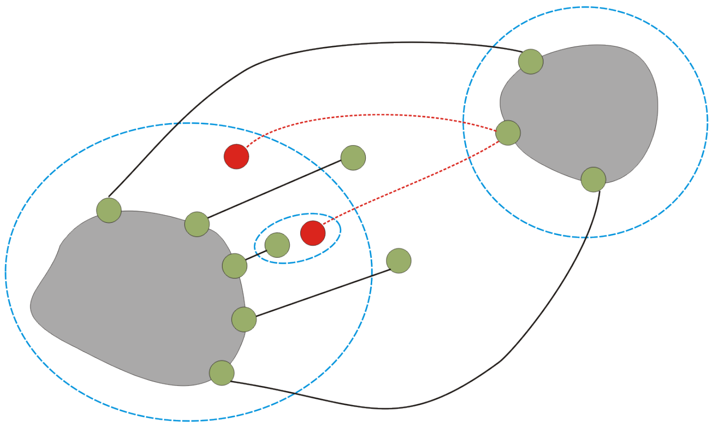


Figure 7.5: A non-c-connected, but connected clustered graph for which the embedding of the underlying graph still allows c-planar and non-c-planar drawings by changing the order of the edges incident to a cluster. Alternative drawings for an edge are drawn in red.

In case C is c-planar, an embedding of C is therefore given by an embedding of G plus the circular ordering of edges crossing the boundary of the region of each cluster. A planar embedding of G is called a c-planar embedding of G and C if it can be extended to an embedding of C that allows a c-planar drawing.
We summarize the known results on c-planarity testing and embedding in the following.

Previous Results Feng et al. [1995a] gave the first c-planarity test for c-connected c-graphs that runs in $O(n^2)$ time and is based on PQ-trees. Using the embedding algorithm by Chiba et al. [1985], their algorithm is also able to return a c-planar embedding within the same time bounds in case the c-graph is c-planar.

A related problem, where the input is given by a replacement system, was studied by Lengauer [1986] already in 1986. Lengauer gave a linear time algorithm concerning the input size of the problem, which is not necessarily in the order of the size of the graph. The first linear time algorithm for testing if a c-connected clustered graph is c-planar was presented by Dahlhaus [1998]. Re-interpretations of Dahlhaus' graph grammar based algorithm in terms of SPQR trees can be found in Dahlhaus et al. [2006], Cortese et al. [2006a,b].

In the following years, there have been many attempts to advance on the way to a solution for the generic problem, including non-c-connected graphs, by examining restricted classes of clustered graphs, either by a specific characterization of the clustering and graph structure allowed, or by the size of the constituting components.

The class of so called *almost c-connected* c-graphs was introduced by Gutwenger et al. [2002]. They consider clustered graphs where either all nodes in the cluster tree T that correspond to non-c-connected clusters, lie on the same path in T starting at the root, or where alternatively for each non-c-connected cluster its super-cluster and all its siblings in T are connected. They use a connectivity augmentation approach based on graph decomposition

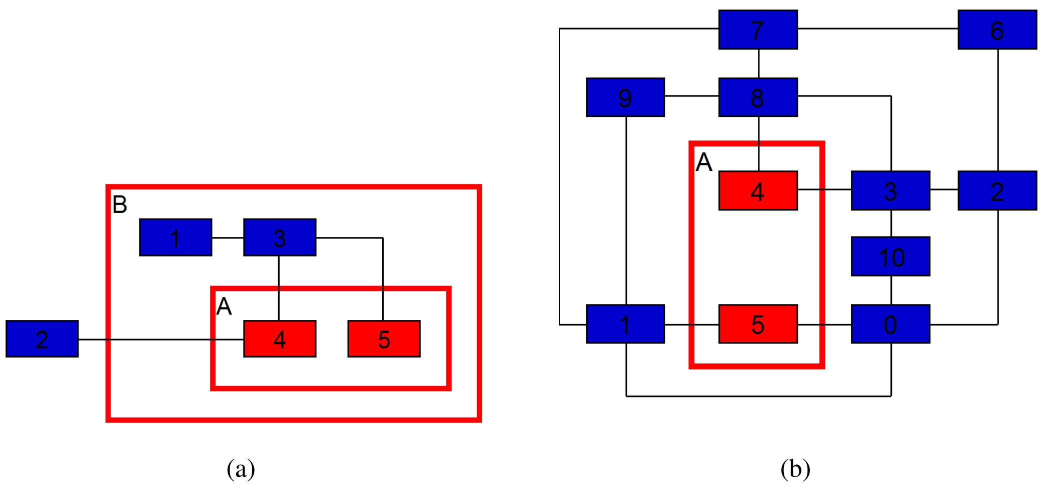


Figure 7.6: (a) Example of a c-graph that is both extrovert and almost c-connected. (b) A c-planar drawing of a c-graph where cluster A is enclosed within a cycle of the root cluster.

with $SPQR$-trees. A quadratic time algorithm for c-planarity testing and embedding is given that extends the test of Feng et al. [1995a] with a *subgraph induced planar connectivity augmentation*. For a graph $G = (V, E)$, such an augmentation is defined on a set of vertices $W \subset V$ as a set of additional edges F connecting vertices in W such that $G' = (V, E \cup F)$ is planar and the subgraph of G' induced by W is connected. The algorithm first splits non-c-connected clusters to achieve a c-connected c-graph that can be tested for c-planarity. In the positive case the algorithm tries to compute a planar augmentation in the resulting representative graph for each non-connected cluster. If none exists, the c-graph is not c-planar.

An algorithm for testing c-planarity of so called *extrovert clustered graphs*, where disconnected clusters need to fulfill a special connectivity property, was given by Goodrich et al. [2005]. A chunk ν_i of a non-connected cluster ν is called *extrovert*, if the parent cluster μ of ν is connected and ν_i has at least one outgoing edge that also leaves the parent cluster μ. If each chunk of a non-connected cluster is extrovert, the cluster is called extrovert, and a c-graph is called extrovert when all of its clusters are either connected or extrovert. A testing and embedding algorithm for this class of c-graphs was presented with $O(n^3)$ running time. Later, this result was improved by Sun and Zhang [2008], who presented an $O(n^2)$ testing algorithm for extrovert c-graphs. As a special case, the classes of extrovert and almost c-connected c-graphs include c-connected clustered graphs. Neither of the two classes however includes the other.

Cortese et al. [2004] considered c-planarity testing of c-graphs where the underlying graph is a cycle. They give a efficient algorithm for graphs that at each level of the inclusion tree have a cycle structure, showing that in this case c-planarity can be tested in time $O(Ln)$, where L is the depth of the cluster tree.

Cornelsen and Wagner [2006] studied completely connected clustered graphs. They show that such a clustered graph is c-planar if and only if the underlying graph is planar; see Section 7.1.2. Even though explicitly stated for the first time in Cornelsen and Wagner [2006], this result already follows from a result in Gutwenger et al. [2002]. We use this relation in a practical c-planarity testing approach by checking all possible extensions of the c-graph to a completely connected planar graph, as described in Section 7.2.2.

When a c-graph is found not to be c-planar, we would like to draw the graph with the smallest number of edge crossings possible. Di Battista et al. [2001] described a planarization based method for crossing minimization: In a first phase a small number of edges is deleted from the clustered graph $C = (G, T)$ such that a c-planar graph C' is left. In the second phase, the deleted edges are re-inserted successively into a c-planar embedding of C' such that only a small number of crossings is produced. This planarization step needs $O(mx + m^2c + mnc)$ time, where n, m, and c are the number of vertices, edges, and clusters of C, respectively, and x is the number of crossings. The authors also adapt the topology-shape-metrics approach to derive an orthogonal drawing algorithm for clustered graphs. Our result from Section 7.2.2 can be used as the first phase within this approach, as it computes a maximum c-planar subgraph.

In several publications, the complexity of the problem is reduced by limiting the size of certain cluster related properties of the input graph, including the number of connected components per cluster [Jelínek et al., 2008a], the cluster size [Jelínková et al., 2009] for special classes of graphs, or the number of outgoing edges per cluster [Jelínek et al., 2008b]. Jelínková et al. [2009] investigated small clusters in Eulerian graphs and cycles. They restrict the clusters to have size at most three and achieve a testing algorithm with running time of $O(|\mathcal{C}|^2 + n)$ for triconnected planar graphs, and of $O(|\mathcal{C}|^3 + n)$ for cycles, where $\mathcal{C}$ is the number of clusters. They generalize the latter result for a special class of Eulerian graphs, so-called k-Rib-Eulerian graphs, leading to a running time of $O(3^k \cdot k \cdot n^3)$. The constant k defines the number of vertices of a triconnected planar graph that is used to obtain the k-Rib-Eulerian graph by multiplying and then subdividing some edges. Di Battista and Frati [2009] discussed the problem of clustered planarity in the setting of a fixed embedding, and give a first result for c-graphs with a *flat* cluster hierarchy, i.e., the length of any path from the root to a leaf of T is two. They show that for embedded flat clustered graphs with at most five vertices per face c-planarity can be tested in linear space and time.

Didimo et al. [2008] investigate c-graphs with overlapping clusters (*oc-graphs*), replacing the inclusion tree by an acyclic digraph such that clusters may share vertices. They investigate the overlapping clustered planarity problem (*oc-planarity*) and give polynomial time oc-planarity tests for several classes of oc-graphs.

Angelini et al. [2010b] introduced the *split-c-planarity problem* that, given a clustered graph C, asks if C can be made c-planar by performing at most k cluster splits (that is, replacing a cluster by two clusters). Testing c-planarity is the special case of split-c-planarity with $k = 0$, i.e., when no splits are allowed. They show that split-c-planarity is NP-complete even for $k = 1$. In Gutwenger et al. [2002], simply the maximum number of splittings is performed to achieve cluster connectivity during the testing algorithm.

There are also approaches for the clustering of edges, where the goal is to increase the readability of given drawings by computing channels for the bundled routing of edges [Cui et al., 2008, Gansner and Koren, 2007, Nachmanson et al., 2011]. Such approaches try to avoid visual clutter in drawings of dense graphs, caused by the large number of edges and their mutual crossings. The vertex positions are already given as input, whereas the clustering of the edges (in contrast to the vertex clustering discussed here) is not input or based on underlying semantics, but computed purely based on geometric properties of the given drawing.

7.1.2 Characterizations of C-Planar Graphs

The characterizations of this section give insight into the relations between graph structure, inclusion relation and c-planarity for several classes of c-graphs and are the base for the existing c-planarity testing algorithms. Even though some characterizations only consider c-connected c-graphs, fundamental results show that non-c-connected c-graphs can be augmented without losing the c-planarity property, and the construction of such an augmentation is part of several algorithms. In the following, let $C = (G, T)$ be a clustered graph.

In case that G is not even connected, more problems arise as we are not allowed to treat the different connected components separately. Components that cross cluster boundaries might influence the c-planarity property and we have to consider them for testing, as can be seen in the simple example of a $K_{3,3}$ structure where edges are replaced by clusters such that connected components contain only a single edge; see Figure 7.12. A trivial case occurs when a connected component cc is completely contained in a single cluster c, i.e., does not cross a cluster boundary. We can then safely treat cc separately for testing, and also insert a drawing of it within c's region in a drawing of $C \backslash cc$.

Feng et al. [1995a] gave the following characterization of c-planar c-graphs:

Theorem 7.1 (External face [Feng et al., 1995a]). *A c-connected clustered graph $C = (G, T)$ is c-planar if and only if graph G is planar and there exists a planar drawing D of G, such that for each node ν of T, all the vertices and edges of $G - G(\nu)$ are in the external face of the drawing of $G(\nu)$.*

This characterization is used in their testing algorithm for c-connected c-graphs by checking for each cluster in the inclusion tree in a bottom-up fashion whether it can be drawn planar with the outgoing edges in the external face of the drawing. The cluster then is replaced in G by a representative graph that models all possible orderings of the outgoing edges for further testing. The main idea is that the connectivity of each subgraph induced by a cluster ν can be used to construct a biconnected graph that includes all outgoing edges of ν, connected to a new virtual vertex; see Figure 7.7. Then the planarity test based on PQ-trees can be used for the resulting graph and, in the case of planarity, the subgraph can be replaced by a construction based on the resulting PQ-tree that reflects the possible orderings of the outgoing edges around the boundary of ν. The PQ-tree data structure can be used to check c-planarity of an n-vertex, connected clustered graph in $O(n^2)$ time. Feng et al. [1995a] also describe an extension that computes a c-planar embedding of the given graph in the same asymptotic time.

From the fact that there exist planar drawings of trees and every planar drawing of a tree has only a single face, it is immediately clear from Theorem 7.1 that this simple case of a c-connected graph is c-planar:

Lemma 7.1. *A c-connected clustered graph whose underlying graph is a tree, is c-planar.*

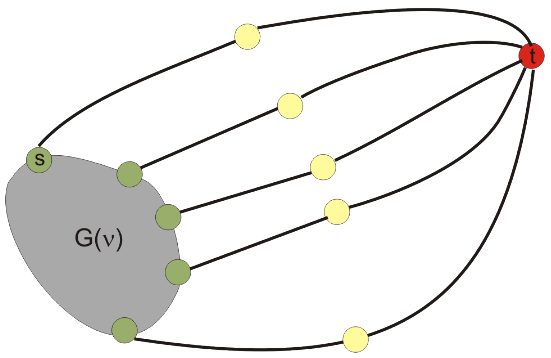


Figure 7.7: Illustration of the testgraph for cluster induced subgraphs that is used as input for the PQ-tree algorithm. The virtual vertex that connects all outgoing edges is colored red.

The following result states that we can extend each c-planar clustered graph to a c-connected c-planar graph, i.e., no aspect of c-planarity is hidden in the non-connectivity of the clusters.

Theorem 7.2 (Sub-clustered graph [Feng et al., 1995a]). *A clustered graph $C = (G,T)$ is c-planar if and only if it is a sub-clustered graph of a c-connected and c-planar clustered graph.*

Even though this motivates the intuitive idea of connecting each cluster before testing for c-planarity, there are no characterizations known for the general case how to do this without possibly destroying the c-planarity property. An interesting question therefore is how to characterize the edges that would cause non-c-planarity, or alternatively the edges that are safe for addition. Gutwenger et al. [2002] give a characterization for a restricted subproblem, using a *subgraph induced planar augmentation* to augment cluster induced subgraphs in a planar way to achieve connectivity. Note that if $G(\nu)$ is connected, the boundary of its external face in any planar drawing of $G(\nu)$ consists of a connected cycle.

Clearly, each c-planar drawing of a c-graph C corresponds to at least one c-connected c-planar extension of C, as we can connect components of a non-connected cluster in a tree-like fashion within the region $R(C)$.

The problem of c-planar augmentation for a cluster in the general case is closely related to the problem of characterizing all possible orderings of the outgoing edges. This has been done by Feng et al. for connected clustered graphs using PQ-trees and wheel graphs, and in the case of almost c-planar c-graphs a planar augmentation suffices. In the general case however, a planar augmentation might restrict the edge order around the vertex.

Given a fixed c-planar embedding of a c-graph, we can even further augment it to obtain a triangulation without destroying c-planarity:

Lemma 7.2 (Triangulation [Jünger et al., 2002]). *Let $C = (G,T)$ be a c-planar embedded c-connected c-graph. C can be triangulated without loosing the c-planar embedding in linear time concerning the number of vertices of G. The triangulated underlying graph G does not contain multiple edges.*

Non-adjacent vertices in a triangulation do not share a common face, therefore vertices from different chunks of a non-connected cluster will lie in separate faces, making it impossible due to the triangulation to achieve a c-planar drawing. This is similar to the situation depicted in Figure 7.10(b). Cornelsen and Wagner [2006] explicitly state an interesting consequence:

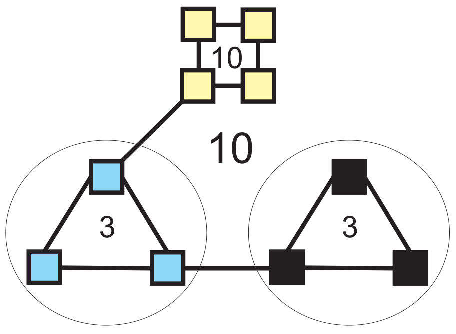
Theorem 7.3 (Complete augmentation [Cornelsen and Wagner, 2006])**.** *Every c-planar clustered graph is a subgraph of a c-planar completely connected clustered graph.*

Lemma 7.3 ([Feng, 1997])**.** *Let $C = (G, T)$ be a clustered graph where G is a triangulation. Then C is c-planar only if C is c-connected.*

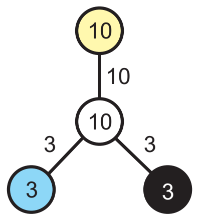
Obviously it is not necessary for vertices of a cluster ν to lie on the external face f of the drawing of $G(\nu)$, as long as they do not have outgoing edges, which would lead to crossings. All that is necessary is that they are not enclosed by cycles from other clusters that separate them from f. Analogously, the vertices of ν on the external face of ν's drawing do not need to be incident to the external face of the parent cluster μ, the drawing of ν might be completely enclosed by a cycle of μ. It suffices that the chunks of ν are adjacent to the external face with respect to their level in the inclusion tree, i.e., when all edges passing clusters that lie on the path between ν and the root of the cluster tree are removed. For instance, vertices of a cluster that is a child of the root cluster do not need to lie on the external face of G, as long as there is a curve to the external face crossing only edges that have at least one end-vertex belonging to the root cluster. For example, the cluster vertices of cluster A in Figure 7.6(b) are enclosed in inner faces of the root cluster,

Dahlhaus [1998] gives a corresponding characterization for clustered planar embeddings of c-connected c-graphs based on a weight criterion on the dual graph. The weight function models the inclusion relation such that an enclosing cycle in the embedding can be detected. Observe that in a c-planar embedding Γ of a c-connected clustered graph $C = (G, T)$, clusters appear as connected areas without holes. That is, for a cluster ν there may not be any parts of a cluster ν', that is not a descendant of ν in the inclusion tree, enclosed by a cycle K consisting only of edges in ν. If such an inclusion occurs in a planar embedding Γ of G, i.e., if a hole exists and therefore Γ is not c-planar, there has to be some path connecting vertices in ν' to vertices on K since G is connected. Due to the c-connectivity of the clustered graph, at least one of the edges on the path has to pass through the least common ancestor of ν and ν' in the cluster tree. This property is exploited to characterize correct embeddings of c-planar graphs.

The weight function is defined as follows: Let $weight(c)$ of a cluster c be the number of vertices in c, i.e., the weight may only increase on a path from a leaf to the root of the cluster tree T. For any edge $e = (v, w)$, let $w_{lca}(e)$ be the weight of the smallest cluster $c \in C$ that contains v and w. This cluster is just the least common ancestor of v and w in T, which is the highest cluster whose region a drawing of e has to touch in a c-planar drawing. The weight of a face f, *weight(f)*, is the maximum weight of an edge that belongs to the face cycle of f. A face f therefore has as weight the maximum weight of the clusters whose regions share some part with f, i.e., the weight of the least common ancestor of the vertices on the face boundary in T. We interpret the weights of the faces and edges of G as weights of the vertices and edges of the dual graph G' of Γ and G. A hole in a cluster then corresponds to a



(a) A clustered planar embedding



(b) and its dual graph

Figure 7.8: A c-connected clustered graph with the face weights of the Dahlhaus criterion, circles denote clusters.

cycle of edges with weight at most $i-1$ that encloses a subgraph with weight at least i. See Figure 7.9(a).

A clustered planar embedding now can be characterized as follows: For each i, let F_i be the set of faces f of Γ with $weight(f) \geq i$ and E_i be the set of edges e of G with $w_{lca}(e) \geq i$. Let E'_i be the set of edges in G' corresponding to the edges in E_i. Then (F_i, E'_i) represents a subgraph of G'. See Figure 7.8 for an example. Note that there cannot be any edges $e = (f_1, f_2)$ with weight k, where $k > weight(f_1)$ or $k > weight(f_2)$.

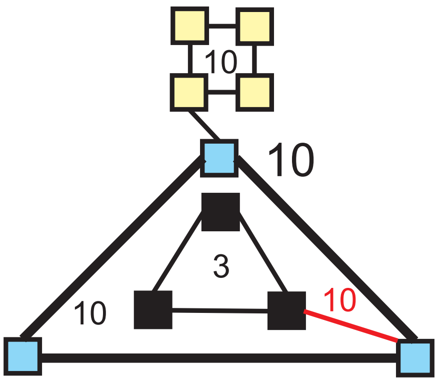
Theorem 7.4 (Clustered Planar Embeddings [Dahlhaus, 1998]). *A planar embedding Γ of $G = (V, E)$ is a clustered planar embedding of G and $C = (G, T)$ if and only if for each i with $F_i \neq \emptyset$, (F_i, E'_i) is a connected subgraph of the dual graph of Γ and a face of maximum weight is selected as external face.*

The theorem's condition guarantees that the drawing of each cluster does not have a hole, ensuring an inclusion representation. As Γ is a planar embedding of G, and due to the c-connectivity, an edge cannot cross the boundary of a cluster twice, Γ allows a clustered planar drawing and is therefore a clustered planar embedding.

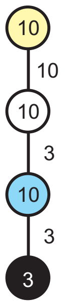
The result of Lemma 7.1 can easily be shown using Dahlhaus' criterion, as there is only a single face in the dual graph of a tree's embedding. See Section 7.4 for a detailed description of a linear time c-planarity testing algorithm for c-connected graphs based on this criterion. Whereas c-connected clustered graphs with tree structure can easily be seen to be c-planar, this is not true for the arbitrary case. We can simply extend the drawing of the clustered $K_{3,3}$ structure with single edge components of Figure 7.12(b) to have an underlying tree, and the resulting c-graph is not c-planar. We can even restrict the extension to a path, resulting in a connected c-graph with non-connected clusters that is not c-planar; see Figure 7.12(a).

Cornelsen and Wagner [2006] showed that in case the c-graph is completely connected, c-planarity can even be reduced to simple planarity:

Theorem 7.5 (Completely connected [Cornelsen and Wagner, 2006]). *Let $C = (G, T)$ be a completely connected clustered graph. Then*



(a) Non c-planar embedding, thick edges denote a cluster enclosing cycle



(b) and its dual graph

Figure 7.9: The same graph as in Fig. 7.8 with an embedding where one child cluster of the root cluster is enclosed by the other. The embedding is therefore not clustered planar, the dual graph is not connected for weights ≥ 10.

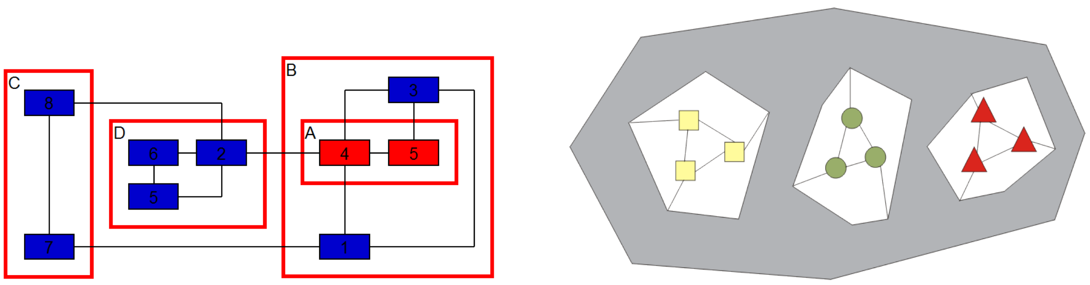
$$C \textit{ is c-planar} \Leftrightarrow \; G \textit{ is planar}$$

When each cluster of C is connected, in a planar drawing of the underlying graph a cluster c might not be split into several components by external edges, as otherwise a crossing would occur. When also the complement of c has to be connected, i.e., C is completely connected, it has to be drawn in the same face of a planar drawing of c, which can be chosen as the external face. The c-planarity then immediately follows from Theorem 7.1. Theorems 7.1 and 7.5 are related to the criteria of Dahlhaus for the same reason: It is not allowed to enclose a cluster c or parts of it by a cycle of edges belonging to another cluster c', if c' is not an ancestor of c in the inclusion tree, as otherwise there would be a hole in c'. If c and $G \setminus c$ are connected, in any planar drawing of G, $G \setminus c$ lies in a single face of the drawing of $G(c)$. In order to avoid holes in clusters in a c-planar drawing, this face has to be the external face.

Goodrich et al. [2005] call an embedding Γ of a cluster or a chunk of a cluster ν *simple* when it is in conformance with the criterion of Theorem 7.1, i.e., outgoing edges are drawn in the external face of ν. A cluster ν with k chunks is called *connectable* when $k-1$ edges ('bridges') can be drawn in Γ such that the chunks are connected to a single component without introducing any edge crossings (a planar augmentation). Sibling non-connected clusters in an extrovert c-graph are called *conflicting* in Γ, if each of them is connectable, but there is no way to connect all of them at the same time without introducing crossings.

Theorem 7.6 (Extrovert [Goodrich et al., 2005]). *An extrovert c-graph $C = (G, T)$ is c-planar $\iff$ G is planar and there exists a planar embedding $D(G)$ of G such that, each chunk of a cluster is simple; each extrovert cluster is connectable; and no sibling extrovert clusters conflict.*

Obviously Theorem 7.6 just combines the augmentation property of Theorem 7.2 with the c-planarity criterion of Theorem 7.1. Similar to Gutwenger et al. [2002], the parent cluster of a disconnected cluster for extrovert c-graphs always has to be connected.



(a) Completely connected

(b) Not completely connected

Figure 7.10: Two clustered graphs. (a) The graph is completely connected. (b) Schema of a not completely connected c-graph C that has a large triconnected component (gray) which constitutes a child cluster c of the root cluster r. Three other child clusters c_1, c_2, c_3 of r are inscribed in different faces of the drawing of c. C cannot be drawn c-planar as c_1, c_2, c_3 cannot be drawn in the same face of a planar drawing of the underlying graph of c, and any attempt to achieve connectivity for the complement of c leads to a non-planar graph.

The following result shows that in case a c-graph is c-planar, we can even change the root of the inclusion tree T to any cluster ν of T and the resulting c-graph is still c-planar.

Lemma 7.4 ([Cornelsen and Wagner, 2006]). *Let (G, T, r) be a c-planar clustered graph and ν a node of T. Then (G, T, ν) is c-planar.*

Given a completely connected c-planar c-graph $C = (G, T, r)$ together with a fixed embedding for G, they investigate the interplay between external face and tree root r. Fixing the external face, they give a characterization which node can be chosen as root and vice versa. Obviously, a node ν is rootable for a fixed external face f_0, due to complete connectivity, if it has at least one vertex incident to f_0. The vertex cannot be separated by an unconnected cluster, i.e. there is no enclosing cycle and if there is no such vertex, the cluster would be enclosed in a cycle such that the embedding of G is not c-planar.

They also show that for any planar embedding and external face a cluster can be chosen as root node such that the embedding is c-planar.

In case we have a non c-planar c-graph, we would like to identify a large subgraph that is c-planar, e.g., for use in a planarization approach. From Lemma 7.5 it immediately follows that we do not need to destroy c-connectivity to compute a c-planar subgraph.

Lemma 7.5 (C-planar subgraph [Di Battista et al., 2001]). *Let $C = (G, T)$ be a connected clustered graph. Then there exists a c-planar connected clustered subgraph of C.*

Such a subgraph can be constructed by a bottom-up visit of the cluster tree that combines spanning trees for the already processed clusters [Di Battista et al., 2001].

Clearly, also a maximum c-planar subgraph has to be c-connected, otherwise we could simply augment it with additional edges in contradiction to the maximality.

The criterion of Feng et al. [1995a] uses PQ-trees to represent the admissible embeddings of cluster-induced subgraphs, as it is necessary to consider them for c-planarity checking. If we would try to simplify c-planarity testing by collapsing a cluster c, such that a vertex representative is inserted into the graph instead, the restrictions on the order of the outgoing edges are relaxed that may be imposed by the cluster's topology. In order to achieve a planar drawing, an order of the edges around the cluster might be necessary that leads to non-planarity when the cluster is fully expanded. For standard planarity testing there are several simple reduction techniques allowed that could reduce the input instance size. Vertices of degree 1 do not matter for planarity and also chains of degree 2 vertices can be replaced by a single edge without changing the planarity status. These reduction can also be applied for c-planarity testing as long as clusters have to be connected. However, in case the clusters do not need to be connected, vertices of degree 1 or 2 can be part of a non-connected cluster, and removing them might impact the c-planarity status.

Cortese et al. [2004] consider c-planarity testing on clustered graphs where the underlying graph and the induced cluster structure on each level of the inclusion tree is a cycle. That is, the define the graph G^l the graph whose vertices are the nodes of T at distance l from the root, where an edge (μ, ν) exists if and only if an edge of G exists incident to both μ and ν. To simulate the closed regions containing the clusters, they add edges to the cycle to make the cluster connected (which is always possible without losing the c-planarity property). Such a set of edges is called *saturator*.

Theorem 7.7 (Saturator [Cortese et al., 2004]). *A c-planar 3-cluster cycle admits a saturator that is the collection of three disjoint paths.*

Theorem 7.8 ([Cortese et al., 2004]). *Given an n-vertex clustered graph C(G,T), such that T has depth L and, for $l > 0$, G^l is a cycle, there exists an algorithm to test if C is c-planar in $O(Ln)$ time.*

For the restricted case of a cycle graph G, Cortese et al. [2004] gave the most simple example of a situation where no c-planar augmentation is possible, a *3-cluster cycle* with six vertices, where representatives of the clusters appear alternately on the cycle.

Note that analyzing the faces is in general not sufficient, as we do not know if we need to connect components of a cluster in a particular face. In case we just connect all components in all faces, we might introduce cycles that enclose parts of other clusters, leading to a non-c-planar drawing. However some connections might be unnecessary, as we might already have created a connecting path between these components due to other augmentation edges.

As we have seen in Theorems 7.1 and 7.4, when each cluster of a clustered graph is connected, we can use the standard definition of a graph embedding to test c-planarity.

When clusters are allowed to be disconnected, the situation is far more difficult. A close relation between vertices may simply be modeled by clusters instead of edges, as in the example of Figure 7.12, where the structure of the non-planar graph $K_{3,3}$ is modeled by a c-graph. Clearly, this c-graph is non-c-planar even though the underlying graph is a set of planar connected components consisting of single edges. In this specific case, there is a unique augmentation to a c-connected graph, which would make the underlying graph non-planar, but for arbitrary graphs it might not be obvious if there is a c-planarity preserving augmentation; see Figure 7.11. In order to search for violations of c-planarity, the simple check for an enclosing cycle in an embedding of the underlying graph is also not possible in such a case. Instead of an edge, also two non-adjacent vertices that belong to the same,

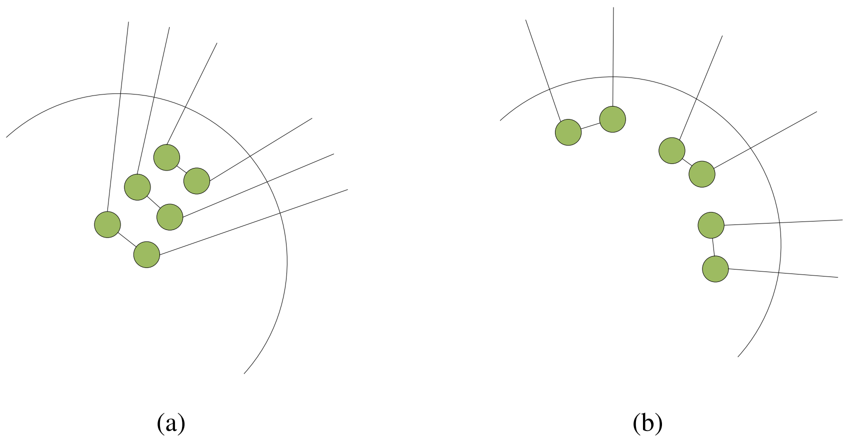


Figure 7.11: Illustration of non-c-connected case: There are multiple positions allowed for components with outgoing edges to be nested in other components. If we use a simple augmentation to make the cluster connected, we might restrict the possible order of edges around the cluster, leading to a non-c-planar c-connected graph.

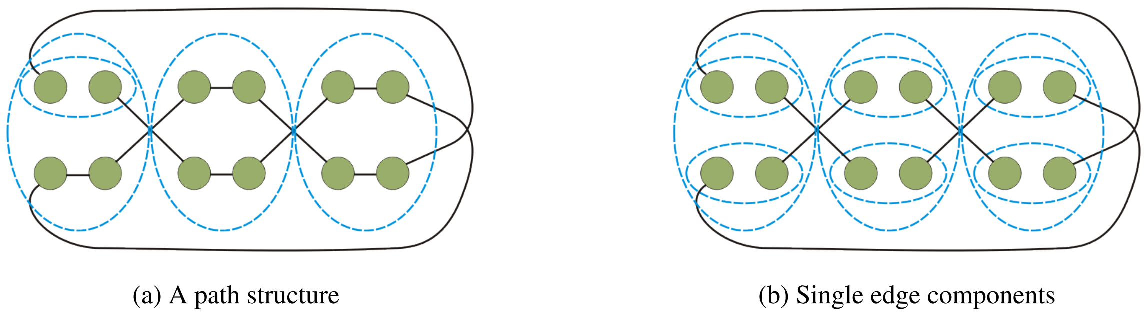


Figure 7.12: $K_{3,3}$ structure modeled by replacing some edges by clusters.

non-connected cluster may be part of the enclosure. This does not lead to a cycle of smaller wight in the dual graph, and we can therefore not apply the criterion of Theorem 7.4. When the underlying graph is not connected, a subgraph may even be inscribed into an enclosing cycle without losing dual graph connectivity (even though the dual graph is not formally defined in this case, due to the drawing we have a topology that can by modeled by a graph that corresponds to the dual graph in the connected case). For example, if we remove the red edge in Figure 7.9, the 'dual' graph is connected even though the drawing is still not c-planar, as the weight of the blue triangle face is three instead of 10. As the induced subgraph is not connected, we can also not apply the modeling of Feng et al. [1995a] that uses the PQ-tree structure.

7.1.3 Compound Graphs

Sugiyama and Misue [1991] introduced a graph model that is more powerful than the clustered graph model, as the inclusion relation is directly modeled on the vertices instead of introducing clusters. This allows to have edges between groups of vertices by connecting the enclosing vertices.

Compound graphs have several important applications, for example in the static analysis of program code. In interprocedural control flow diagrams that represent the control flow in programs (see, e.g., Sander [1999]), the vertices of each module should be grouped together. Also in hardware design compound graphs can be used to model circuit diagrams, where the elements of each module should be surrounded by rectangles indicating its borders.

Formally, Sugiyama and Misue define a *compound digraph* as a triple $D = (V, E, F)$, where V is a set of vertices, E a set of directed inclusion edges such that u includes $v \iff (u, v) \in E$, and F a set as adjacency edges as in the standard graph concept. Sugiyama and Misue give a hierarchical drawing algorithm for a restricted class of compound graphs, where the inclusion relations form a tree, and there are no adjacency relations between a pair of vertices (u, v) such that u is the ancestor of v or vice versa.

Compound-Compound edges can easily be modeled in a clustered graph scenario in the case were adjacency inbetween ancestors in the inclusion tree is forbidden (e.g., to avoid cycles in hierarchical graph drawing, where by convention edges are drawn from the bottom of one node to the top of the other). Then, the edges always leave a compound ν to a compound on the path from ν to the root of the inclusion tree, and can be modeled by a connecting edge between otherwise isolated vertices inside the corresponding compounds. Edges between a node ν and its descendant τ would stay inside of the cluster that is higher in the hierarchy. Such an edge therefore would not model the restriction that the region of τ in a drawing must not be completely enclosed as we need to route an edge from ν's boundary to τ's boundary. But in this case the compound adjacency edge can be modeled by an edge connecting a new vertex in the τ compound and a new vertex in the parent of the ν compound.

Here we can see an interesting connection to our ec-embedding constraint model: If we allow free edges at vertices with embedding constraints, then ec-planarity testing is similar to a special case of compound graph planarity. In order to check ec-planarity with free edges, we first detach these edges from their end vertices, introducing a new end vertex for each detached end. Note that an edge $e = (u, v)$ may be a free edge at one or both end vertices and has to be detached only from a vertex v when it is free at v. We then conduct the ec-expansion for the constrained edges as usual. To guarantee that a free edge can be routed such that it connects the replacement regions of u and v, we enclose the new end vertices with the respective replacement structure by a (non-connected) cluster. Hence, the free edges are modeled similar to the compound connection edges above, and c-planarity testing for the resulting c-graph of depth 1 can be used within the ec-planarity test for the original graph.

7.2 The Maximum C-Planar Subgraph Problem

In this section, we introduce an approach to solve the *general* clustered planarity problem using integer linear programming (ILP) techniques. We give an ILP formulation that also includes the natural generalization of c-planarity testing—the *Maximum C-Planar Subgraph Problem* (MCPSP)—and solve this ILP with a branch-and-cut algorithm. Given a clustered graph $C = (G, T)$, the maximum c-planar subgraph problem asks for a c-planar subgraph of C with a maximum number of edges. Our computational results show that this approach is already successful for many clustered graphs of small to medium size and thus can be the foundation of a practically efficient algorithm that integrates further sophisticated ILP techniques. Our solution to the MCPSP is also suitable for the first phase of the planarization approach for c-graphs.

The formulation uses the result by Cornelsen and Wagner [2006] which shows that each c-planar clustered graph is a subgraph of a c-planar completely connected clustered graph. We present a branch-and-cut algorithm based on the LP-relaxation of our ILP formulation. In order to experimentally evaluate our approach, we introduce a benchmark set of clustered graphs.

This section is organized as follows. After stating the problem description, Section 7.2.2 describes our ILP formulation for the MCPSP and a branch-and-cut algorithm based on the LP-relaxation of the ILP. An experimental evaluation of this algorithm with new benchmark instances is given in Section 7.2.3. Section 7.2.4 concludes with an outlook on future work.

7.2.1 Problem Description and Related Work

Definition 7.4 (Maximum C-planar Subgraph Problem)**.** Given a clustered graph $C = (G = (V, E), T)$ find a c-planar clustered graph $C' = (G' = (V, E'), T)$ with $E' \subseteq E$ such that E' has maximum cardinality.

This problem generalizes the problem of c-planarity testing, for which no polynomial time algorithm is known, as well as the *Maximum Planar Subgraph Problem* (MPSP), which is known to be NP-hard [Liu and Geldmacher, 1977], in a natural way. Obviously, the MCPSP is therefore also NP-hard. Each instance graph G of the MPSP can easily be transformed to an instance of the MCPSP by constructing a clustered graph $C = (G, T)$ where T contains only the root cluster.

Our approach is based on two results of Cornelsen and Wagner [2006]: Theorem 7.5 and 7.3. These results show that it is sufficient to test the graph G for planarity, if the clustered graph (G, T) is completely connected, and that every c-planar clustered graph is a subgraph of a c-planar completely connected clustered graph. This fact can be used to test a given clustered graph for c-planarity by augmenting it (in the correct way) to a completely connected graph. In order to compute the maximum c-planar subgraph, we therefore may have to remove edges from the input graph as well as to add edges to achieve a completely connected c-planar graph.

7.2.2 Solving the MCPSP via an ILP and Branch-and-Cut

Throughout this section let $C = (G = (V, E), T)$ be the given clustered graph with edge set E, and let F denote the complement of E (i.e., the potential edges for augmentation). For a cluster ν in C let $E(\nu)$ denote the edge set induced by the vertices $V(\nu)$ in cluster ν, and let $E(\bar{\nu})$ denote the edge set induced by the vertices in $V(\bar{\nu}) = V \setminus V(\nu)$.

Our approach is based on the idea of constructing a completely connected clustered subgraph $C' = (G' = (V, E'), T)$ of the (complete) clustered graph $C^* = ((V, E \cup F), T)$

such that G' is planar. By Theorem 7.5 such a graph C' induces a c-planar subgraph $C[E'] = ((V, E' \cap E), T)$ of C. In order to guarantee that $C[E']$ contains a maximum number of edges from E—and thus solves the MCPSP—we assign costs to the edges such that the original edges from E are preferred over all the edges from F. If C is c-planar, then E' will contain all edges of C. Hence, our algorithm is also able to test c-planarity of a general clustered graph.

Corollary 7.1. *$C[E']$ is a maximum c-planar subgraph of C if and only if it is the largest subgraph with the property that there exists a completely connected clustered graph C' such that (a) $C[E']$ is its subgraph and (b) the underlying graph of C' is planar. If $C[E'] = C$, C is c-planar.*

The ILP Formulation

Based on the above theoretical results, we can formulate an integer linear program to model the MCPSP as follows. We define the variables

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{7.1}$$
$$y_f \in \{0,1\} \qquad \forall f \in F \tag{7.2}$$

which are 1 if the corresponding edge is contained in the solution graph, and 0 otherwise. As objective function we use:

$$\max \sum_{e \in E} x_e - \varepsilon \sum_{f \in F} y_f \tag{7.3}$$

We want to maximize the number of original edges in the solution and use as few augmenting edges as possible to obtain a completely connected clustered graph. In order for the latter criterion to not interfere with the main optimization goal, we restrict its influence by multiplying the corresponding variables by a small constant ε: we know that we will not require more than $3n$ non-edges due to Euler's formula, otherwise the graph will not be planar. Choosing $\varepsilon := \frac{0.1}{3n}$ therefore guarantees that the second term in 7.3 does not grow larger than 0.1.

We have two sets of constraints: the first set guarantees that the clustered graph C' induced by the edge set E' is completely connected. The second set ensures planarity of C'.

Connectivity Constraints. A completely connected clustered graph $C' = (G' = (V, E'), T)$ has the property, that for each cluster ν, the induced graphs $G'[V(\nu)]$ and $G'[V(\bar{\nu})]$ are connected. A graph $G = (V, E)$ is connected if and only if it contains a path between all pairs of its vertices. A *cut set* $(W|A)$ with $W \subseteq V$ and $A \subseteq E$ in the graph $G = (V, E)$ is defined as the set of edges restricted to A which are incident to exactly one vertex of W. In a connected graph $G = (V, E)$, we clearly have $|(W|E)| \geq 1$ for any cut set $(W|E)$ with $\emptyset \neq W \subset V$ and $W \neq V$. For a vertex set $N \subseteq V$ in $G = (V, E)$, we define

$\mathcal{P}(N) := \{W \mid \emptyset \subset W \subset N \wedge 2|W| \leq |N|\}$. We can then define the connectedness constraints as:

$$\sum_{e\in(W|E(\nu))} x_e + \sum_{f\in(W|F(\nu))} y_f \geq 1 \qquad \forall\, \nu \in T,\ \forall\, W \in \mathcal{P}(V(\nu)) \tag{7.4}$$

$$\sum_{e\in(W|E(\bar\nu))} x_e + \sum_{f\in(W|F(\bar\nu))} y_f \geq 1 \qquad \forall\, \nu \in T,\ \forall\, W \in \mathcal{P}(V(\bar\nu)) \tag{7.5}$$

While the constraints (7.4) guarantee the c-connectivity, the constraints (7.5) guarantee the connectedness of the cluster complements, and the combination therefore ensures complete connectivity.

Kuratowski Constraints. Based on Theorem 7.5, it is sufficient to achieve planarity of the underlying graph G', if the considered clustered graph $C' = (G', T)$ is completely connected, to guarantee c-planarity. We use Kuratowski constraints as introduced for the maximum planar subgraph problem [Jünger and Mutzel, 1996]) to guarantee planarity of the solution graph. These constraints are based on Kuratowski's theorem [Kuratowski, 1930] which states that a graph is planar if and only if it does not contain a subdivision of K_5 or $K_{3,3}$. We call these subdivisions *Kuratowski subdivisions* and represent them by their edge set. Let H be a non-planar graph, thus containing some Kuratowski subdivision with edge set K. Any planar subgraph H' will not contain all edges of K, as this would be contradictory to the planarity of H'.

Let $\mathcal{K}$ be the set of all Kuratowski subdivisions in the (complete) graph $(V, E \cup F)$. We can formulate the Kuratowski constraints as

$$\sum_{e\in K} x_e + \sum_{e\in K} y_e \leq |K| - 1 \quad \forall K \in \mathcal{K}. \tag{7.6}$$

The constraints model the requirement that we are not allowed to select all edges of any Kuratowski subdivision, i.e., at least one of them will be eliminated for the solution graph. From the discussion above we have the following theorem:

Theorem 7.9. *Let $C = (G = (V, E), T)$ be a clustered graph, $C^* = ((V, E \cup F), T)$, and the variables and constraints as defined above.*

- *Then any feasible solution of the inequality system consisting of the constraints (7.1), (7.2), and (7.4) to (7.6) corresponds to a completely connected c-planar subgraph of C^* and vice versa:*
- *any completely connected c-planar subgraph of C^* fulfills the constraints (7.1), (7.2), and (7.4) to (7.6);*
- *the objective function (7.3) and the constraints (7.1), (7.2), and (7.4) to (7.6) are a correct integer linear programming formulation for the maximum c-planar subgraph problem for $C = (G, T)$. An optimum solution to the MCPSP is given by the edges corresponding to the variables x_e with $x_e = 1$ in the optimum solution of the ILP; and*
- *if $x_e = 1$ for all $e \in E$, then C is c-planar.*

The Branch-And-Cut Algorithm

Both constraint sets contain an exponential number of constraints and hence it is not practicable to generate all constraints in advance. We therefore solve the ILP within a branch-and-cut framework: We start with a small subset of constraints, drop the integrality constraints, and apply cutting-plane algorithms to add additional constraints as required. The problem to identify necessary additional constraints after obtaining a fractional solution, i.e., after solving the partial LP-relaxation, is called the *separation problem*.

Separation Routines. Separating the connectivity constraints can be done in polynomial time by computing minimum cuts in the graph, using the fractional solution as edge capacities. So far no polynomial time algorithm for the Kuratowski constraint separation is known. Hence, we resort to a heuristic separation routine, similar to the ones described in Jünger and Mutzel [1996] for the MPSP. Note that we separate all cut constraints before separating any Kuratowski constraints. Hence we have a connected subgraph, which uses few augmentation edges due to their negative coefficient in the objective function. The separation heuristic rounds the fractional solution to an integer solution. We call the resulting subgraph the *support graph* S. We then try to identify Kuratowski subdivisions in S. Traditional planarity testing algorithms extract a single such subgraph as a witness for non-planarity. To obtain multiple, say k, subdivisions, one has to run the planarity test k times, resulting in an overall runtime of $O(kn)$. We use an extended test algorithm presented in Chimani et al. [2007], which is able to separate multiple different subdivisions in linear time (in the size of the output). For each subdivision K obtained by this procedure, we can then test whether the current fractional solution violates the corresponding constraint induced by K, and add it to the ILP in this case.

Branching and Primal Heuristic. If we have a fractional solution, but cannot find any violated constraints, we have to resort to branching, as in any branch-and-cut algorithm. In this situation, good adaptive LP-based heuristics become crucial, to be able to prune nodes in the branch-and-bound tree early. Our heuristic works as follows: In a first step, we compute a spanning tree recursively for each cluster in a bottom-up scheme on T. Let c be the current cluster, and H the complete graph on the vertices $V(\nu) \cup children(\nu)$, i.e., the set of all vertices and subclusters directly in ν. We compute the minimum spanning tree of H, using the fractional values as the negative weight of the corresponding edges. Merging all these minimum spanning trees, we obtain a spanning tree R for C^*, which is by construction c-connected and c-planar. After sorting the remaining edges based on their fractional value, we iteratively try to add them to R in decreasing order. This can be done in polynomial time, since planarity testing of a c-connected clustered graph is polynomial. We obtain a maximal c-connected, c-planar subgraph R of C^* that also implies a c-planar subgraph of C.

7.2.3 Computational Experiments and Discussion

In this section we report the results of our experimental evaluation and specify the experimental settings. The main intention is to show the feasibility of our approach, as no speed-up techniques like strong heuristics, preprocessing, and column generation are used.

Benchmark Set and Experimental Setting

We created a benchmark set based on the Rome graph library [Di Battista et al., 1997] by generating cluster hierarchies on top of each graph of the library. The library contains planar and non-planar graphs; key properties are shown in Table 7.13(top).

We create a cluster structure by randomly picking vertices in a cluster ν, starting with the root cluster, and after each pick, a random decision is made if a new cluster is generated with the vertices picked so far, up to a maximum number of 9 clusters. We restrict the maximum cluster tree depth to two levels (in addition to the root cluster), the number of edges to 30, and divide the created clustered graphs into two groups depending on the planarity of the underlying graph. The benchmark set can be found online [Klein, 2008].

We implemented our approach as a module within the Open Graph Drawing Framework [OGDF] using the branch-and-cut framework ABACUS [Jünger and Thienel, 2000] with CPLEX as LP-solver and run the approach on each graph in our benchmark set. The experiments were run on a 2.33GHz Intel Xeon machine with 2GB RAM per process, and we set a limit of 30 minutes CPU time and return the best solution if the computation of the optimal solution exceeds the limit.

In addition to solving the MCPSP, we also experimented with a variant where only c-planarity is tested; in this case no maximum c-planar subgraph needs to be computed and subproblems are pruned as soon as their dual bound proves that an original edge would have to be deleted.

Results and Discussion

Figure 7.14 shows the running times required by our approach, relative to the graph size; the table on the bottom of Figure 7.13 summarizes the runtime performance of the instances, depending on the planarity of the underlying graph. We see that restricting the computation to pure c-planarity testing by pruning leads to decreases in the overall average computation time, but does not necessarily speed up the computation for each instance, because subproblems containing the maximum c-planar subgraph may be pruned, which extends the search in the branch tree.

Our main observation is that the performance on most of the test graphs is promising: only 2 non-planar and 17 planar graphs could not be solved within the time limit; the 95%-percentile shows that long running cases are extremely rare. The average running time of the c-connected clustered graphs is below 0.02 seconds, indicating that the ILP performs well on this polynomial time solvable class. We therefore conjecture that the ILP may be useful as a tool when developing c-planarity tests for special graph classes, as the ILP may give hints on the classes' hardness.

7.2.4 Conclusion

We introduced the maximum c-planar subgraph problem and presented an ILP-formulation together with a branch-and-cut approach to solve it to optimality. An experimental evaluation showed the general feasability of our concept. Our approach can be used for small to medium sized clustered graphs with a limited number of clusters. The results show that the

	# inst.	*c-plan.*	*c-con.*	*compl. con.*	*Clusters min*	*Clusters avg*	*Clusters max*	*Vertices max*	*Edges max*
Planar graphs	1815	1494	25	2	3	4	9	29	30
Non-planar graphs	116	0	3	0	3	5.2	9	26	30

	Running time (sec) min	*avg*	*95%*	*max*
P-Sub	0.01	4.9	9.6	1460.9
NP-Sub	0.01	40.9	18.7	1456.5
P-CPl	0.01	4.3	6.1	249.9

Figure 7.13: **(top)** Properties of the benchmark instances. **(bottom)** Average runtime performance of the branch-and-cut algorithm. P-Sub and NP-Sub are running times for solving the MCPSP on the (non-)planar graphs, respectively. P-CPl denotes the runtime for the c-planarity test on the instances with underlying planar graphs. 95% denotes the 95%-percentile.

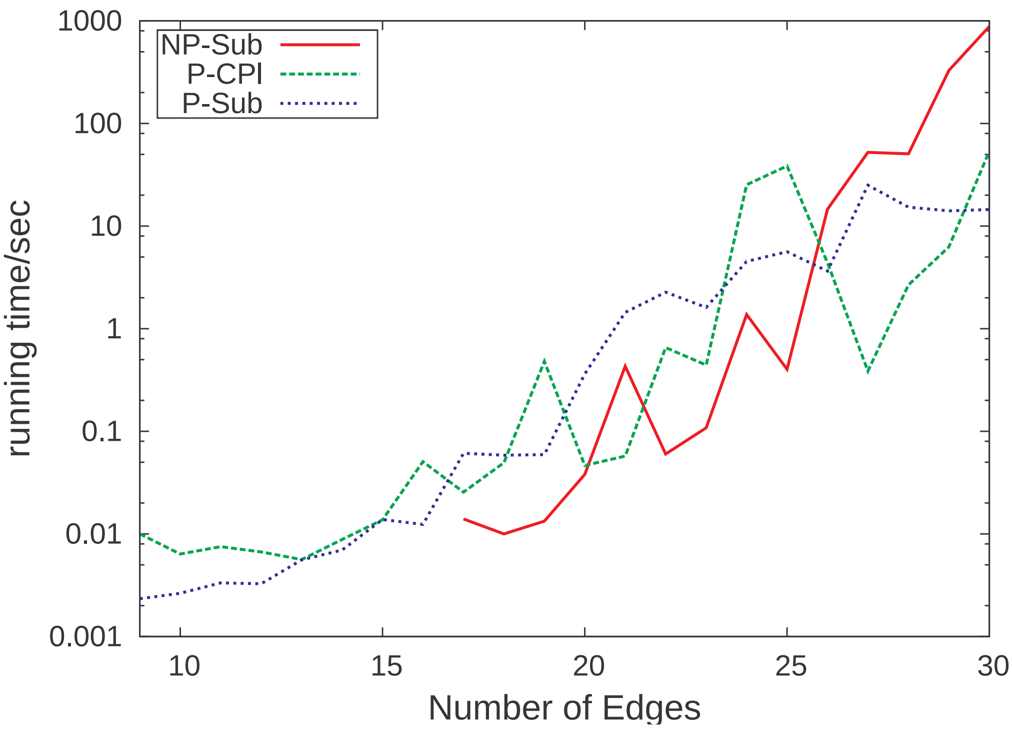


Figure 7.14: Average running time of the branch-and-cut algorithm for the planar (P-Sub) and non-planar (NP-Sub) instances and for the c-planarity test shortcut (P-CPl). Note that the y-axis uses log scale.

number of chunks has a large influence on the efficiency of our branch-and-cut approach. We believe that our branch-and-cut approach can be improved to also cope with harder instances, especially by using stronger heuristics and preprocessing to reduce the search space. The results on the c-connected graphs encourage a closer investigation of the behavior of our approach for other polynomial time solvable classes, especially with regard to the level of c-connectivity of clustered graphs. In addition, it would be worthwhile to analyze the algorithm's behavior with regard to added constraints as well as added and deleted edges for larger benchmark sets. As most of the introduced variables will not be needed during computation, we can use column generation instead of adding all possible variables in advance.

7.3 A Pricing Scheme for Clustered Planarity Testing

In this section we present a branch-cut-and-price approach for the c-planarity testing problem for arbitrary clustered graphs. Our approach is based on the integer linear programming (ILP) formulation from Section 7.2 and improves its results for the case of pure c-planarity testing instead of the search for a maximum c-planar subgraph.

7.3.1 Motivation and Basic Idea

The branch-and-cut approach presented in the previous section allows a flexible computation of a maximum c-planar subgraph. The flexibility is needed to allow deletion of edges that hinder c-planarity of the completely connected extension of the input graph. When only c-planarity testing is needed, we can reduce the flexibility, allowing us to also drastically reduce the search space. Clearly, if the underlying graph G is not planar, the clustered graph C cannot be c-planar. This can be checked by a simple planarity test prior to c-planarity testing, we therefore assume that the underlying graph is planar in the following.

First of all, we do not need to introduce variables for the original edges of the graph, as all of them need to be part of a feasible solution. We only add variables for potential augmentation edges, which might be necessary to achieve complete connectivity. The overall algorithmic idea then is the same as for the MCPSP – try to construct a planar and completely connected extension of the input graph corresponding to the requirements of Theorem 7.5. But in fact we can even do better—we know that typically only a small number of variables will be needed to achieve this, and we therefore apply a pricing scheme, hoping that we will only generate a small subset of the potential variables until we find a solution or decide that none exists. We can apply our knowledge of the clustered graph structure to prefer certain connection edges over others or to decide that an edge might never help us in obtaining a solution.

The description of our approach is organized as follows. We first describe our ILP formulation for the CPP, and then discuss the branch-cut-and-price algorithm based on the LP-relaxation of the ILP. We conclude with an outlook on future work.

7.3.2 An ILP Formulation for the CPP

Throughout this section, let $C = (G = (V, E), T)$ be the given clustered graph with edge set E, and let F denote the complement of E (i.e., the potential augmentation edges not in E). For a cluster ν in C let $E(\nu)$ denote the edge set induced by the vertices $V(\nu)$ in cluster ν, let $E(\bar{\nu})$ denote the edge set induced by the vertices in $V(\bar{\nu}) = V \setminus V(\nu)$, and $F(\nu)$ denote the potential augmentation edges in ν (also called *non-edges* in the following).

If C is c-planar, there exists a (possibly empty) augmenting edge set $F_A \subset F$ such that $C_A = (G_A = (V, E \cup F_A), T)$ is c-planar and completely connected. Our approach is based on the idea of constructing a completely connected clustered subgraph $C' = (G' = (V, E'), T)$ of the (complete) clustered graph $C^* = ((V, E \cup F), T)$ such that $E \subset E'$ and G' is planar. By Theorem 7.5 such a graph C' is c-planar, contains C as a c-planar subgraph and we have $F_A = E' \setminus E$.

If C is not c-planar, then no such augmenting edge set F_A exists and the problem of constructing C' is infeasible. Hence, our algorithm is able to test c-planarity of an arbitrary clustered graph by either finding a valid solution for this *c-planar complete connectivity augmenting edge set problem* or proving infeasibility of our formulation.

Variables and Objective Function. Based on the above theoretical results, we can formulate an integer linear program as follows. We define the variables

$$y_e \in \{0, 1\} \qquad \forall e \in F \tag{7.7}$$

that are 1 if the corresponding non-edge is contained in F_A, and 0 otherwise. Although we have a pure decision problem where the number of additional non-edges (which are necessary to obtain a completely connected clustered graph) is not decisive, we try to keep the set of non-edges as small as possible to minimize the computational effort during the optimization. As our objective function we use:

$$\min \sum_{e \in F} y_e \tag{7.8}$$

Similar to the branch-and-cut approach for the MCPSP, we have two sets of constraints according to the requirements of Theorem 7.5: the first set guarantees that the clustered graph C' induced by the edge set E' is completely connected. The second set uses Kuratowski constraints to guarantee planarity of C'. The difference in the problem formulation allows us, however, to use simplified versions of the constraints compared to the formulation for the MCPSP.

Connectivity Constraints. As for the MCPSP, we try to achieve connectivity by requiring a lower bound on the minimum cut for the graph. Yet, for the pure c-planarity testing, we can restrict the selection of cut sets that need to be considered. As the edges of G have to be part of the solution, we only need to assure connectivity between cluster-induced connected components (chunks) of C. Let $CC(N)$ denote the set of connected components of the subgraph of G induced by some vertex set N, and $V(S_c)$ denote the set of vertices in some $S_c \subset CC(N)$. We define $\mathcal{P}_c(N) := \{S_c \mid \emptyset \subset S_c \subset CC(N) \wedge 2|S_c| \leq |CC(N)|\}$.

We can then define the connectivity constraints as:

$$\sum_{e \in (V(S_c)|F(\nu))} y_e \geq 1 \qquad \forall\, \nu \in T,\ \forall\, S_c \in \mathcal{P}_c(V(\nu)) \tag{7.9}$$

$$\sum_{e \in (V(S_c)|F(\bar{\nu}))} y_e \geq 1 \qquad \forall\, \nu \in T,\ \forall\, S_c \in \mathcal{P}_c(V(\bar{\nu})) \tag{7.10}$$

The constraints (7.9) guarantee the connectivity within the cluster ν, as they require an outgoing edge of each proper subset of chunks that connects the set to the other chunks. The constraints (7.10) ensure the connectivity outside of ν.

Kuratowski Constraints. As our input is planar, only the addition of non-edges may create Kuratowski subdivisions. We use the Kuratowski constraints to forbid at least one of the involved augmentation edges for the edge set K of each Kuratowski subdivision found. Let $\mathcal{K}$ be the set of all Kuratowski subdivisions in the (complete) graph $(V, E \cup F)$ and let $K_F := K \cap F$ be the set of non-edges of a Kuratowski subdivision edge set $K \in \mathcal{K}$. We can formulate the Kuratowski constraints as

$$\sum_{e \in K_F} y_e \leq |K_F| - 1 \quad \forall K \in \mathcal{K}. \tag{7.11}$$

From the discussion above we have the following theorem.

Theorem 7.10. *Let $C = (G = (V, E), T)$ be a clustered graph, $C^* = ((V, E \cup F), T)$, F and the variables and constraints as defined above.*

- *Any feasible solution of the inequality system consisting of the constraints (7.7), (7.9)–(7.11) corresponds to an augmenting edge set F_A such that $C_A = (G_A = (V, E \cup F_A), T)$ represents a completely connected c-planar subgraph of C^*; and vice versa:*
- *any completely connected c-planar subgraph $C' = (G = (V, E'), T)$ of C^* with $E \subset E'$ satisfies the constraints (7.7), (7.9)–(7.11); and*
- *the objective function (7.8) and the constraints (7.7), and (7.9) to (7.11) are a correct integer linear programming formulation for the minimum c-planar complete connectivity augmenting edge set problem for $C = (G, T)$. Clustered planarity of C is then equivalent to the existence of a feasible solution, and the edges corresponding to the variables y_e with $y_e = 1$ in the optimum solution of the ILP define a minimum augmenting edge set F for C such that the resulting clustered graph is completely connected.*

7.3.3 Pruning the Search Space

Focusing on pure c-planarity testing instead of maximum c-planar subgraph computation, we can establish some restrictions on the set of variables for the edges to be added and also on the constraints that we need compared to the approach from Section 7.2. Keeping the number of variables and constraints low may reduce the complexity of our optimization and therefore speed up the optimization process. Compared to the MCPSP approach, we first discard the variables for the original edges. In addition, not all of the potential edges have to be regarded as candidates for complete connectivity, and some of the constraints described above may be obsolete due to the specific structure of the given clustered graph. Some edges may not be needed in any case, and some might always be necessary. allow a simple decision if an edge or constraint is needed, and use such categorizations obtained by a preprocessing step to further reduce the number of possible variables.

Note that the restricted set of edges for the connectivity constraints above can still be much larger than necessary in practice, as it may contain edges that will definitely never be part of a solution. For example, if we have a large triconnected chunk tc that has an outgoing edge leaving the cluster, the embedding (up to mirroring) and the external face of the chunk are fixed. A vertex tc that is not adjacent to this external face cannot be used to connect chunks without introducing crossings. In general, when we have sets of augmentation edges that connect two chunks, we may exploit the similarity of the edges with respect to planarity issues: For sets of edges that, in combination with all possible augmentation sets, are equivalent with respect to the creation of Kuratowski subdivisions, the restriction of the order of outgoing edges of a cluster, and the connectivity obtained, we only need to consider a single representative. Clearly we cannot expect to find a full specification of these equivalence classes, but may already benefit from simple cases.

A first classification uses the concept of so-called bags. Bags describe the structural situation within a cluster c on a finer level compared to just the graph topology. A bag is a maximum set of chunks that are connected by subclusters.

Definition 7.5 (Bag)**.** Given a clustered graph $C = (G = (V, E), T)$ and a cluster ν of C, a *bag* is a maximum set b of chunks ch_j in ν, such that for each pair $ch_s, ch_t \in b$ there exists a sequence $ch_s = ch_1, \dots, ch_k = ch_t$ such that for $ch_i, ch_{i+1}, 1 \leq i < k$, there exists a subcluster ν' of ν that contains vertices of both ch_i and ch_{i+1}.

Note that also a single chunk may constitute a bag. The set of sub-clusters is called *bonding* clusters of b. Let the *collapse graph* of a bag b of ν be the graph that is obtained by collapsing each chunk in b to a vertex and connecting each pair of those vertices if and only if there exists a sub-cluster ν' that contains vertices of both corresponding chunks. A bag in ν is called *leashed* if it contains an extrovert vertex with respect to ν.

The idea behind the bag concept is the following: due to the sub-cluster connectivity constraints, we already know that the bag will be augmented to a connected component of ν's induced subgraph. Let us have a look at the situation after connecting disconnected sub-clusters of ν via augmenting edges: When two or more chunks are connected via sub-clusters, the induced subgraph consisting of those chunks will be connected after the clusters' induced subgraphs are connected. Therefore no additional connection between those chunks is needed when we want to achieve connectivity for ν. If a cluster ν contains a single bag, we do not need any connectivity constraints for ν, otherwise we can restrict the connectivity constraints to the connection between the bags in ν. Note that this does not allow to omit the remaining variables, as we still may need them to obtain complete connectivity: When a bag b consists of more than a single chunk, we might want to reduce the variables to the vertex pairs of each bonding cluster of b. However, as we also need to assure that the complement of each cluster is connected, we might need to add an edge outside of the bonding clusters. If for example two chunks are connected via a sub-cluster ν', we need a path connecting the subgraphs that remain after the removal of ν'. In other words, we can only safely restrict the edge addition between chunks to the vertex pairs in bonding clusters if there are at least two disjoint paths between the chunks' representatives in the collapse graph.

Consider a bag b that is not leashed. Neither does it restrict the embedding of $C \backslash b$, nor is its embedding restricted by $C \backslash b$. We can therefore safely remove b from ν and treat it separately.

7.3.4 The Branch-Cut-And-Price Algorithm

Both our two constraint sets contain an exponential number of constraints, and similar to the MCPSP we can solve the ILP with a branch-and-cut approach. Although the number of variables is quadratic in the number of vertices of the clustered graph C, only a linear number of them can be part of the solution. Otherwise the resulting graph will not be planar. We expect that only a small number of the non-edges will be needed to obtain a completely connected graph and hence use column generation to add variables only when required.

Adding constraints in the separation step, or fixing variables in the branching step, may result in an infeasible system of inequalities. This may either be the case when the graph is non-c-planar, i.e., no feasible solution can be found, or when necessary variables are not in the set of currently active variables. We need to efficiently decide which case applies and in the latter case, find a small set of variables that can be added to obtain feasibility again. Depending on the reason that caused the infeasibility, we apply different pricing methods that take advantage of problem-specific information. These methods are embedded into a branch-cut-and-price framework that is presented in the following section. Note that we do have a very specific setting here: Pricing is not needed to improve a feasible solution. Instead we add variables to make the system feasible. If at a certain point of the computation we obtain a valid solution, we can stop as we have found an augmentation that makes the input graph completely connected while preserving planarity.

Branch-Cut-and-Price Framework. In the following, we describe how the different steps of our approach work. The framework is depicted in Figure 7.1. In each iteration, we solve the relaxation of the current subproblem. If the subproblem is infeasible, we need to select variables to activate. We distinguish the following cases for pricing:

KuratowskiPricing: We add a Kuratowski constraint, and the system becomes infeasible. Then none of the involved edges can be deselected to satisfy the Kuratowski constraint (i.e., set the corresponding variable to 0) without violating a connectivity cut. For each edge in the Kuratowski subdivision we detect the set of connectivity cuts that become violated by setting the corresponding variable to 0. Then we activate variables that satisfy as many cuts as possible. We can apply several strategies here, e.g., either resolve after adding the first variable or after achieving a good cover of the violated cuts. As we are solving a linear program, we can also exploit the slack values of the cuts. When y_d is the variable we set to zero, we will only violate those with $slack - value(y_d) < 1$, and may repair strongly violated cuts first.

CutPricing: The system is infeasible due to a newly added cut constraint. We have to add a corresponding variable. If we cannot find a variable, Kuratowski constraints prevent addition of further variables and the current subproblem is indeed infeasible.

When the subproblem is feasible, we try to separate new constraints, see below.

Finding a Feasible Initial Solution. We start with a formulation that contains constraints to assure that each chunk of a cluster has to be connected to the remaining chunks in the same cluster. Therefore we can easily find a first feasible variable set by adding edges to satisfy the constraints.

```
Require: Clustered graph $C = (G, T)$
Ensure: Returns true if the clustered graph is clustered planar, and false otherwise
 1: Find edges $E'$ whose addition would make $C$ completely cluster connected
 2: Subproblem $s = (\{y_e \mid e \in E'\}, \emptyset, \emptyset)$
 3: Queue $Q = \langle s \rangle$
 4: while $Q$ not empty do                                        ▷ Main loop
 5:     Extract a subproblem $s = (Y_{\text{act}}, C_{\text{act}}, C_{\text{crit}})$
 6:     loop                                                      ▷ Solve subproblem
 7:         Solve LP on variables $Y_{\text{act}}$ with constraints $C_{\text{act}} \rightarrow$ solution $\bar{y}$
 8:         if infeasible then                                    ▷ Identify neccessary variables
 9:             if $C_{\text{crit}}$ contains cut-constraints then
10:                 Identify new variables $Y_{\text{new}}$ (CutPricing)
11:             else                                              ▷ $C_{\text{crit}}$ contains Kuratowksi constraints
12:                 Identify new variables $Y_{\text{new}}$ (KuratowskiPricing)
13:             end if
14:             if $Y_{\text{new}} \neq \emptyset$ then
15:                 $Y_{\text{act}} = Y_{\text{act}} \cup Y_{\text{new}}$
16:                 continue                                      ▷ Resolve subproblem
17:             end if
18:             break                                             ▷ Subproblem is indeed infeasible. Try next.
19:         else                                                  ▷ Identify neccessary constraints
20:             Find violated connectivity constraints (7.9), (7.10) $\rightarrow C_{\text{new}}$
21:             if $C_{\text{new}} = \emptyset$ then
22:                 Find violated Kuratowski constraints (7.11) $\rightarrow C_{\text{new}}$
23:             end if
24:             if $C_{\text{new}} \neq \emptyset$ then
25:                 $C_{\text{crit}} = C_{\text{new}}$
26:                 $C_{\text{act}} = C_{\text{act}} \cup C_{\text{crit}}$
27:                 continue                                      ▷ Resolve subproblem
28:             else if $\bar{y}$ is integral then
29:                 return true                                   ▷ c-Planar!
30:             end if
31:         end if
32:         Identify a variable $y_e$ to branch on                ▷ We need to branch
33:         $C_{\text{crit}} = \{c \in C_{\text{act}} \mid c$ is a cut-constraint using $y_e\}$
34:         $Q$.push( $(Y_{\text{act}}, C_{\text{act}} \cup \{y_e = 0\}, C_{\text{crit}})$ )
35:         $C_{\text{crit}} = \{c \in C_{\text{act}} \mid c$ is a Kuratowski-constraint using $y_e\}$
36:         $Q$.push( $(Y_{\text{act}}, C_{\text{act}} \cup \{y_e = 1\}, C_{\text{crit}})$ )
37:         break                                                 ▷ Tackle next subproblem
38:     end loop
39: end while
40: return false                                                  ▷ Not c-planar!
```

Algorithm 7.1: Testing c-planarity

Separation Routines. When we have a feasible subproblem, we try to separate new constraints. The basic strategy for separation is similar to the MCPSP case. We first try to obtain new connectivity constraints. Separating the connectivity constraints can be done in polynomial time using a minimum cut algorithm. If no violated connectivity constraints can be found, we resort to Kuratowski separation. The violated constraints are added to our problem and we continue. If no constraints are violated and the solution is integral, we have found a valid solution and can terminate. Otherwise, we need to branch. Adding connectivity or planarity constraints may render the inequality system infeasible under the current set of variables. Edges that may be needed for a valid solution might not be represented in the set of active variables. We then need to efficiently identify variables that can be added to obtain feasibility again if possible.

Compared to the MCPSP, we can strengthen our constraints by using specific information on the relation of the Kuratowski subdivision paths and the cluster connection edges. As a basic example, consider the case where chunks in a cluster all lie on a single path of a Kuratowski subdivision, and do not contain a Kuratowski minor vertex. Even though we could add a Kuratowski constraint that contains the connection edges between the chunks, we can use the information to achieve a stronger result. We know that these chunks will be connected in any completely connected augmentation, so we do not need to care about the connectivity edges. In order to achieve a planar augmentation, some edge in the remainder of the Kuratowski subdivision needs to be deleted. We therefore discard the chunk connection edges from our Kuratowski constraint and get a stronger constraint, as we prevent any construction that involves the remainder of the Kuratowski subdivision, but uses an alternative path that involves the chunks.

Branching. If we have a fractional solution, but cannot find any violated constraints, we have to resort to branching, as in any branch-and-cut algorithm. Fixing a variable in the branching step may render some of the inequalities (in which the variable is involved) invalid. We call these constraints *critical constraints after branching* and denote the set of critical constraints with C_{crit}. In case we fix a variable to 0, some of the already added cut constraints may become violated, and if we fix a variable to 1, Kuratowski constraints may become violated. In the latter case, the system may become infeasible if the violated constraints cannot be repaired due to cut constraints that prevent variables from being deselected. In both cases of infeasibility we therefore arrive in a situation similar to our *CutPricing* and *KuratowskiPricing* and may proceed accordingly.

We can however improve the performance by using the information that the infeasibility arises from a branching step: If we make the current formulation infeasible by fixing a variable to 0, we can try to reobtain feasibility by adding a currently inactive variable. If a valid solution is still possible, such a variable always exists and can be found as follows: Let S_{vc} be the set of constraints that become violated if variable y_e is set to 0. Clearly, $S_{vc} \subseteq \{\text{Cuts } k : \text{coeff}(k, y_e) = 1\}$ and for $k \in S_{vc}$ we have $lhs(k) - val(y_e) < 1$ before fixing y_e. S_{vc} cannot contain Kuratowski constraints, because otherwise these constraints would be violated for any value of y_e and no valid solution would be possible. We would like to find a variable that allows to make as many constraints as possible valid again and therefore search for a y that maximizes $\sum_{k \in S_{vc}} \text{coeff}(k, y)$. We add y to the set of active variables and remove all constraints k with coeff$(k, y) = 1$ from S_{vc}. We do this iteratively until all constraints of C_{crit} are satisfied or the LP is proven to be infeasible.

7.3.5 Conclusion and Future Work

We presented a Branch-Cut-and-Price approach for the clustered planarity testing problem. This approach improves upon our formulation for the maximum c-planar subgraph problem when only c-planarity testing is needed. Our approach will be implemented in OGDF to test the practical performance. A direction of further research is to establish strong general rules to reduce the number of possible variables. Examples are rules to find vertices that are never involved in a planar augmentation and can be left out, and sets of vertices and edges that are equivalent for the solution of the problem and therefore can be modeled by a single representative.

7.4 Linear Time Planarity Testing for C-connected Clustered Graphs

In this section we present a linear time algorithm for testing clustered planarity of a c-connected c-graph $C = (G, T)$, and for computing a clustered planar embedding for C. The algorithm resembles the algorithm of Dahlhaus [1998], but we give a modified version that uses a decomposition of the input graph based on SPQR-trees instead of graph grammars.

Let us first recall the basic ideas from Dahlhaus [1998]. Whereas the algorithm by Feng et al. [1995a] checks planarity of a c-connected c-graph by representing all possible orderings of the outgoing edges of a cluster with the help of the PQ-tree structure, Dahlhaus takes a different approach. Instead of modeling the freedom in the order of edges, it fixes a canonical embedding, the *normal form embedding* and checks this embedding for c-planarity. A main component of the c-planarity test is the criterion from Theorem 7.4 which states that a fixed embedding Γ of G is a c-planar embedding when there is no enclosing cycle for a cluster ν of T, and the external face is selected correctly. Note that due to the c-connectivity, ν can only either be not enclosed, or completely enclosed, it is not possible in a planar embedding that parts of ν are separated from each other.

The idea of the algorithm now is as follows: We apply an approach based on graph decomposition that uses SPQR-trees together with the fixed embedding criterion for testing clustered planarity of a c-connected c-graph. First we define a normal form for clustered planar embeddings and show how to compute such an embedding for the input graph in linear time. Then we show that c-planarity of C is equivalent to clustered planarity of any of its normal form embeddings. The final step then is to apply the testing criterion to the computed normal form embedding in order to test clustered planarity of the input graph. As the graph decomposition based on SPQR-trees only works for biconnected graphs, we give an extension of the approach to general connected graphs.

The combinatorial characterization of clustered planar embeddings is given in Section 7.4.1, where the main components of this characterization are also extended to be used within skeletons of SPQR-trees. A normal form of clustered planar embeddings is presented in Section 7.4.2. The clustered planarity test for biconnected graphs is described in Sections

7.4.3 and 7.4.4, the computation of a clustered planar embedding is given in Section 7.4.4, Section 7.4.4 deals with the time complexity of our approach, and Section 7.4.4 presents the extension to general connected graphs.

7.4.1 Clustered Planar Embeddings

In this section we give a combinatorial characterization of c-planar embeddings based on a weighting criterion on the dual graph. The combinatorial characterization allows us to judge if a fixed embedding is a clustered planar embedding. But in order to check clustered planarity efficiently we have to consider all possible embeddings without enumerating them. We will exploit the fact that SPQR-trees can be used to represent all possible combinatorial embeddings of a planar biconnected graph. Therefore we also discuss how to extend the notion of an edge and face weight to embeddings of the skeletons in SPQR-trees.

Combinatorial Characterization

In a c-planar embedding Γ of a c-connected clustered graph $C = (G, T)$, clusters appear as connected areas without holes. Recall the characterization of such a hole as a cluster part enclosed by a cycle of edges from another cluster that is not an ancestor. As we have seen, such a cycle can be detected for a given embedding using a weight function $w_{lca}()$ on the faces and edges.

We first restate Theorem 7.4 together with a proof:

Let $weight(c)$ of a cluster c be the number of nodes in c, for any edge $e = (v, w)$, let $w_{lca}(e)$ be the weight of the smallest cluster $c \in C$ that contains v and w, and the weight of a face f, $w_{lca}(f)$, is the maximum weight of an edge that belongs to the face cycle of f. We interpret the weights of the faces and edges of G as weights of the nodes and edges of the dual graph G' of Γ and G. A clustered planar embedding now can be characterized as follows: For each i, let F_i be the set of faces f of Γ with $w_{lca}(f) \geq i$ and E_i be the set of edges e of G with $w_{lca}(e) \geq i$. Let E'_i be the set of edges in G' corresponding to the edges in E_i. Then (F_i, E'_i) represents a subgraph of G'.

Theorem (Clustered Planar Embeddings)**.** *A planar embedding Γ of $G = (V, E)$ is a clustered planar embedding of G and $C = (G, T)$ if and only if for each i with $F_i \neq \emptyset$, (F_i, E'_i) is a connected subgraph of the dual graph of Γ and a face of maximum weight is taken as external face.*

Proof. Suppose (F_i, E_i) is not connected. Then at most one of its components C_i can contain the vertex representing the external face. Let us w.l.o.g. assume that C_1 does not contain the external face. The faces in F_i have an outer cycle Z in G. The edges of Z are represented in G' by edges that leave C_1. Because C_1 is a connected component of the subgraph consisting of edges with weight at least i, the edges on Z are of weight at most $i - 1$. Therefore the vertices that appear on Z are in a common cluster c of weight at most $i-1$. On the other hand each $f \in C_1$ is of weight at least i and therefore contains at least one edge e with weight $\geq i$. But then the vertices incident to e cannot be in c, i.e., Z encloses vertices that are not in c and c has a hole. Using the same reasoning, we can show that a face of maximum weight needs to be chosen as external face. If a face that has not maximum weight is chosen as the external face, then the outer cycle is in a common cluster that is not of maximum weight and therefore the outer cycle surrounds faces of larger weight and c has therefore a hole. Vice versa, suppose the cluster c of weight $i - 1$ has a hole with vertex set H and let Z be the innermost cycle of c that surrounds H. Then all inner faces f that share an edge with Z are of weight at least i. All these faces f can reach the external face (of maximum weight) only through edges of Z. Therefore the external face and the inner faces of Z sharing an edge with Z are in different connected components of (F_i, E_i). □

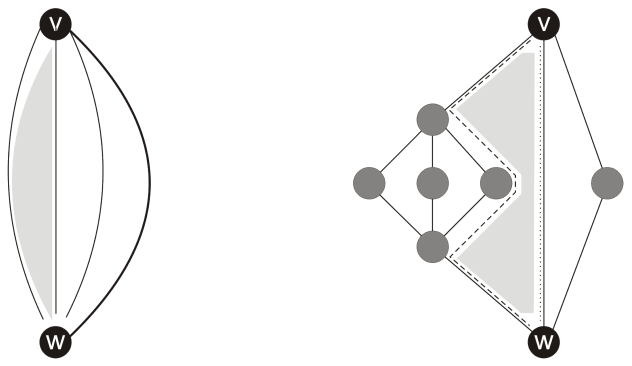
Weight Property for SPQR-Trees

We use SPQR-trees to represent all possible embeddings and need to apply the weight function $w_{lca}()$ to detect forbidden cycles. Therefore we give an extension of the weight property to the faces and edges of the skeletons in the SPQR-tree. The extension will then be used in Section 7.4.2 to compute a normal form embedding. We start our graph decomposition by selecting an edge with maximum weight as the reference edge; this edge will later define the external face.

Correlation of Faces in Skeletons and Pertinent Graphs. Faces in an embedding of the skeleton of a tree node μ correspond to faces in the corresponding embedding of the pertinent graph of μ in the following way. Let G be the given graph and S_μ denote the pertinent graph associated with μ, that is, S_μ is a subgraph of G. If μ is for example the Q node associated with the reference edge, then S_μ is the whole graph G. We say that f is a face of S_μ if all its boundary edges belong to S_μ. We call f a *proper face* of S_μ if it is a face of S_μ but not of any $S_{\mu'}$, such that μ' is a child of μ in the SPQR-tree. This means that there has to be a face in the embedded skeleton of μ that can be identified with f up to replacement of border edges as follows. Let f be a proper face of S_μ, μ' a child node of μ, such that $S_{\mu'}$ contains boundary edges of f, and v and w be the poles of μ'. Then the boundary edges of f that belong to $S_{\mu'}$ form a sub-path $p_{\mu'}$ of the face cycle of f starting at v and ending at w (see Figure 7.15). Replacing each $p_{\mu'}$ by the (virtual) edge between the split pair of μ' defines a face f' of the embedded skeleton of μ. We call such a face a *proper face* of a skeleton to show the correlation to proper faces of the pertinent graphs. Let Sk_μ denote the skeleton graph of node μ in the following. The proper faces of Sk_μ are exactly the faces that are different from the external face, i.e., the faces that are not adjacent to the virtual edge connecting the poles of μ.

Weights in the SPQR-tree. We would like to associate weights of faces of the embedding of G with weights of corresponding faces (as described above) in the skeletons of the SPQR-tree. Recall that the weight of a face f of G (and thus also of a proper face of some S_μ) is the maximum weight of a boundary edge of f. Therefore we would also like to give skeleton edges in the SPQR-tree weights such that the weight of a face in a skeleton can be defined too as the maximum weight of the boundary (skeleton) edges and is equal to the face weight of the corresponding proper face in the pertinent graph.

Let $g_1, \ldots, g_k$ be the boundary edges of a proper skeleton face f_μ of a node μ and let f be the corresponding proper face of S_μ. For each $g_i = (v_i, w_i)$ we consider a path p_i in



(a) Skeleton (b) Pertinent graph

Figure 7.15: A proper face of the skeleton in a P node, and the corresponding face in the pertinent graph after expansion. The dashed and the dotted line denote the two sub-paths belonging to child nodes.

S_μ from v_i to w_i such that the maximum weight of an edge on p_i is minimized (i.e., a path whose vertices have the smallest least common ancestor in the cluster tree T). We call these skeleton edge weights *axis weights*, denoted by aw_{g_i}, and the paths p_i with maximum edge weight equal to the axis weight *axis*. The concatenation of the paths p_i forms a cycle C of G that separates f from the external face of G.

Lemma 7.6. *The weight of any proper face f of S_μ in a clustered planar embedding Γ is the maximum axis weight of a boundary edge of the corresponding proper face f_μ of Sk_μ.*

Proof. If Γ is a clustered planar embedding and C is the cycle formed by the concatenation of the axes p_i of the boundary edges around f_μ, then by Theorem 7.4, the maximum weight of an edge of C and therefore the maximum axis weight aw_{max} of a boundary edge cannot be less than the weight of f, otherwise C would be included in a cluster c of size aw_{max} and enclose vertices that are not in c. On the other hand, the maximum weight of an edge e of C, contained in a path p_i, cannot exceed the weight of f, because then there would be a path on the boundary of f connecting v_i and w_i with maximum weight less then the weight of e, contradicting the definition of the axis weight. □

We associate the axis weights of the boundary edges of face f_μ with the corresponding pertinent nodes adjacent to μ in the SPQR-tree. Let aw_μ denote the axis weight of a node μ in the following, let max_μ be the maximum weight of an edge in S_μ, and for a face f_μ in the skeleton let aw_{f_μ} denote the maximum axis weight of a boundary edge of f_μ.

The boundary edges of the external face of the graph S_μ can be split into two paths p_1^μ and p_2^μ connecting the poles of μ. We call these paths the *outer paths* of S_μ. The weight max_μ cannot appear exclusively in the interior of the pertinent graphs of the nodes in the SPQR-tree, there has to be an edge of weight max_μ on one of the outer paths, otherwise a face of weight max_μ would be separated by the outer paths from the external face. This fact will be used to obtain a unique normal form embedding in Section 7.4.2.

An outer path of S_μ containing an edge of weight max_μ is called a *dominating outer path*. Figure 7.16 illustrates some of the terms from this section.

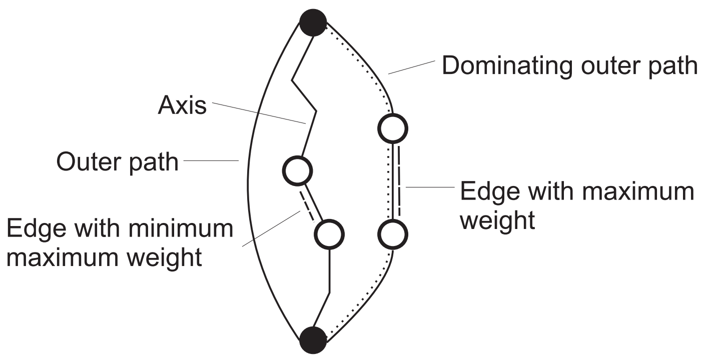


Figure 7.16: Illustration of an axis and a dominating outer path in a pertinent graph S_μ.

7.4.2 Normal Forms of Clustered Planar Embeddings

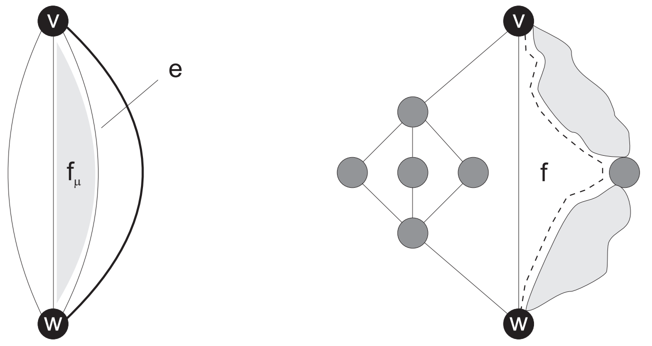
In this section we develop a normal form of clustered planar embeddings which has some special properties that we can exploit later on to check clustered planarity. We use the idea to swap subgraphs at poles to transform given clustered planar embeddings into a corresponding normal form. We then show how to convert a clustered planar embedding into a normal form that is unique if we have determined a dominating outer path for each pertinent graph S_μ. If both outer paths of a pertinent graph are dominating, we select one of them arbitrarily as *the* dominating outer path. The main result of this section is that a clustered graph has a clustered planar embedding iff any of its normal form embeddings is clustered planar.

Characterization of Normal Forms

First, we discuss properties of clustered planar embeddings regarding the dominating outer paths of subgraphs, then we give a characterization of a normal form of clustered planar embeddings based on these properties.

Characteristics of clustered planar embeddings regarding the weight property. Suppose f_μ is a proper face of the skeleton of a node μ in the SPQR-tree and f is the corresponding proper face of S_μ. If e is a boundary edge of f_μ and ν is the pertinent node of e, then one of the outer paths of S_ν is a sub-path of the cycle of the boundary edges of f; see Figure 7.17. The dominating outer path of S_ν can only belong to f in a clustered planar embedding, if the weight of f, and therefore aw_{f_μ}, is at least as large as max_ν, which is a weight on the dominating outer path. Otherwise the dominating path would be separated and there would be a hole in the cluster. The reverse conclusion allows a swapping of pertinent graphs in the embedding:

Lemma 7.7. *Let Γ be a clustered planar embedding and let μ and ν be two nodes in the SPQR-tree, where μ is the parent of ν. Suppose an outer path of the pertinent graph S_ν belongs to the boundary of a proper face f of S_μ and the weight of f is at least max_ν. Then Γ can be converted into a clustered planar embedding Γ_r, such that the dominating outer path of S_ν belongs to the boundary of f.*



(a) Skeleton of node μ (b) Pertinent graph of μ

Figure 7.17: The outer path (dashed line) of the pertinent graph of a skeleton edge e that lies on the boundary of a proper face f_μ of node μ is a part of the boundary of the corresponding face f in the pertinent graph of μ.

Proof. The result is trivial if the dominating outer path of S_ν already belongs to the boundary of f. Otherwise we want to swap S_ν and show that the resulting embedding is still clustered planar. We assume that in Γ, the dominating outer path of S_ν does not belong to the boundary of f but to the boundary of another face f' of G and Γ. Then f' also has a weight of at least max_ν. All faces that belong to S_ν have a weight of at most max_ν. Let the sets F_i of faces be defined as in Theorem 7.4. Recall that by Theorem 7.4, each F_i is connected in the dual graph of G and Γ. Denote the set of proper faces in S_ν that are in F_i by $F_{i,\nu}$. Then $F_{i,\nu}$ can only be non-empty if $i \leq max_\nu$. Since F_i is connected and f and f' are the only faces not belonging to S_ν that share boundary edges with S_ν, we can leave $F_{i,\nu}$ only through f or f', i.e., the connected components of $F_{i,\nu}$ are adjacent with f or f' (i.e., share boundary edges). When we swap S_ν the connected components of $F_{i,\nu}$ adjacent with f become adjacent with f' and vice versa. Therefore F_i remains connected and the embedding $\Gamma_r$we obtain by swapping S_ν is also a clustered planar embedding by Theorem 7.4 and the dominating outer path of S_ν belongs to the boundary of f. □

We can use the swapping to construct an embedding that takes into account the dominating outer paths. For the P- and S-nodes of the SPQR-tree, we can even give a more specific characterization. First we discuss the S-node case.

Corollary 7.2. *Suppose μ is a S-node with children $\nu_1, \ldots, \nu_k$ in the SPQR-tree representing the clustered planar embedding Γ. Then Γ can be converted into a clustered planar embedding Γ_r such that the dominating outer path of S_μ is the concatenation of the dominating outer paths of S_{ν_i}.*

Proof. Let p be the dominating outer path of S_μ and let p_i be the sub-path of p in S_{ν_i}. Let f be the face of G and Γ having p on its boundary. Note that f does not belong to S_μ. The weight of f is at least max_μ. Each p_i belongs to the boundary of f and max_{ν_i} is at most max_μ and therefore at most the weight of f. Due to Lemma 7.7, either p_i ist the dominating path of S_{ν_i} or we can swap S_{ν_i}, replacing p_i by the dominating outer path of S_{ν_i}. □

Next we consider the case of a P-node, which is slightly more difficult than the serial case. First we show that in a clustered planar embedding, there has to be a particular order of the parallel edges with respect to their axis and maximum weights, because there must not be any cycles of smaller weight around edges of higher weight. Therefore there may not be any local maximum of the axis weights besides the weights of the first and/or last edge in the edge sequence.

Lemma 7.8. *Let μ be a P-node with children $\nu_1, \dots, \nu_k$ in the SPQR-tree representing the clustered embedding Γ, and let the skeleton edges e_i corresponding to the child nodes appear in the sequence $e_1, \dots, e_k$ in Γ. Then the sequence of the axis weights of the e_i splits into a decreasing sequence $aw_{e_1}, \dots, aw_{e_l}$ and an increasing sequence $aw_{e_{l+1}}, \dots, aw_{e_k}$, where one of the sequences might be empty. Moreover, for $i \leq l$, $max_{e_i} \leq aw_{e_{i-1}}$ and, for $i > l$, $max_{e_i} \leq aw_{e_{i+1}}$.*

Proof. If the axis weights would not split into a decreasing and an increasing sequence, there would be positions $i < j < l$ such that $aw_{e_j} > aw_{e_i}, aw_{e_l}$. The concatenation of the axes of $S_{pert(e_i)}$ and $S_{pert(e_l)}$ would form a cycle of maximum edge weight $< aw_{e_j}$ that separates an edge and therefore also a face of weight aw_{e_j} from the external face, contradicting Theorem 7.4. To prove the second statement, we first consider the case $i < l$, i.e., e_i is not the last edge in the decreasing subsequence. If $max_{e_i} > aw_{e_{i-1}}$ then the axes of $S_{pert(e_{i-1})}$ and $S_{pert(e_{i+1})}$ which both have a maximum weight $< max_{e_i}$ separate an edge and therefore a face of weight $\geq max_{e_i}$ from the external face contradicting Theorem 7.4. For symmetry reasons, we also have $max_{e_i} \leq aw_{e_{i+1}}$ for $i > l + 1$. We consider the position l and observe that $max_{e_l} \leq aw_{e_{l+1}}$ or $max_{e_l} \leq aw_{e_{l-1}}$, for the same reasons as above. In the first case, we consider $aw_{e_1}, \dots, aw_{e_{l-1}}$ as the decreasing and $aw_{e_l}, \dots, aw_{e_k}$ as the increasing sequence. In the second case, we check whether $max_{e_{l+1}} \leq aw_{e_{l+1}}$. If this is not the case, for symmetry reasons, we put e_{l+1} into the decreasing sequence. □

Let $e_1, \dots, e_k$ and l be as in the previous Lemma. Then $e_1, \dots, e_l$ is called *the decreasing sequence* and $e_{l+1}, \dots, e_k$ is called *the increasing sequence* of $e_1, \dots, e_k$.

We now provide a method to convert the clustered planar embedding Γ into a clustered planar embedding Γ_r such that the maximum weight of the non-dominating outer path of S_μ is minimized.

Assume that e_i and e_j belong both to the decreasing subsequence or both to the increasing subsequence. Then either $aw_{e_i} \leq max_{e_i} \leq aw_{e_j}$ or vice versa.

Now assume that e_i and e_j are any parallel edges in Sk_μ. We say that e_i and e_j *overlap* if neither $aw_{e_i} \leq max_{e_i} \leq aw_{e_j}$ nor $aw_{e_j} \leq max_{e_j} \leq aw_{e_i}$. That means if e_i and e_j overlap then they cannot both belong to the increasing or the decreasing subsequence. The *overlap graph of* μ, denoted by O_μ, consists of the vertex set $\{e_1, \dots, e_k\}$ and the edges (e_i, e_j) such that e_i and e_j overlap. Connected components of O_μ are called *overlap components* of μ. Observe that e_i and e_j belong to the same subsequence if they can be joined by a path of the overlap graph of even length and that e_i belongs to the decreasing and e_j belongs to the increasing subsequence (or vice versa) if they can be joined by an odd length path of the overlap graph, just because subsequence membership alternates along a path in an overlap component.

From our observations it follows that if we fix for one edge e_i in a overlap component C of μ that it belongs to the increasing subsequence then we also know for each e_j of C whether it belongs to the increasing or to the decreasing subsequence of $e_1, \ldots, e_k$ independent of the particular clustered planar embedding. We extend the notions of maximum weight and axis weight to overlap components. Let max_C denote the maximum max_{e_i} with $e_i \in C$ and aw_C the minimum aw_{e_i} with $e_i \in C$.

Lemma 7.9. *If C and C' are different overlap components of μ then either $aw_C \leq max_C \leq aw_{C'}$ or $aw_{C'} \leq max_{C'} \leq aw_C$.*

Proof. Two edges e_i and e_j overlap if and only if the open intervals (aw_{e_i}, max_{e_i}) and (aw_{e_j}, max_{e_j}) intersect or if for one of them, say e_i, $aw_{e_i} = max_{e_i}, aw_{e_i} \in (aw_{e_j}, max_{e_j})$. For any overlap component C, the union of the intervals (aw_{e_i}, max_{e_i}) with $e_i \in C$ is (aw_C, max_C): The union of the intervals (aw_{e_i}, max_{e_i}) with $e_i \in C$ is a union of open intervals (a, b) with $a > aw_C$ and $b < max_C$. We only have to show that each $x \in (aw_C, max_C)$ belongs to some (aw_{e_i}, max_{e_i}) with $e_i \in C$. Assume this is not the case. Then we can partition the $e_i \in C$ into *small* e_i where $(aw_{e_i}, max_{e_i}) \subset (aw_C, x)$ and *large* e_i where $(aw_{e_i}, max_{e_i}) \subset (x, max_{e_i})$. No small e_i overlaps with a large e_j and there are small and large e_i. Therefore C is not connected in the overlap graph which is a contradiction. As a consequence, the intervals (aw_C, max_C) and $(aw_{C'}, max_{C'})$ must be disjoint. If for one of the overlap components, say for C, $aw_C = max_C$, then $aw(C) \notin (aw_{C'}, max_{C'})$. This proves the Lemma. □

Lemma 7.10. *A clustered planar embedding Γ can be converted into a clustered planar embedding Γ' with the following property: If μ is a P-node with children $\nu_1, \ldots, \nu_k$ in the SPQR-tree representing Γ, and the subgraphs S_{ν_i} appear in this sequence in Γ', then for each overlap component C of μ, the $e_i \in C$ with $max_{e_i} = max_C$ appears in the increasing subsequence of $e_1, \ldots, e_k$, where e_i is the skeleton edge corresponding to node ν_i.*

Proof. Let C be an overlap component. Assume that the e_i with $max_{e_i} = max_C$ belongs to the decreasing subsequence of $e_1, \ldots, e_k$. Let $C_{\leq}$ be the union of overlap components C' with $max_{C'} \leq max_C$ and let g_j be the largest index, such that $g_j \in C_{\leq}$. Note that $aw_{e_{i-1}}$ and $aw_{e_{j+1}}$ are at least max_C. Therefore also the weights of the face f_j consisting of an outer path of $S_{pert(e_j)}$ and an outer path of $S_{pert(e_{j+1})}$ and of the face f_{i-1} consisting of an outer path of $S_{pert(e_{i-1})}$ and an outer path of $S_{pert(e_i)}$ are at least max_C because the (axis) weight of the corresponding faces f_μ^j and f_μ^{i-1} in Sk_μ are at least $aw_{e_{j+1}}$. We can view $C_{\leq}$ as a component G_μ and set $S_C := \cup_{e_i \in C_{\leq}} S_{pert(e_i)}$ (the edges in the pertinent graphs of the skeleton edges in $C_{\leq}$). The outer paths of S_C are the outer paths of $S_{pert(e_i)}$ and $S_{pert(e_j)}$ that belong to f_{i-1} and f_j. The dominating outer path of C is the outer path belonging to $S_{pert(e_i)}$. We can swap H_C and the resulting planar embedding is still clustered planar by Lemma 7.7. The maximum weight edge e_i of C now belongs to the increasing subsequence. We have to do this with every overlap component.

While swapping $C_{\leq}$, also all C' with $max(C') < max(C)$ are swapped. Therefore we start with the overlap component C with max_C maximal. If max_C belongs to the decreasing sequence we only reverse the enumeration $e_1, \ldots, e_k$. Then we put the overlap component with the second largest maximum weight into the right position as explained above, and we continue with the overlap component with the third largest maximum weight and so on. □

The Normal Form Embedding

Using the results from Section 7.4.2, we can state the following Theorem:

Theorem 7.11. *Each clustered planar embedding $\varGamma$ can be converted into a clustered planar embedding $\varGamma_{nf}$ that satisfies the following conditions: Let μ be a node in the SPQR tree that represents the embedding $\varGamma_{nf}$.*

(1) If μ is an S-node with children $\nu_1, \ldots, \nu_k$, then the dominating path of S_μ is the concatenation of the dominating paths of the subgraphs S_{ν_i}.

(2) If μ is a P-node with children $\nu_1, \ldots, \nu_k$, where in $\varGamma_{nf}$ the S_{ν_i} appear in the given sequence from 1 to k, then either the maximum weight max_{ν_i} of each overlap component appears in the decreasing subsequence of $e_1, \ldots, e_k$ where ν_i is the pertinent node of skeleton edge e_i or the maximum weight of each overlap component appears in the increasing subsequence of $e_1, \ldots, e_k$.

(3) If e is a skeleton edge of μ then one of the following two conditions is satisfied:

- *e is an inner edge of Sk_μ (i.e., the outer paths of $S_{pert(e)}$ are not on an outer path of S_μ), and for the proper faces f_1 and f_2 of S_μ that contain the outer paths of $S_{pert(e)}$ (i.e., the corresponding proper faces f'_1 and f'_2 in Sk_μ share e as their boundary edge), the dominating outer path of $S_{pert(e)}$ is at the boundary of the f_j that has the larger weight.*
- *e is not an inner edge of Sk_μ (i.e., the outer path of S_μ contains an outer path of $S_{pert(e)}$), f is the proper face of S_μ that shares an outer path with $S_{pert(e)}$ (i.e., the corresponding proper face f' of Sk_μ is the only proper face of Sk_μ that has e as boundary edge), and the dominating outer path of $S_{pert(e)}$ is on the boundary of f if and only if $max_e \leq weight(f)$.*

We call the embedding Γ_{nf} a *normal form embedding*.

Construction of a Unique Normal Form Embedding

In order to have a *unique* normal form embedding that is independent from the given particular embedding Γ, regardless whether it is clustered planar or not, we introduce the following rules that make our decisions well-defined:

(1) If both outer paths of S_μ are dominating, we select one of them as *the* dominating outer path.

(2) We sort the faces of G with respect to the axis weight of the corresponding face in some skeleton to obtain a sequence $f_1, \ldots, f_k$ and if we have to select one of the faces f_i and f_j of maximum weight, we select the one with maximum index.

(3) We sort the nodes of the SPQR-tree in the first priority by their axis weight and in the second priority by their maximum weight to obtain a sequence $\mu_1, \ldots, \mu_k$. Note that if μ is a P-node with skeleton edges $e_1, \ldots, e_k$, then we can construct a normal form embedding such that the decreasing/increasing subsequence is also decreasing/increasing with respect to the index in the sorted list.

We construct Γ_{nf} bottom up. The embedding of each Q-node is uniquely determined. For S- and R-nodes, the embeddings are uniquely determined up to reversal. For P-nodes, there is only one sequence $e_1, \ldots, e_k$ such that for each overlap component C, $max(C)$ appears in the increasing subsequence when the decision rules are applied. If the embedding of the skeleton of each child node ν of a node μ in the SPQR-tree is uniquely determined up to reversal, then the dominating outer path of each S_ν is uniquely determined by our decision rules. Therefore also the embedding of S_μ is uniquely determined up to reversal when we apply the decision rules.

Note that the construction of Γ_{nf} is independent of any particular embedding Γ. We can convert any embedding Γ to Γ_{nf} and if Γ is a clustered planar embedding, we can convert it to the clustered planar embedding Γ_{nf}. Vice versa, we can reverse the conversion, and if Γ is also a normal form embedding then clustered planarity is preserved. We therefore get the following result:

Theorem 7.12. *A graph G and a clustering C have a clustered planar embedding if and only if any normal form embedding of G and C is clustered planar.*

7.4.3 Clustered Planarity Testing Algorithm

In Sections 7.4.3 and 7.4.4 we describe the clustered planarity testing algorithm that is based on the concepts introduced in the previous sections. Section 7.4.3 states how the axis and maximum weights as well as the dominating outer paths and also the skeleton embeddings can be determined. Section 7.4.4 shows how to compute the normal form embedding and how to apply Theorem 7.4 efficiently. We first consider the case where the whole graph is biconnected and then give a sketch of the algorithm for the general case of connected graphs.

Overview

The main steps of the clustered planarity testing algorithm are as follows:

(1) First we compute an SPQR-tree for the given graph and determine the axis weights and the maximum weights of the skeleton edges, which are independent of the embeddings of the skeletons.

(2) We determine embeddings for the skeletons Sk_μ of the SPQR-tree according to the rules of our normal form embedding.

(3) We determine the (axis) weights of the proper faces of each Sk_μ depending on the computed embedding of the skeleton.

(4) We determine the dominating outer paths of the skeletons, which are projections of the dominating outer paths of the subgraphs S_μ to Sk_μ. Note that each path of S_μ joining two vertices of Sk_μ projects to a path of Sk_μ.

(5) For each node μ with child node ν in the SPQR-tree, we determine how S_ν has to be embedded relative to S_μ to get a normal form embedding. This *relative embedding of μ and ν* depends only on the skeletons of μ and ν and the axis and maximum weights of their edges.

(6) We merge the relative embeddings into one embedding Γ that is a normal form embedding.

(7) We check that Γ is a clustered planar embedding using Theorem 7.4.

Assignment of Axis and Maximum Weights for the Skeleton Edges

The maximum weights can be assigned as follows: For each leaf-node (Q-node) μ with edge e, we just assign $max_\mu := w_{lca}(()e)$. If μ is a node with children $\nu_1, \ldots, \nu_k$ in the SPQR-tree (not a leaf), then $max_\mu := max(max_{\nu_1}, \ldots, max_{\nu_k})$. This can be done in linear time with respect to the size of the tree and therefore also in linear time with respect to the size of the graph G.

The axis weights are more difficult to compute. We use a minimum spanning tree with respect to the weights of the edges and take care that we do not lose our linear time bound. The edges of G can be divided into two classes depending on their position in the spanning tree and the class types can then be used to derive the axis weights.

First we compute a minimum spanning tree T_S of the whole graph G with respect to the edge weights in linear time using the following result:

Lemma 7.11. *Given a clustered graph $C = (G, T)$, a spanning tree T_S of G is a minimum spanning tree of G with respect to the edge weights if and only if for each cluster c of C, T_S restricted to c is a single tree (not a forest).*

Proof. First we show that any spanning tree where some cluster of C does not induce a subtree is not a minimum spanning tree, i.e., for every minimum spanning tree, each cluster induces a subtree. Let S be T_S restricted to c. We assume that S is not a tree, i.e., S is not connected. Since G restricted to c is connected, we can find two connected components S_1 and S_2 of S, such that there is an edge (u, v) of G that joins a vertex u of S_1 with a vertex v of S_2. There is also a path p from a vertex u' in S_1 to a vertex v' in S_2 such that all inner vertices of p do belong to T_S but not to S_1 or S_2. Let (u', x) be the first edge of p. x is also not in c, otherwise x would belong to S_1. Therefore the weight of (u', x) is greater than the size of c. On the other hand, the weight of (u, v) is at most the size of c. Therefore, if we replace in T_S the edge (u', x) by (u, v), we still get a tree, and the sum of the weights decreases. Vice versa, we show that all spanning trees with the property that each cluster induces a subtree have the same weight sum. Note that T_S remains a tree if we shrink any cluster to a vertex, because any cluster induces a subtree. Let c be a cluster and $c_1, \ldots, c_k$ be its child clusters. Then the number of edges (u, v) of T_S, such that c is the smallest cluster containing u and v is $k - 1$. The sum of the edge weights of these edges (u, v) is therefore $(k-1)|c|$. Let k_c be the number of child clusters of the cluster c. Then the weight sum of T_S is therefore $\sum_{c \in C} (k_c - 1)|c|$. This is true for every T_S such that each cluster $c \in C$ induces a subtree. $\square$

Using Lemma 7.11, we determine T_S in linear time as follows:

For each cluster c, let E_c be the set of edges $e \in E$ such that c is the smallest cluster containing both end vertices of e. We contract the child cluster of c, i.e., we replace each edge $e = uv$ of E_c by $c_u c_v$ where c_u and c_v are the child clusters of c that contain u and v respectively. For each cluster c, we compute a spanning tree T_c of E_c. Note that all edges in E_c have the same weight. Now the tree T'_c arises from T_c by replacing each edge $c_u c_v$ of T_c by one edge uv. Then the union of all T'_c is a minimum spanning tree T_S, i.e., a spanning tree, such that each cluster induces a subtree.

Next we root T_S at the vertex x where x is one of the two vertices of the reference edge. Now consider any inner node μ of the SPQR-tree with poles u and v and pertinent graph S_μ. We divide the inner nodes into two classes depending on the position of their poles within the spanning tree.

Introverted nodes are nodes whose poles are on a single path from the root to a leaf of the spanning tree, i.e., the parent of u or the parent of v in T_S is in S_μ; see Figure 7.18.

Social nodes are nodes whose poles are in two different paths from the root to a leaf of T_S, i.e., the parents of u and of v in T_S are not in S_μ.

Observe that for a social node μ the spanning tree T_S restricted to S_μ splits into two trees T^u_μ with root u and T^v_μ with root v. Any path of S_μ from u to v contains an edge e of G with one vertex in T^u_μ and the other vertex in T^v_μ. We call such an edge a *connecting edge* of S_μ.

No we can easily derive the axis weights from the type of the node as follows:

Lemma 7.12. *If μ is an introverted node of the SPQR-tree with poles u and v, then the axis weight aw_μ of μ is the maximum weight of an edge on the unique path from u to v in T_S. If μ is a social node, then aw_μ is the minimum weight of a connecting edge of S_μ.*

Proof. Consider the smallest cluster c that contains u and v and that remains connected if we restrict it to S_μ. We use the fact that aw_μ is the size of c. This follows from the following observations:

- If there is a path from u to v in S_μ with maximum weight w then u and v are in a connected component of a cluster c of size w restricted to S_μ. Since clusters are connected in G and u and v separate S_μ from the rest of the graph, G restricted to S_μ $\cup c$ is connected.

- If $S_\mu \cup c$ is connected and contains u and v, then there is a path from u to v using only vertices in $S_\mu \cup c$. Such a path has maximum weight $\leq |c|$.

From Lemma 7.11 we know that the unique path from u to v in T_S solely consists of vertices of c if μ is introverted. Therefore the maximum weight on this path is at most $|c|$, i.e., the axis weight aw_μ. If the maximum weight would be less, this path would contradict the definition of the axis weight.

If μ is social then the weight of any connecting edge of S_μ cannot be smaller than the axis weight of μ. But then there has to be a connecting edge with weight aw_μ because aw_μ is the size of c. □

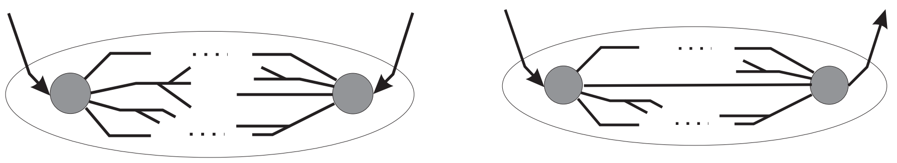

(a) Social node (b) Introverted node

Figure 7.18: Location of the poles of social (a) and introverted (b) nodes in the spanning tree T_S, thick edges denote tree edges, dotted lines denote parts that are not in the spanning tree.

To compute the axis weights efficiently, we proceed recursively, i.e., determine the axis weights of a node μ from the axis weights of the child nodes using Lemma 7.12. If we know the axis weights of all skeleton edges of μ, then we can derive the axis weight of μ in the same way as we determine the axis weight of μ from the weights of the edges in the pertinent graph S_μ.

Lemma 7.13. *Let μ be a non-leaf node in the SPQR-tree with poles u and v.*

(1) If μ is introverted then the skeleton edges of μ whose pertinent node is introverted form a tree T_μ with the same root as T_S restricted to S_μ. The axis weight of μ, aw_μ, is the maximum axis weight of an edge on the unique path from u to v in T_μ.

(2) If μ is social then the introverted skeleton edges of μ form two trees T_μ^u and T_μ^v with u and v respectively as root. The axis weight of μ, aw_μ, is the minimum axis weight of a skeleton edge joining a vertex of T_μ^u and a vertex of T_μ^v (we call these edges the connecting edges *of the skeleton of μ).*

Proof. Let T_μ be the set of introverted edges of Sk_μ and let x be a vertex of Sk_μ. There is exactly one edge (x, y) of T_S, such that y is the T_S-parent of x. There is at most one edge e_x of Sk_μ, such that (x, y) is in $S_{pert(e_x)}$. Moreover, if x is not u or v, there exists such an edge e_x and this edge is introverted. Furthermore, each introverted edge of Sk_μ is such an e_x for some x. Let x and p_x be the vertices incident with e_x, i.e., nodes in Sk_μ. Then we can consider p_x as the parent of x in T_μ and p_x is an ancestor of x in T_S. Therefore T_μ has no cycle and is a forest. If μ is social then T_μ splits into two trees, otherwise μ is introverted and T_μ is a tree. Suppose μ is introverted. Then each edge on the unique path from u to v in T_S is in some $S_{pert(e')}$, such that e' is on the unique path from u to v in Sk_μ. This is in particular true for the maximum weight edge e on the unique path from u to v in T_S. By Lemma 7.12, the edge e' of Sk_μ such that $S_{pert(e')}$ contains e has the same axis weight as the weight of e. Therefore the axis weight of μ is the maximum axis weight of an edge of T_μ on the unique path from u to v. Now suppose μ is social and let e be a connecting edge of S_μ. Note that also the edge e' of Sk_μ, such that $S_{pert(e')}$ contains e is social. Moreover, all connecting edges of $S_{pert(e')}$ are also connecting edges of S_μ. If e is a connecting edge of S_μ of minimum weight, then e is also a connecting edge of $S_{pert(e')}$ of minimum weight. On the other hand, if e' joins a vertex of T^u_μ and of T^v_μ, then also e' is social, and all connecting edges of $S_{pert(e')}$ are connecting edges of S_μ. Each connecting edge of such $S_{pert(e')}$ therefore has a weight that is at least the axis weight of μ. Therefore the axis weight of μ is the minimum over the axis weights of edges e' of Sk_μ connecting a vertex of T^u_μ with a vertex of T^v_μ. $\square$

We can efficiently check if a node μ with poles u and v is introverted, by checking that one of the Q-nodes, that correspond to edges connecting u and v with its respective parent in T_S, is a descendant of μ in the SPQR-tree. This can be done in linear time with a simple preprocessing of the SPQR-tree. Let μ be introverted. Note that for the components $S_{pert(e)}$ that correspond to skeleton edges e that are passed by the unique path from u to v the nodes $pert(e)$ are also introverted. We determine a subforest T_I of the SPQR-tree consisting of the introverted nodes. The children of μ in T_I are pertinent nodes of the edges in T_μ that are on the unique path from u to v in T_μ. The axis weight of μ is the maximum axis weight of a child of μ in T_I.

If μ is social, then we determine the subforest T_{Soc} of the SPQR-tree consisting of the social nodes. The children of μ in T_{Soc} are the nodes with one pole vertex in T^u_μ and the other one in T^v_μ, i.e., the connecting edges of μ. The axis weight of μ is the minimum axis weight of a child of μ in T_{Soc}.

Following Lemma 7.13, we get the axis weight of μ by determining the axis weight of the children of μ in T_I or T_{Soc} and taking the maximum or minimum, respectively. This can be done in linear time. We therefore have

Proposition 7.13. The axis weights of the skeleton edges can be computed in linear time.

Computation of Embeddings for the Skeletons

We need to compute embeddings for the skeletons of the S, P, and R-nodes of the SPQR-tree. For the rigid case in R-nodes there is a unique combinatorial embedding up to reversal. The combinatorial embedding of the path for the sequential case in S-nodes is unique, too. For the skeletons of P-nodes we have to find a normal form embedding, i.e., a sequence $e_1, \ldots, e_k$ of the parallel edges that splits into a decreasing and an increasing sequence that satisfy the conditions of Theorem7.4. The strategy to compute an embedding for the skeleton of a P-node μ is as follows:

(1) Instead of computing the whole overlap graph of μ we only compute a spanning forest O'_μ. We then also know the overlap components.

(2) For each edge in Sk_μ we determine whether it belongs to the decreasing or the increasing subsequence. For each overlap component C, we select a e_C with maximum maximum weight and put it into the increasing sequence.

(3) We know for each overlap component the elements of the decreasing subsequence and of the increasing subsequence and can therefore derive the sequence $e_1, \ldots, e_k$ which gives us an embedding of the skeleton.

A Spanning Forest of the Overlap Graph. Recall that the union of the intervals (aw_{e_i}, max_{e_i}) of an overlap component C forms an open interval (aw_C, max_C). Therefore if $aw_C < a < max_C$ then there is an $e \in C$ with $aw_e < a < max_e$. If some $e' \in C$ has smaller axis weight than e, then $aw_C < aw_{e'}$. On the other hand, if aw_e overlaps with some e'' then $aw_e < max_{e''}$ and therefore $aw_e < max_C$. It follows that if there is a e' with smaller axis weight than e in the same overlap component C, then $aw_C < aw_e < max_C$. We then can conclude that there is a $e''' \in C$ with $aw_{e'''} < aw_e < max_{e'''}$, i.e., e''' overlaps with e and has smaller axis weight than e. Hence we have the following:

Lemma 7.14. *If the overlap component containing edge e also contains an edge e' with smaller axis weight than e, then e overlaps with an e' with smaller axis weight than e.*

We know now, that if an e overlaps with an e' of smaller axis weight, we can use such an e' as parent of e in the spanning forest of the overlap graph. But this does not determine the whole spanning forest. We only know that if e has no parent, then $aw_e = aw_C$ where C is the overlap component containing e. There might be several edges e_i in C with $aw_{e_i} = aw_C$. Observe that if $aw_C < max_C$ such an e_i can only belong to C iff $aw_{e_i} < max_{e_i}$. If $aw_{e_i} = max_{e_i}$ its maximum weight is less than or equal to any axis weight of edges on C. For the backward direction, the intersection of the intervals (aw_{e_i}, max_{e_i}) and (aw_C, max_C) is non-empty and we can pick any x of the intersection and any $e' \in C$ with $aw_{e'} < x < max_{e'}$ so that e and e' overlap.

Observe also in general that if $aw_e = aw_{e'} < max_{e'} \leq max_e$ then e and e' overlap. If there is a e' with $aw_{e'} = aw_e < max_{e'} < max_e$ we can take such a e' as parent of e.

We now have the following situation: If e and e' belong to the same overlap component and have no parent then they coincide in the weight and in the maximum weight and the axis weight is smaller than the maximum weight. Therefore if the parent of e is not defined but $max_e > aw_e$ then we have to check whether there is another e' with $aw_{e'} = aw_e$ and $max_{e'} = max_e$. We consider e and e' as equivalent if they conincide in max and aw and the axis and the maximum value are different.

We observe that we can order the equivalent edges e in any way, and just the predecessor of edge e can be taken as parent of e. We use the observations above and proceed as follows:

(1) We sort the skeleton edges of μ in the first priority by their axis weight and in the second priority by their maximum weights to obtain a sequence $e_1, \ldots, e_k$. This can be done in linear time.

(2) If there is a e_j with $j < i$ and $max_{e_j} > aw_{e_i}$ then we can take such e_j as parent $par(e_i)$ of e_i. To do this efficiently, we proceed as follows:

   (a) We determine for each $i > 1$, $max'(e_i) := max_{j<i}(max_{e_j})$, i.e., the maximum max value of a e_j with index smaller then i.

   (b) For each maximum weight m, we determine the smallest index $i(m)$, such that $max_{e_{i(m)}} = m$. This can be done in linear time by bucket sort.

   (c) For each e_i with $max'(e_i) > aw_{e_i}$ we assign $par(e_i) := e_{i(max'(e_i))}$, i.e., the parent of e_i is an e_j with $max(e_j) = max'(e_i)$ of smallest index j.

Our main result is now

Proposition 7.14. Spanning forests of the overlap graphs O_μ of μ can be determined in linear time.

The partition of overlap components into decreasing and increasing subsequence. First we check whether an edge e_i of Sk_μ belongs to the increasing or the decreasing subsequence of the skeleton edges $e_1, \ldots, e_k$. Note that overlap components and nodes of O'_μwithout a parent are in a 1-1-correspondence. We proceed as follows:

(1) We determine for each overlap component C of μ, max_C and a $e_C \in C$ with $max_{e_C} = max_C$. That means, we determine for each e of Sk_μ that has no parent in O'_μ, the number max_C of that C with $e \in C$, i.e., the maximum $max_{e'}$ of a descendent e' of e in O'_μ(including e). For this C, we determine e_C. If e has proper descendants, i.e., C has more than one element, this e_C is the e_i of minimum index such that $max_{e_i} = max_C$, which means $e_C = i(max_C)$. If e has no proper descendants then e_C is just e.

(2) To check whether e belongs to the increasing or the decreasing sequence, we check if it has an odd or even distance from some e_C in the spanning forest O'_μ.

We know now whether an edge e in Sk_μ belongs to the decreasing or the increasing subsequence of $e_1, \ldots, e_k$, but we do not know the position of e in $e_1, \ldots, e_k$ yet.

We use our sorting $e_1, \ldots, e_k$ with respect to the axis and maximum weights from Section 7.4.3 as follows: Let $F_1(e_i) = 1$ and $F_2(e_i) = 0$ if e_i belongs to the decreasing subsequence and $F_1(e_i) = 0$ and $F_2(e_i) = 1$ if e_i belongs to the increasing subsequence. This e_i is the l^{th} element of the decreasing subsequence if $\sum_{j \geq i}^{F1(e_j)} = l$ and e_i belongs to the decreasing subsequence, and it it the l^{th} element of the increasing subsequence if $\sum_{j \leq i}^{F2(e_j)} = l$ and it belongs to the increasing subsequence.

The final sequence $e_1, \ldots, e_k$ is then just the concatenation of the decreasing and the increasing subsequence.

All steps can be done in linear time. We can now state the main result of this subsection:

Proposition 7.15. For all P-nodes, the embeddings of the skeleton edges represented by the sequence $e_1, \ldots, e_k$ can be computed in linear time.

Determining the Dominating Outer Paths of the Skeletons

We assume that μ is a P- or R-node in the SPQR-tree, so that the embedding of the skeleton is now fixed up to reversal (Remember that in an S-node, the skeleton consists simply of a path between the poles representing the serial structure, and an additional edge connecting the poles). Let u and v be the poles of μ. Note that any path p from u to v in S_μ projects to a path p' from u to v in Sk_μ. This path p' contains just those edges e_i in the skeleton such that $S_{pert(e_i)}$ contains at least one edge of p. The outer paths of S_μ project to *outer paths of the skeleton of* μ, i.e., the two paths from u to v in the skeleton that form the outer cycle of the skeleton. We would like to find the outer path of the skeleton that is the projection of the dominating outer path of S_μ to the skeleton.

Lemma 7.15. *Let a clustered planar embedding Γ of the graph G and a corresponding SPQR-tree be given. If there is a proper face f_μ in Sk_μ with axis weight $aw_{f_\mu} = max_\mu$ then there is an edge e on an outer path of the skeleton with $aw_e = max_\mu$.*

Proof. The proper face f in S_μ that is the corresponding face to f_μ also has weight max_μ. Let C be the concatenation of the axes of the edges e' that belong to the outer cycle of Sk_μ. One of the edges of C must have weight max_μ, because otherwise a face of this weight is separated by a cycle of smaller weight from the external face. Therefore one of the edges on the outer cycle of Sk_μ must have axis weight max_μ. □

The *dominating outer path* of a skeleton Sk_μ is the projection of the dominating outer path of S_μ into Sk_μ, i.e., an edge e of Sk_μ belongs to the dominating outer path of Sk_μ if and only if $S_{pert(e)}$ contains edges of the dominating outer path of S_μ.

Lemma 7.16. *A dominating outer path of Sk_μ can be determined as follows:*

(1) If Sk_μ contains a proper face f_μ with weight $aw_{f_\mu} = max_\mu$ then an outer path of Sk_μ containing an edge e with $aw_e = max_\mu$ can be selected as the dominating outer path.

(2) If Sk_μ does not contain a proper face f with weight $aw_{f_\mu} = max_\mu$ then an outer path of Sk_μ containing an edge e with $max_{pert(e)} = max_\mu$ can be selected as the dominating outer path.

Proof. We have to show that the selected outer path of Sk_μ is the projection of a dominating outer path of S_μ into Sk_μ.

If Sk_μ contains a proper face of weight max_μ then Lemma 7.15 applies, i.e., there is an edge e on one of the outer paths p' of Sk_μ, such that $aw_e = max_\mu$. This path p' is the projection of an outer path p of S_μ that also passes $S_{pert(e)}$. If u and v are the endnodes of e then p restricted to $S_{pert(e)}$ is a path from u to v in $S_{pert(e)}$. Since $aw_e = max_\mu$, p restricted to $S_{pert(e)}$ passes an edge of weight $\geq max_\mu$. Since $max_{pert(e)} \leq max_\mu$, such an edge must have weight max_μ.

If Sk_μ does not contain a proper face of weight max_μ then we consider any edge e of S_μ on an outer path of S_μ with weight max_μ. This edge e belongs to some $S_{pert(e')}$ with e' belonging to Sk_μ. Such a e' belongs to one of the outer paths of Sk_μ, and $max_{pert(e')} = max_\mu$. Now consider any e' in Sk_μ that belongs to an outer path with $max_{pert(e')} = max_\mu$. Let f_μ be the proper face of Sk_μ that contains e' and let f be the corresponding face of S_μ. Since the weight of f_μ and therefore the weight of f are smaller than max_μ, the dominating outer path of $S_{pert(e')}$ (which contains an edge of $max_\mu = max_{pert(e')}$) is not a sub-path of the boundary of f. Therefore the dominating path of $S_{pert(e')}$ is a sub-path of a dominating path of S_μ. This means that e' is an edge of the projection of a dominating path of S_μ into Sk_μ. Therefore the outer path of Sk_μ containing e' can be taken as dominating outer path of Sk_μ. □

The dominating outer path of Sk_μ now can be determined as follows:

(1) We determine the outer paths of Sk_μ. Note that a planar embedding of Sk_μ together with the edge connecting the poles is known. The two faces containing this edge determine the outer paths of Sk_μ.

(2) We check whether there is a proper face of Sk_μ with weight max_μ. Note that each face is determined by the sequence of its boundary edges. The maximum axis weight of a boundary edge of any face can hence be determined in linear time.

(3) We proceed as in Lemma 7.16 to determine the dominating outer paths. Since we know the face weights and the maximum and axis weights of the edges in Sk_μ, we can determine the maximum axis or maximum weight of each outer path in linear time and hence perform this step also in linear time.

As a result of this subsection, we get the following:

Proposition 7.16. If μ is a P- or R-node then a dominating path of Sk_μ can be determined in linear time.

7.4.4 Computation of the Clustered Planar Embedding

In this section we show how the final clustered planar embedding can be computed. Our final embedding has to satisfy the requirements of a normal form embedding. We can assume from the results of the previous sections that the embedding of each Sk_μ in the SPQR-tree is known. If μ is an S- or an R-node, we are done, in the case of a P-node we compute the decreasing and increasing subsequence of the parallel edges and sort both sets with respect to the axis weight. We also know the dominating paths, i.e., we know if the left or right outer path of Sk_μ is dominating.

Now we have to find the relative orientation of a child component Sk_ν of Sk_μ. Note that the axis in the pertinent graph of a skeleton edge e splits the pertinent graph into two parts. We have to guarantee that for each face f incident with e, for the part that in the final embedding will be turned in direction of f, its maximum weight is at most the axis weight of f. Otherwise the part P_{max} with the higher maximum weight would be enclosed by a circle with edge weights bounded by the axis weight of f. We therefore have to check if we have to swap the embedding of $Sk_{pert(e)}$. This can be done by first computing the relative orientation of a child component within its parent component by a bottom-up traversal of the SPQR-tree, followed by a top-down traversal that uses the relative orientations to compute the absolute orientation as follows:

For each node ν in the SPQR-tree, we store an orientation value $or(\nu)$ that specifies the relative embedding of a child component Sk_ν within its parent component Sk_μ. If e is an inner edge of Sk_μ then we have to swap $Sk_{pert(e)}$ if the part P_{max} is directed to the face of smaller axis weight. If e is an outer edge, and μ is not an S-node, then $Sk_{pert(e)}$ has to be swapped if the incident face f different from the external face has smaller axis weight than the weight of P_{max} and P_{max} is directed to this face or f has an axis weight that is at least the maximum weight in P_{max} and P_{max} is directed to the external face of Sk_μ. If μ is an S-node, the parts with the larger weight are oriented in the same direction.

Now we know for each child component its relative embedding within its parent component. We can derive the absolute orientation $abs(\mu)$ just by checking how often a component would be swapped either directly or by swapping an ancestor component. If there is an odd number of swaps, the component needs to be swapped in order to achieve the correct final orientation. We therefore only need to compute

$$abs(\mu) := \Pi_{(\nu\ ancestor\ of\ \mu\ in\ SPQR-tree)\ \vee\ (\nu=\mu)}\ or(\nu)$$

which can be done in linear time by a top-down traversal.
Let emb_μ be the original embedding of Sk_μ. Then

$$emb_\mu^{final} := \begin{cases} emb_\mu, & \text{if } abs(\mu) = 1 \\ \text{reversal of } emb_\mu, & \text{if } abs(\mu) = -1 \end{cases}$$

The final embedding emb^{final} of G can be defined as follows: We determine the embedding emb_μ^{pre} of S_μ recursively knowing the embeddings $emb_{pert(e_i)}^{pre}$ of $S_{pert(e_i)}$ for all e_i in Sk_μ and the embedding emb_μ^{final} of Sk_μ.

emb_μ^{pre} is determined as follows: For a skeleton edge e in Sk_μ with incident vertices u and v in emb_μ^{final}, in the clockwise enumeration of the incident edges of u and v in Sk_μ, e is replaced by the edges of $S_{pert(e)}$ that are incident with u, and with v, respectively. They appear in emb_μ^{pre} in the same order as in $S_{pert(e)}$. After processing all nodes in the SPQR-tree, we know the embedding of the root node and therefore we can derive the embedding for the graph G. We finally insert the root edge connecting the poles u and v of the root node between the first and last edge of u and v, respectively, and get an embedding for the input graph G that satisfies the requirements of a normal form.

How to Derive the Normal Form Efficiently

If v is a vertex in G, a node in the SPQR-tree $\mathcal{T}$ whose skeleton contains v is called an *allocation node* of v. The allocation nodes of v form a subtree T_v of T and if v is not a pole of the root of the SPQR-tree, then the root of T_v is the only node μ of T_v such that v is not a pole of μ, otherwise the root of T_v is the root of $\mathcal{T}$. Note that the number of roots of the trees T_v is equal to the number of vertices in G. The number of non-root nodes is also linear in the size of G, because each node μ appears as non-root node in at most two trees T_u and T_v where u and v are the poles of μ, and the size of the SPQR-tree of a planar graph is linear in the number of vertices of the graph. We can therefore determine the trees T_v by a traversal of the SPQR-tree in linear time.

We use the trees T_v to efficiently determine the final embedding emb^{final} of G. We may assume that the embeddings emb_μ^{pre} are known, where emb_μ^{pre} is the projection of a normal form embedding emb^{final} of G into Sk_μ.

We call a vertex v an *inner vertex* of Sk_μ, if v belongs to Sk_μ, but is not a pole of μ. For each inner vertex v of Sk_μ, emb_μ^{pre} determines the cyclic enumeration of the edges incident with v in Sk_μ. For each pole of μ, we have a cyclic enumeration of the edges incident with v where the first and last edge are fixed. Therefore we have for each node μ of T_v an enumeration of the children of μ that corresponds to the clockwise enumeration of the edges of Sk_μ incident with v.

The leaves of T_v are just the Q-nodes representing edges that are incident with v (except for the special case if v is a pole of the root of the SPQR-tree). We determine a postorder of the nodes in T_v and restrict this order to the leaves of T_v (plus the root if necessary). This gives us exactly the enumeration of edges incident with v as in emb^{final}, because if $e_1, \ldots, e_k$ is the enumeration of edges incident with v in Sk_μ that corresponds to the embedding emb_μ^{pre} then in emb^{final} the edges of $S_{pert(e_1)}$ are enumerated first, the descendants of $S_{pert(e_2)}$ second and so on, which is also true for the post order enumeration of T_v.

Therefore we get the following result:

Theorem 7.17. *The normal form embedding emb^{final} of G can be determined in linear time if the axis and maximum edge weights and the SPQR-tree are known.*

Proof. A postorder traversal can be done in linear time. Selecting and renumbering the leaves can be done in the same time bounds (for each leaf of T_v, determine the number of leaves of T_v with a smaller postorder number). □

Checking Clustered Planarity of the Final Embedding

At this point we have an embedding emb^{final} of G that is a normal form embedding. We know that this embedding is a clustered planar embedding if and only if G has a clustered planar embedding. We check this using Theorem 7.4 as follows:

We determine a spanning tree T_F of the dual graph of G such that T_F restricted to the sets F_i (see Section 7.4.1) is a spanning tree of (F_i, E_i) for each i. The existence of such a spanning tree is equivalent to the statement that any (F_i, E_i) is connected. Therefore, by Theorem 7.4, the fact that T_F restricted to F_i is a spanning tree of (F_i, T_i) for each i, is equivalent to the statement that emb^{final} is a clustered planar embedding.

Let $F_{=i}$ be the set of faces of weight i and $E_{=i}$ be the set of edges of weight i. Clearly, if i is the maximum weight of an edge of G, then $F_{=i} = F_i$ and $E_{=i} = E_i$ and, in a clustered planar embedding, such a $(F_{=i}, E_{=i})$ therefore has to be connected. If i is not the maximum weight, then in a clustered planar embedding for each connected component C of $(F_{=i}, E_{=i})$ there is an $f \in F_{=i}$ and an $f_1 \in F_{i+1}$ (i.e. with a higher weight than f), such that f and f_1 share an edge of weight i: If there is no such edge of weight $\geq i$, then (F_i, E_i) could not be connected. As the maximum weight of an edge at the boundary of f is i, the weight must be exactly i.

We compute the spanning tree T_F as follows:

(1) Determine the connected components of $(F_{=i}, E_{=i})$ and the spanning trees T_C for each of these components.

(2) For each connected component C of some $(F_{=i}, E_{=i})$ with i not maximum edge weight, try to determine an $f_C \in C$ and an $f_{1,C}$ of weight $> i$, such that f_C and $f_{1,C}$ share an edge e_C of weight i.

(3) T_F consists of the edges of the trees T_C and the edges e_C.

The computation steps obviously can be done in linear time. Regarding the correctness, we show the following:

Proposition 7.18. T_F is a tree if and only if the given embedding emb^{final} is a clustered planar embedding.

Proof. Using Theorem 7.4, we show that T_F is a tree $\iff$ for each i, (F_i, E_i) is connected. We use the following auxiliary result.

Lemma 7.17. *T_F is a tree if and only if for each non-maximum weight i and each connected component C of $(F_{=i}, E_{=i})$, the tree edge e_C of weight i and with incident faces f_C in C and $f_{1,C}$ with weight greater than i exists and for the maximum edge weight i in G, $(F_{=i}, E_{=i})$ is connected.*

Proof. T_F restricted to each component C of some $(F_{=i}, E_{=i})$ is a tree, therefore T_F is a tree if and only if T_F is a tree if we contract each such component C to a node. After contraction of these connected components, T_F is a forest and only the edges e_C remain. Therefore T_F is a tree $\iff$ for all but one component C, the edge e_C is defined. For components C of $(F_{=i}, E_{=i})$ with maximum weight, e_C is not defined. Therefore T_F is a tree $\iff$ there is exactly one connected component of the $(F_{=i}, E_{=i})$ of maximum weight, and for each non-maximum weight i and each connected component C of $(F_{=i}, E_{=i})$, e_C is defined. □

Now we assume that T_F is a tree. We root T_F as follows: For each non-maximum weight i in G and each connected component C of $(F_{=i}, E_{=i})$, we take f_C as root of T_F restricted to C. The parent of f_C is $f_{1,C}$. For the maximum weight i we take any $f \in F_{=i}$ as root. The weight of the parent of f is at least the weight of f and the weight of the edge joining f and its parent is the weight of f. Therefore T_F restricted to F_i is a spanning tree of (F_i, E_i) and therefore (F_i, E_i) is connected for each i. Vice versa, if for each i, (F_i, E_i) is connected then we know from the discussion above that for each connected component C of $(F_{=i}, E_{=i})$, there is an edge e_C with weight i at the boundary of some face $f \in C$ and of another face f' of weight $> i$. We chose one such edge as edge of T_F and obtain a tree. □

Complexity Analysis

An SPQR-tree for a given biconnected graph can be constructed in linear time as shown by Gutwenger and Mutzel [2001]. The recursive procedure to compute the maximum weights needs linear time with respect to the size of the SPQR-tree and therefore also linear time with respect to the size of the graph G. From Proposition 7.13 we know that the axis weights can be determined in linear time.

The computation of the embeddings of the skeletons can be done in linear time. For the P-nodes we need to sort the subsequences of the parallel edges, but as they are sorted by their axis weights, this can be done in linear time, too. The face weights are determined by the computed axis weights and the computation of the dominating paths can be done in linear time by Proposition 7.16. After obtaining a normal form embedding, we can check in linear time, if it is clustered planar, which is true if and only if the clustered graph is clustered planar.

We get the following main result:

Theorem 7.19. *If G is a biconnected graph then we can check in linear time whether $C = (G, T)$ has a clustered planar embedding and we can construct such a clustered planar embedding within the same time bound.*

Extension to Connected Graphs

We give a sketch of how to extend the algorithm to the case of connected graphs. A connected, but not biconnected graph G can be decomposed into its biconnected components in linear time [Hopcroft and Tarjan, 1973b, Tarjan].

We select in G an edge e of maximum weight and consider the block b_r containing e as root of the block tree of G, $\mathcal{B}$. This defines, for every other block and each vertex b, a parent $P(b)$ of b in $\mathcal{B}$.

The algorithmic idea now is as follows:

(1) We determine clustered planar embeddings emb_b of each of the blocks b, such that the parent $P(b)$, which has to be a vertex, appears at the external face.

(2) Let v be the parent of a block b and b' the parent of v, i.e., v separates b and b'. We embed b into a face f of b' that contains v as boundary vertex and has a weight that is at least as large as the maximum weight of an edge in b or a descendant of b.

To guarantee that the weight of a face into which a block is embedded is large enough, we redefine the maximum weights.

(1) We define weights of vertices. Note that the decision to embed a block b into a certain face f means that all descendants of b in the block tree are enclosed by the face boundary of f. The weight of a vertex v is therefore defined as the maximum weight of an edge that appears in a block that is a descendant of v in $\mathcal{B}$.

(2) Let μ be a node in the SPQR-tree of block b. The axis weight is defined as before. The maximum weight of μ is the maximum weight of an edge of S_μ or an *inner vertex* of S_μ (i.e., a vertex different from the poles of μ).

(3) If max_μ is the weight of an edge of S_μ then the dominating path of S_μ is the outer path containing such an edge. If max_μ is instead the weight of an inner vertex of S_μ, then the dominating outer path is one that contains an inner vertex of S_μ of maximum weight.

We can determine max_μ in the same recursive manner as in the biconnected case. We determine the weights of the vertices and the maximum weights of the skeleton edges and select the maximum of these weights as max_μ.

The next step is to determine the normal form embeddings of all the blocks b. It is clear that each vertex v appears as boundary vertex of some face of b that has a weight that is at least as large as the weight of v. In particular, this is true for the parent $P(b)$ of block b. The weight of $P(b)$ is at least the maximum over all vertex and edge weights of b. Therefore $P(b)$ appears at a face of maximum weight of b, this face can then be taken as external face of b.

After determining the embeddings of all blocks, we embed each block b into the face of the parent of $P(b)$ of largest weight that has the vertex $P(b)$ at its boundary. This defines the final embedding. The time bound is the same as in the case of biconnected components.

7.4.5 Conclusion

We presented a linear time algorithm for c-planarity testing of c-connected c-graphs. Overall, the computations of the normal form embedding and the underlying theoretical foundations are quite complex, the algorithm therefore does not seem to be a good starting point for a practical approach to solve the general case for non-c-connected c-graphs. Nonetheless the concepts used for the characterization in Section 7.4.1 might be useful, but as clusters need not induce connected subgraphs, we have to take care about edge-region crossings in this case.

Part III

INTERACTIVE GRAPH DRAWING

8. INTERACTION

The sum of being consists of the two systems of substantial forms and interactional relations.

J. F. Smith in Encycl. Brit. XXI. 412/1 (1886)

In the context of this work, *interaction* refers to the process in which a user interacts with graphical representations of graphs, either to explore the underlying data or to produce a nice representation, e.g., for publishing or presentation purposes. Interactive graph editing and exploration facilities have become components in visualization systems from many application areas. Depending on the application area, the task at hand, and the software tool used, the interaction may range from simple changes of graphical attributes, as, e.g., colors or vertex size, over the generation of a new layout, to the modification of the graph structure.

Interactive graph drawing means computing 'iterative' layouts on a changing graph, i.e., to draw a series of graphs G_i where the difference between subsequent graphs is small and often consists only of a single vertex or edge addition. Interactive graph drawing is useful in the context of iterative processes for information gathering and modeling, and there are two main aspects associated with it. The first one is how to preserve as much as possible the user's mental map, i.e., the characteristics and landmarks of the drawing that the user memorizes and exploits for orientation. Certainly, the new drawing should also offer a good quality with respect to the layout aesthetics in the static case. The second aspect deals with the interaction concept, i.e., the operations allowed for the user to change the representation or data, including navigation methods to move from the view of one data subset to another. An important requirement for these operations is that the user should be able to work in 'real time' interactively without having to wait for results too long. The recomputation should therefore happen efficiently, for example by reusing parts of the previous layout instead of creating the new one from scratch. However, there are also cases where the sequence of all changes is known in advance; then the layouts can be computed in a preprocessing step (*offline drawing*) and the change information can be used to improve the quality of the intermediate layouts.

In the literature, the term 'dynamic graph drawing' is used quite frequently in a meaning similar to the definition of interactive graph drawing here. As dynamic changes in graphs do not require user interaction, but may for example be the result of a change in the data, the concepts are not equivalent. Nonetheless they are closely related and the resulting scientific questions have a large overlap, we therefore discuss the related topics together. A restricted scenario is considered in *incremental graph drawing*, where a layout for a graph is computed by adding vertices one at a time.

Even though interaction is an additional requirement both for the software tool and for the layout approach, well-designed interactive navigation techniques may reduce both the computational and visual complexity. Interactive visualization coupled with analytical methods is therefore employed in practice to support the user in complex data exploration and decision making processes, and the quality of the realization will have significant impact on the success of these processes. Munzner [2000] states:

> Interactivity is the great challenge and opportunity of computer-based visualization.

Further important applications, besides data exploration, include the design of software or business processes in modeling tools. There, the user memorizes the overall architecture and expects the system to preserve both global and local layout characteristics to a certain extent.

Clearly navigation and interaction issues are closely related, as navigation needs user interaction and results in changes of the data representations. The treatment of interaction in this thesis is not intended to completely cover all corresponding issues, but to treat some of the main topics with regard to the resulting requirements for graph drawing solutions, discussed using exemplary interaction use cases from practical applications. Before we describe these use-cases in Chapter 9, we give an overview on the relevant interaction and navigation concepts and techniques in this chapter.

8.1 Interaction Concepts

There are several general concepts and terms related to interaction which categorize the operations on the user interface, the graph representation, and the graph layout. These concepts concern the user's underlying intentions and goals when performing an operation, and also the resulting events.

The development of a taxonomy that describes the design space of interaction techniques is still a question of current research. Yi et al. [2007] identify seven categories of interaction techniques that they propose as a basic framework:

- Select: Mark something as interesting.
- Explore: Show something else.
- Reconfigure: Show different arrangements.
- Encode: Show different representations.
- Abstract/Elaborate: Show more or less detail.
- Filter: Show something conditionally.
- Connect: Show related items.

Lin and Eades [1995] define the requirements of what they call a *Diagram User Interface (DUI)* as:

- Readability, representing rules such as minimization of crossings, symmetry, and uniform distribution of vertices.
- Conformance to the syntax and semantics of the application's diagrammatic representation. The term 'conformance criteria' denotes the rules that define how the layout conforms to the application.
- Controllability, which denotes the user interaction, for example placing vertices to the center, or restricting the drawing width. The corresponding rules are called 'preference criteria'.
- Reasonable response time, i.e., the interactive system should not exceed the user's tolerance; therefore a computationally efficient implementation is needed.

Clearly, the requirements on the appearance of the user interface and also the response time may also vary depending on the role of the user. Parameter setting interfaces to influence the layout algorithm, and interaction techniques to modify and represent constraints should be adapted to expert and non expert users. Often multiple views show different aspects of the same data set at the same time. Linking the corresponding views, such that a change in the representation in one of them is adopted in others, as, e.g., selection of a color for a data subset, helps in improving the knowledge gain and at the same time reducing the confusion from multiple simultaneous views on the data. Also brushing techniques, like simultaneous selection in all views, can improve the value of multiple views in the analysis task.

The most basic conceptual distinction for layout operations is given by Lin and Eades. They distinguish between two layout operations: 'layout creation', i.e., creating a layout from scratch, and 'layout adjustment', i.e., updating an existing layout after modifications. In Lin and Eades [1995] and Misue et al. [1995] the term *layout adjustment* is used to describe the necessary update of a drawing after a change performed by either the user or the system. Such changes may for example be addition or deletion of objects, or also the modification of the layout due to a change in the data, or based on personal preferences. The update then is required to beautify the drawing again or to guarantee that the drawing conforms to a specified drawing convention.

There is a large number of potential user actions that may trigger a change in the layout, involving the following:

- A change of the graph structure, including addition or removal of objects, edge splitting, or changing the direction of an edge. A special case of this change occurs when an expand-and-collapse mechanism is used for navigation, allowing to expand/collapse groups of vertices. Besides direct manipulation over the graph representation, the change may also be caused indirectly, for example with a filter operation that eliminates graph objects from the view or from the database.
- A change in the cluster structure, either by moving vertices or by adding or removing clusters.
- A change in the data annotations, e.g., adding edge weights that have to be reflected by the layout method.

- A change in the object semantics, e.g., changing the type of a vertex or edge. In UML class diagrams for example, changing the edge type from association to generalization should result in a modified layout that reflects the new inheritance structure.

- A change of the constraint set, by either adding/removing a constraint or modifying an existing one.

- A manual change of the layout. The user may either want the layout to remain otherwise unchanged in this case, or allow the layout method to adapt the layout to the change, e.g., by repositioning neighbors of a moved vertex or by creating the space needed around the new position.

- A change in the abstraction level of detail by a semantical zoom operation may add/remove additional objects, data annotations, or both.

Interaction thus occurs to directly or indirectly manipulate (visual) representations, and to modify or enter data. It may also be used to give the system hints to achieve good results in an optimization process [do Nascimento and Eades, 2002, 2005, 2008]. Conversely, the system should give hints to the user where important patterns (regarding semantic or structure) are hidden in the diagram. An interesting question in this context is if even in the case when a change in the graph can be automatically animated, user input should be at least allowed: It may be beneficial for the knowledge transfer to allow or even require some kind of user interaction guiding the animation. The user's active participation may help to support the knowledge discovery and the development of a mental map regarding the structure of the data.

An aspect that is especially interesting for interactive graph drawing is the influence of performance issues on user acceptance. When only static representations have to be computed just once, for example for metro map layouts, performance issues are of minor significance. In interactive environments, however, it is very important to have an acceptable response time, and for some applications even a real time environment is needed. Even though several methods allow to handle even large graphs within a few seconds with acceptable layout quality (see, e.g., Section 4.2.3), interactive graph drawing poses a number of problems that need to be solved. Multilevel drawing approaches for example start from scratch, that is they do not build upon the existing layout, and can therefore not easily be used to preserve a drawing.

A general goal for layout changes is *stability*, i.e., the preservation of layout characteristics which allow the user to orient himself in the drawing. However, there are no clear rules how to exactly specify and achieve stability. In the following section we present and discuss existing approaches. Note that it is also possible to use different views to either find 'the' view that suits best the visualization needs or to explore different and unrelated characteristics of the data. As the idea is to create diverse and unrelated layouts, there is no need in this case to try to achieve stability.

Stability and the Mental Map

I end here as I did in 1981: cognitive maps may be impossible figures.

BARBARA TVERSKY, 1992

When graphs or their representations change over time, we would like to adapt the drawing in a way that allows the user to keep his orientation and to experience the change as a smooth transition from one drawing to the next. This goal is commonly denoted by the terms *stability* and *mental map preservation*. See Figure 8.1. As the cognitive processes underlying the perception of a diagram are only understood to a small part, there is no exact definition for the mental map, and it is even unlikely that there ever will be one [Tversky, 1992]. Clearly, the main characteristics of the mental map may differ individually and depend on several further aspects, therefore no general specification of drawing constraints can be given to guarantee mental map preservation in all cases.There are however attempts to capture some of the main characteristics which we will describe next.

Disruption of the mental map can be partially prevented by using some kind of *animation* (see Section 8.3.1), but when the change in the structure or the layout is too large, this won't help much. Instead, the new layout should comprise some characteristics that can be matched by the user with characteristics from the previous drawing. This might either be characteristic structures of the graph drawn in a predefined and similar way, or anchor landmarks that allow orientation. Even though stability is a major issue in interactive graph drawing, mainly practical algorithms are published, and only a few theoretical treatments and empirical evaluations deal with the mental map [Lin and Eades, 1995, Misue et al., 1995, North, 1996, Nesbitt and Friedrich, 2002, Purchase et al., 2007, Purchase and Samra, 2008]. This is at least partially due to the fact that a thorough treatment of these problems needs to involve perceptional, psychological, and application specific issues in addition to graph drawing considerations. Experimental studies are also difficult to design properly, and labor-intensive to conduct.

North [1996] defines the *incremental layout problem* as:

> Given a sequence of graphs $G_0, G_1, G_2, \ldots, G_n$ interpreted as successive versions of a graph G, find a 'good' sequence of Layouts $L_0, L_1, L_2, \ldots, L_n$.

The question here is, how do we define 'good'? North gives a set of desirable characteristics and proposes the following order of importance:

- *Consistency:* The adherence to layout style rules.
- *Stability*: The preservation of the mental map.
- *Readability* create a pleasing and easy to read diagram.

North [1996] justifies a higher priority for consistency by the assumption that fundamental changes in the graph structure may be expected to cause large changes in layouts. Even though this is a reasonable assumption, it is difficult to define a mapping of the degree of fundamentality of structural change to the degree of layout change. Moreover, in the context

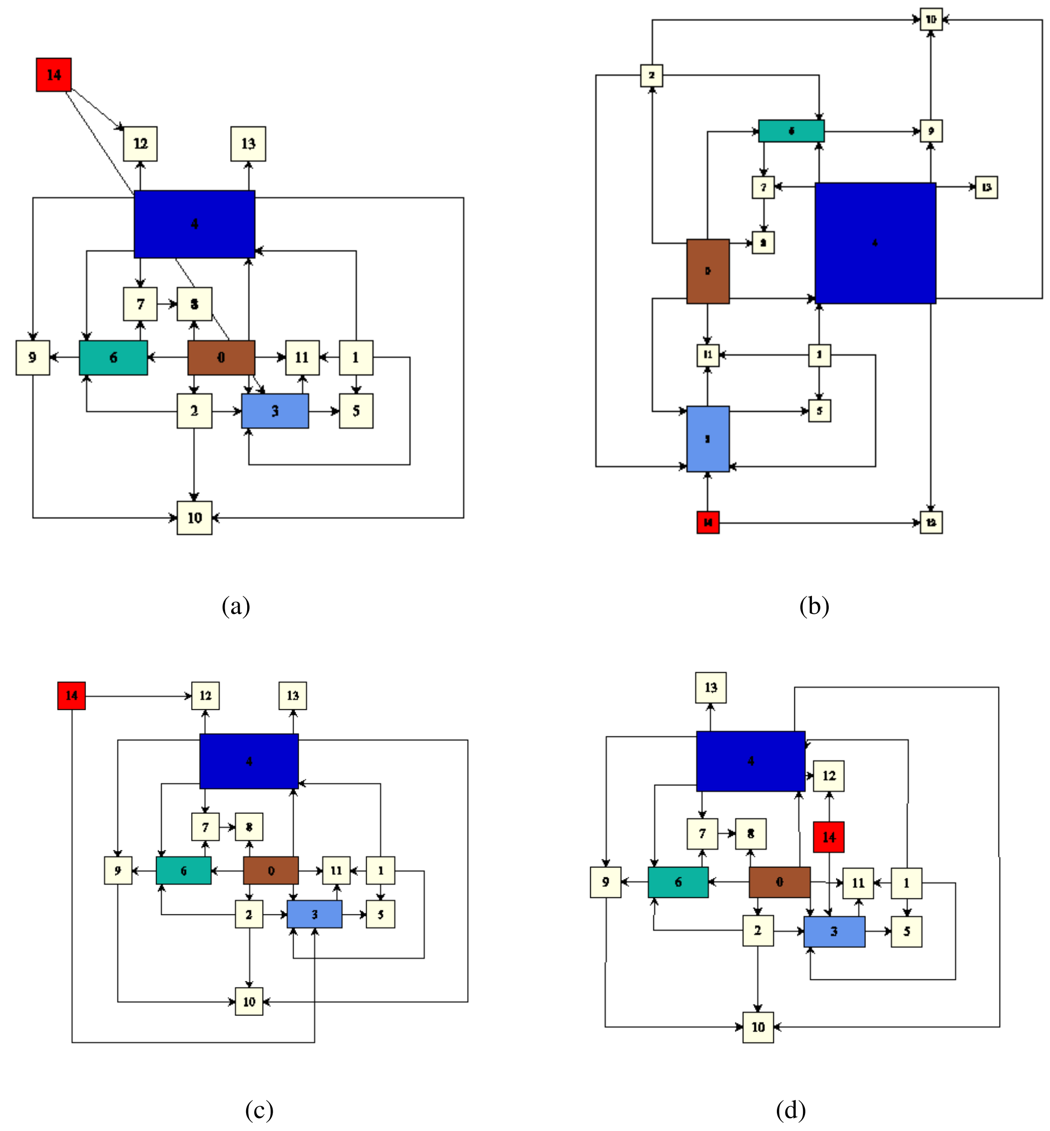


Figure 8.1: Effects on the layout by a change in the graph structure: (a) A graph drawn in orthogonal style using the GIOTTO approach, with a manually added vertex (red vertex). (b) The layout recomputed without any layout preservation. Orthogonal ordering, topology, and even the vertex sizes are changed. (c) Full preservation of the previous layout. (d) A balance between minimum change and high layout quality, placing the new vertex close to his neighbors, and minimizing edge bends at the expense of an edge crossing.

of interactive graph drawing applications it is questionable whether the user is always aware of the fact that he is causing a fundamental change of the structure in a graph theoretic sense by a simple operation.

As most important factors for stability characterization North identifies position (geometry) and order (topology), both as absolute properties and relative to a logic or geometric neighborhood. He suggests that some kind of 'age' or 'memory' could be employed to keep track of recent changes and to prefer recently changed graph objects and their neighborhood for further movement in subsequent changes. In addition, North proposes that edge routing stability is less important than vertex stability, as edges are traced on the fly to discover connections. This is however not a reasonable assumption for application areas where edges and their routing may be part of characteristic substructures with established drawing styles, as the alignment of the main pathway in a biological network or the representation of a network motif. In contrast, Nesbitt and Friedrich [2002] propose to highlight key structural edges, to improve the perception of either semantically important or perceptually dominant edges. Nesbitt and Friedrich suggest that the mental map of information provided by a graph drawing is very dependent on the perceived form or structure in the overall layout and investigate the application of perception principles for animation purposes.

The most simplistic concept of layout preservation clearly is to keep the previous layout fixed. This may quickly lead to bad layout quality even after a few graph changes, and does not allow modification of the layout to adhere to drawing conventions that might require updates from the previous layout. Even 'small' local changes like the addition of a vertex may be in conflict to the optimization goals in a way that makes a global rearrangement necessary.

Papakostas and Tollis [1996] propose several scenarios for interactive orthogonal graph drawing for 4-planar graphs, where only elementary changes to the graph structure are allowed:

- No-change: the previous layout is preserved.
- From-scratch: the layout is recomputed for the updated graph.
- Relative-coordinates: the general shape should stay the same, but small changes are allowed.
- Full-control: the user is in control of placing the new vertex or edges.

The most interesting of these cases in the context of this work is the relative coordinates scenario. Depending on the definition of 'small changes', it comprises all aspects of layout updates discussed here.

Misue et al. [1995] and Eades et al. [1991] propose orthogonal ordering, clusters, and topology as the layout properties that should be maintained to preserve the mental map. Orthogonal ordering captures the idea to preserve the relative ordering of point pairs in both dimensions – if v is southeast of w in the first drawing, it should remain there in the new drawing. Clusters reflect the idea of proximity – what is close together should stay close together, and topology preserves the sense of inside and outside curves, typically captured by the planar embedding. Based on these properties, Diehl and Görg [2002] formally define metrics to measure how close the mental maps of two given layouts are. They call the resulting class of metrics *mental distances* defined as follows: Let **Layout** be the set of all layouts.

Mental Distance [Diehl and Görg, 2002] Let $l_1, l_2 \in$ **Layout** be two layouts. Then the function Δ : **Layout** $\times$ **Layout** $\rightarrow \mathcal{R}_0^+$ is a metric for how good l_2 preserves the mental map of l_1. In particular $\Delta(l_1, l_2) = 0$ means that l_1 and l_2 have the same mental map.

Diehl and Görg note that such a metric does not need to be restricted to the layout of the common subgraph in case of a dynamic graph operation. As examples for mental distances they define and implement within their approach a *Euclidean mental distance* that is defined as the sum over all vertices of the Euclidean distances between the positions of a vertex in the two layouts. This is not a very adequate measure in practice, because it does not take into account the amount of move for each single vertex, and it also completely ignores that vertices may move together in a way that does not distort the mental map. The *orthogonal mental distance* is defined to be proportional to the sum of the number of all changes in the orthogonal ordering between each pair of vertices in x and y direction. Frishman and Tal [2008], too, propose vertex displacement as a metric to measure dynamic stability. Lyons [1992] proposes an approach called *cluster busting*, which aims at distributing vertices more evenly while preserving the shape of the overall layout. This concept use the idea of a 'shape' of a set of vertices, ignoring the edges. A difference metric based on the sum of distances that the vertices have moved as well as a metric based on the clockwise geometric ordering of the vertices are proposed. Further metrics are surveyed in Branke [1999]. Such metrics are rather basic indicators for changes in the layout and do not cover well mental map issues in real-world applications, where drawing conventions and semantic information may be important aspects. Freire and Rodríguez [2006] investigate the preservation of the mental map for interactive navigation using the focus+context approach. This approach relies on frequent changes of the viewpoint and therefore is susceptible to a disruption of the user's orientation. They classify approaches for mental map preservation according to three factors: the predictability of navigational actions, the degree of change from one view to the next, and the traceability of changes once they occur.

Bridgeman and Tamassia [2000] propose a framework for *difference metrics* for defining and validating metrics to measure the difference between two drawings of the same graph. They define the following six metric categories:

- Distance: metrics based on the distance between points or the distance points move between drawings.
- Proximity: metrics based on the nearness of points and the clustering of points according to the distance between them.
- Partitioning: metrics based on partitioning points according to measures other than proximity, e.g., identifying groups that have the same relative position in both drawings.
- Orthogonal ordering: metrics based on the relative angle between pairs of points.

- Shape: metrics based on the sequence of horizontal and vertical segments of the graph's edges.

- Topology: metrics based on the embedding of the graph.

This categorization builds upon and extends previously suggested criteria: Proximity, orthogonal ordering, and topology are the criteria already suggested by Misue et al. [1995] and Eades et al. [1991], distance was suggested by Lyons [1992], and shape tries to cover edge routing in orthogonal drawings similar to the approach by Brandes and Wagner [1998]. Bridgeman and Tamassia [2000] perform an experimental study and conclude that orthogonal ordering achieved the best performance, but is far away from optimal in capturing layout distances based on human judgment. As a further main result they suggest that combinations of these metrics may perform significantly better. In Bridgeman and Tamassia [2002], an extended experimental study is performed that investigates similarity for orthogonal drawings. Rotation issues are taken into account, and also the magnitude of difference is assessed by evaluating response times on comparison tasks.

Those metrics cover the changes between a pair of drawings, but there is no direct extension to a series of drawings, as in the offline drawing scenario, besides the pairwise evaluation between pairs of subsequent drawings. An additional aspect that is not covered by such metrics is the case where instead of performing elementary operations for a graph change, graphs are combined to form a union graph, as for example when a package vertex in a software diagram is expanded by the graph that represents the contained structures. Due to drawing style rules or rules based on the semantic, changes in the layouts might occur that are perceived as quite natural and do not disrupt the user's mental map.

Strategies to achieve a mental map preservation include either to reuse the given layout and adjust it at the location of change, or compute a new one with constraints that guarantee a certain stability. A lot of publications discuss approaches to preserve certain characteristics of a given layout during the computation of a new one (e.g., [Bridgeman et al., 1997, Brandes and Wagner, 1998, Brandes et al., 2002, Dwyer et al., 2009]), a few complexity results on underlying problems exist (e.g., [Brandes and Pampel, 2008]). Freire and Rodríguez [2006] discuss factors that affect the mental map preservation in interactive graph-based interfaces. For navigation in interactive Focus+Context graphs they identify three factors that affect the mental map: the predictability of navigational actions, the degree of change from one view to the next, and the traceability of changes once they occur.

Preservation of the mental map is not only necessary when graph changes occur, but also when a number of graphs need to be compared. In this case, use of constraints may allow the generation of an appropriate mental map that facilitates comparison. Sometimes two or more graphs with only small differences have to be displayed consecutively or simultaneously, e.g., when graph sequences represent time series, or several data sets have to be compared. In such cases, layouts that allow to match conserved characteristics and parts of the graphs will facilitate the analysis. See also Sections 3.2.1 for a use-case from biology.

Interaction operations may also include manual editing of the layout, introducing additional constraints for a subsequent automatic drawing to preserve them.

8.2 Navigation

It is always wise to look ahead, but difficult to look further than you can see.

WINSTON CHURCHILL

As the amount of information to be analyzed and explored gets larger and larger, it is impossible to visualize all data items at the same time. Practical applications may lead to hundreds of thousands of vertices. Even if it would still be possible to physically display medium sized data sets, the visualization would be of minimal use, as dense and cluttered layouts with a large number of overlaps and edge crossings are unavoidable for most real-world data. Hence there is a need to navigate through the data with only a moderate number of data items displayed at a time.

Navigation is sometimes specified as the change of the viewpoint or the position of an object in a scene (e.g., Munzner [2000]). We extend this specification to take into account that for huge data sets there is no pre-specified scene for the whole data set, and we would like navigation to include also the exploration of different parts of the data. Navigation concepts allow to move from a visualization of a subset of the data, the *focus region*, to a different visualization either of the same or of a different subset of the data. In the most basic case, these subsets will be closely related, for example in expand-and-collapse approaches one of the subsets will contain the other. A view on the focus region is often complemented with additional contextual information, as for example an overview window that shows a larger part of the data or an abstraction. Efficient navigation methods have to be combined with concise visualizations for the focus region and smooth transitions at region changes to allow a simple, fast, and valuable exploration process. Use cases for such an exploration process are not restricted to the case where a specific data item of interest has to be retrieved, but can also include a knowledge discovery process, e.g., to gain information on the characteristics of a data set or on the contained structural relations. We give an overview on the goals and concepts for interactive navigation in this chapter.

8.2.1 Goals for Navigation

Navigation should allow to explore and access the data without much effort. A key issue to achieve this is the orientation of the user within the data space. During an exploration process the user needs to have information about the location of the currently displayed data subset in the whole data set, and about its relations to the hidden parts of the data. This can either be done by using a *focus+context* approach which displays some kind of contextual information in addition to the data in focus, or by using a strict classification of the data for navigation. Navigation can be done directly along the structural information, e.g., along edges or paths, or over additional classification information, e.g., a hierarchical clustering structure.

A commonly accepted goal for navigation in graphs is stability, i.e., during navigation the drawing should not change too much. Stability of the drawing is however not the only aspect of stability for navigation. Even when a fixed layout of the whole graph is preserved during navigation, a change of the focus region might distort the user's orientation. Changing the viewport in a navigation process should therefore not result in a 'jump' but in a smooth transition.

A simple idea to cope with the layout stability problem would be to provide a fixed drawing of the graph and to show only small parts of it, but such an approach won't work in general, as the resource requirements will be too large, with respect to both computation time and space. In addition, the necessary structural information won't be available, either because it cannot be retrieved as a whole, or because the graph structure itself is generated on the fly corresponding to the user's input, e.g., by applying filtering techniques on data annotations during exploration. Approaches with static precomputed layouts also interfere with the idea of interactivity, where the user can select and modify views and also add constraints to adapt a layout to his needs.

A second goal for navigation is that the user is *guided* so that he can easily identify promising paths to follow. This requires that task information and value metrics are integrated in the objective function of the navigation approach.

8.2.2 Basic Concepts and State of the Art

There are a number of general concepts and techniques to handle navigation requirements. According to Eades [2010] only two basic operations need to be supported for the navigation of large graphs: Drill down and drill up, meaning expanding groups of vertices and collapsing them again. Eades states that the drill down operation should be triggered by the user, whereas the drill up should be automatically performed by the system. Such a concept may however lead to problems when multiple focii and their relation may be of interest. These two operations also only cover a part of the navigation issues: the change of the graph in the focus region with regard to an abstraction concept. Navigation concepts also need to consider additional aspects like the representation of contextual information or a change of the viewport. In the following, we will shortly discuss related techniques, for a detailed discussion see, e.g., Shneiderman and Plaisant [2009], Cockburn et al. [2009], Aigner, Herman et al. [2000].

Overview+Detail In an Overview+Detail approach, an abstraction of the whole data set, e.g., a bird's eye view, is visualized complementary to a detailed view of a data subset or region of interest, with a spatial separation between both views. A navigation operation allows to jump not just to a neighboring data set but to far regions, possibly disrupting the users orientation. This problem can at least partially be remedied by using animation techniques, zooming out before a move and zooming in to the focus region again.

Focus+Context Detail information and contextual information are visualized together in a single continuous view.

Pan-and-Zoom Zooming allows visualization of focus and context with a temporal separation, i.e. users magnify a subgraph for detail (zoom in) and demagnify for context (zoom out), panning allows to move the viewport.

Semantic zooming Whereas geometric zooming just changes the size of the viewport, semantic zooming changes the shape or context in which the information is presented. The level of detail information for objects is changed, e.g., by expanding the depiction of a biological reaction from an edge to a structure that also includes co-substrates and by-products.

Filter and Select Filtering allows to suppress the visualization of objects based on rules. Selection allows to assign a subset of objects a priority status, e.g., to highlight them or define them as targets for further action.

Distortion Spatial distortion techniques allow focus regions to occupy a larger area, for example fish-eye views magnify the focus region and display it within its surrounding context, which is distorted relative to the distance to the focus region.

According to Shneiderman and Plaisant [2009], there are three main features for navigation interfaces:

(1) rapid situation awareness through effective overviews,

(2) reduced number of actions to accomplish tasks,

(3) prompt, meaningful feedback for user actions.

There are several reasons for problems in navigation approaches: Panning over long distances is an annoying task that may make the user lose orientation. Zooming out to avoid this problem may hide details that are landmarks for orientation, especially when a path that is tracked is routed close to other vertices and edges, or has many crossings. Link sliding [Moscovich et al., 2009] allows to simplify the tracking by constraining motion to a single path, automated zooming can help to keep a balance between the moving speed and the zoom level [Igarashi and Hinckley, 2000]. Such techniques may also be enriched by using information on priorities for the objects (defining landmarks) and navigation paths to remedy a second problem: Often the focus region does not contain information on the data outside the focus region, and how the focus is embedded in it. A natural question then is how to represent landmarks within the focus view. A basic information that is of interest for navigation is the topological neighborhood of an object in focus, which can be dynamically brought into the focus to help identifying valuable navigation paths [Moscovich et al., 2009].

Early work for online animated graph drawing was done by Huang et al. [1998] and Eades et al. [1997]. Here, the scenario is that only small parts of the graph may be known and only a certain part is displayed on the screen. Navigation operations result in a transition from one subgraph to the next, which only differ by a small number of vertices between the two successive visible parts. A change of the visible part should then result in changes of the drawing that are perceived as a smooth transition, preserving the users mental map. The exploration of a graph is performed by moving from one *focus vertex* to the next, and the subgraph visualized is computed from the neighborhood of the sequence of recently visited focus vertices. A force-directed algorithm is modified to visually separate the neighborhoods of the focus nodes, using gravitational repulsion forces.

Eades and Huang [2000] present a force-directed model that allows to navigate clustered graphs. They try to restrict vertices to a cluster region by introducing a *virtual node* for each cluster, which is connected by *virtual edges* to the vertices of the cluster. Spring forces are then divided into internal, external and virtual springs to reflect the clustering structure.

Abello et al. [2004] use a technique that they call *compound-fisheye view* in combination with tree-maps to navigate large graphs. Instead of using a geometric distortion, the compound-fisheye uses a hierarchical clustering of the graph and displays regions far away from the focus with coarser representations from the cluster tree. Abello et al. [2006] present a large scale graph visualization system where the user navigates by iteratively expanding clusters. The authors use a force-directed algorithm [Barnes and Hut, 1986] for the expanded subgraph's layout, but their approach allows that an arbitrary layout algorithm can be used.

Gansner et al. [2005] propose a slightly related *topological zooming* method to visualize large graphs, including also an animation scheme for transition between successive views. They use a coarsening similar to the multilevel paradigm, where the merging is done based on the neighborhood in a proximity graph in addition to the graph's edges. After selection of a focal vertex, a graph representation is computed such that the focus area is represented in detail (i.e., using lower levels in the coarsening hierarchy), whereas more distant parts are shown in less detail. As the focus area would be very dense with such an approach, a distortion is used to get a better density distribution in the final layout. Auber et al. [2003] present an approach for the interactive navigation of small world networks, where the network is decomposed into a hierarchy of highly connected sub-networks. A force-directed layout algorithm is applied both in overview and detail visualizations.

8.3 Interactive Graph Layout

The changes to a graph or its underlying model, performed by either the user or the system, have to be reflected in the visual representation of the model. Not only need addition or deletion of objects be reflected by adding or deleting corresponding representatives, but also the layout of the graph needs to be adapted to respect the change in the graph structure or semantics. The distances of the objects for example may reflect the strength of the relation and strongly connected regions should be grouped close together in the layout. The addition of an edge might therefore connect regions that then have to be drawn near each other. The amount of change in the layout should be minimal with respect to the user's orientation within the diagram. The corresponding metrics were already discussed in Section 8.1, in the following we describe existing strategies that try to observe them for layout computation.

Already a lot of strategies exist for the mental map preservation, some of which are based on a specific layout method, whereas others are realized by independent techniques. Eades et al. [1991] are the first to describe and formally discuss the problem of keeping the user's mental map in an interactive drawing system. Their approach mainly aims at preserving the 'shape' of the diagram, modeled by proximity, orthogonal ordering, and topology. They state that in this case the user's mental map is hardly disturbed at all, and present several simple techniques for vertex overlap removal and focus representation. As already stated in Section 5.1.2, one of the first works on stability in automatic graph drawing was the publication of Böhringer and Paulisch [1990]. They call the issues related to user-defined constraints *structural stability*, and in addition also propose an approach to achieve stability in incremental layered drawings through constraints. For this *dynamic stability* problem, constraints are automatically generated after each automatic layout to fix the layout structure, and those constraints then are weakened in the vicinity of a subsequent graph change.

Diehl et al. [2001] and Diehl and Görg [2002] consider *offline* drawings of a sequence of graphs, i.e., when all changes are known in advance. In Diehl and Görg [2002], general adjustment strategies independent of a layout method are described. Diehl et al. [2001] describes *foresighted* layout, where a global layout for the sequence of graphs is computed based on the supergraph that consists of the union of the sequence graphs. To avoid unused space for graph objects that only occur in few steps, objects with disjoint life cycles are grouped together and may occupy the same layout position.

Storey et al. [1999] adapt a fisheye-view based technique to help in preserving the mental map. Independent of a specific layout method, their `SHriMP` technique grows a vertex of interest by pushing neighboring vertices outside. Storey et al. describe how to adapt the moves of the pushed vertices to maintain the stability criteria of Misue et al. [1995] separately. Lee et al. [2006b] integrate the distance metrics of Bridgeman and Tamassia [2002] into the cost function of a simulated annealing approach.

There is already a fair amount of work concerned with mental map preservation for orthogonal layouts. Besides the embedding, the bends on the edges and the angles between consecutive edges at a vertex are of major influence. Miriyala et al. [1993] discuss edge routing in orthogonal drawings for use in an incremental orthogonal drawing scenario and give a heuristic A^* algorithm that avoids vertex overlap and tries to minimize bends, edge lengths, and crossings. In Papakostas and Tollis [1996], Papakostas et al. [1997], and Papakostas and Tollis [1998] update techniques for orthogonal drawings are discussed. For several scenarios, a discussion of the different insertion cases is given in detail, and the performance is analyzed and evaluated. From a practical point of view, these techniques may quickly lead to a decline in layout quality, and may not allow to maintain compliance with additional constraints. Interactive Giotto [Bridgeman et al., 1997] is an algorithm for interactive orthogonal graph drawing within the No-change Scenario of Papakostas and Tollis [1996]. Brandes and Wagner [1997] propose a Bayesian paradigm, with a cost model that allows a trade-off between optimization criteria and the minimization of changes. Brandes and Wagner [1998] try to cover edge routing preservation in a computation of an orthogonal shape for 4-planar graphs. They suggest to use a penalized flow network that aims at preserving bend and angles. Biedl and Kaufmann [1997] propose an incremental drawing approach for orthogonal grid drawings of arbitrary graphs that works in linear time and guarantees upper bounds on the number of bends and the size of the grid, at the cost of a large number of crossings. Brandes and Wagner [1998] develop an approach for the dynamic orthogonal grid drawing of 4-graphs that preserves the embedding and penalizes changes in the orthogonal shape, leading to a weighted trade-off between the total number of bends and the number of changes. This approach is extended in Brandes et al. [2000a] to cope with high-degree vertices. Brandes et al. [2002] give an orthogonal drawing algorithm that uses a drawing 'sketch' of the graph as input. The algorithm produces an orthogonal drawing with few bends in the Kandinsky model and tries to preserve the general appearance of the sketch.

Frishman and Tal [2004] describe an algorithm that allows the dynamic drawing of clustered graphs, where after each graph update also the corresponding layout is updated. To minimize the change in the drawing, placeholder vertices may be inserted after vertex deletion to stabilize the cluster size, and the movement of 'old' vertices and clusters is restricted.

Wybrow et al. [2005] describe an efficient method to incrementally compute minimal length object-avoiding poly-line connector routings, allowing the use in an interactive environment. Dwyer et al. [2006b] show how an edge routing approach can be integrated into a force-directed approach using stress majorization.

Based on force-directed layout techniques, Frishman and Tal [2008] propose an algorithm for large graphs that uses a modification of the algorithm from Fruchterman and Reingold [1991] for execution on a GPU. Their measure of dynamic layout quality consists of the average displacement of vertices, and each vertex is assigned an individual convergence scheme with a pinning weight that controls the allowed displacement.

Cohen et al. [1992] describe a framework for dynamic drawing of planar graphs. Their model is based on a set of queries and updates that can be performed on an implicit representation of the drawing. These operations include drawing queries that return the drawing of a subgraph and update operations like vertex insertion and deletion and replacement of an edge by a graph. Cohen et al. distinguish between two types of quality measures in dynamic graph drawing with an inherent tradeoff in-between: aesthetic properties that have to be maintained, and the space-time complexity of queries and updates. A solution to the dynamic graph drawing problem then is an algorithm that maintains a drawing satisfying a static and a dynamic drawing predicate under a sequence of operations. The static drawing predicate expresses the aesthetic properties, e.g., 'The drawing is planar, straight-line, grid, upward, with $O(n^2)$ area'. The dynamic drawing predicate expresses similarity properties maintained between subsequent drawings, 'The drawing of a subtree is not affected by the update up to translation'. Techniques with poly-logarithmic query and update time are given that maintain drawings which are optimal with respect to sets of aesthetic criteria for rooted trees, series-parallel graphs, and planar st-digraphs.

8.3.1 Animation

Animation is used to allow a smooth transition from one layout to the next in an interactive or dynamic drawing context. Vertices are moved to their new position through a sequence of intermediate images called *frames*. The changes over consecutive frames then should be perceived as movements of the vertices by the human brain. A direct linear interpolation is the most simplistic form of such an animation. Clearly, such an animation is in general not a good choice, as it may introduce a large number of crossings during the move and also multiple vertices may occlude each other when they pass through a single position at the same time. An animation needs to communicate changes in the graph structure and move prominent substructures during the move in a way such that they stay recognizable. Such a behavior is in conformance with the *common fate law*, which states that objects that move together are perceived to also belong together. Besides layout issues, techniques like fade-in/fade-out can be used to highlight changes in the graph, so that the observer has some time to recognize the change. A simple principle to help the user track the layout transformation is slow-in slow-out animation, where near the beginning and the end of the movement more frames are used. The increased time to accelerate and slow down the movement helps in recognizing transition paths.

Friedrich and Eades [2002] and Friedrich and Houle [2001] describe models for good animations and techniques for smooth transformations between a source and a destination layout. The approach by Friedrich and Eades is based on the idea that an animation of substructures can be interpreted by the human brain as the movement of a three-dimensional object in space. The algorithm first divides the animation into a series of translation, rotation, scaling, and shearing operations. Applying an interpolation of these operations to the original graph provides a smooth animation to the final drawing. Some layout styles require special animation methods. The movement of vertices on a layer in a radial drawing to a new position may need a specific treatment to avoid a confusing animation. For example, the linear movement between the old and the new position on the same ring may mean that the vertex has to leave the ring first and then return to a position on the ring later. Two independent vertex movements may cross each other without need. The transition paths of many vertices may cross in the middle of the display causing the user to lose track of the different vertex paths. Yee et al. [2001] used a transition based on polar coordinates to avoid such problems. They state two transition constraints to maintain consistency between the old and the new layout, where a spanning tree is used as underlying structure. Pavlo et al. [2006] presented an approach that also allows to smoothly change to a spanning tree that also may be built from a different set of edges.

9. APPLICATIONS

Biologists currently waste a lot of time and effort in searching for all of the available information about each small area of research.

GENEONTOLOGY.ORG

We have seen several visualization approaches for data from various application areas in the previous sections. In practical applications, these visualizations need to be complemented with analysis methods, for example data clustering and classification, to structure the data and filter the substantial information out of complex data sets. To communicate this information in an appropriate way to humans, the visualizations have to be embedded in an interactive framework that supports the user's workflow and guides him during the knowledge discovery process. This process then can be directed by the user based on his expert knowledge, where decisions are not feasible for fully automatic analysis. The development of appropriate methods and tools is a challenge for current research in several application domains. The European *Innovative Medicines Initiative (IMI)* states that 'Gaps in information technology, and the lack of platforms to analyze large amounts of information in an integrated and predictive way is a major pre-competitive barrier in the current biomedical R&D process' [IMI].

As we will see in the following examples, the intuitive data representation combined with easy to use navigation methods may sometimes be more important than sophisticated graph drawing methods alone. We show applications from two different areas, biology and chemistry, that use interaction and drawing constraints to support the user's knowledge discovery process. Section 9.1 presents Scaffold Hunter, a tool for the analysis of chemical compound databases, Section 9.2 presents the integrative visual analysis of protein domain interaction networks, and Section 9.3 presents an approach to combine 3D and 2D structure analysis for protein residue networks.

9.1 Scaffold Hunter

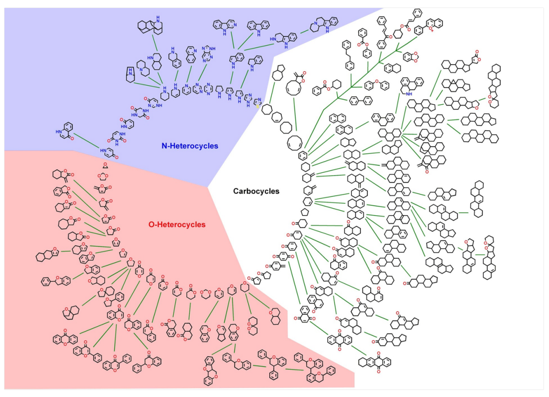
The search for a potential drug over large compound databases can be a tedious challenge, as the behavior and impact of a chemical compound cannot be easily predicted or derived from simple molecular properties. In addition, the effects of a compound may change significantly depending on several parameters like dose or the conditions of the organism at the time of application. The drug discovery process therefore involves decisions based on expertise and intuition of the experienced chemist that cannot be replaced by automatic processes. Nonetheless this process can be greatly supported by an intuitive representation of the available data and by navigation approaches that allow for organized exploration of chemical space.

Scaffold Hunter is a software tool for the exploration and analysis of chemical compound databases. It is designed to support the chemist in the search for drug candidates out of a large pool of compounds. Scaffold Hunter allows navigation in the chemical space spanned by the database compounds with the help of a hierarchical classification based on compound structure.

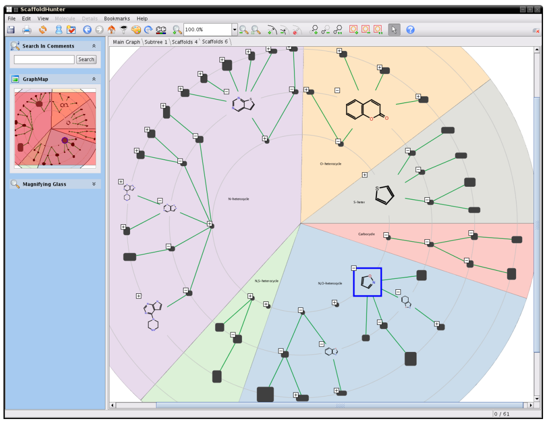
9.1.1 Motivation

In modern drug discovery and development, chemical synthesis is one of the key technologies. Due to the cost and effort involved in synthesis and experiments for the evaluation of drug candidates, efficient identification of promising test compounds is important for the development of new drugs. One of the main problems is that within the huge chemical space of synthesizable compounds (approximately 10^{60} molecules), there is only a small fraction of potentially active compounds of interest for further investigation. Even though the drug discovery pipeline, which aims at detecting small molecules that bind to protein targets involved in a disease process, involves high throughput screenings in the early stages to identify potentially active molecules, only a small part of this space can be covered by such a physical screening. The chemist's workflow therefore can be supported by automatic identification of regions that may contain good candidates with high probability and by enriching the navigation with pointers to these region within a visual exploration process.

Orientation within this chemical space is difficult, as on the one hand there is only partial knowledge about molecule properties, and on the other hand a large number of potentially relevant annotations exist, as physical and chemical properties, target information, side effects, and many more. Some of these annotations also may be either predicted with a certain confidence or result from experiments, with uncertainty and sometimes even contradicting information. Nonetheless, there are some approaches to classify and cluster compounds for navigation. A number of properties might be good indicators for drug-like molecule characteristics, and there are several physico-chemical properties that allow to discard molecules for reasons like bad oral availability, insufficient stability, or others. Prospective drug molecules need to show sufficient biological activity with respect to a protein target (or a small set of targets, which can sometimes be beneficial [Mencher and Wang, 2005]), but have to be selective enough not to show harmful side-effects, and need to have physical properties that allow them to be easily synthesized, sufficiently stable, as well as certain pharmacokinetical properties to reach the disease site, as, e.g., bio-availability (the fraction of an administered drug dose that reaches systemic circulation). At a later stage, also additional information like patent status might be of interest. This wealth of possibly relevant information make efficient drug testing difficult. In addition, although much of this information is publicly available, it is difficult to effectively utilize public domain resources for drug discovery research. Information is spread across many resources, access interfaces for these resources differ, and even the unambiguous identification of compounds is difficult as different methods are used to compute identifiers. Integration of these data resources in a visual analysis tool with an intuitive navigation concept facilitates drug discovery processes to a large extent.



(a)



(b)

Figure 9.1: Views on the hierarchical classification of molecules using the scaffold tree approach. (a) Manual drawing giving the inspiration for the Scaffold Hunter visualization. (b) Scaffold Hunter screenshot. A mapping of a scaffold property to the scaffold representation size is used, several scaffolds are enlarged, indicating extraordinarily large values.

9.1.2 Goals and Challenges

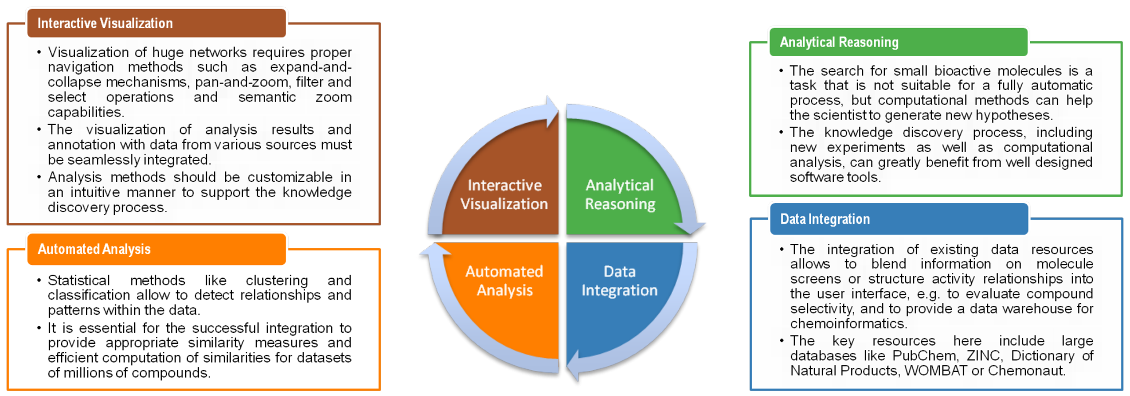
Our main goal in the development of Scaffold Hunter was to facilitate the interactive exploration of chemical space in an intuitive way. We wanted to develop and implement a software tool that integrates drug discovery data and allows to browse through the structures and data in an interactice visual analysis approach.

The development of Scaffold Hunter was motivated by a manually generated visualization of a structural classification of natural products in Koch et al. [2005], see Figure 9.1(a). Using this classification, the authors were able to 'chart the known chemical space explored by nature'. The combination of a simple organizing principle, which arranges representatives for subsets of molecules in a tree-like fashion, with the 2D depiction of chemical structures allowed an intuitive representation of a biologically relevant fraction of chemical space. However, this was only a static view for a specific task, and it required several days of work to manually create it.

Several goals guided the design and implementation of Scaffold Hunter:

- Our tool should allow to automatically create views that represent a space of chemical compounds in an intuitive fashion for chemists.
- The user should be able to integrate public data resources as well as his own compound databases.
- Interaction with the views should be possible to adapt them to the needs of a specific task, and to allow an analysis of the underlying data.
- Guided navigation within the compound space should be possible, to focus on regions of interest and to drill down to promising drug candidates.

When these goals are satisfied, the tool enables a visual analysis workflow that allows to efficiently identify drug candidates based on the combined information available. See Figure 9.2 for a model of this workflow.



Interactive Visualization

- Visualization of huge networks requires proper navigation methods such as expand-and-collapse mechanisms, pan-and-zoom, filter and select operations and semantic zoom capabilities.
- The visualization of analysis results and annotation with data from various sources must be seamlessly integrated.
- Analysis methods should be customizable in an intuitive manner to support the knowledge discovery process.

Automated Analysis

- Statistical methods like clustering and classification allow to detect relationships and patterns within the data.
- It is essential for the successful integration to provide appropriate similarity measures and efficient computation of similarities for datasets of millions of compounds.

Analytical Reasoning

- The search for small bioactive molecules is a task that is not suitable for a fully automatic process, but computational methods can help the scientist to generate new hypotheses.
- The knowledge discovery process, including new experiments as well as computational analysis, can greatly benefit from well designed software tools.

Data Integration

- The integration of existing data resources allows to blend information on molecule screens or structure activity relationships into the user interface, e.g. to evaluate compound selectivity, and to provide a data warehouse for chemoinformatics.
- The key resources here include large databases like PubChem, ZINC, Dictionary of Natural Products, WOMBAT or Chemonaut.

Figure 9.2: Interactive visual analysis of the correlation between chemical structure and biological activity. Graphic provided by Nils Kriege, used by permission.

Several challenges make a straightforward realization of these goals difficult:

- The set of chemical compounds under investigation may be huge—up to several million compounds. This raises both efficiency and visualization problems.
- In order to provide the user with sufficient information on the compounds, interfaces to quite diverse data resources have to be developed, including large online databases as PubChem, Zinc, or ChEMBL.
- Chemists are not used to advanced visual analysis concepts and only have moderate confidence in on-screen analysis so far.
- No visualization or navigation paradigms are established for the exploration of large chemical databases. We therefore need to develop new interaction and navigation concepts using visualization paradigms especially suited for the chemist's workflow. The interfaces and analysis methods need to be accessible for chemists without expert knowledge in cheminformatics, mathematics, or statistics.

9.1.3 Approaches and Use-Cases

In order to organize chemical space and to reduce the number of objects that have to be visualized, we use the *scaffold tree* approach of Schuffenhauer et al. [2007]. This approach computes an abstraction of the molecule structures that allows to represent subsets of molecules with single representatives for navigation. The scaffold tree algorithm is applied to construct a unique tree hierarchy, where for each molecule a representative underlying structure, the so-called *scaffold*, is computed based on a set of deterministic rules. In a step-by-step process, those scaffolds then are further reduced up to a single ring by cutting off parts that are

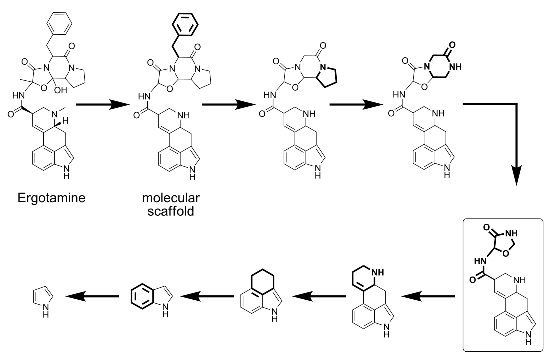


Figure 9.3: Creation of a branch in the scaffold tree: Molecules are reduced to scaffolds, and scaffolds are reduced by removing a ring structure in each step.

considered less important for biological activity. A hierarchy determined by the substructure relation is then defined for the resulting set of scaffolds. In this step a unique parent is selected. See Figure 9.3. The resulting set of trees is combined at a virtual root in a single tree which can be visualized using graph layouts.

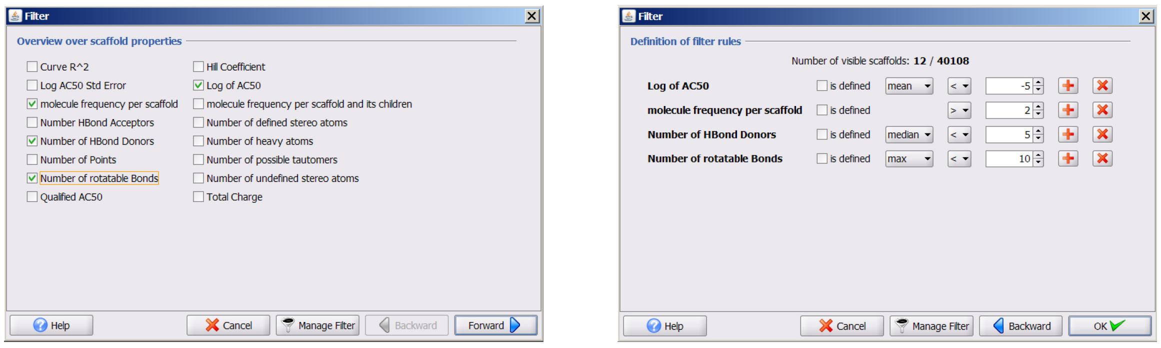
Each scaffold represents a set of molecules that are similar in the sense that they share a common molecular framework. Experimental results show that these molecules also share common biological properties, making the classification suitable for the identification of previously unknown bioactive molecules [Schuffenhauer et al., 2007]. As, depending on the task at hand, differing aspects may be relevant in order to measure molecule similarity, the user can customize the rules for scaffold tree generation.

Classification by scaffold hierarchies also facilitates *Scaffold Hopping*, an established technique in drug discovery, where one tries to find new bioactive molecules by starting at a molecule with known activity, and modifies the central structure.

The classification into scaffold trees leads to a strong reduction of the number of objects that have to be visualized. A flexible filter mechanism allows to further reduce the number of visible scaffolds in a way that is suitable for the typical drill-down approach in a chemical workflow. Filter rules can be created based on all scaffold properties deposited in the database, see Figure 9.4.

We decided to represent scaffolds using a 2D structure, as the 2D information is sufficient for a good estimation of the chemical behavior for the purpose of classification and search of drug candidates. The combination of this 2D visualization with the classification of the molecules enables the chemist to explore data in an intuitive visual fashion by navigating along well-defined relations. Chemists are used to these 2D representations and 3D information is not needed in most of the use-cases. Integrating 3D facilities however would complicate the user interface as well as the implementation and maintenance effort, and would also increase the hardware resources needed.

Compounds in a chemical database will not completely cover the chemical space spanned by the created scaffolds. Scaffolds that are not a representative of molecules, but solely created during the scaffold tree reduction step, are nonetheless inserted into the visualization. These *virtual scaffolds* represent ‘holes’ in the database and may be of particular interest as a starting point for subsequent synthesis. They represent previously unexamined molecules



Overview over scaffold properties

☐ Curve R^2	☐ Hill Coefficient
☐ Log AC50 Std Error	☑ Log of AC50
☑ molecule frequency per scaffold	☐ molecule frequency per scaffold and its children
☐ Number HBond Acceptors	☐ Number of defined stereo atoms
☑ Number of HBond Donors	☐ Number of heavy atoms
☐ Number of Points	☐ Number of possible tautomers
☑ Number of rotatable Bonds	☐ Number of undefined stereo atoms
☐ Qualified AC50	☐ Total Charge

Definition of filter rules

Number of visible scaffolds: **12** / **40108**

Log of AC50	☐ is defined	mean	<	-5
molecule frequency per scaffold	☐ is defined		>	2
Number of HBond Donors	☐ is defined	median	<	5
Number of rotatable Bonds	☐ is defined	max	<	10

Figure 9.4: Filter dialogs in Scaffold Hunter: (a) This dialog allows to select properties to be used for filtering out of the set of all scaffold properties. (b) The dialog for filter rule definition.

that may for example exhibit higher potency.

Besides navigation and orientation, data integration and efficient automated data analysis are key challenges. We developed a generic interface layer that allows to integrate interfaces to online resources or external databases. The integrated data then can be merged with data from various local sources. Additional descriptive statistical measures for the relevant structures are currently under development and will be provided in order to further guide navigation and to annotate the scaffold structures.

Even though chemical databases may contain millions of compounds, the interface capabilities are designed and restricted to the visualization of dozens to only several thousand compounds, as the visualization of all database entries as distinct entities at the same time is hardly ever of interest for chemists. Instead, two main scenarios are of interest:

First, an overview on the database contents is needed, both for evaluation and for comparison. Applications include visualization of several data sets at the same time, for instance results of several assays that have to be compared, or data sets stemming from multiple, heterogeneous databases where the covered region of chemical space, and also spots that are not represented in the databases are of interest. A typical use case is to compare an internal and a commercial database to see to what extent purchasing would increase the coverage of promising chemical space.

Second, another main task is the search for biologically active molecules that may be promising for synthesis to check suitability as potential drugs. Here, spots of large potential biological activity have to be identified. Note that biological activity for the largest part of the chemical space is not known, as the molecules are not tested or not even synthesized, but can only be derived indirectly, e.g., from the values of similar molecules with known activity. In addition, there are also many other required or desired properties, as for example synthesizability or bio-availability, which need to be estimated, and are often approximated best by experienced chemists. The visual representation of the known data combined with the navigation along paths of structural similarity help to gain insight into patterns of the molecule properties, and to drill down the data to small molecule sets that are especially suited as drug candidates.

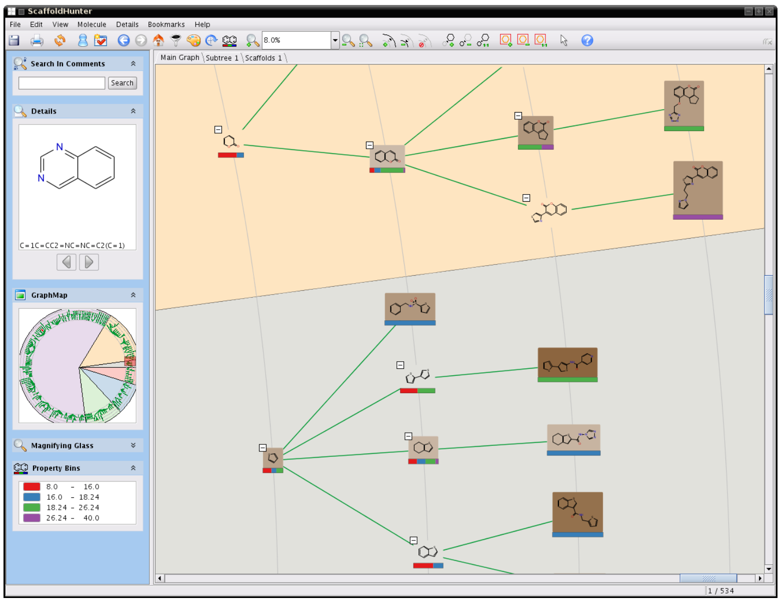


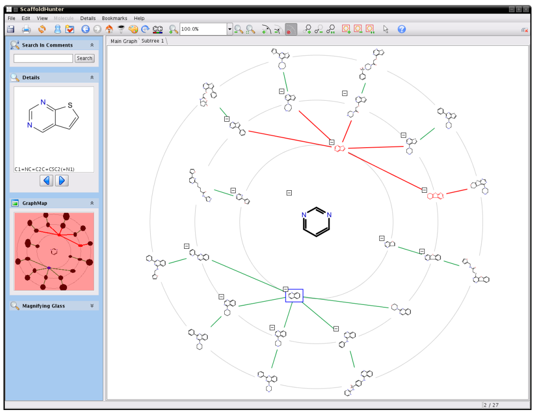
Figure 9.5: Close-up view of shaded scaffold tree visualization, with property bins that show the distribution of a value among the molecules under each scaffold.

9.1.4 Interaction and Navigation Paradigm

Based on the scaffold classification concept, Scaffold Hunter's main view represents the scaffold tree. The user can freely navigate in the view, as the user interface allows grab-and-drag operations and zooming. Zooming can be done either manually in direction of the mouse cursor, or automatically when the user switches between selected regions of interest. The system then moves the viewport in an animation to the new focus region, first zooming out automatically to allow the user to gain orientation. At the new focus region, the system zooms in again. For realization of the Overview-plus-Detail concept, we implemented a minimap. The minimap shows the whole scaffold tree and the position of the viewport and allows to keep orientation even at large zoom scales. Both the main view and the minimap allow pan-and-zoom operations. See Figure 9.5.

On startup, a user-defined number of levels is shown in the radial layout style (see Section 9.1.5). An expand-and-collapse mechanism is implemented that allows the user to either remove unwanted subtrees from the view or to explore deeper into subtrees of interest. In a sequence of filter and selection methods the user can define scaffold and molecule subsets of interest that can be independently visualized and analyzed.

The layout of the classification tree is always centered at the virtual root. We decided not to allow the selection of a new root for the following reason: As drug candidates need to meet certain requirements regarding their biological activity and bio-availability, it will rarely be necessary to explore trees over more than a few (< 8) levels. The molecules on deeper levels will be too large and have to many rings to be relevant for further consideration. However, Scaffold Hunter allows to open additional tabbed views, where subsets of the scaffold tree can be visualized centered on a scaffold of interest, as shown in Figure 9.6(b). Such detail views of filtered subsets or subtrees help to analyze and compare multiple sets with different


(a) Subtree Layout

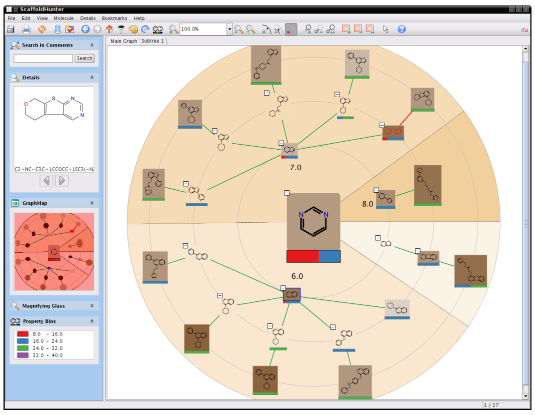

(b) Sorting and Color Shading

Figure 9.6: Subtrees rooted at a scaffold of interest can be visualized in separate tabs. Color shading allows to highlight scaffolds with desirable properties, as, e.g., biological activity. A sorting with respect to a scaffold property can be applied to define the order of the scaffold trees in clockwise order, a background color shading of the resulting segments reveals the corresponding distribution.

properties, see Figure 9.6(a).

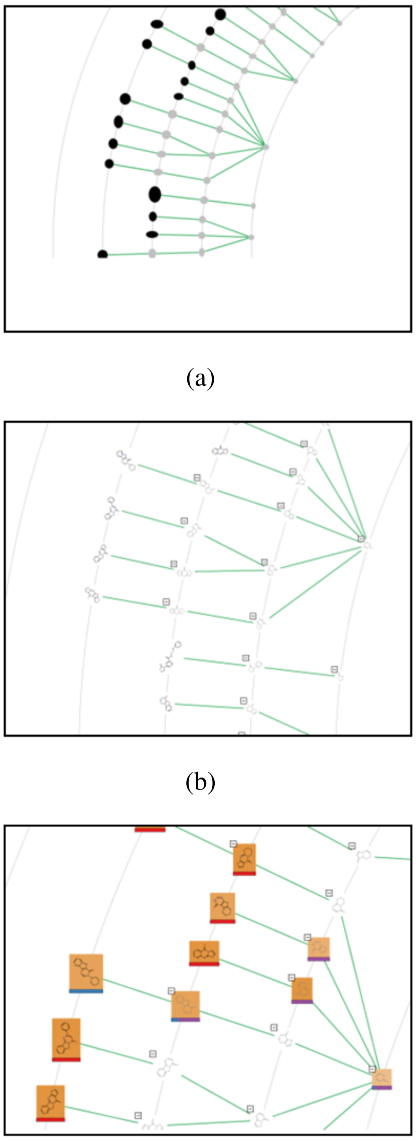
Complementary views show additional data, as, e.g., molecule sets, property spread sheets, or dendrogram views resulting from a hierarchical clustering. A detail window shows selected scaffolds together with a set of user-selected properties. This allows to quickly iterate through a selected set of scaffolds.

In order to guide the chemist in his search for a new drug molecule, combined information on the structure and the biological activity are included in the visual representation. Property bins can be defined to show the distribution of a selected molecule property for scaffolds. These property bins are displayed under the respective scaffold image and can help to indicate interesting subtrees for further exploration, see Figure 9.5. Property bins may optionally indicate the values of the molecule subset represented by a scaffold, or give the cumulated values of the subtree rooted at the scaffold. The cumulative view can help to select subtrees for deeper exploration.

The user interface provides detail on demand, passing the mouse over a scaffold of interest brings up a tooltip that shows in detail the scaffold structure and selected properties. Data annotations can also be represented by several graphical features: Activity data, or any other relevant annotation, can be blended into the representation using color shadings, see Figure 9.6(b). Scaffold structures, scaffold background, and canvas background can be colored according to such data. This allows both to get an overview on the distribution of annotation values and to focus on regions with specific values of interest. Annotation values can also be mapped onto the size of a scaffold representation, see Figure 9.1(b).

Bookmarks allow to persistently store and annotate specific scaffolds of interest.

All graphical views provide a kind of semantic zoom that increases the level of graphical data annotations with increasing zoom level. See Figure 9.7. During navigation in zoom out mode, structure information on scaffolds in the mouse pointer region is displayed in a magnifying glass window that can optionally be opened in the left side pane.



(a)

(b)

(c)

Figure 9.7: Increasing level of detail with semantic zoom. Simplified representative shapes (a) are first replaced by structure images (b), and finally the full set of currently selected data annotations is shown.

9.1.5 Layouts

There are several requirements for layout methods within Scaffold Hunter which result from the goals we defined for the application and also the approach taken. Minimum layout requirements are:

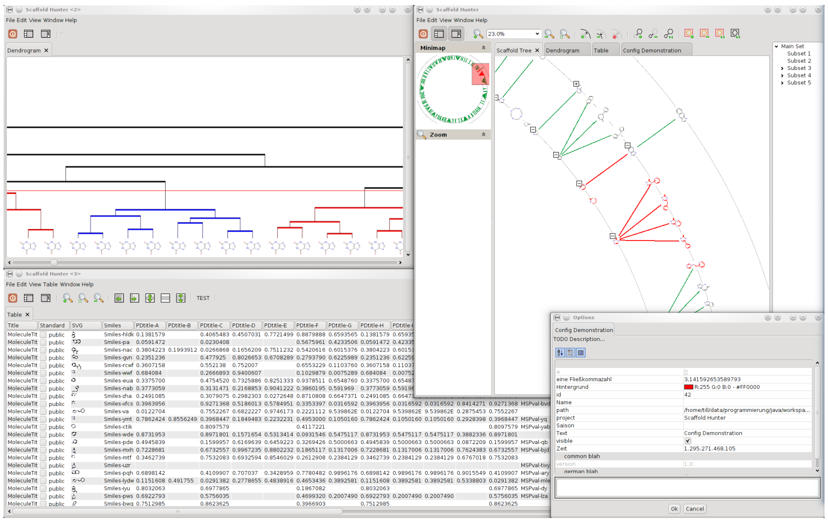
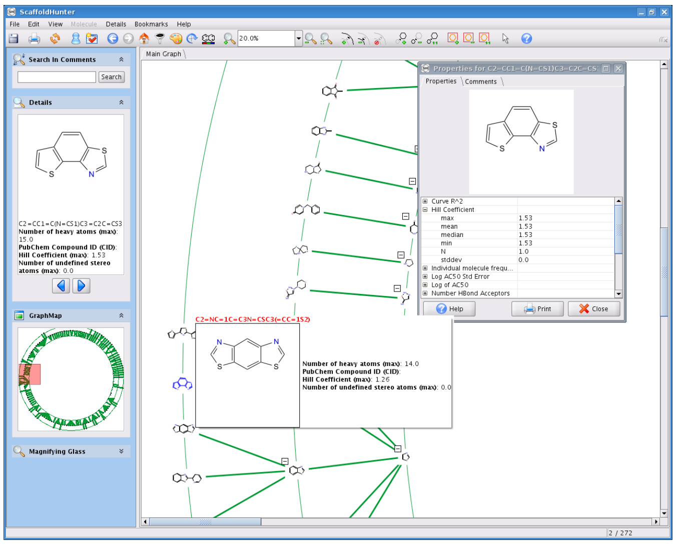


Figure 9.8: User interface of Scaffold Hunter. Several views show data and statistical analysis results complementary to the scaffold tree navigation. A dendrogram view shows results of a hierarchical clustering, and a spread sheet view shows a textual representation of molecule properties.



C2=CC1=C(N=CS1)C3=C2C=CS3
Number of heavy atoms (max): 15.0
PubChem Compound ID (CID):
Hill Coefficient (max): 1.53
Number of undefined stereo atoms (max): 0.0

C2=NC=1C=C3N=CSC3(=CC=1S2)
Number of heavy atoms (max): 14.0
PubChem Compound ID (CID):
Hill Coefficient (max): 1.26
Number of undefined stereo atoms (max): 0.0

Curve R^2	
Hill Coefficient	
max	1.53
mean	1.53
median	1.53
min	1.53
N	1.0
stddev	0.0
Individual molecule frequ...	
Log AC50 Std Error	
Log of AC50	
Number HBond Acceptors	

Figure 9.9: Detail on demand: The screenshot shows three ways to display detail information on scaffolds. 1) The 'details' view in the left sidebar displays a close-up view of the scaffold structure and properties for selected scaffolds. Using the buttons allows to iterate through the selected scaffolds. 2) Hovering over a scaffold image with the mouse pointer opens a tooltip with the same information. 3) A properties dialog can be opened from the context menu. It shows the full property information and the structure view for the scaffold. Comments can be entered that are persistently stored for the scaffold. The user can customize the first two views to show a selection of properties.

- Representation of the hierarchy: As we compute a scaffold hierarchy based on the scaffold tree algorithm, the layout has to reflect this hierarchy so that the hierarchical relations can be clearly identified. This includes both the top-down direction as well as a visual separation of different subtrees. Also the level to which a scaffold belongs has to be identifiable.

- Non-overlap: Edge crossings, vertex-edge or vertex overlap are unacceptable for scaffold tree visualization.

- Vertex sizes: Vertex sizes are determined by sizes of SVG images that depict the 2D scaffold structure. These vertex size have to be respected.

- Circular order: Chemists can sort the scaffold subtrees according to a scaffold property of choice, and the layout has to reflect this order. This edge order requirement is trivially supported by the implemented radial layout version.

Further criteria that can be used to evaluate the quality of the drawing are:

- Visual separation of the hierarchy levels. Levels should not be too close to each other, and subtrees should be visually separated.

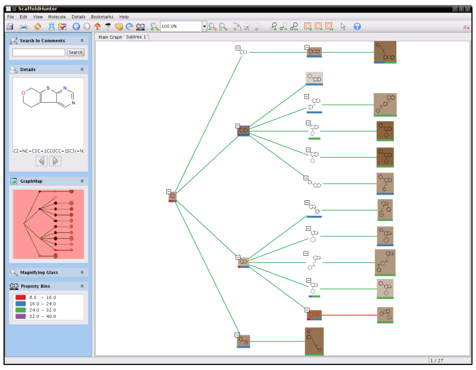
- Space utilization. The layout should not exhibit much white space.

- With respect to a given clockwise sorting and the requirement that subtrees and hierarchy levels have to be identifiable, vertices should be distributed nicely (uniformly).

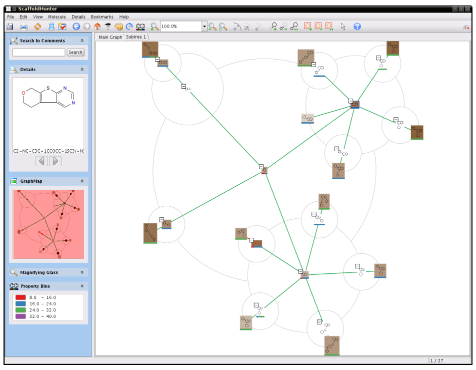
Several layouts are implemented to display the scaffold hierarchy, including radial, balloon, and tree layout. All of them easily allow to satisfy our edge order, distance, crossing restriction, and vertex size constraints. However, the space consumption is quite high. See Figure 9.10. The main layout is the radial layout. We give visual cues for the level affiliation of a scaffold by visualizing the radial circles as thin background lines. In addition, we use a dynamic distance between layers which is adapted according to the zoom level. This allows to achieve good separation of hierarchy levels and a clear depiction of the tree structure in lower zoom levels, whereas in close-up zooms scaffolds can still be represented together with at least one child level.

9.1.6 Realization and Further Features

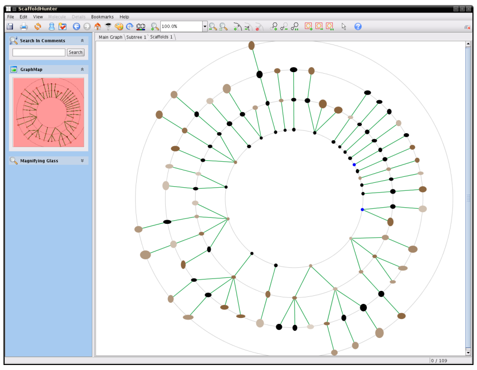
Scaffold Hunter was implemented as a prototype application with the features described as above in 2007 and then tested in practice. Since then the tool has been continuously maintained and updated. Scaffold Hunter is implemented in Java to allow platform independent use, and freely available as open source. We have developed new concepts to extend and improve Scaffold Hunter, stemming from the feedback of the user community and additional input of chemists from both academia and industry. A student project group is currently working on the implementation of several of these concepts; see Figure 9.8 for an example screenshot of the new, enhanced user interface.



(a) Tree layout



(b) Balloon layout



(c) Radial layout

Figure 9.10: Scaffold Hunter provides several layout algorithms, including radial, tree, and balloon layout.

A frequent task during the analysis of chemical compounds is the search for structurally similar compounds of a particular compound in a database, including molecules that contain the search compound or are contained by it. Web interfaces of public compound databases like PubChem therefore contain corresponding structure or substructure search facilities. Structures may be entered using a structure editor or established representations such as SMILES strings or the recently developed IUPAC International Chemical Identifier (InChI). We have implemented a fast graph-based substructure search approach tailored to the special characteristics of chemical molecules [Klein et al., 2011]. A structure editor allows to create search patterns graphically, alternatively also string identifier in standard formats can be used. See Figure 9.11. Scaffold Hunter also provides image export for presentation purposes in SVG, TIFF, and PNG format.

Data Integration

As already stated, chemical data on compounds is collected in different databases that are accessible over web or programmatical interfaces. Due to the sheer amount of data, data handling and integration must be organized in an efficient manner. Realtime processing of the complete data set is not feasible for interactive navigation. In the prototype system

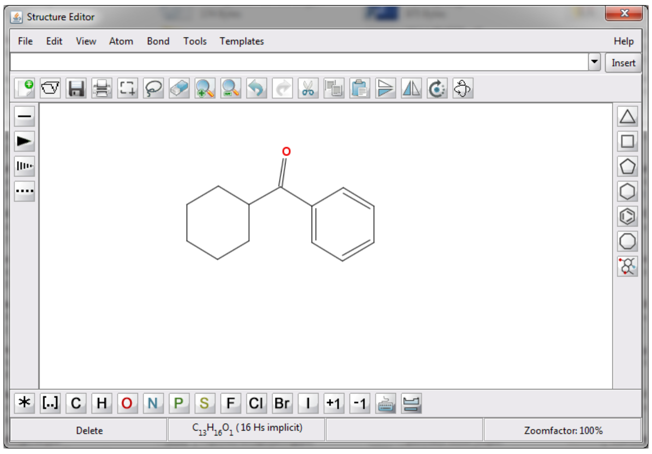

Structure Editor
File Edit View Atom Bond Tools Templates Help
Insert
O
C H O N P S F Cl Br I +1 -1
Delete
$C_{13}H_{16}O_1$ (16 Hs implicit)
Zoomfactor: 100%

Figure 9.11: Graphical structure editor for substructure search.

only static precomputation of the scaffold tree for a local database is implemented. We are currently working on the generic interface layer to allow the integration of arbitrary online or database resources. The new interface allows a flexible integration of resources, which can be dynamically changed when the user wants to select other data sources. Predefined interfaces and data mappings for main online resources will be provided.

9.1.7 Impact and Related Approaches

Compared to other application areas, especially biology, the support of the analysis workflow in chemistry by integrated tools that combine both advanced visualization and interaction as well as analysis methods is rather weak even though the need for such tools has been formulated quite often. Scaffold Hunter has been the first tool that allows to navigate in the hierarchical chemical space defined by the scaffold tree. Recently there have been further attempts to create new software to remedy the situation, namely the tools Molwind, SARANEA [Lounkine et al., 2010], iPHACE [García-Serna et al., 2010] and Scaffold Explorer [Agrafiotis and Wiener, 2010] which also aim at facilitating the visual navigation within chemical compound databases. The server-based tool Molwind, which has been inspired by Scaffold Hunter, has been developed by researchers at Merck-Serono. It is based on the World Wind concept by NASA and maps structural spaces to geospatial layers. However, the Molwind approach neglects the edges of the hierarchical graph and therefore does not contain all the information needed for navigation. Compared to these approaches, the web based tool iPHACE introduces basic additional features for visual analysis, namely interaction heat maps, and focuses on the drug-target interactions. iPHACE contains interaction data extracted from two databases (IUPHAR and PDSP), but lacks a sophisticated graphical navigation concept as well as integration of external data sources. The application SARANEA focuses on the visualization of structure-activity and structure-selectivity relationships by means of 'network-like similarity graphs'. The tool neglects structural classification schemes which are advisable for large datasets and is dependent on externally calculated fingerprints for the calculation of similarities. The most recent approach to provide tools for the analysis of chemical data sets is Scaffold Explorer, which allows the user to define the scaffolds with respect to his task-specific needs, but is targeted more towards the analysis of small data sets.

Although these first approaches received a very positive feedback from the pharmaceutical community, they are more or less in a prototypical stage and have only gained a small user base. The most likely explanation is that chemists first need to familiarize with such approaches, as there have not been established ways for the integrated visual analysis of chemical data so far.

9.1.8 Evaluation and Outlook

We have used Scaffold Hunter successfully in a study that proved the effectiveness of the approach and the usefulness of our implementation [Wetzel et al., 2009]. There is also already an active user community that provides valuable feedback. The main concept of bioactivity guided navigation of chemical space seems to be promising, which is also backed by recent results [Bon and Waldmann, 2010].

Nonetheless the software is still missing some features that would make it a valuable tool for a broader community, and there are also several possible extensions that would help to allow a smoother integration into a chemist's workflow. There are several workflows along the drug discovery process that are related, but require slightly different views on the data. Supporting these views in a more flexible setting, including also better analysis capabilities, could help to boost the use of Scaffold Hunter in real-world environments.

The issues that need to be considered for improvement include:

Usability. Chemists are not used to complex visual analysis tools so far. In order to stimulate use of such tools, established representations and interactions need to be included, also to increase user's confidence in automatically generated results. Visual representations like heat maps and dendrograms are already used and intuitively understood, but combination in an integrated interactive environment is not yet widespread. When multiple views of the data are provided, intuitive linking is of utmost importance for acceptance by chemists. Brushing and switching of views, e.g., from classification representations like dendrograms to spreadsheets, are intuitive actions in the chemist's knowledge discovery process, and need to be supported in a way that allows to keep the orientation. This also includes labeling of the views to be able to identify their source and how they were created (e.g. using the corresponding parameters and data subsets), links back to upper levels in a drill-down process to get back from dead ends and fathomed areas of the chemical space under investigation. Users suggested that the tool should provide simple annotation facilities for temporary and persistent labeling. This includes visual features like setting flags to support orientation when moving back and forth through several views. We are currently working on including these features in Scaffold Hunter. Several other tools allow to visually align 2D drawings of compounds, i.e., draw them such that their layout is most similar. This can be useful to compare compounds that share a pharmacophore, i.e., a set of structural features responsible for a molecule's biological activity, determined by the spatial arrangement of functional groups essential for the biological activity. Implementation of such a feature may further improve

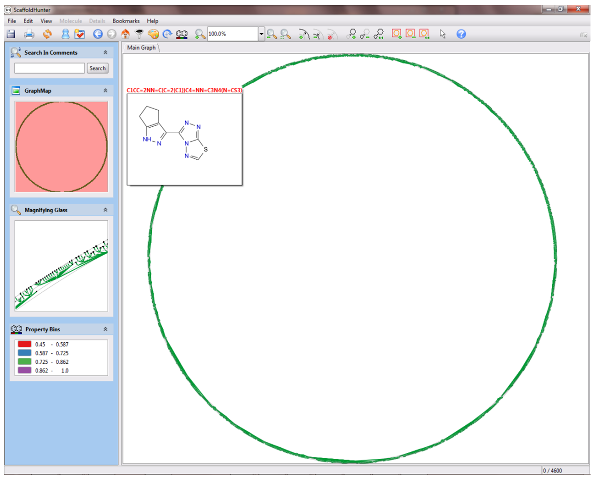


Figure 9.12: Bad radial overview layout - no structural information is conveyed to the user.

usability.

Layouts. Currently three layout methods are implemented, and the radial layout is the most intuitive for representation of and navigation in the scaffold tree. There are however several drawbacks. First of all, area utilization is poor when many molecules have to be drawn at the first level. See Figure 9.12. The overview then does not convey any information to the user. Specific tree layout algorithms like Walker's algorithm might prove useful as an alternative, even though they do not yield the nice concentric appearance of the radial layout. Recently, an extension of multidimensional scaling for radial constraints was presented [Brandes and Pich, 2009b], and we currently are working on a modification of this approach adapted to requirements in Scaffold Hunter. Several issues regarding a consistent distance model, overlap avoidance, and inclusion of fixed circular order of the scaffolds have to be considered to get satisfactory results.

Free area inside the inner circle could also be used to represent additional information, e.g., representatives of the segments resulting from sorting (as in figure 9.6(b)).

Classification. The navigation approach implemented in Scaffold Hunter is based on the tree-like structures of the scaffold tree classification. One the one hand, this makes orientation an easy task, on the other hand it might be an oversimplification for various use-cases, as information on molecule similarity is lost. Information on alternative parent molecules and structurally closely related molecules might allow for a more network-like structure without compromising the orientation too much. When ring-free molecules have to be investigated or functional side-chains are much more important than the scaffold structure, alternative classifications and navigation approaches might be better suited than the scaffold tree. We will therefore extend the scaffold tree approach to allow flexible generation of hierarchies, e.g., to take into account side chains and functional groups, which also facilitates the use of structure-activity relationships.

9.2 Protein-Domain Interaction Networks

Protein interaction networks are the essential underlying dynamic structures for many biological processes. Their visualization and analysis can therefore help to gain new insights into these processes and the underlying principles. However proteins are not the smallest biological unit that can be considered for investigation, as they are made up of amino acid chains and can be further subdivided into *domains*. A protein domain is a part of the protein sequence that can function and exist independently, and is often conserved over different species. Many proteins consist of several domains which may either act together or perform independent functions. The investigation of the corresponding domain interactions is of interest for a deeper understanding of biological processes. With a deeper insight in the way the interplay between proteins is disturbed during diseases, new ways to control or modify the behavior by new therapeutics can be developed. We have developed an interactive visualization for protein interaction networks that takes into account the domain substructures.

9.2.1 Problem Overview

All information required for the functioning of an organism is encoded in its genome. Genes in eukaryotic genomes are composed of sequences of *exons* (coding sequences) and *introns* (non-coding sequences). The encoded information is required for the construction of proteins. In the process of translating gene information into protein information, introns are removed and consecutive exons are joined together. This procedure is called *splicing*. Proteins play an important role in almost all biological processes performing a variety of functions critical for the viability of cells. Since most protein functions are based on molecular interactions between different proteins, recent research has focused on detecting protein interactions. Proteins frequently contain several domains, and the same domain can usually be found evolutionarily conserved within different proteins [Finn et al., 2008]. Regarding the interactions between proteins, it has been shown that protein-protein interactions are often mediated by the physical contact of protein domains [Albrecht et al., 2005]. In the last years, comprehensive studies of protein and domain interaction networks have produced large amounts of interaction data. This results in sizable amounts of experimentally derived protein interaction data, which is publicly available and complemented by even larger datasets of computationally predicted interactions. Since protein-protein interactions are mainly formed by specific domain-domain interactions, it is crucial to understand biological mechanisms that alter the domain structure or the domain composition of a protein. One such mechanism that commonly occurs in eukaryotic cells is *alternative splicing*. Alternative splicing is the process in which various exon combinations are joined together, leading to different protein isoforms obtained from one gene. Recent work revealed that alternative splicing is a major cause of the observed protein and interaction diversity [Matlin et al., 2005]. Assembling different combinations of exons can be responsible for changes in protein sequence and domain composition. In many cases, protein-protein interactions can thus be prevented or promoted by disabling or enabling certain domains. This is of great importance as some splice isoforms are also involved in human diseases. Special *microarrays* now allow for measuring gene expression at the level of the coding sequences of genes (*exons*), which makes it possible to identify alternative splicing events. With the so-called 'exon arrays',

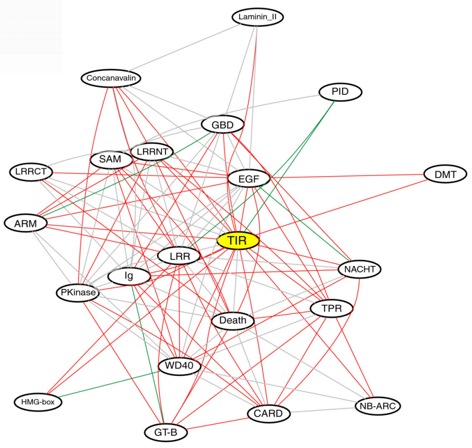


Figure 9.13: Example of a protein-domain network that shows the standard layout visualization commonly used [Zhang et al., 2008]. Proteins and domains are depicted and treated in the same way.

alternatively spliced proteins and their domain compositions can be detected. In addition, detecting differences between healthy and diseased cells concerning alternative splicing patterns becomes feasible, which might lead to the identification of potential targets for drug development. For a deeper discussion of the biological background, see for example Alberts et al. [2007].

For the visualization of protein interaction networks, several algorithms already exist that consider biological information for layout computation. Friedrich and Schreiber [2003] propose the animated visualization of typed interaction networks, where a subnetwork of proteins that support a specific, user-selected type of interaction is displayed using a force-directed algorithm, while the remaining proteins are placed on a circle around the drawing of the subnetwork. In order to reduce the number of crossings in the layout, Kato et al. [2005] add a crossing cost penalty to a grid-based layout algorithm, which can also deal with placement constraints to model sub-cellular localization information. Li and Kurata [2005] present a grid-based layout algorithm that allows clustering the objects in the drawing based on their membership in functional modules. None of these approaches consider the domain structure of the proteins. Even though the roles of proteins and domains can be distinguished, and the distinction is of interest for analysis of the interaction structures, proteins and domains are often treated equally for visualization purposes, see Figure 9.13. To visualize the impact of alternative splicing on protein interaction networks, we developed new graph visualization paradigms that respect the domain structure of the proteins. Our approach is implemented in the plugin DomainGraph for Cytoscape, an established open-source software platform for the analysis and visualization of biological networks. DomainGraph comprises two main functionalities: the decomposition of a user-imported protein interaction network into the underlying domain-domain interactions and the integration of exon array expression data for highlighting the effects of alternative splicing events. The resulting protein and domain networks can be visualized using our layout method.

9.2.2 Domain Graph Definition and Construction

The DomainGraph plugin is fully integrated into the Cytoscape application. In a first step, the user imports a file containing pairwise protein-protein interactions, specifies the species, and selects a suitable database containing domain-domain interaction information. Based on this information, the domain interactions underlying the imported protein interaction network are computed. This network is modeled as a graph as follows: We define a domain graph as an undirected graph $G = (V, E)$, where the vertices in the graph represent proteins and the constituent domains. For each protein and each domain, a vertex in the domain graph is created. Each edge describes either the occurrence of a specific domain in a protein (pd-edge), or the presence of an interaction between two proteins (pp-edge) or domains (dd-edge). The following definitions hold for a domain graph $G = (V, E)$:

- V_p is the set of protein vertices;
- V_d is the set of domain vertices, multiple vertices may represent the same domain iff it occurs in multiple proteins;
- V is the union set of protein and domain vertices;

  $V = V_p \cup V_d$
- E_{pp} is the set of pp-edges representing protein-protein interactions;

  $E_{pp} = \{(p_i, p_j) : p_i, p_j \in V_p \text{ and } p_i, p_j \text{ interact}\}$
- E_{dd} is the set of dd-edges representing domain-domain interactions;

  $E_{dd} = \{(d_i, d_j) : d_i, d_j \in V_d \text{ and } d_i, d_j \text{ interact}\}$
- E_{pd} is the set of pd-edges linking proteins to their constituent domains;

  $E_{pd} = \{(p_i, d_i) : p_i, d_i \in V \text{ and } p_i \text{ contains } d_i\}$
- E is the union set of protein-protein interaction and domain-domain interaction edges and protein-domain linkers;

  $E = E_{pp} \cup E_{dd} \cup E_{pd}$

9.2.3 Domain Graph Visualization

Several graphical attributes are used to allow an interpretation of the protein-domain network with regard to the vertex type and also the alternative splicing events. We assign characteristic shapes and colors for each type of vertex to distinguish between proteins and domains, and corresponding colors to the different types of edges. In addition, expression data information can be integrated into the visualization to represent specific splicing effects. After exon array data has been integrated into the domain graph, occurrences and effects of alternative splicing events as well as gene suppression events are highlighted. For each of the different events, a different visual style is defined that makes them clearly visible; see Figure 9.17(b).

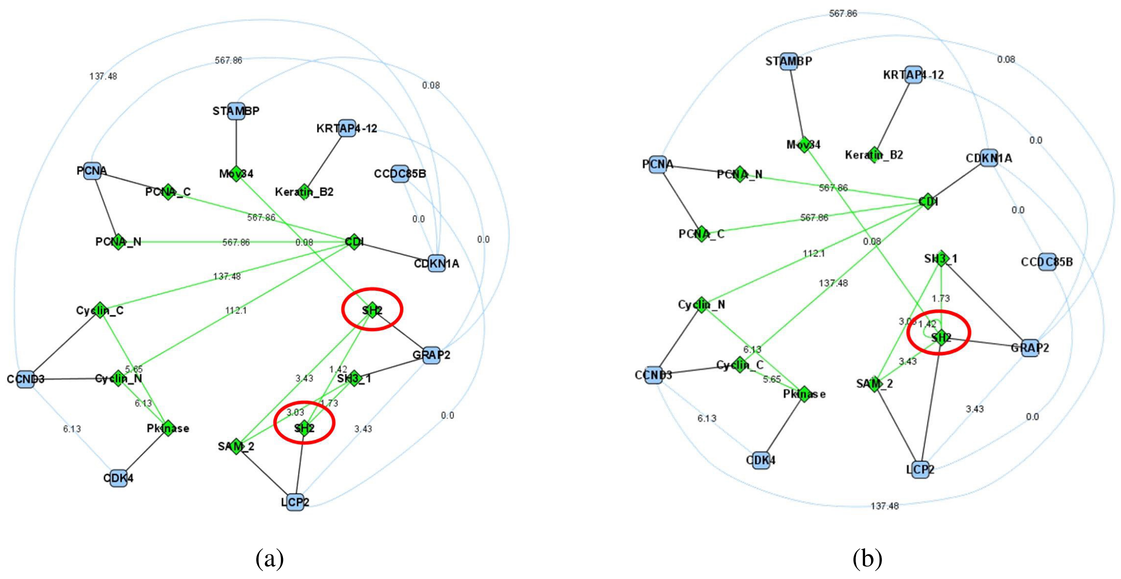


Figure 9.14: Domain graph in extended (a) and compact view after one compaction step, where the two SH2 domain vertices are merged (b). Protein vertices (squares) are on the outer circle, domain vertices (diamonds) on the inner circle. Edge label numbers denote confidence values.

DomainGraph provides three different network views, which can be changed at any time by the user to customize the domain graph to his needs. The extended view is the most detailed view displaying all constituent domains for each protein separately. Therefore, the number of shown vertices and edges is fairly high in the domain graph. In contrast, the compact view provides a domain graph with a decreased number of vertices and edges by merging identical domains contained in different proteins into a single meta-vertex, see Figure 9.14. Since a domain may occur in many different proteins, this view may reduce the number of depicted vertices and edges considerably. The third viewing mode is the protein network view, which by default shows only the vertices and edges of the protein interaction network. In addition, the user can select the proteins for which the underlying domain-domain interactions are visualized subsequently, see Figure 9.15. This third view has the advantage that the user can individually choose the size of the domain graph and is able to focus on protein interactions of special interest.

9.2.4 Splicing Pattern Analysis Methods

DomainGraph provides different methods for comparing domain graphs or calculating the intersection, union, or difference of pairs of domain graphs, see Figure 9.16. To this end, DomainGraph does not only take the structure of the domain graph and the two main classes of vertices, proteins and domains, into account, but also the types of the graph objects (for example, a spliced domain). Distinguishing these types is especially important for comparing domain graphs with integrated expression data. For instance, if a domain vertex is colored as being normally expressed in one domain graph and displayed as being spliced out in

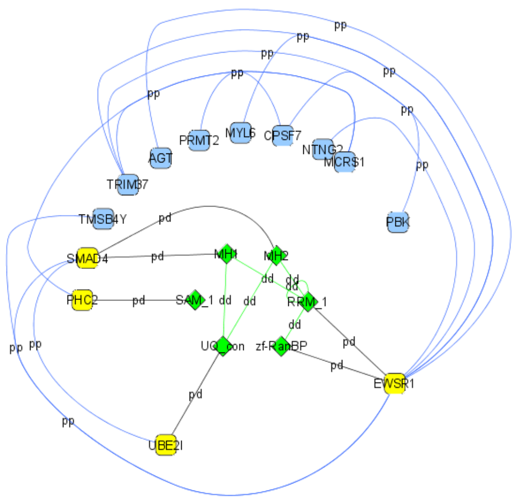


Figure 9.15: Domain graph in protein network view that shows additional domains and their interactions for the four selected proteins (yellow).

another graph, this vertex will be highlighted in the domain graph resulting from the analysis. Using such analysis methods, varying gene expression patterns in different tissues caused by alternative splicing events can easily be recognized. Moreover, variations due to different splicing patterns can be detected between healthy and diseased tissues.

9.2.5 Domain Graph Layout Requirements

Even though existing graph drawing algorithms like the automatic layouts provided by Cytoscape may be a good starting point for the development of methods for visualizing biological networks, they need to be adapted to reflect specific biological information in the computed drawing. Moreover, applying generic approaches to dense biological networks often leads to cluttered layouts with a large number of edge and object intersections. The results may be far away from biologically meaningful layouts that could help the biologist to investigate and evaluate experimental data, see Figure 9.17(a) for an example of a domain graph visualized using Cytoscape's force-directed layout. Several edges are concealed and it is difficult to recognize the structure of the domain interactions. In the case of a domain graph, the layout should allow a clear visual distinction between protein and domain vertices and the corresponding distinct interaction types. Although DomainGraph provides a graphical representation for the protein and domain vertices that separates them by their visual appearance, the application of generic network layout algorithms described above may make it difficult to quickly identify the topology of the protein and domain interactions. We therefore developed and implemented a radial layout algorithm that specifically takes the integrated biological information for the networks into account and emphasizes the corresponding topological characteristics. In our opinion, this improves the visual presentation of the protein domain graphs considerably.

A natural problem for the identification of protein and domain interaction partners and of structural interaction patterns are edge-edge and edge-vertex crossings and also overlapping

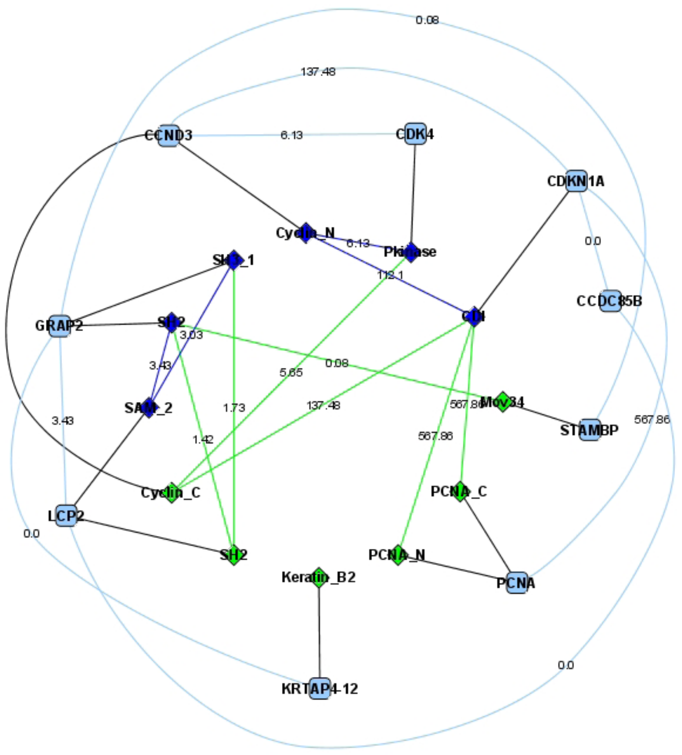


Figure 9.16: Intersection graph computed of two protein-domain networks. Vertices and edges contained in both domain graphs with different type annotation are colored dark blue.

vertices. A good layout thus needs a clear visual separation of the objects without occlusion, a minimization of edge-edge and edge-vertex crossings, a maximization of the angular resolution, and it should allow an easy identification of the different vertex and interaction types. In addition, protein vertices should be placed close to the domains they comprise.

9.2.6 Radial Layout Algorithm

Some of the layout aesthetics and optimization goals described compete with each other and therefore cannot be met simultaneously. As a compromise, we decided to implement an algorithm that combines elements of circular, radial, and layered layout algorithms, which we call RadialLayout.

In order to allow a clear separation of protein and domain vertices, proteins and domains are placed on separate concentric circles around a common center. The domain vertices are placed on the inner circle such that the domain interactions in the focus of the user's interest are located in the center of the drawing and proteins are placed on the outer circle as close as possible to their constituent domains see Figure 9.17(b). The use of separate circles for the two object types allows for the clear distinction between proteins and domains as well as between the interaction and linkage edge types in the drawing. A further separation of the domain vertices is achieved by highlighting the occurrences of alternative splicing events and grouping the domain vertices on the domain circle with regard to their splicing pattern.

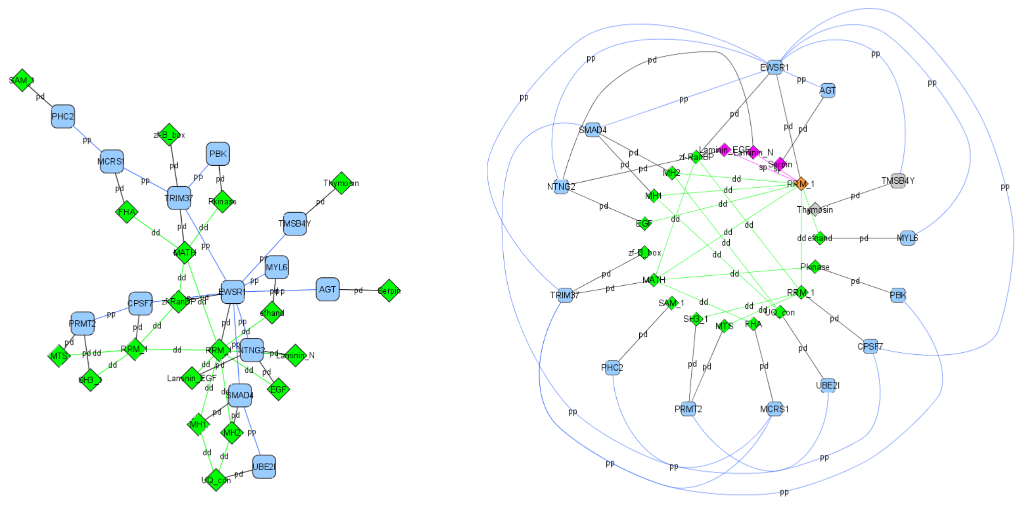


Figure 9.17: Comparison of domain graph visualization using Cytoscape's force-directed layout (left) and RadialLayout algorithm (right), with protein vertices represented by squares and domain vertices represented by diamonds. Domains affected by alternative splicing are colored pink, and domains forming interactions with spliced domains are colored orange. Missing proteins due to gene suppression are gray. Proteins and their constituent domains are placed close to each other. Edge crossings are minimized and pp-edges are routed around the protein circle.

In addition, the protein vertices are placed as close as possible to the domains they contain. This facilitates assessing the influence of alternative splicing events on the functioning of a certain protein and its interactions.

In order to reveal the interaction topology, the protein and domain vertices need to be positioned in a way that minimizes the edge crossings since they complicate the identification of corresponding connected objects. As the domain interactions often are in the focus of investigation, one could expect that circular crossing minimization could be applied. However, due to the special structure of the protein-domain networks this does not significantly improve readability, but leads to problems with the visual association of domains to their containing proteins and the length and crossings of pd-edges. These effects are much more problematic than the domain edge crossings, where interesting relation structures often still can be perceived well. We therefore decided to minimize crossings between pd-edges. This problem is similar to the crossing reduction in the bilayer crossing minimization, which has an important application in hierarchical graph drawing. The order of the objects on each of the two circles has to be fixed such that the number of crossings is minimized. The bilayer edge crossing minimization problem is NP-hard, but it can be solved to optimality for small instances in acceptable time, and heuristics have been developed that also perform very well on larger instances (see, e.g., Jünger and Mutzel [1997]). For DomainGraph, we decided to implement a heuristic in order to layout also larger instances of up to a few hundred vertices in reasonable time. The basic notion of our approach is to start with an initial permutation of the vertices on both circles. Then the order on one circle is fixed and the algorithm tries to

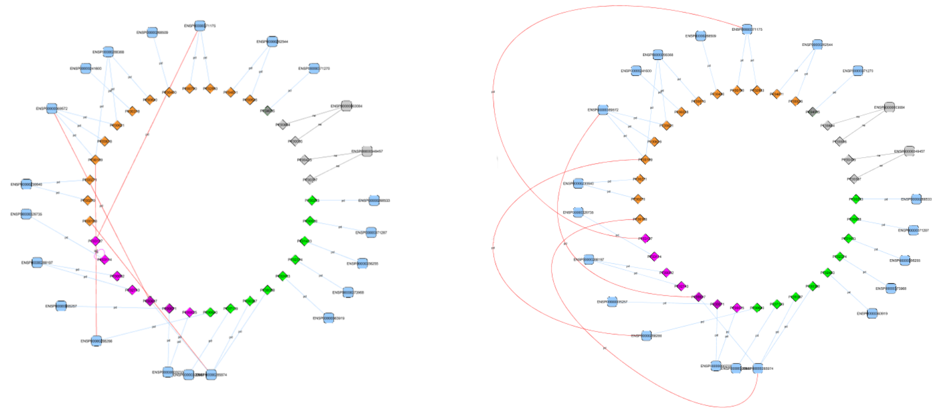


Figure 9.18: Handling of pd-edges crossing the inner domain circle (left) (dd-edges are hidden for this picture): Curved routing removes edge-vertex occlusion and helps to better recognize domain interactions (right).

find an order on the second circle with a decreased number of crossings. In our implementation, this is achieved by using an adaption of the well-known barycenter heuristic [Sugiyama et al., 1981]. This heuristic places each vertex at the barycenter of its neighbors on the second circle. This crossing reduction is iteratively performed with alternating roles of the two circles until the number of crossings cannot be reduced any further. The computation of the number of edge crossings in each step can be a bottleneck with regard to the running time [Waddle and Malhotra, 1999]. Therefore, we use an implementation of the bilayer cross counting method. This approach proved to be very fast in experiments [Barth et al., 2004] and has $O(|E|log|V_{small}|)$ asymptotic running time where V_{small} is the smaller set of protein and domain vertices. In case of expression data integration, the set of domain vertices is additionally split into subsets according to their vertex types, and the crossing reduction step starts with a permutation that reflects the grouping of the domains into domain types. Our implementation of the barycenter heuristic is then applied to arrange the vertices in each subtype group, minimizing the crossings in the drawing. The inner circle area is reserved for dd-edges, while pp-edges and pd-edges that cross domain vertices on the inner circle or in the inner circle area are routed as curved splines around the protein circle to further reduce crossings and to facilitate the visual recognition of the domain interactions, see Figure 9.18.

In summary, the main requirements met by RadialLayout are the visual separation of protein and domain vertices, the focus on domain interactions and the impact of alternative splicing, the avoidance of vertex occlusion, and the minimization of edge crossings in the drawing.

9.2.7 Interaction and Navigation Features

We implemented several features to improve the interactive exploration of the networks. The user may select interesting vertices for highlighting either by moving the mouse over a particular vertex or by clicking on several proteins to select a subset. When the user moves the mouse pointer over a protein or a domain of interest, the neighbors of the object and the cor-

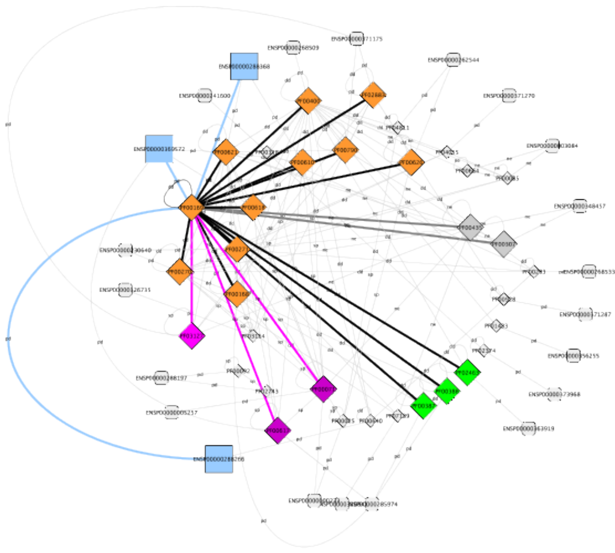


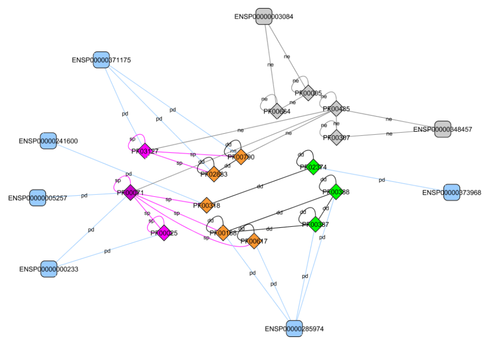
Figure 9.19: Dynamic highlighting of a direct vertex neighborhood. An adaption of the vertex positions is used to improve the layout of the highlighted subset. Unrelated vertices are grayed out.

responding links are highlighted, see Figure 9.19. Optionally, the user can select to either use color change, a change of the vertex sizes, or both for highlighting. The vertex positions can be adapted for such a highlighting view to improve the depiction of the related neighborhood with respect, e.g., to confidence values. The graph structure can also be dynamically compacted by use of the compact and protein network view. As shown in Figure 9.15, domains not contained in selected proteins are hidden, reducing the number of vertices on the inner ring and facilitating better recognition of the interaction structure. Tooltips in the graphics provide experimental and structural information. Additional graphical information is displayed if the user double-clicks on a protein vertex. The domain architecture of the respective protein is then shown together with the underlying exon structure. Moreover, all vertices are linked to the respective entries in the source databases giving more detailed information in a web browser. When double-clicking on a protein vertex, a graphical representation of the selected protein and its domain architecture is presented in a new window.

If the domain graph is derived from a dataset of predicted domain interactions, confidence scores are provided with the protein and domain interaction edges. These scores can be used for filtering. Interactions with a confidence score below a user-defined threshold can be discarded from the domain graph to focus on more meaningful interactions. Confidence scores of predicted domain interactions can be visualized by representing more confident interactions with wider edges.

9.2.8 Alternative Layouts

Because the circle structure of RadialLayout is not always well suited to quickly comprehend the decomposition of the interaction network with regard to the splicing types, we experimented with alternative layouts. These break the strict assignment to positions on the circle and allow groups of domains belonging to the same splicing event to move and shape ac-


(a)

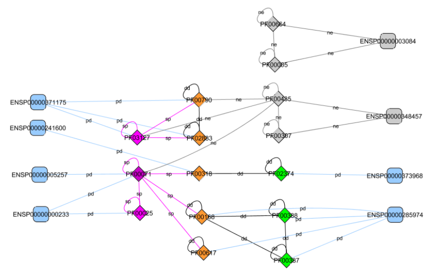

(b)

Figure 9.20: Alternative layouts: (a) The SemiCircle layout breaks the circular vertex configuration to improve the angular resolution of domain-domain interactions. It allows domain groups to bend stronger towards their neighbors. (b) The LinearLayout draws domain groups as lines instead of arcs, independent of their neighbors. It has a stronger focus on the proportions of the different domain types. They are therefore drawn closer together and aligned to make the size of each group visible.

cording to their neighborhood structure. The layouts differ in the way the groups may differ from the arc shape and circle positions, see Figure 9.20. We however avoided introducing completely different layouts, as the user should be able to switch from one layout style to another without losing his orientation.

9.2.9 Conclusions

DomainGraph is a powerful tool for the visual analysis of protein and domain interaction networks. Our visualization enables the straightforward identification of alternative splicing events and their direct and indirect impact on the protein interaction network, based on the integrated exon array data. Additional analysis methods that take vertex types into account are especially useful for recognizing expression differences, for instance, between tissues or healthy and diseased cells. For domain graphs, the development and application of our RadialLayout algorithm has shown to be a major advantage over conventional layout algorithms in practice because it is specifically designed for the integrated biological information and the visual differentiation of protein and domain interactions. Moreover, the crossing minimization of the edges aims at providing the best possible overview of the domain graph, its topology and interactions.

Clearly, as the crossing reduction only takes into account the protein-domain edges, the overall visualization could be improved by a combined crossing minimization of both domain-domain as well as protein-domain edges. Several approaches for the crossing minimization on a cycle exist [Baur and Brandes, 2004, Gansner and Koren, 2007]. There are also other ideas for further reduction of visual clutter, as, e.g., the techniques from Gansner and Koren [2007], but not all of them can easily be applied in our scenario. The idea to route some of the edges outside the circle is not applicable, as a specific type of edges, the pp-edges, are already routed outside the circle. As one of our main goals is crossing minimization, and at least a partial ordering of the vertices is already given, the idea to minimize edge lengths instead of the number of crossings is also not directly applicable. However, edge bundling might be helpful for large and dense graphs, and also an angular maximization approach might help to increase readability.

9.3 Visualizing Residue Networks of Protein Structures

In the last section we have seen how the subdivison of proteins into domains can be used in a visualization approach that allows a closer investigation of protein interaction structures. A protein, however, is not just a sequence of domains, but a chain (or a set of chains) of amino acids that folds in 3D space into a characteristic structure. Properties of this structure influence the protein's behavior and function. The folding is determined to a large extent by the protein's *primary structure*, the sequence of amino acids. Side chains of amino acids linked with their neighbors in this sequence are called *residues*. Besides the sometimes quite complex folding in 3D, local segments of the protein's backbone may form quite regular structures, which can be categorized as either *helix* or *sheet*. The arrangements make up the *secondary structure* of the protein. Figure 9.21(a) shows helices colored in red and sheets colored in blue. The study of individual amino acid residues and their molecular interactions in protein structures is crucial for understanding structure-function relationships. Recent work indicates that residue networks derived from three-dimensional protein structures provide additional insights into the structural and functional roles of interacting residues. Our main contribution here is an approach to visualize the residue networks for a combined analysis of 2D and 3D structures. It is included in the new software RINalyzer [Doncheva et al., 2011], a plugin for the Cytoscape platform.

9.3.1 Motivation

In recent years, thousands of 3D protein structures have been determined by X-ray crystallography and NMR spectroscopy, and many more have been computationally modeled. Commonly, protein structures are analyzed in 3D molecular viewers, a large variety of which are now available [O'Donoghue et al., 2010]. Although 3D visualization still plays a vital role in the analysis of protein structure and function, complementary approaches based on 2D representations of residue interaction networks (RINs) have been proposed more recently [Csermely, 2008, Vishveshwara et al., 2009]. RINs decrease the visual complexity of 3D protein structures and allow focusing on individual residues and their molecular interactions. Basically, a RIN is derived from the atomic 3D coordinates of a protein structure model that has been either determined experimentally or predicted by computational methods. Each RIN consists of vertices representing amino acid residues as well as edges corresponding to non-covalent interactions between residues, see Figure 9.21. Exploring and analyzing the network of interacting residues was shown to provide additional insights into the structural and functional role of residues [Csermely, 2008, Vishveshwara et al., 2009]. RINs can be applied to study residue interactions in a number of relevant application scenarios, for instance, regarding protein dynamics and engineering, folding, similarity, structure-function relationships, protein and ligand binding, and the impact of residue mutations. The network models of many proteins show small-world characteristics, i.e., most vertices are connected over a short network distance. They contain a few hundred vertices which exhibit specific *motif* patterns, assemblies of a few network elements, and *modules*, densely connected groups of residues that often correspond to protein domains. For a detailed discussion of protein structure network analysis, see Böde et al. [2007].

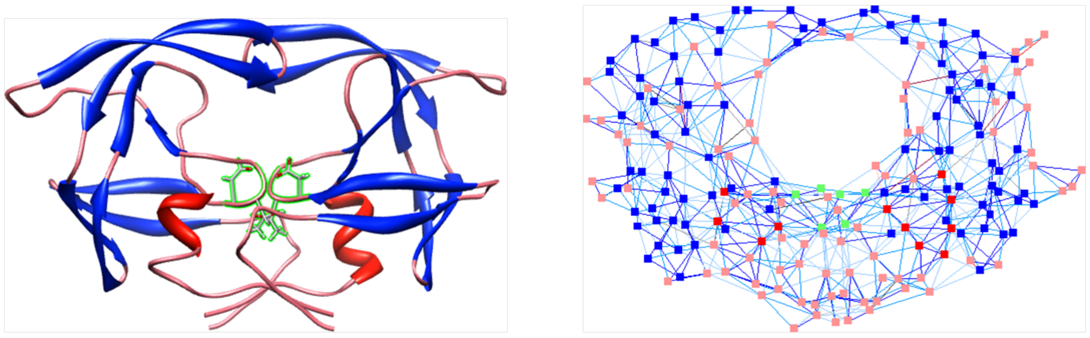

(a) (b)

Figure 9.21: (a) A 3D structure representation of the HIV-1 protease in Chimera. Such a structure can be converted into a 2D network representation of residue interactions. (b) The corresponding 2D network representation visualized in Cytoscape. The RINalyzer tool allows to simultaneously explore both the 3D protein structure and the 2D network, where the 2D layout is automatically computed and tries to preserve the shape of the 3D structure.

9.3.2 Realization

Despite the many applications of RINs, 3D molecular graphics software for protein structures has not been combined yet with network analysis tools although a close integration would be very useful for the investigation of protein structure and function. Therefore, novel software is desirable that supports the simultaneous, interactive 2D visualization of a RIN together with the corresponding 3D protein structure. Such integration is provided by RINalyzer, a new software tool that provides versatile and interactive structure analysis tools for RINs and enables dynamic 2D and 3D views as shown in Figure 9.22. In this integrative approach, residue vertices selected in a RIN are automatically highlighted in the protein structure visualized by molecular graphics and vice versa. This approach also contrasts with the generation of merely static 2D representations of molecules and their interactions [Zhou and Shang, 2009]. Coloring is used to highlight different structural subunits and interaction types to facilitate the analysis. Edges are labeled with the number of interactions of the corresponding type.

RINalyzer is available as a plugin for the established network visualization platform Cytoscape. In general, this platform can easily be extended by plugins, many of which are developed by other users to provide additional software functionalities. For instance, the Cytoscape plugin structureViz supports the structural analysis of protein-protein interaction networks [Morris et al., 2007]. Like structureViz, RINalyzer links Cytoscape with the well-known molecular modeling system UCSF Chimera [Pettersen et al., 2004] for 3D protein structure visualization. RINalyzer is complemented by the RINerator module, which generates RINs from a 3D protein structure.

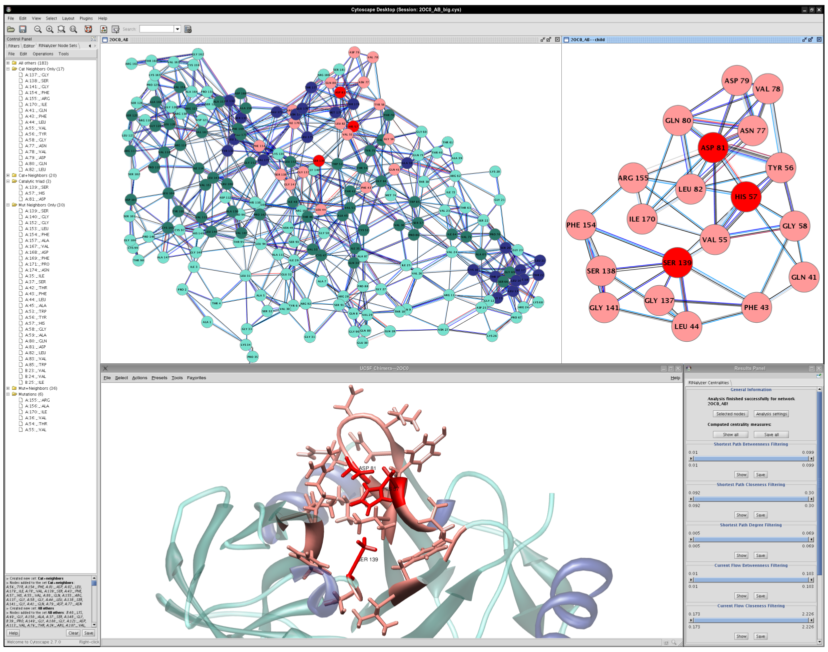


Figure 9.22: Simultaneous viewing of the 3D protein structure of the NS3-4A protease of the hepatitis C virus and the corresponding residue interaction network in 2D. The 2D layout and the 3D views are used complementary for interactive visual analysis of the protease structure. The molecular graphics visualization of the 3D protease structure is provided by UCSF Chimera. Different views of the corresponding residue interaction network are displayed by RINalyzer within Cytoscape, including overviews of the protease domain and the catalytic site. The vertex set interface (left) and the centralities panel (bottom right) of RINalyzer provide additional software functionality for the visual exploration of RINs and their topological properties. Picture kindly provided by Nadya Doncheva.

9.3.3 Network Layout

The simultaneous visualization of both 3D structure and 2D network should provide additional value to the user and make it possible to gain insight into structural and functional properties better than one of the views alone. Both views should complement each other, therefore the 2D layout should not just use the coordinates of the 3D structure information. Instead, our goal for the 2D layout is that it mainly highlights the network characteristics. However, the overall shape as well as the relative positions of characteristic substructures are constrained to be similar to the one in the 3D view. Requirements for the layout therefore include both geometric as well as topological constraints, the latter ones restricted to structural subunits of the residue network.

Our layout is based on the grid version of the Fruchterman-Reingold layout. Even though this algorithm is only suited for networks of medium size, compared to more recent multi-level layout approaches, we think it is responsive enough for RINs and often produces nice layouts. We tuned the algorithm to achieve layouts that allow to recognize the protein folding structure better and to separate the secondary structure parts. As the algorithm computes the layout based on the attraction of vertices connected by an edge and repulsion of vertices close to each other, we adjust the attraction and repulsion values with respect to the 3D layout given by Chimera and the protein structure. Vertices from the same part of the secondary structure repel each other less than vertices from different parts. If consecutive vertices from a helix or sheet are not directly connected by an edge, we simulate the attraction effect of the edge such that the vertices are not drawn too far apart from each other. We start the algorithm on a layout generated by a projection of the 3D structure positions to a 2D plane and anchor the vertices at the position given by the Chimera layout. The vertices then are allowed to move away from these positions depending on the force, i.e., anchoring is a soft constraint in our approach, such that the resulting layout is not too far apart from the view in Chimera, but the significant network relations have an adequate impact on the final layout.

Both views are linked, such that brushing and selecting in one view also leads to a selection in the other view.

9.3.4 Network Approaches to Protein Structure Analysis

The visual combination of RINs with 3D structures affords novel complementary methods to analyzing protein structures. One main feature of RINalyzer is the computation and illustration of a comprehensive set of well-known topological network centrality measures (based on shortest paths, current flows, or random walks) for relating spatially distant residue vertices and discovering critical residues and their long-range interaction paths in protein structures. Another software feature is the ability to perform comparative analysis of residue interactions by constructing a combined RIN that highlights the residue interaction similarities and differences between two proteins, see Figure 9.23. This enables the detailed comparison of residue interactions between homologous proteins or between different variants or mutants of a given protein. In addition, RINalyzer can be used in combination with other Cytoscape plugins such as NetworkAnalyzer for topological network analysis or MCODE and clusterMaker for finding network modules and clusters.

9.3.5 Concluding Remarks

We described innovative approaches to the investigation of complex protein structure-function relationships using RINs. In particular, RINalyzer complements current 3D structure viewers and modeling tools and enables the interactive, visual analysis of protein structures with RINs. Future extensions should aim at automating large-scale analysis and offering even more user options for structure exploration and visualization with flexible network layouts.

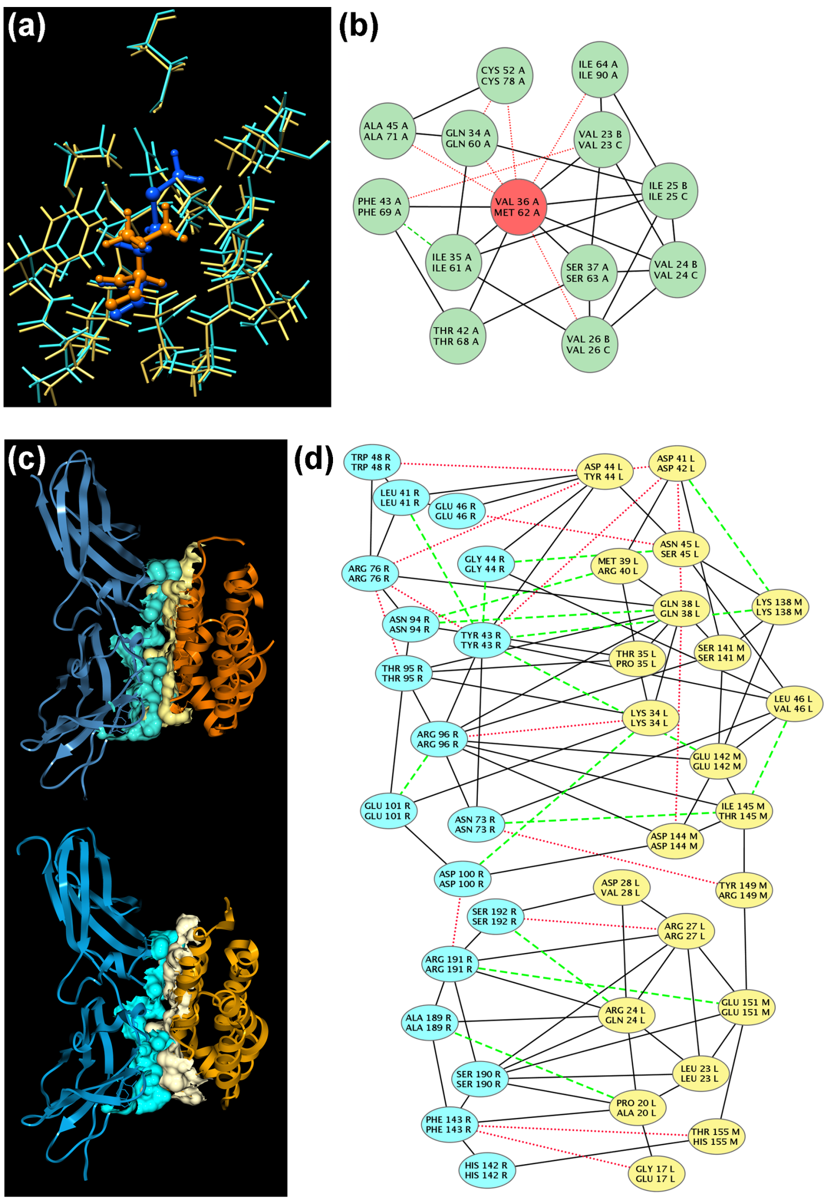


Figure 9.23: Comparison of protein structures using residue interaction networks. (a) A wild-type protease structure of the hepatitis C virus is compared with a variant structure associated with drug resistance. The superimposed structures are visualized in UCSF Chimera with focus on the mutation (wild-type in orange and variant in blue). The mutated residue is displayed using balls and sticks, its neighbors are shown as lines. (b) RINalyzer is used to construct a combined network for comparing the residue interactions of the wild-type and the variant structures. Solid black lines represent non-covalent residue interactions preserved in both structures, the green dashed line corresponds to an interaction present only in the wild-type structure, and the red dotted lines indicate interactions observed only in the variant form. (c) The interfaces of two protein complexes are compared in UCSF Chimera. (d) RINalyzer is applied to compare the interfaces between the two complexes using a combined network. Vertices corresponding to residues of the two complexes are colored blue and yellow, respectively. Solid black lines represent non-covalent residue interactions preserved in both binding interfaces, green and red dashed lines show interactions observed only in one interface.

10. CONCLUSION, FURTHER WORK, AND GENERAL REMARKS

Visualisation is an important weapon in the management and control of the vast flood of data now generated.

CLAIRE KNIGHT AND MALCOLM MUNRO

There is a huge demand for efficient interactive graph visualization approaches and tools in many application areas, and the gap between state-of-the-art findings in graph drawing research and the methods used in practice is quite large. On the one hand, it is difficult for practitioners to employ the latest approaches due to the lack of appropriate software libraries, on the other hand the methods developed by graph drawing experts still fail to meet all requirements needed in practical applications.

Graph drawing has come a long way from purely aesthetic layouts to sophisticated methods for some of the most complex problems motivated from practice. We are however still far away from solving all problems of practitioners. The evaluation of the proposed methods, if any, is mainly done on a few, ad hoc selected examples and rated from a graph drawing perspective. Studies on the practical impact, the robustness, and effectiveness of the developed methods for real-world scenarios are rarely done. Instead, often implementations are provided for purely experimental platforms.

Even though practitioners are willing to include graph drawing solutions into their own implementations, they often do not know how to realize the methods, or how to use and integrate existing graph drawing software. Easy-to-use interfaces and clear and well documented specifications need to be provided to overcome the lack of state-of-the-art graph drawing methods in most of the relevant tools from practice. Publicly available and free open-source graph drawing libraries that support both relevant exchange formats as well as established drawing notations will facilitate the use of graph drawing algorithms in real-world applications. However, such projects need to be reliably maintained and updated over a longer period of time to increase the users' confidence, allowing them to build upon graph drawing libraries to create long-term solutions. With regard to these points, our graph drawing library OGDF needs to be extended further. Support for notations as SGBN and UML, a simple interface that allows a mapping from drawing requirements to constraint specification also for non-experts in graph drawing, respective specifications how to store these constraints also with our file format OGML, and a support for a larger number of constraints in key layout algorithms are either under way or planned.

We investigated drawing requirements stemming from practical applications, and presented an extensive collection of these requirements together with a discussion of the corresponding data, tasks, and workflows. Our collection shows that most of the specific requirements from different application areas can be reduced to rather generic and application-independent requirements. These requirements can be mapped to drawing constraints which make an algorithmic treatment of the corresponding drawing problems possible. We gave a survey on the current state of the art regarding graph drawing with constraints, and presented a categorization of the constraints. An overview of the methods in graph drawing to cope with the related problems was presented, including interaction and navigation approaches as well as the integration of constraints into the optimization process.

For two important types of constraints–clustering and embedding constraints–we described models and solutions. For the case of cluster constraints we described an approach to compute the maximum c-planar subgraph, which is useful in the context of a clustered planarization approach. We made the branch-and-cut approach for the maximum c-planar subgraph computation publicly available in the open-source graph drawing library OGDF. This implementation is also useful for c-planarity testing of non-c-connected clustered graphs of small size, and constitutes the first practical c-planarity testing approach for general clustered graphs. As c-planarity testing is an important task independently of the maximum c-planar subgraph computation, we presented an improvement for the case of pure c-planarity testing which is based on a branch-and-price approach. Several concepts that exploit the specific structural relations in the c-graphs constructed during our approach are used to prune the search space. However, the complexity of c-planarity testing is still unknown, and there is a lack of significant classes beyond c-connected c-graphs for which efficient c-planarity testing can be done. In addition, theoretical foundations on the augmentation problem for non-c-connected graphs are required to advance on the path to a solution for the general testing problem. Rules to determine needed and unnecessary connectivity edges could greatly improve both the theoretical understanding and the efficiency of practical approaches such as our branch-and-price approach. An interesting field for further investigation might be the interplay between Kuratowski subdivisions and the cluster structure with respect to the necessary connectivity paths.

For the ec-embedding constraints, we developed a linear time testing and embedding algorithm and showed that the edge insertion problem can still be solved in linear time under these constraints. Even though in practice often all edges are subject to constraints and therefore our model is directly applicable, still the problem of free edges that are not constrained has to be solved. As these edges do not fit into our expansion-based planarity testing approach, other ways to consider them are needed.

We also described visualization paradigms for use within visual analysis approaches for biological and chemical data. An implementation of the navigation concept for chemical compound spaces is implemented in the Scaffold Hunter tool. Scaffold Hunter is an interactive exploration tool that allows data integration of multiple local and online resources and provides complementary, linked views on both aggregated navigation structures and the underlying data. Even though the tool already proved to be useful for real-world problem solutions, the implemented interaction, visualization, and navigation concepts need to be further evaluated in practice and refined to allow a smooth integration into the different task-dependent workflows of chemists. Data integration from various online resources is only partially achieved so far, but will greatly improve the tool's value. Integrated visual analysis of chemical data is just in the beginning, and there is still much work to be done to develop and evaluate better interaction and visualization paradigms.

A similar challenge arises for our visualization approaches for biological networks. Even though the quality of our protein-domain network layout clearly improves upon existing solutions, there is still much room for improvement with a focus on the integration of such layouts in an interactive visualization that fosters visual analysis and fits well into the biologist's workflow.

A further problem in practice is the handling of huge graphs. In recent years, a large number of drawing and navigation approaches for such graphs were proposed, where multilevel approaches seem to be the most promising technique. Although the methods proposed were successful in fast and high quality computation of layouts, the discussion about large graph layout is often decoupled from the considerations of real world requirements. If any, only clustering constraints are integrated in these approaches [Bourqui et al., 2007], as the cluster structure is also used for navigation purposes. A further investigation of both, new techniques for integration of more constraints, and also the underlying theoretical fundamentals, is needed.

BIBLIOGRAPHY

Graph drawing toolkit. http://www.dia.uniroma3.it/ gdt/gdt4/. 103

IEEE VGTC Pacific Visualization Symposium 2008, PacificVis 2008, Kyoto, Japan, March 5-7, 2008, 2008. IEEE. 239, 250

J. Abello, S. G. Kobourov, and R. Yusufov. Visualizing large graphs with compound-fisheye views and treemaps. In Pach [2004], pages 431–441. ISBN 3-540-24528-6. 193

J. Abello, F. van Ham, and N. Krishnan. Ask-graphview: A large scale graph visualization system. *IEEE Trans. Vis. Comput. Graph.*, 12(5):669–676, 2006. 193

D. K. Agrafiotis and J. J. M. Wiener. Scaffold explorer: An interactive tool for organizing and mining structure-activity data spanning multiple chemotypes. *Journal of Medicinal Chemistry*, 53(13):5002–5011, 2010. doi: 10.1021/jm1004495. URL `http://pubs.acs.org/doi/abs/10.1021/jm1004495`. PMID: 20524668. 209

A. V. Aho, D. S. Johnson, R. M. Karp, S. R. Kosaraju, C. C. McGeoch, C. H. Papadimitriou, and P. A. Pevzner. Emerging opportunities for theoretical computer science. *SIGACT News*, 28(3):65–74, 1997. 4

W. Aigner. Infovis wiki. `http://www.infovis-wiki.net/`. 191

T. Aittokallio and B. Schwikowski. Graph-based methods for analysing networks in cell biology. *Brief Bioinform*, 7(3):243–255, 2006. doi: 10.1093/bib/bbl022. URL `http://bib.oxfordjournals.org/cgi/content/abstract/7/3/243`. 32

B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell, Fifth Edition*. Garland Science, 5 edition, 2007. 213

M. Albrecht, C. Huthmacher, S. C. E. Tosatto, and T. Lengauer. Decomposing protein networks into domain-domain interactions. In *ECCB/JBI*, page 221, 2005. 212

M. Albrecht, A. Kerren, K. Klein, O. Kohlbacher, P. Mutzel, W. Paul, F. Schreiber, and M. Wybrow. On open problems in biological network visualization. In Eppstein and Gansner [2010], pages 256–267. ISBN 978-3-642-11804-3. 6, 25, 27

P. Aloy and R. B. Russell. Structure-based systems biology: a zoom lens for the cell. *FEBS Letters*, 579(8):1854 – 1858, 2005. ISSN 0014-5793. doi: DOI:10.1016/j.febslet.2005.02.014. URL `http://www.sciencedirect.com/science/article/B6T36-4FH0T2R-4/2/5c9588641ea59c66b2eb01d2a8ed6b49`. Systems Biology. 20

S. W. Ambler. *The Elements of UML 2.0 Style*. Cambridge University Press, Cambridge, UK, 2005. ISBN 978-0-521-61678-2. 44

Y. An, J. Janssen, and E. E. Milios. Characterizing and mining the citation graph of the computer science literature. *Knowl. Inf. Syst.*, 6(6):664–678, 2004. 40

P. Angelini, G. Di Battista, F. Frati, V. Jelínek, J. Kratochvíl, M. Patrignani, and I. Rutter. Testing planarity of partially embedded graphs. In M. Charikar, editor, *SODA*, pages 202–221. SIAM, 2010a. 83, 96, 97

P. Angelini, F. Frati, and M. Patrignani. Splitting clusters to get c-planarity. In Eppstein and Gansner [2010], pages 57–68. ISBN 978-3-642-11804-3. 2009. 133

M. Ankerst, S. Berchtold, and D. A. Keim. Similarity clustering of dimensions for an enhanced visualization of multidimensional data. In *INFOVIS '98: Proceedings of the 1998 IEEE Symposium on Information Visualization*, page 52, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-9093-3. 15

B. Aranda, P. Achuthan, Y. Alam-Faruque, I. Armean, A. Bridge, C. Derow, M. Feuermann, A. T. Ghanbarian, S. Kerrien, J. Khadake, J. Kerssemakers, C. Leroy, M. Menden, M. Michaut, L. Montecchi-Palazzi, S. Neuhauser, S. Orchard, V. Perreau, B. Roechert, K. van Eijk, and H. Hermjakob. The IntAct molecular interaction database in 2010. *Nucleic Acids Research*, 38:D525–D531, 2010. 22, 33

D. Archambault, T. Munzner, and D. Auber. Topolayout: Multilevel graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):305–317, 2007. ISSN 1077-2626. doi: http://dx.doi.org/10.1109/TVCG.2007.46. 66, 71

M. Ashburner. Gene ontology: Tool for the unification of biology. *Nature Genetics*, 25: 25–29, 2000. 23

D. Auber, Y. Chiricota, F. Jourdan, and G. Melançon. Multiscale visualization of small world networks. In *INFOVIS*. IEEE Computer Society, 2003. 193

G. J. Badros. *Extending Interactive Graphical Applications with Constraints*. PhD thesis, University of Washington, 2000. 80

S. T. Barnard and H. D. Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency - Practice and Experience*, 6(2):101–117, 1994. 70

J. E. Barnes and P. Hut. A hierarchical O(n-log-n) force calculation algorithm. *Nature*, 324: 446–449, 1986. 193

A. Barsky, T. Munzner, J. Gardy, and R. Kincaid. Cerebral: Visualizing multiple experimental conditions on a graph with biological context. *IEEE Transactions on Visualization and Computer Graphics*, 14:1253–1260, 2008. ISSN 1077-2626. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2008.117. 33, 34

G. Bartel, C. Gutwenger, K. Klein, and P. Mutzel. An experimental evaluation of multi-level layout methods. In *18th International Symposium on Graph Drawing 2010 (GD10)*, number 6502 in LNCS, pages 80–91. Springer-Verlag, 2010. 71

W. Barth, M. Jünger, and P. Mutzel. Simple and efficient bilayer cross counting. *Journal of Graph Algorithms and Applications (JGAA)*, 8(2):179–194, 2004. http://www.cs.brown.edu/publications/jgaa/. 219

V. Batagelj, H.-H. Bock, A. Ferligoj, and A. Žiberna. *Data Science and Classification (Studies in Classification, Data Analysis, and Knowledge Organization)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 3540344152. 15

C. Batini, M. Talamo, and R. Tamassia. Computer aided layout of entity relationship diagrams. *Journal of Systems and Software*, 4:163–173, 1984. 58, 67, 107

C. Batini, E. Nardelli, and R. Tamassia. A layout algorithm for data flow diagrams. *IEEE Trans. Softw. Eng.*, 12(4):538–546, 1986. ISSN 0098-5589. 61, 81

M. Baur. *visone - Software for the Analysis and Visualization of Social Networks*. PhD thesis, Universität Karlsruhe (TH), 2008. 37, 39

M. Baur and U. Brandes. Crossing reduction in circular layouts. In J. Hromkovic, M. Nagl, and B. Westfechtel, editors, *WG*, volume 3353 of *Lecture Notes in Computer Science*, pages 332–343. Springer-Verlag, 2004. ISBN 3-540-24132-9. 221

M. Baur and U. Brandes. Multi-circular layout of micro/macro graphs. In Hong et al. [2008], pages 255–267. ISBN 978-3-540-77536-2. 37

M. Baur, U. Brandes, J. Lerner, and D. Wagner. Group-level analysis and visualization of social networks. In J. Lerner, D. Wagner, and K. A. Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 330–358. Springer-Verlag, 2009. ISBN 978-3-642-02093-3. 37

M. Y. Becker and I. Rojas. A graph layout algorithm for drawing metabolic pathways. *Bioinformatics*, 17(5):461–467, 2001. 25, 85

C. Bennett, J. Ryall, L. Spalteholz, and A. Gooch. The aesthetics of graph visualization. In D. W. Cunningham, G. W. Meyer, L. Neumann, A. Dunning, and R. Paricio, editors, *Computational Aesthetics*, pages 57–64. Eurographics Association, 2007. ISBN 978-3-905673-43-2. 62

F. Bertault. A force-directed algorithm that preserves edge-crossing properties. *Inf. Process. Lett.*, 74(1-2):7–13, 2000. 83, 96, 97

F. Bertault and M. Miller. An algorithm for drawing compound graphs. In Kratochvíl [1999], pages 197–204. ISBN 3-540-66904-3. 66

P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. *IEEE Trans. on Comp.*, 49:826–840, 2000. 69

P. Bertolazzi, G. D. Battista, and W. Didimo. Quasi-upward planarity. *Algorithmica*, 32(3): 474–506, 2002. 82

T. C. Biedl and M. Kaufmann. Area-efficient static and incremental graph drawings. In R. E. Burkard and G. J. Woeginger, editors, *ESA*, volume 1284 of *Lecture Notes in Computer Science*, pages 37–52. Springer-Verlag, 1997. ISBN 3-540-63397-9. 194

T. C. Biedl, J. Marks, K. Ryall, and S. Whitesides. Graph multidrawing: Finding nice drawings without defining nice. In S. Whitesides, editor, *Graph Drawing*, volume 1547 of *Lecture Notes in Computer Science*, pages 347–355. Springer Verlag, 1998. ISBN 3-540-65473-9. 60, 101

C. Binucci, E. D. Giacomo, W. Didimo, A. Estrella-Balderrama, F. Frati, S. G. Kobourov, and G. Liotta. Upward straight-line embeddings of directed graphs into point sets. *Comput. Geom.*, 43(2):219–232, 2010. 95

BioCarta. BioCarta Pathway Project homepage. `http://biocarta.com`. 24

BioCyc. The BioCyc collection of pathway/genome databases. `http://biocyc.org`. 24

BioGRID. BioGRID is an online interaction repository with data compiled through comprehensive curation efforts. `http://www.thebiogrid.org/`. 22, 96

J. Blythe, C. McGrath, and D. Krackhardt. The effect of graph layout on inference from social network data. In Brandenburg [1996], pages 40–51. ISBN 3-540-60723-4. 40

C. Böde, I. A. Kovacs, M. S. Szalay, R. Palotai, T. Korcsmaros, and P. Csermely. Network analysis of protein dynamics. *FEBS Letters*, 581(15):2776–2782, 2007. ISSN 00145793. doi: 10.1016/j.febslet.2007.05.021. URL `http://dx.doi.org/10.1016/j.febslet.2007.05.021`. 223

H. L. Bodlaender and G. Tel. A note on rectilinearity and angular resolution. *J. Graph Algorithms Appl.*, 8:89–94, 2004. 100

K.-F. Böhringer and F. N. Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *Proc. of CHI-90*, pages 43–51, Seattle, WA, 1990. 81, 97, 194

R. Bon and H. Waldmann. Bioactivity-guided navigation of chemical space. *Acc Chem Res.*, 43(8):1103–14, 2010. 210

K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using P-Q tree algorithms. *Journal of Computer and System Sciences*, 13: 335–379, 1976. 108, 109

R. Bourqui, D. Auber, V. Lacroix, and F. Jourdan. Metabolic network visualization using constraint planar graph drawing algorithm. In IV2006, pages 489–496. 25, 85

R. Bourqui, D. Auber, and P. Mary. How to draw clustered weighted graphs using a multilevel force-directed graph drawing algorithm. In *IV*, pages 757–764. IEEE Computer Society, 2007. 85, 231

J. Boyer and W. Myrvold. On the cutting edge: Simplified $O(n)$ planarity by edge addition. *Journal on Graph Algorithms and Applications*, 8(3):241–273, 2004. 108

BPEL. Web services business process execution language version 2.0 OASIS standard, organization for the advancement of structured information standards (OASIS). `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`, 2007. 49

BPMN. Business process modeling notation (BPMN) homepage, object management group (OMG). `http://www.bpmn.org`, a. 2, 49

BPMN. BPMN example document, object management group (OMG). `http://www.omg.org/cgi-bin/doc?dtc/10-06-02`, 2010b. 50

F.-J. Brandenburg, editor. *Graph Drawing, Symposium on Graph Drawing, GD '95, Passau, Germany, September 20-22, 1995, Proceedings*, volume 1027 of *Lecture Notes in Computer Science*, 1996. Springer-Verlag. ISBN 3-540-60723-4. 236, 258, 262

U. Brandes and S. R. Corman. Visual unrolling of network evolution and the analysis of dynamic discourse? *Information Visualization*, 2(1):40–50, 2003. 39, 85

U. Brandes and B. Pampel. On the hardness of orthogonal-order preserving graph drawing. In Tollis and Patrignani [2009], pages 266–277. ISBN 978-3-642-00218-2. 100, 189

U. Brandes and C. Pich. An experimental study on distance-based graph drawing. In *Proc. Graph Drawing 2008*, volume 5417 of *LNCS*, pages 218–229, 2009a. 73, 84

U. Brandes and C. Pich. More flexible radial layout. In Eppstein and Gansner [2010], pages 107–118. ISBN 978-3-642-11804-3. 67, 73, 83, 97, 211

U. Brandes and B. Schlieper. Angle and distance constraints on tree drawings. In Kaufmann and Wagner [2007], pages 54–65. ISBN 978-3-540-70903-9. 97, 100

U. Brandes and D. Wagner. A bayesian paradigm for dynamic graph layout. In G. Di Battista, editor, *Graph Drawing*, volume 1353 of *Lecture Notes in Computer Science*, pages 236–247. Springer-Verlag, 1997. ISBN 3-540-63938-1. 194

U. Brandes and D. Wagner. Dynamic grid embedding with few bends and changes. In *Proceedings of the 9th International Symposium on Algorithms and Computation*, ISAAC '98, pages 89–98. Springer-Verlag, 1998. ISBN 3-540-65385-6. URL `http://portal.acm.org/citation.cfm?id=646341.686589`. 189, 194

U. Brandes, M. Güdemann, and D. Wagner. Fully dynamic orthogonal graph layout for interactive systems. Technical report, Universität Konstanz, 2000a. 194

U. Brandes, G. Shubina, R. Tamassia, and D. Wagner. Fast layout methods for timetable graphs. In Marks [2001], pages 127–138. ISBN 3-540-41554-8. 97

U. Brandes, M. Eiglsperger, M. Kaufmann, and D. Wagner. Sketch-driven orthogonal graph drawing. In *Proc. 10th Intl. Symp. Graph Drawing (GD 2002)*, volume 2528 of *LNCS*, pages 1–11, 2002. 189, 194

U. Brandes, T. Dwyer, and F. Schreiber. Visual understanding of metabolic pathways across organisms using layout in two and a half dimensions. *J. Integrative Bioinformatics*, 1(1), 2004. 85, 87

U. Brandes, C. Erten, J. J. Fowler, F. Frati, M. Geyer, C. Gutwenger, S.-H. Hong, M. Kaufmann, S. G. Kobourov, G. Liotta, P. Mutzel, and A. Symvonis. Colored simultaneous geometric embeddings. In G. Lin, editor, *COCOON*, volume 4598 of *Lecture Notes in Computer Science*, pages 254–263. Springer-Verlag, 2007. ISBN 978-3-540-73544-1. 84

J. Branke. Dynamic graph drawing. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs*, volume 2025 of *Lecture Notes in Computer Science*, pages 228–246. Springer-Verlag, 1999. ISBN 3-540-42062-2. 188

S. Brasch, L. Linsen, and G. Fuellen. Vanlo - interactive visual exploration of aligned biological networks. *BMC Bioinformatics*, 10(1):327, 2009. ISSN 1471-2105. doi: 10.1186/1471-2105-10-327. URL `http://www.biomedcentral.com/1471-2105/10/327`. 30

P. Braß, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. B. Mitchell. On simultaneous planar graph embeddings. *Comput. Geom.*, 36(2):117–130, 2007. 84

S. S. Bridgeman and R. Tamassia. Difference metrics for interactive orthogonal graph drawing algorithms. *J. Graph Algorithms Appl.*, 4(3):47–74, 2000. 188, 189

S. S. Bridgeman and R. Tamassia. A user study in similarity measures for graph drawing. *J. Graph Algorithms Appl.*, 6(3):225–254, 2002. 189, 194

S. S. Bridgeman, J. Fanto, A. Garg, R. Tamassia, and L. Vismara. Interactivegiotto: An algorithm for interactive orthogonal graph drawing. In Di Battista [1997], pages 303–308. ISBN 3-540-63938-1. 189, 194

C. Buchheim, M. Jünger, and S. Leipert. Improving Walker's algorithm to run in linear time. In Kobourov and Goodrich [2002], pages 344–353. ISBN 3-540-00158-1. 67

C. Buchheim, M. Jünger, A. Menze, and M. Percan. Bimodal crossing minimization. In Chen and Lee [2006], pages 497–506. ISBN 3-540-36925-2. 82, 83

ChEMBL. ChEMBL database, european bioinformatics institute (EBI), part of the european molecular biology laboratory (EMBL). `https://www.ebi.ac.uk/chembl/`. 15

C. Chen. *Information Visualization: Beyond the Horizon*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 184628340X. 3

D. Z. Chen and D. T. Lee, editors. *Computing and Combinatorics, 12th Annual International Conference, COCOON 2006, Taipei, Taiwan, August 15-18, 2006, Proceedings*, volume 4112 of *Lecture Notes in Computer Science*, 2006. Springer-Verlag. ISBN 3-540-36925-2. 238

N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *Journal of Computer and System Sciences*, 30(1):54–76, 1985. 108, 114, 122, 131

M. Chimani and K. Klein. Algorithm engineering: Concepts and practice. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuśs, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 131–160. Springer Verlag, 2010. 18

M. Chimani, G. W. Klau, and R. Weiskircher. Non-planar orthogonal drawings with fixed topology. In *Proc. of SofSem 2005*, volume 3381 of *LNCS*, pages 96–105. Springer-Verlag, 2005. ISBN 3-540-24302-X. 96

M. Chimani, P. Mutzel, and J. M. Schmidt. Efficient extraction of multiple Kuratowski subdivisions. Technical Report TR07-1-002, Chair for Algorithm Engineering, Dep. of CS, University Dortmund, 2007. See `http://ls11-www.cs.uni-dortmund.de/people/chimani/files/extractkura-TR.pdf`. 146

M. Chimani, C. Gutwenger, M. Jansen, K. Klein, and P. Mutzel. Computing maximum c-planar subgraphs. In Tollis and Patrignani [2009], pages 114–120. ISBN 978-3-642-00218-2. 5

M. Chimani, M. Jünger, and M. Schulz. Crossing minimization meets simultaneous drawing. In *PacificVis* DBL [2008], pages 33–40. 84

A. Cockburn, A. Karlson, and B. B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv.*, 41:2:1–2:31, January 2009. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/1456650.1456652. URL `http://doi.acm.org/10.1145/1456650.1456652`. 191

R. F. Cohen, G. Di Battista, R. Tamassia, I. G. Tollis, and P. Bertolazzi. A framework for dynamic graph drawing. In *Symposium on Computational Geometry*, pages 261–270, 1992. 195

M. Coleman and D. Parker. Aesthetics-based graph layout for human consumption. *Software - Practice and Experience*, 26(12):1415–1438, 1996. 61, 66

M. K. Coleman. *Aesthetics-based Graph Layout for Human Consumption*. PhD thesis, University of California, Los Angeles, 1993. 59

C. S. Collberg, S. G. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In Diehl et al. [2003], pages 77–86, 212–213. ISBN 1-58113-642-0. 45

S. Cornelsen and D. Wagner. Completely connected clustered graphs. *J. Discrete Algorithms*, 4(2):313–323, 2006. 127, 128, 129, 130, 132, 136, 137, 139, 143

P. F. Cortese, G. Di Battista, M. Patrignani, and M. Pizzonia. Clustering cycles into cycles of clusters (extended abstract). In *Proc. of The 12th Int. Symposium on Graph Drawing (GD 2004)*, 2004. 132, 140

P. F. Cortese, G. Di Battista, F. Frati, M. Patrignani, and M. Pizzonia. C-planarity of c-connected clustered graphs: Part I – characterization. Technical Report RT-DIA-109-2006, Dip. Informatica e Automazione, Univ. Roma Tre, 2006a. 131

P. F. Cortese, G. Di Battista, F. Frati, M. Patrignani, and M. Pizzonia. C-planarity of c-connected clustered graphs: Part II – testing and embedding algorithm. Technical Report RT-DIA-110-2006, Dip. Informatica e Automazione, Univ. Roma Tre, 2006b. 131

D. Croft, G. O'Kelly, G. Wu, R. Haw, M. Gillespie, L. Matthews, M. Caudy, P. Garapati, G. Gopinath, B. Jassal, S. Jupe, I. Kataskaya, S. Mahajan, B. May, N. Ndegwa, E. Schmidt, V. Shamovsky, C. Yung, E. Birney, H. Hermjakob, P. D'Eustachio, and L. Stein. Reactome: a database of reactions, pathways and biological processes. *Nucleic Acids Research*. doi: 10.1093/nar/gkq1018. URL `http://nar.oxfordjournals.org/content/early/2010/11/23/nar.gkq1018.abstract`. 24

P. Csermely. Creative elements: network-based predictions of active centres in proteins and cellular and social networks. *Trends Biochem Sci*, 33:569–576, 2008. 223

W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE Trans. Vis. Comput. Graph.*, 14(6):1277–1284, 2008. 133

E. Dahlhaus. A linear time algorithm to recognize clustered planar graphs and its parallelization. In *LATIN: Latin American Symposium on Theoretical Informatics*, 1998. 5, 125, 130, 131, 136, 137, 156

E. Dahlhaus, K. Klein, and P. Mutzel. Planarity testing for c-connected clustered graphs. Technical Report TR06-1-01, Chair of Algorithm Engineering (Ls 11), Dep. of CS, University Dortmund, 2006. See `http://ls11-www.cs.uni-dortmund.de/people/klein/CClusterTR.pdf`. 5, 131

J. Demeter, C. Beauheim, J. Gollub, T. Hernandez-Boussard, H. Jin, D. Maier, J. C. Matese, M. Nitzberg, F. Wymore, Z. K. Zachariah, P. O. Brown, G. Sherlock, and C. A. Ball. The stanford microarray database: implementation of new analysis tools and open source release of software. *Nucleic Acids Res*, 35(Database issue), January 2007. ISSN 1362-4962. URL `http://view.ncbi.nlm.nih.gov/pubmed/17182626`. 18

E. Dengler, M. Friedell, and J. Marks. Constraint-driven diagram layout. In *Proc. of the 1993 IEEE Symposium on Visual Languages*, pages 330–335, Bergen, Norway, 1993. 102

G. Di Battista, editor. *Graph Drawing, 5th International Symposium, GD '97, Rome, Italy, September 18-20, 1997, Proceedings*, volume 1353 of *Lecture Notes in Computer Science*, 1997. Springer-Verlag. ISBN 3-540-63938-1. 238, 243

G. Di Battista and F. Frati. Efficient c-planarity testing for embedded flat clustered graphs with small faces. *J. Graph Algorithms Appl.*, 13(3):349–378, 2009. 133

G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996. ISSN 0097-5397. doi: http://dx.doi.org/10.1137/S0097539794280736. 10, 116

G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom.*, 7:303–325, 1997. 147

G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Orthogonal and quasi-upward drawings with vertices of prescribed size. In J. Kratochvíl, editor, *Proc. Graph Drawing '99*, volume 1731 of *LNCS*, pages 297–310. Springer-Verlag, 1999a. URL `http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=1731&spage=297`. 84, 97

G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, 1999b. 69, 81

G. Di Battista, W. Didimo, and A. Marcandalli. Planarization of clustered graphs. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing, 9th International Symposium, GD 2001 Vienna, Austria*, volume 2265 of *LNCS*, pages 60–74. Springer-Verlag, 2001. ISBN 3-540-43309-0. 85, 97, 127, 133, 139

G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Drawing database schemas. *Software: Practice and Experience*, 32(11):1065–1098, 2002. ISSN 0038-0644. doi: http://dx.doi.org/10.1002/spe.474. 82

G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Dbdraw - automatic layout of relational database schemas. In M. Jünger and P. Mutzel, editors, *Graph Drawing Software*, Mathematics and Visualization, pages 237–256. Springer-Verlag, 2004. 82

W. Didimo, F. Giordano, and G. Liotta. Overlapping cluster planarity. *J. Graph Algorithms Appl.*, 12(3):267–291, 2008. 133

S. Diehl and C. Görg. Graphs, they are changing. In Kobourov and Goodrich [2002], pages 23–30. ISBN 3-540-00158-1. URL `http://portal.acm.org/citation.cfm?id=647554.729718`. 187, 188, 194

S. Diehl, C. Goerg, and A. Kerren. Preserving the mental map using foresighted layout. In *In Proceedings of Joint Eurographics Ű IEEE TCVG Symposium on Visualization VisSymŠ01*, pages 175–184. Springer Verlag, 2001. 194

S. Diehl, J. T. Stasko, and S. N. Spencer, editors. *Proceedings ACM 2003 Symposium on Software Visualization, San Diego, California, USA, June 11-13, 2003*, 2003. ACM. ISBN 1-58113-642-0. 240, 244, 250

R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, fourth edition, 2010. 7

H. A. D. do Nascimento and P. Eades. User hints for directed graph drawing. In *Graph Drawing, 9th International Symposium, GD 2001 Vienna, Austria*, volume 2265 of *Lecture Notes in Computer Science*, pages 205–219, 2002. 60, 184

H. A. D. do Nascimento and P. Eades. User hints: a framework for interactive optimization. *Future Generation Comp. Syst.*, 21(7):1171–1191, 2005. 184

H. A. D. do Nascimento and P. Eades. User hints for map labeling. *J. Vis. Lang. Comput.*, 19(1):39–74, 2008. 184

U. Dogrusöz, E. Giral, A. Cetintas, A. Civril, and E. Demir. A compound graph layout algorithm for biological pathways. In Pach [2004], pages 442–447. ISBN 3-540-24528-6. URL `http://dblp.uni-trier.de/db/conf/gd/gd2004.html`. 85, 91, 97

U. Dogrusöz, E. Z. Erson, E. Giral, E. Demir, O. Babur, A. Cetintas, and R. Colak. Patikaweb: a web interface for analyzing biological pathways through advanced querying and visualization. *Bioinformatics*, 22(3):374–375, 2006. 32

U. Dogrusöz, E. Giral, A. Cetintas, A. Civril, and E. Demir. A layout algorithm for undirected compound graphs. *Inf. Sci.*, 179(7):980–994, 2009. 85, 86

N. T. Doncheva, K. Klein, F. S. Domingues, and M. Albrecht. Analyzing and visualizing residue networks of protein structures. *Trends in Biochemical Sciences (TIBS)*, 2011. doi: doi:10.1016/j.tibs.2011.01.002. 5, 223

C. Dornheim. Planar graphs with topological constraints. *Journal on Graph Algorithms and Applications*, 6(1):27–66, 2002. URL `http://www.cs.brown.edu/publications/jgaa/accepted/2002/Dornheim2002.6.1.pdf`. 83, 97, 100

C. Dunne and B. Shneiderman. Improving graph drawing readability by incorporating readability metrics: a software tool for network analysts. Technical Report HCIL-2009-13, University of Maryland, 2009. URL `http://www.cs.umd.edu/localphp/hcil/tech-reports-search.php?number=2009-13`. 61

T. Dwyer. Scalable, versatile and simple constrained graph layout. *Comput. Graph. Forum*, 28(3):991–998, 2009. 89, 90

T. Dwyer and Y. Koren. Dig-cola: Directed graph layout through constrained energy minimization. In *INFOVIS*, page 9. IEEE Computer Society, 2005. ISBN 0-7803-9464-X. 83, 89, 97

T. Dwyer and K. Marriott. Constrained stress majorization using diagonally scaled gradient projection. In Hong et al. [2008], pages 219–230. ISBN 978-3-540-77536-2. 89

T. Dwyer and G. G. Robertson. Layout with circular and other non-linear constraints using procrustes projection. In Eppstein and Gansner [2010], pages 393–404. ISBN 978-3-642-11804-3. 89, 95, 97, 99

T. Dwyer and M. Wybrow. Adaptagrams project homepage. `http://adaptagrams.sourceforge.net/`. 103

T. Dwyer, Y. Koren, and K. Marriott. Stress majorization with orthogonal ordering constraints. In Healy and Nikolov [2006], pages 141–152. ISBN 3-540-31425-3. 89, 97

T. Dwyer, Y. Koren, and K. Marriott. Ipsep-cola: An incremental procedure for separation constraint layout of graphs. *IEEE Trans. Vis. Comput. Graph.*, 12(5):821–828, 2006a. 89

T. Dwyer, K. Marriott, and M. Wybrow. Integrating edge routing into force-directed layout. In Kaufmann and Wagner [2007], pages 8–19. ISBN 978-3-540-70903-9. 97, 195

T. Dwyer, K. Marriott, and M. Wybrow. Dunnart: A constraint-based network diagram authoring tool. In Tollis and Patrignani [2009], pages 420–431. ISBN 978-3-642-00218-2. 101, 103

T. Dwyer, K. Marriott, and M. Wybrow. Topology preserving constrained graph layout. In *Graph Drawing: 16th International Symposium, GD 2008, Heraklion, Crete, Greece, September 21-24, 2008. Revised Papers*, pages 230–241, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-00218-2. doi: http://dx.doi.org/10.1007/978-3-642-00219-9_22. 83, 89, 96, 97, 189

P. Eades. A heuristic for graph drawing. In *Congressus Numerantium*, volume 42, pages 149–160, 1984. 70

P. Eades. Drawing free trees. *Bulletin of the institute for Combinatorics and its Applications*, 5:10–36, 1992. 65

P. Eades. Invited talk GD'10, 2010. 191

P. Eades and Q.-W. Feng. Drawing clustered graphs on an orthogonal grid. In Di Battista [1997], pages 146–157. ISBN 3-540-63938-1. 85, 97

P. Eades and M. L. Huang. Navigating clustered graphs using force-directed methods. *J. Graph Algorithms Appl.*, 4(3):157–181, 2000. 85, 127, 192

P. Eades, W. Lai, K. Misue, and K. Sugiyama. Preserving the mental map of a diagram. In *Proceedings of COMPUGRAPHICS '91*, pages 34–43, 1991. 187, 189, 193

P. Eades, R. F. Cohen, and M. L. Huang. Online animated graph drawing for web navigation. In Di Battista [1997], pages 330–335. ISBN 3-540-63938-1. 192

P. Eades, Q. Feng, X. Lin, and H. Nagamochi. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In *Algorithmica*, pages 113–128. Springer-Verlag, 1999a. 127

P. Eades, Q.-W. Feng, and H. Nagamochi. Drawing clustered graphs on an orthogonal grid. *J. Graph Algorithms Appl.*, 3(4):3–29, 1999b. 127

P. Effinger, M. Siebenhaller, and M. Kaufmann. An interactive layout tool for bpmn. In B. Hofreiter and H. Werthner, editors, *CEC*, pages 399–406. IEEE Computer Society, 2009. ISBN 978-0-7695-3755-9. 50

H. Eichelberger. Nice class diagrams admit good design? In Diehl et al. [2003], pages 159–167. ISBN 1-58113-642-0. 44

Eigenfactor. Eigenfactor ranking web page. `http://eigenfactor.org`. 42

M. Eiglsperger, U. Fößmeier, and M. Kaufmann. Orthogonal graph drawing with constraints. In *Proc. of the 11th ACM-SIAM Symp. on Discr. Alg.*, SODA '00, pages 3–11, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. ISBN 0-89871-453-2. URL `http://portal.acm.org/citation.cfm?id=338219.338225`. 83, 97

M. Eiglsperger, M. Kaufmann, and F. Eppinger. An approach for mixed upward planarization. *J. Graph Algorithms Appl.*, 7(2):203–220, 2003. 83

M. Eiglsperger, C. Gutwenger, M. Kaufmann, J. Kupke, M. Jünger, K. Klein, S. Leipert, P.Mutzel, and M. Siebenhaller. Automatic layout of uml class diagrams in orthogonal style. *Information Visualization*, 3:189–208, 2004. 68, 83, 97

W. D. Ellis. *A source book of gestalt psychology,*. Routledge & Kegan Paul, January 1969. 13

D. Emig, M. S. Cline, K. Klein, A. Kunert, P. Mutzel, T. Lengauer, and M. Albrecht. Integrative visual analysis of the effects of alternative splicing on protein domain interaction networks. *J. Integrative Bioinformatics*, 5(2), 2008a. 5

D. Emig, K. Klein, A. Kunert, P. Mutzel, and M. Albrecht. Visualizing domain interaction networks and the impact of alternative splicing events. *Medical Information Visualisation; BioMedical Visualisation, International Conference on*, 0:36–43, 2008b. doi: http://doi.ieeecomputersociety.org/10.1109/MediVis.2008.16. 5

D. Eppstein and E. R. Gansner, editors. *Graph Drawing, 17th International Symposium, GD 2009, Chicago, IL, USA, September 22-25, 2009. Revised Papers*, volume 5849 of *Lecture Notes in Computer Science*, 2010. Springer-Verlag. ISBN 978-3-642-11804-3. 233, 234, 237, 243

C. Erten and S. G. Kobourov. Simultaneous embedding of planar graphs with few bends. *J. Graph Algorithms Appl.*, 9(3):347–364, 2005. 84

C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. V. Yee. Graphael: Graph animations with evolving layouts. In Liotta [2004], pages 98–110. ISBN 3-540-20831-3. 40, 99

A. Estrella-Balderrama, E. Gassner, M. Jünger, M. Percan, M. Schaefer, and M. Schulz. Simultaneous geometric graph embeddings. In Hong et al. [2008], pages 280–290. ISBN 978-3-540-77536-2. 84

A. Estrella-Balderrama, J. J. Fowler, and S. G. Kobourov. Graphset, a tool for simultaneous graph drawing. *Softw., Pract. Exper.*, 40(10):849–863, 2010. 84

Q. W. Feng. *Algorithms for Drawing Clustered Graphs*. PhD thesis, The University of Newcastle, 1997. 136

Q.-W. Feng, R. F. Cohen, and P. Eades. Planarity for clustered graphs. In P. G. Spirakis, editor, *ESA: Annual European Symposium on Algorithms*, volume 979 of *Lecture Notes in Computer Science*, pages 213–226. Springer-Verlag, 1995a. ISBN 3-540-60313-1. 8, 97, 126, 129, 131, 132, 134, 135, 139, 141, 156

Q.-W. Feng, R. F. Cohen, and P. Eades. How to draw a planar clustered graph. In D.-Z. Du and M. Li, editors, *COCOON*, volume 959 of *Lecture Notes in Computer Science*, pages 21–30. Springer-Verlag, 1995b. ISBN 3-540-60216-X. 127

R. D. Finn, J. G. Tate, J. Mistry, P. C. Coggill, S. J. Sammut, H.-R. Hotz, G. Ceric, K. Forslund, S. R. Eddy, E. L. L. Sonnhammer, and A. Bateman. The pfam protein families database. *Nucleic Acids Research*, 36(Database-Issue):281–288, 2008. 212

C. J. Fisk and D. D. Isett. "accel" automated circuit card etching layout. In *DAC '65: Proceedings of the SHARE design automation project*, pages 9.1–9.31, New York, NY, USA, 1965. ACM. doi: http://doi.acm.org/10.1145/800266.810762. 70

M. Forster, A. Pick, M. Raitner, F. Schreiber, and F.-J. Brandenburg. The system architecture of the biopath system. *In Silico Biology*, 2:37, 2002. 28, 29

U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. *Lecture Notes in Computer Science*, 1027:254–266, 1996. ISSN 0302-9743. 68

J. J. Fowler, M. Jünger, S. G. Kobourov, and M. Schulz. Characterizing simultaneous embedding with fixed edges. *Electronic Notes in Discrete Mathematics*, 31:41–44, 2008. 84

C. Fraley and A. E. Raftery. How many clusters? Which clustering method? Answers via model-based cluster analysis. *Comput. J.*, 41(8):578–588, 1998. 18

F. Frati, M. Kaufmann, and S. G. Kobourov. Constrained simultaneous and near-simultaneous embeddings. *J. Graph Algorithms Appl.*, 13(3):447–465, 2009. 84

L. C. Freeman. Graphic techniques for exploring social network data. In J. S. P. J. Carrington and S. Wasserman, editors, *Models and Methods in Social Network Analysis,*. Cambridge University Press, 2004. 36

M. Freire and P. Rodríguez. Preserving the mental map in interactive graph interfaces. In A. Celentano, editor, *AVI*, pages 270–273. ACM Press, 2006. ISBN 1-59593-353-0. 188, 189

A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In *GD '94: Proceedings of the DIMACS International Workshop on Graph Drawing*, pages 388–403, London, UK, 1995. Springer-Verlag. ISBN 3-540-58950-3. 70

A. Frick, C. Keskin, and V. Vogelmann. Integration of declarative approaches. In S. C. North, editor, *Graph Drawing*, volume 1190 of *Lecture Notes in Computer Science*, pages 184–192. Springer-Verlag, 1996. ISBN 3-540-62495-3. 67

C. Friedrich and P. Eades. Graph drawing in motion. *J. Graph Algorithms Appl.*, 6(3): 353–370, 2002. 196

C. Friedrich and M. E. Houle. Graph drawing in motion ii. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 220–231. Springer-Verlag, 2001. ISBN 3-540-43309-0. 196

C. Friedrich and F. Schreiber. Visualisation and navigation methods for typed protein-protein interaction networks. *Applied Bioinformatics*, 2, 2003. 213

C. Friedrich and F. Schreiber. Flexible layering in hierarchical drawings with nodes of arbitrary size. In V. Estivill-Castro, editor, *ACSC*, volume 26 of *CRPIT*, pages 369–376. Australian Computer Society, 2004. ISBN 1-920682-05-8. 84, 97

Y. Frishman and A. Tal. Dynamic drawing of clustered graphs. In *INFOVIS '04: Proceedings of the IEEE Symposium on Information Visualization*, pages 191–198, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8779-3. doi: http://dx.doi.org/10.1109/INFOVIS.2004.18. 45, 194

Y. Frishman and A. Tal. Multi-level graph layout on the GPU. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1310–1319, 2007. ISSN 1077-2626. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2007.70580. 71

Y. Frishman and A. Tal. Online dynamic graph drawing. *IEEE Trans. Vis. Comput. Graph.*, 14(4):727–740, 2008. 188, 195

T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, 1991. ISSN 0038-0644. doi: http://dx.doi.org/10.1002/spe.4380211102. 70, 97, 195

M. Gaertler. Clustering. In U. Brandes and T. Erlebach, editors, *Network Analysis*, volume 3418 of *Lecture Notes in Computer Science*, pages 178–215. Springer-Verlag, 2004. ISBN 3-540-24979-6. 18

P. Gajer and S. G. Kobourov. GRIP: Graph drawing with intelligent placement. *J. Graph Algorithms Appl.*, 6(3):203–224, 2002. 40

P. Gajer, M. T. Goodrich, and S. G. Kobourov. A multi-dimensional approach to force-directed layouts of large graphs. In Marks [2001], pages 211–221. ISBN 3-540-41554-8. 70

E. R. Gansner and Y. Hu. Efficient, proximity-preserving node overlap removal. *J. Graph Algorithms Appl.*, 14(1):53–74, 2010. 41, 83, 97, 99

E. R. Gansner and Y. Koren. Improved circular layouts. In *Proceedings of the 14th international conference on Graph drawing*, GD'06, pages 386–398, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-70903-9. URL `http://portal.acm.org/citation.cfm?id=1758612.1758651`. 65, 133, 221

E. R. Gansner, Y. Koren, and S. C. North. Graph drawing by stress majorization. In Pach [2004], pages 239–250. ISBN 3-540-24528-6. 70, 72

E. R. Gansner, Y. Koren, and S. C. North. Topological fisheye views for visualizing large graphs. *IEEE Trans. Vis. Comput. Graph.*, 11(4):457–468, 2005. 193

R. García-Serna, O. Ursu, T. I. Oprea, and J. Mestres. iphace: integrative navigation in pharmacological space. *Bioinformatics*, 26(7):985–986, 2010. 209

M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):312–316, 1983. ISSN 0196-5212. 67, 107

A. Garg. New results on drawing angle graphs. *Comput. Geom. Theory Appl.*, 9(1-2):43–82, 1998. ISSN 0925-7721. doi: http://dx.doi.org/10.1016/S0925-7721(97)00016-3. 100

B. Genc and U. Dogrusöz. A constrained, force-directed layout algorithm for biological pathways. In Liotta [2004], pages 314–319. ISBN 3-540-20831-3. 85

E. D. Giacomo, W. Didimo, G. Liotta, H. Meijer, and S. K. Wismath. Constrained point-set embeddability of planar graphs. In Tollis and Patrignani [2009], pages 360–371. ISBN 978-3-642-00218-2. 95

A. Godiyal, J. Hoberock, M. Garland, and J. C. Hart. Rapid multipole graph drawing on the GPU. In *Proc. Graph Drawing 2008*, volume 5417 of *LNCS*, pages 90–101, 2009. 71

A. Goesmann, M. Haubrock, F. Meyer, J. Kalinowski, and R. Giegerich. Pathfinder: reconstruction and dynamic visualization of metabolic pathways. *Bioinformatics*, 18(1): 124–129, 2002. 25

J. R. Goodall, G. J. Conti, and K.-L. Ma, editors. *Visualization for Computer Security, 5th International Workshop, VizSec 2008, Cambridge, MA, USA, September 15, 2008. Proceedings*, volume 5210 of *Lecture Notes in Computer Science*, 2008. Springer-Verlag. ISBN 978-3-540-85931-4. 249, 257

M. T. Goodrich, G. S. Lueker, and J. Z. Sun. C-planarity of extrovert clustered graphs. In Healy and Nikolov [2006], pages 211–222. ISBN 3-540-31425-3. 132, 138

C. Gutwenger and P. Mutzel. A linear time implementation of SPQR trees. In J. Marks, editor, *Proc. 8th Intl. Symp. Graph Drawing (GD 2000)*, volume 1984 of *LNCS*, pages 77–90. Springer-Verlag, 2001. 10, 113, 122, 176

C. Gutwenger and P. Mutzel. An experimental study of crossing minimization heuristics. In G. Liotta, editor, *Proc. 11th Intl. Symp. Graph Drawing (GD 2003)*, volume 2912 of *LNCS*, pages 13–24. Springer-Verlag, 2004. 67, 107

C. Gutwenger, M. Jünger, K. Klein, J. Kupke, S. Leipert, and P. Mutzel. Caesar automatic layout of uml class diagrams. In *Graph Drawing*, pages 461–462, 2001. 83

C. Gutwenger, M. Jünger, S. Leipert, P. Mutzel, M. Percan, and R. Weiskircher. Advances in C-planarity testing of clustered graphs. In *Graph Drawing: Proc. Graph Drawing (GD)*, 2002. 131, 132, 133, 135, 138

C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. *Algorithmica*, 41(4):289–308, 2005. 108, 114, 115, 122

C. Gutwenger, K. Klein, and P. Mutzel. Planarity testing and optimal edge insertion with embedding constraints. In Kaufmann and Wagner [2007], pages 126–137. ISBN 978-3-540-70903-9. 5, 100

C. Gutwenger, K. Klein, and P. Mutzel. Planarity testing and optimal edge insertion with embedding constraints. *J. Graph Algorithms Appl.*, 12(1):73–95, 2008. 5, 83, 97

GVSR. Graph visualization software references. `http://gvsr.polytech.univ-nantes.fr/GVSR/`. 74

S. Hachul and M. Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In J. Pach, editor, *Proc. Graph Drawing 2004*, volume 3383 of *LNCS*, pages 285–295. Springer-Verlag, 2004. 70

S. Hachul and M. Jünger. Large-graph layout algorithms at work: An experimental study. *J. Graph Algorithms Appl.*, 11(2):345–369, 2007. 71

R. Hadany and D. Harel. A multi-scale algorithm for drawing graphs nicely. *Discrete Applied Mathematics*, 113(1):3–21, 2001. 70

D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *J. Graph Algorithms Appl.*, 6(3):179–202, 2002. 70

M. Harrigan and P. Healy. Practical level planarity testing and layout with embedding constraints. In Hong et al. [2008], pages 62–68. ISBN 978-3-540-77536-2. 83, 97

W. He and K. Marriott. Constrained graph layout. *Constraints*, 3(4):289–314, 1998. ISSN 1383-7133. doi: http://dx.doi.org/10.1023/A:1009771921595. 81, 88

P. Healy and N. S. Nikolov, editors. *Graph Drawing, 13th International Symposium, GD 2005, Limerick, Ireland, September 12-14, 2005, Revised Papers*, volume 3843 of *Lecture Notes in Computer Science*, 2006. Springer-Verlag. ISBN 3-540-31425-3. 243, 248, 256, 263

J. Heer and D. Boyd. Vizster: Visualizing online social networks. In *IEEE Information Visualization (InfoVis)*, pages 32–39, 2005. URL `http://vis.stanford.edu/papers/vizster`. 38

N. Henry and J.-D. Fekete. Matlink: Enhanced matrix visualization for analyzing social networks. In *Proceedings of the International Conference Interact*, pages 288–302, 2007. 36

N. Henry, J.-D. Fekete, and M. J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1302–1309, 2007. 36

I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Trans. Vis. Comput. Graph.*, 6(1):24–43, 2000. 61, 191

J. Homer, A. Varikuti, X. Ou, and M. A. McQueen. Improving attack graph visualization through data reduction and attack grouping. In Goodall et al. [2008], pages 68–79. ISBN 978-3-540-85931-4. 48

S.-H. Hong and M. Mader. Generalizing the shift method for rectangular shaped vertices with visibility constraints. In Tollis and Patrignani [2009], pages 278–283. ISBN 978-3-642-00218-2. 85, 97

S.-H. Hong and H. Nagamochi. Convex drawings of hierarchical planar graphs and clustered planar graphs. *J. of Discrete Algorithms*, 8:282–295, September 2010. ISSN 1570-8667. doi: http://dx.doi.org/10.1016/j.jda.2009.05.003. URL `http://dx.doi.org/10.1016/j.jda.2009.05.003`. 127

S.-H. Hong, T. Nishizeki, and W. Quan, editors. *Graph Drawing, 15th International Symposium, GD 2007, Sydney, Australia, September 24-26, 2007. Revised Papers*, volume 4875 of *Lecture Notes in Computer Science*, 2008. Springer-Verlag. ISBN 978-3-540-77536-2. 235, 243, 245, 248

J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974. 108, 113

J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973a. 122

J. E. Hopcroft and R. E. Tarjan. Efficient algorithms for graph manipulation. *Commun. ACM*, 1(6):372–378, June 1973b. 177

M. L. Huang, P. Eades, and J. Wang. On-line animated visualization of huge graphs using a modified spring algorithm. *J. Vis. Lang. Comput.*, 9(6):623–645, 1998. 192

W. Huang, S.-H. Hong, and P. Eades. How people read sociograms: a questionnaire study. In Misue et al. [2006], pages 199–206. ISBN 1-920682-41-4. 63

W. Huang, S.-H. Hong, and P. Eades. Effects of sociogram drawing conventions and edge crossings in social network visualization. *J. Graph Algorithms Appl.*, 11(2):397–429, 2007. 40

W. Huang, S.-H. Hong, and P. Eades. Effects of crossing angles. In *PacificVis* DBL [2008], pages 41–46. 61, 62

W. Huang, P. Eades, and S.-H. Hong. Measuring effectiveness of graph visualizations: a cognitive load perspective. *Information Visualization*, 8(3):139–152, 2009. ISSN 1473-8716. doi: http://dx.doi.org/10.1057/ivs.2009.10. 63

W. Huang, P. Eades, S.-H. Hong, and C.-C. Lin. Improving force-directed graph drawings by making compromises between aesthetics. In C. D. Hundhausen, E. Pietriga, P. Díaz, and M. B. Rosson, editors, *VL/HCC*, pages 176–183. IEEE, 2010. ISBN 978-0-7695-4206-5. 59

C. Huttenhower, S. Mehmood, and O. Troyanskaya. Graphle: Interactive exploration of large, dense graphs. *BMC Bioinformatics*, 10(1):417, 2009. ISSN 1471-2105. doi: 10.1186/1471-2105-10-417. URL `http://www.biomedcentral.com/1471-2105/10/417`. 32

T. Igarashi and K. Hinckley. Speed-dependent automatic zooming for browsing large documents. In *UIST*, pages 139–148, 2000. 192

IMI. The innovative medicines initiative, second call for proposals 2009. 197

IV2006. *10th International Conference on Information Visualisation, IV 2006, 5-7 July 2006, London, UK*, 2006. IEEE Computer Society. 236, 252

T. Jacobs and B. Musial. Interactive visual debugging with uml. In Diehl et al. [2003], pages 115–122. ISBN 1-58113-642-0. 45

V. Jelínek, E. Jelínková, J. Kratochvíl, and B. Lidický. Clustered planarity: Embedded clustered graphs with two-component clusters. In Tollis and Patrignani [2009], pages 121–132. ISBN 978-3-642-00218-2. 133

V. Jelínek, O. Suchý, M. Tesar, and T. Vyskocil. Clustered planarity: Clusters with few outgoing edges. In Tollis and Patrignani [2009], pages 102–113. ISBN 978-3-642-00218-2. 133

E. Jelínková, J. Kára, J. Kratochvíl, M. Pergel, O. Suchý, and T. Vyskocil. Clustered planarity: Small clusters in cycles and eulerian graphs. *J. Graph Algorithms Appl.*, 13(3): 379–422, 2009. 133

M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16(1):33–59, 1996. 145, 146

M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Algorithms Appl.*, 1:1–25, 1997. 218

M. Jünger and P. Mutzel. *Graph Drawing Software*. Springer-Verlag, 2003. 7, 32

M. Jünger and S. Thienel. The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. *Software: Practice and Experience*, 30(11):1325–1352, 2000. 147

M. Jünger, S. Leipert, and M. Percan. Triangulating clustered graphs. Technical report, Zentrum für Angewandte Informatik Köln, Lehrstuhl Jünger, December 2002. 135

B. Junker, C. Klukas, and F. Schreiber. Vanted: A system for advanced data analysis and visualization in the context of biological networks. *BMC Bioinformatics*, 7(1):109, 2006. 25, 32

T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989. ISSN 0020-0190. 59, 70, 72, 88

T. Kamada and S. Kawai. A general framework for visualizing abstract objects and relations. *ACM Trans. Graph.*, 10(1):1–39, 1991. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/99902.99903. 81, 88

P. D. Karp and S. M. Paley. Automated drawing of metabolic pathways. In H. L. H., C. C. C., and R. Robbins, editors, *Proceedings of the Third International Conference on Bioinformatics and Genome Research*, pages 225–238. World Scientific Publishing Co., 1994a. 25, 85

P. D. Karp and S. M. Paley. Representations of metabolic knowledge: Pathways. 1994b. URL `citeseer.ist.psu.edu/karp94representations.html`. 85

G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. In *SC*, 1995. 70

K. Kato, M. Nagasaki, and A. D. ans S. Miyano. Automatic drawing of biological networks using cross cost and subcomponent data. *Genome Informatics*, 16(2):22–31, 2005. 92, 97, 213

M. Kaufmann and D. Wagner, editors. *Drawing Graphs*, volume 2025 of *LNCS*. Springer-Verlag, 2001. 69

M. Kaufmann and D. Wagner, editors. *Graph Drawing, 14th International Symposium, GD 2006, Karlsruhe, Germany, September 18-20, 2006. Revised Papers*, volume 4372 of *Lecture Notes in Computer Science*, 2007. Springer-Verlag. ISBN 978-3-540-70903-9. 237, 243, 248

KEGG. Kyoto encyclopedia of genes and genomes (kegg) homepage. http://www.genome.jp/kegg/. 24

D. A. Keim, F. Mansmann, J. Schneidewind, and H. Ziegler. Challenges in visual data analysis. In IV2006, pages 9–16. 12

D. A. Keim, F. Mansmann, J. Schneidewind, J. Thomas, and H. Ziegler. Visual analytics: Scope and challenges. In S. J. Simoff, M. H. Böhlen, and A. Mazeika, editors, *Visual Data Mining*, volume 4404 of *Lecture Notes in Computer Science*, pages 76–90. Springer-Verlag, 2008. ISBN 978-3-540-71079-0. 13

S. Killcoyne, G. W. Carter, J. Smith, and J. Boyle. Cytoscape: a community-based framework for network modeling. *Methods Mol Biol.*, 2009. 32

K. Klein. Benchmark set of clustered graphs for the MCPSP experiments. `ls11-www.cs.uni-dortmund.de/people/klein/clusterbenchmarks08.zip`, 2008. 147

K. Klein, N. Kriege, and P. Mutzel. Ct-index: Fingerprint-based graph indexing combining cycles and trees. In *27th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2011. to appear. 6, 35, 208

A. Klippel, L. Knuf, B. Hommel, and C. Freksa. Perceptually induced distortions in cognitive maps. In C. Freksa, M. Knauff, B. Krieg-Brückner, B. Nebel, and T. Barkowsky, editors, *Spatial Cognition*, volume 3343 of *Lecture Notes in Computer Science*, pages 204–213. Springer-Verlag, 2004. ISBN 3-540-25048-4. 84

C. Klukas and F. Schreiber. Dynamic exploration and editing of kegg pathway diagrams. *Bioinformatics*, 23(3):344–350, 2007. 17

C. Klukas and F. Schreiber. Integration of -omics data and networks for biomedical research. *Journal of Integrative Bioinformatics*, 7(2), 2010. 33

S. G. Kobourov. *Visualization of Large Graphs*. PhD thesis, Johns Hopkins University, 2000. 18

S. G. Kobourov and M. T. Goodrich, editors. *Graph Drawing, 10th International Symposium, GD 2002, Irvine, CA, USA, August 26-28, 2002, Revised Papers*, volume 2528 of *Lecture Notes in Computer Science*, 2002. Springer-Verlag. ISBN 3-540-00158-1. 238, 241

I. Koch, W. Reisig, and F. Schreiber. *Modeling in Systems Biology The Petri Net Approach*, volume 16 of *Computational Biology*. Springer-Verlag, 2011. 20

M. A. Koch, A. Schuffenhauer, M. Scheck, S. Wetzel, M. Casaulta, A. Odermatt, P. Ertl, and H. Waldmann. Charting biologically relevant chemical space: a structural classification of natural products (SCONP). *Proceedings of the National Academy of Sciences of the United States of America*, 102(48):17272–17277, November 2005. ISSN 0027-8424. doi: 10.1073/pnas.0503647102. URL `http://dx.doi.org/10.1073/pnas.0503647102`. 199

K. W. Kohn, M. I. Aladjem, J. N. Weinstein, and Y. Pommier. Molecular interaction maps of bioregulatory networks: A general rubric for systems biology. *Mol. Biol. Cell*, 17(1):1–13, 2006. doi: 10.1091/mbc.E05-09-0824. URL `http://www.molbiolcell.org/cgi/content/abstract/17/1/1`. 24

K. Kojima, M. Nagasaki, E. Jeong, M. Kato, and S. Miyano. An efficient grid layout algorithm for biological networks utilizing various biological attributes. *BMC Bioinformatics*, 8:76+, March 2007. ISSN 1471-2105. doi: 10.1186/1471-2105-8-76. URL `http://dx.doi.org/10.1186/1471-2105-8-76`. 92, 93, 97

K. Kojima, M. Nagasaki, and S. Miyano. Fast grid layout algorithm for biological networks with sweep calculation. *Bioinformatics*, 24(12):1433–1441, 2008. 33, 92, 97

K. Kojima, M. Nagasaki, and S. Miyano. An efficient biological pathway layout algorithm combining grid-layout and spring embedder for complicated cellular location information. *BMC Bioinformatics*, 11:335, 2010. 32, 33, 92, 97

C. Kosak, J. Marks, and S. M. Shieber. Automating the layout of network diagrams with specified visual organization. *IEEE Transactions on Systems, Man, and Cybernetics*, 24 (3):440–454, 1994. 102

J. Kratochvíl, editor. *Graph Drawing, 7th International Symposium, GD'99, Stirín Castle, Czech Republic, September 1999, Proceedings*, volume 1731 of *Lecture Notes in Computer Science*, 1999. Springer-Verlag. ISBN 3-540-66904-3. 235, 262

K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930. 145

A. Lander. The edges of understanding. *BMC Biology*, 8(1):40, 2010. ISSN 1741-7007. doi: 10.1186/1741-7007-8-40. URL `http://www.biomedcentral.com/1741-7007/8/40`. 20, 21

B. Lee, C. S. Parr, C. Plaisant, B. B. Bederson, V. D. Veksler, W. D. Gray, and C. Kotfila. Treeplus: Interactive exploration of networks with enhanced tree layouts. *IEEE Transactions on Visualization and Computer Graphics*, 12:1414–1426, November 2006a. ISSN 1077-2626. doi: http://dx.doi.org/10.1109/TVCG.2006.106. URL `http://dx.doi.org/10.1109/TVCG.2006.106`. 38

Y.-Y. Lee, C.-C. Lin, and H.-C. Yen. Mental map preserving graph drawing using simulated annealing. In Misue et al. [2006], pages 179–188. ISBN 1-920682-41-4. 194

T. Lengauer. Hierarchical planary testing algorithms. In L. Kott, editor, *ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 215–225. Springer-Verlag, 1986. ISBN 3-540-16761-7. 131

I. Letunic, T. Yamada, M. Kanehisa, and P. Bork. ipath: interactive exploration of biochemical pathways and networks. *Trends in Biochemical Sciences*, 33(3):101 – 103, 2008. ISSN 0968-0004. doi: DOI:10.1016/j.tibs.

2008.01.001. URL `http://www.sciencedirect.com/science/article/B6TCV-4RTTKPF-1/2/0b4c5cbd71abcde678714a9ba515798c`. 25

W. Li and H. Kurata. A grid layout algorithm for automatic drawing of biochemical networks. *Bioinformatics*, 21(9):2036–2042, May 2005. ISSN 1367-4803. doi: 10.1093/bioinformatics/bti290. URL `http://dx.doi.org/10.1093/bioinformatics/bti290`. 32, 90, 91, 92, 97, 213

T. Lin and P. Eades. Integration of declarative and algorithmic approaches for layout creation. In *GD '94: Proceedings of the DIMACS International Workshop on Graph Drawing*, pages 376–387, London, UK, 1995. Springer-Verlag. ISBN 3-540-58950-3. 67, 182, 183, 185

G. Liotta, editor. *Graph Drawing, 11th International Symposium, GD 2003, Perugia, Italy, September 21-24, 2003, Revised Papers*, volume 2912 of *Lecture Notes in Computer Science*, 2004. Springer-Verlag. ISBN 3-540-20831-3. 245, 247

P. Liu and R. Geldmacher. On the Deletion of Nonplanar Edges of a Graph. In *10th Conference on Combinatorics, Graph Theory and Computing*, pages 727–738, 1977. 143

E. Lounkine, M. Wawer, A. M. Wassermann, and J. Bajorath. Saranea: A freely available program to mine structure-activity and structure-selectivity relationship information in compound data sets. *Journal of Chemical Information and Modeling*, 50(1):68–78, 2010. doi: 10.1021/ci900416a. URL `http://pubs.acs.org/doi/abs/10.1021/ci900416a`. PMID: 20053000. 209

K. A. Lyons. Cluster busting in anchored graph drawing. In *CASCON '92: Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research*, pages 327–337. IBM Press, 1992. 188, 189

B. A. Malloy and J. F. Power. Exploiting uml dynamic object modeling for the visualization of C++ programs. In T. L. Naps and W. D. Pauw, editors, *SOFTVIS*, pages 105–114. ACM, 2005. ISBN 1-59593-073-6. 45

J. Marks, editor. *Graph Drawing, 8th International Symposium, GD 2000, Colonial Williamsburg, VA, USA, September 20-23, 2000, Proceedings*, volume 1984 of *Lecture Notes in Computer Science*, 2001. Springer-Verlag. ISBN 3-540-41554-8. 238, 247, 259

A. J. Matlin, F. Clark, and C. W. J. Smith. Understanding alternative splicing: towards a cellular code. *Nature Reviews Molecular Cell Biology*, 6:386–398, 2005. 212

C. McGrath, J. Blythe, and D. Krackhardt. The effect of spatial arrangement on judgments and errors in interpreting graphs. *Social Networks*, 19(3):223–242, 1997. doi: http://dx.doi.org/10.1016/S0378-8733(96)00299-7. URL `http://www.sciencedirect.com/science/article/B6VD1-3SWYCMP-8/2/90db2cbc12cefc1a2b9d4cde251af062`. 40

K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16:233–242, 1996. 108

K. Mehlhorn and P. Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer-Verlag, 1 edition, July 2008. ISBN 3540779779. URL `http://www.worldcat.org/isbn/3540779779`. 18

S. Mencher and L. Wang. Promiscuous drugs compared to selective drugs (promiscuity can be a virtue). *BMC Clinical Pharmacology*, 5(1):3, 2005. ISSN 1472-6904. doi: 10.1186/1472-6904-5-3. URL `http://www.biomedcentral.com/1472-6904/5/3`. 198

D. Merico, D. Gfeller, and G. D. Bader. How to visually interpret biological data using networks. *Nat Biotech*, 27(10):921–924, October 2009. ISSN 1087-0156. doi: 10.1038/nbt.1567. URL `http://dx.doi.org/10.1038/nbt.1567`. 23, 32

G. Michal. Biochemical pathways. Poster, 2005. 24

MIM. Molecular interaction maps. `http://discover.nci.nih.gov/mim/`. 24

K. Miriyala, S. W. Hornick, and R. Tamassia. An incremental approach to aesthetic graph layout. In *Proceedings of the International Workshop on Computer-Aided Software Engineering*, pages 297–308, 1993. 194

K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *J. Vis. Lang. Comput.*, 6(2):183–210, 1995. 42, 183, 185, 187, 189, 194

K. Misue, K. Sugiyama, and J. Tanaka, editors. *Asia-Pacific Symposium on Information Visualisation, APVIS 2006, Tokyo, Japan, February 1-3, 2006*, volume 60 of *CRPIT*, 2006. Australian Computer Society. ISBN 1-920682-41-4. 250, 253

J. L. Moreno. *Who shall survive? Foundations of Sociometry, Group Psychotherapy and Sociodrama*. Beacon House Inc., 2 edition, 1953. 36, 37

J. H. Morris, C. C. Huang, P. C. Babbitt, and T. E. Ferrin. structureViz: linking Cytoscape and UCSF Chimera. *Bioinformatics*, 23(17):2345–2347, 2007. doi: 10.1093/bioinformatics/btm329. URL `http://bioinformatics.oxfordjournals.org/content/23/17/2345.abstract`. 224

T. Moscovich, F. Chevalier, N. Henry, E. Pietriga, and J.-D. Fekete. Topology-aware navigation in large networks. In D. R. O. Jr., R. B. Arthur, K. Hinckley, M. R. Morris, S. E. Hudson, and S. Greenberg, editors, *CHI*, pages 2319–2328. ACM, 2009. ISBN 978-1-60558-246-7. 192

T. Munzner. *Interactive Visualization of Large Graphs and Networks"*. PhD thesis, Stanford University, 2000. 18, 182, 190

P. Mutzel and R. Weiskircher. Optimizing over all combinatorial embeddings of a planar graph. In G. Cornuéjols, R. E. Burkard, and G. J. Woeginger, editors, *IPCO*, volume 1610 of *Lecture Notes in Computer Science*, pages 361–376. Springer-Verlag, 1999. ISBN 3-540-66019-4. 10

L. Nachmanson, S. Pupyrev, and M. Kaufmann. Improving layered graph layouts with edge bundling. In *Proceedings of the 18th International Symposium on Graph Drawing (GD'10)*, volume 6502 of *LNCS*. Springer-Verlag, 2011. 133

H. Nagamochi and K. Kuroya. Drawing c-planar biconnected clustered graphs. *Discrete Appl. Math.*, 155:1155–1174, May 2007. ISSN 0166-218X. doi: 10.1016/j.dam.2006.04.044. URL `http://portal.acm.org/citation.cfm?id=1240325.1240386`. 127

M. Nagasaki, A. Saito, E. Jeong, C. Li, K. Kojima, E. Ikeda, and S. Miyano. Cell illustrator 4.0: A computational platform for systems biology. *In silico biology*, 2010. 22

K. V. Nesbitt and C. Friedrich. Applying gestalt principles to animated visualizations of network data. In *IV*, pages 737–743, 2002. 57, 185, 187

D. Nicholson. Nicholson maps online. `http://www.iubmb-nicholson.org/index.html`. 24

S. Noel and S. Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In C. E. Brodley, P. Chan, R. Lippman, and W. Yurcik, editors, *VizSEC*, pages 109–118. ACM, 2004. ISBN 1-58113-974-8. ix, 48, 49

M. Nöllenburg. Automated drawing of metro maps. Technical Report 2005-25, Universiät Karlsruhe, 2005. 87

M. Nöllenburg and A. Wolff. A mixed-integer program for drawing high-quality metro maps. In Healy and Nikolov [2006], pages 321–333. ISBN 3-540-31425-3. 85, 87

M. Nöllenburg and A. Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Transactions on Visualization and Computer Graphics*, 99 (RapidPosts), 2010. ISSN 1077-2626. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2010.81. 87, 88

S. C. North. Incremental layout in dynadag. In *Proc. 3rd Intl. Symp. Graph Drawing (GD '95)*, volume 1027 of *LNCS*, pages 409–418. Springer-Verlag, 1996. ISBN 3-540-60723-4. 185, 187

N. L. Novere, M. Hucka, H. Mi, S. Moodiet, F. Schreiber, A. Sorokin, E. Demir, K. Wegner, M. I. Aladjem, S. M. Wimalaratne, F. T. Bergman, R. Gauges, P. Ghazal, H. Kawaji, L. Li, Y. Matsuoka, A. Villeger, S. E. Boyd, L. Calzone, M. Courtot, U. Dogrusoz, T. C. Freeman, A. Funahashi, S. Ghosh, A. Jouraku, S. Kim, F. Kolpakov, A. Luna, S. Sahle, E. Schmidt, S. Watterson, G. Wu, I. Goryanin, D. B. Kell, C. Sander, H. Sauro, J. L. Snoep, K. Kohn, and H. Kitano. The systems biology graphical notation. *Nature Biotechnology*, 27(8):735–741, 2009. ISSN 1087-0156. doi: 10.1038/nbt.1558. URL http://dx.doi.org/10.1038/nbt.1558. 2, 26

S. O'Donoghue, D. Goodsell, A. Frangakis, F. Jossinet, R. Laskowski, M. Nilges, H. Saibil, A. Schafferhans, R. Wade, E. Westhof, and A. J. Olson. Visualization of macromolecular structures. *Nature Methods*, 7(3(S)):S42–S55, 2010. 223

OGDF. The Open Graph Drawing Framework. http://www.ogdf.net. 103, 147

S. O'Hare, S. Noel, and K. Prole. A graph-theoretic visualization approach to network risk analysis. In Goodall et al. [2008], pages 60–67. ISBN 978-3-540-85931-4. 48

Orion. Orion telecom homepage. http://www.oriontelecom.com/. 1

J. Pach, editor. *Graph Drawing, 12th International Symposium, GD 2004, New York, NY, USA, September 29 - October 2, 2004, Revised Selected Papers*, volume 3383 of *Lecture Notes in Computer Science*, 2004. Springer-Verlag. ISBN 3-540-24528-6. 233, 242, 247

S. E. Palmer. *VISION SCIENCE: Photons to Phenomenology*. Bradford Books, MIT Press Cambridge, 1999. 13

A. Papakostas and I. G. Tollis. Issues in interactive orthogonal graph drawing. In *Proceedings of the Symposium on Graph Drawing*, GD '95, pages 419–430. Springer-Verlag, 1996. ISBN 3-540-60723-4. URL http://portal.acm.org/citation.cfm?id=647547.728743. 187, 194

A. Papakostas and I. G. Tollis. Interactive orthogonal graph drawing. *IEEE Trans. Comput.*, 47:1297–1309, November 1998. ISSN 0018-9340. doi: 10.1109/12.736444. URL http://portal.acm.org/citation.cfm?id=305086.305097. 194

A. Papakostas, J. M. Six, and I. G. Tollis. Experimental and theoretical results in interactive orthogonal graph drawing. In *Proceedings of the Symposium on Graph Drawing*, GD '96, pages 371–386. Springer-Verlag, 1997. ISBN 3-540-62495-3. URL http://portal.acm.org/citation.cfm?id=647548.728766. 194

M. Patrignani. Visualization of large graphs. Doctoral Thesis. Universita' degli Studi di Roma "La Sapienza", Dottorato di Ricerca in Ingegneria Informatica, XIII Ciclo, 2001. 18

A. Pavlo, C. Homan, and J. Schull. A parent-centered radial layout algorithm for interactive graph visualization and animation. *CoRR*, abs/cs/0606007, 2006. 196

G. Pavlopoulos, A.-L. Wegener, and R. Schneider. A survey of visualization tools for biological network analysis. *BioData Mining*, 1(1):12, 2008. ISSN 1756-0381. doi: 10.1186/1756-0381-1-12. URL http://www.biodatamining.org/content/1/1/12. 32

PDB. Protein data bank homepage. http://www.rcsb.org/pdb/. 22

M. Petre. Why looking isn't always seeing: Readership skills and graphical programming. *Commun. ACM*, 38(6):33–44, 1995. 54

M. Petre. Mental imagery and software visualization in high-performance software development teams. *J. Vis. Lang. Comput.*, 21(3):171–183, 2010. 54

E. F. Pettersen, T. D. Goddard, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, and T. E. Ferrin. UCSF Chimera–a visualization system for exploratory research and analysis. *Journal of computational chemistry*, 25(13):1605–1612, October 2004. ISSN 0192-8651. doi: 10.1002/jcc.20084. URL `http://dx.doi.org/10.1002/jcc.20084`. 224

PG478. Final report: Project group 478 OGDF: An open graph drawing framework. `https://eldorado.tu-dortmund.de/handle/2003/23280`. 104

D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow map layout. In *Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, pages 219–224, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7803-9464-x. doi: 10.1109/INFOVIS.2005.13. URL `http://portal.acm.org/citation.cfm?id=1106328.1106595`. 42

T. Poranen, E. Mäkinen, and J. Nummenmaa. How to draw a sequence diagram. In P. Kilpeläinen and N. Päivinen, editors, *SPLST*, pages 91–102. University of Kuopio, Department of Computer Science, 2003. ISBN 951-781-265-5. 44, 80

PSR. Visualization of affiliations in a social network. `http://powerstructureresearch.wordpress.com/2009/05/27/`, 2009. 41

PubChem. Pubchem databases, national center for biotechnology information. `http://pubchem.ncbi.nlm.nih.gov/`. 15

H. Purchase and A. Samra. Extremes are better: Investigating mental map preservation in dynamic graphs. In G. Stapleton, J. Howse, and J. Lee, editors, *Diagrammatic Representation and Inference*, volume 5223 of *Lecture Notes in Computer Science*, pages 60–73. Springer Berlin / Heidelberg, 2008. URL `http://dx.doi.org/10.1007/978-3-540-87730-1_9`. 185

H. Purchase, E. Hoggan, and C. Görg. How important is the 'mental map'? an empirical investigation of a dynamic graph layout algorithm. In M. Kaufmann and D. Wagner, editors, *Graph Drawing*, volume 4372 of *Lecture Notes in Computer Science*, pages 184–195. Springer Berlin / Heidelberg, 2007. URL `http://dx.doi.org/10.1007/978-3-540-70904-6_19`. 185

H. C. Purchase. Metrics for graph drawing aesthetics. *J. Vis. Lang. Comput.*, 13(5):501–516, 2002. 61

H. C. Purchase, R. F. Cohen, and M. I. James. Validating graph drawing aesthetics. In Brandenburg [1996], pages 435–446. ISBN 3-540-60723-4. 59

H. C. Purchase, J.-A. Allder, and D. A. Carrington. User preference of graph layout aesthetics: A uml study. In Marks [2001], pages 5–18. ISBN 3-540-41554-8. 44, 63

H. C. Purchase, M. McGill, L. Colpoys, and D. A. Carrington. Graph drawing aesthetics and the comprehension of uml class diagrams: An empirical study. In P. Eades and T. Pattison, editors, *InVis.au*, volume 9 of *CRPIT*, pages 129–137. Australian Computer Society, 2001. ISBN 0-909-92587-9. 43, 44, 63

A. J. Quigley and P. Eades. Fade: Graph drawing, clustering, and visual abstraction. In Marks [2001], pages 197–210. ISBN 3-540-41554-8. 70

E. F. Reid and H. Chen. Mapping the contemporary terrorism research domain. *Int. J. Hum.-Comput. Stud.*, 65:42–56, January 2007. ISSN 1071-5819. doi: 10.1016/j.ijhcs.2006.08.006. URL `http://portal.acm.org/citation.cfm?id=1222244.1222621`. 40

K. Ryall. Glide. In *Graph Drawing*, pages 479–480, 2001. 101, 102

G. Sander. Graph layout for applications in compiler construction. *Theor. Comput. Sci.*, 217 (2):175–214, 1999. 142

SBGN. Systems biology graphical notation web page. `http://www.sbgn.org/`. 54

A.-W. Scheer, O. Thomas, and O. Adam. Process modelling using event-driven process chains. In *Process-Aware Information Systems*, pages 119–146. Wiley, 2005. 49

F. Schreiber. High quality visualization of biochemical pathways in biopath. *In Silico Biology*, 2:6, 2002. 25

F. Schreiber. Visual comparison of metabolic pathways. *J. Vis. Lang. Comput.*, 14(4):327–340, 2003. 28

F. Schreiber, T. Dwyer, K. Marriott, and M. Wybrow. A generic algorithm for layout of biological networks. *BMC Bioinformatics*, 10:375, 2009. 25, 67, 89, 90, 97

A. Schuffenhauer, P. Ertl, S. Roggo, S. Wetzel, M. A. Koch, and H. Waldmann. The scaffold tree - visualization of the scaffold universe by hierarchical scaffold classification. *J. Chem. Inf. Model.*, 47(1):47–58, 2007. 200, 201

B. Schwikowksi and T. Ideker. Personal communication, 2007. 20

B. Schwikowski, P. Uetz, and S. Fields. A network of protein-protein interactions in yeast. *Nat Biotech*, 18(12):1257–1261, 2000. doi: \url{http://dx.doi.org/10.1038/82360}. 20

R. Sharan and T. Ideker. Modeling cellular machinery through biological network comparison. *Nature biotechnology*, 24(4):427–433, April 2006. ISSN 1087-0156. doi: 10.1038/nbt1196. URL `http://dx.doi.org/10.1038/nbt1196`. 28

R. Sharan, S. Suthram, R. M. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R. M. Karp, and T. Ideker. Conserved patterns of protein interaction in multiple species. *Proceedings of the National Academy of Sciences of the United States of America. Copyright (2005) National Academy of Sciences, USA*, 102(6):1974–1979, February 2005. ISSN 0027-8424. doi: 10.1073/pnas.0409522102. URL `http://dx.doi.org/10.1073/pnas.0409522102`. 28, 29

C. Shirky. People on page: Yasns . . . corante's many-to-many. `http://many.corante.com/archives/2003/05/12/people_on_page_yasns.php`, 2003. 38

B. Shneiderman and C. Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Pearson, 5th edition, 2009. 57, 191, 192

C. Shulyupin. Linux kernel map. `http://www.makelinux.net/kernel_map.shtml`. 46

M. Siebenhaller. *Orthogonal Graph Drawing with Constraints: Algorithms and Applications*. PhD thesis, Universität Tübingen, Wilhelmstr. 32, 72074 Tübingen, 2009. URL `http://tobias-lib.uni-tuebingen.de/volltexte/2009/4448`. 83, 97

SocialAction. SocialAction project homepage. `http://www.cs.umd.edu/hcil/socialaction/`. 38

M. Spönemann, H. Fuhrmann, R. von Hanxleden, and P. Mutzel. Port constraints in hierarchical layout of data flow diagrams. In *Proceedings of the 17th International Symposium on Graph Drawing (GD'09)*, volume 5849 of *LNCS*, pages 135–146. Springer-Verlag, 2010. doi: 10.1007/978-3-642-11805-0_14. 42, 82, 97

S. Stein. *Modelling Method Extension for Service-Oriented Business Process Management*. PhD thesis, Christian-Albrechts-Universität zu Kiel, 2008. 51

U. Stelzl, U. Worm, M. Lalowski, C. Haenig, F. H. Brembeck, H. Goehler, M. Stroedicke, M. Zenkner, A. Schoenherr, S. Koeppen, J. Timm, S. Mintzlaff, C. Abraham, N. Bock, S. Kietzmann, A. Goedde, E. Toksöz, A. Droege, S. Krobitsch, B. Korn, W. Birchmeier, H. Lehrach, and E. E. Wanker. A human protein-protein interaction network: a resource for annotating the proteome. *Cell*, 122(6):957–968, Sep 2005. ISSN 0092-8674. 1

M.-A. D. Storey, F. D. Fracchia, and H. A. Müller. Customizing a fisheye view algorithm to preserve the mental map. *J. Vis. Lang. Comput.*, 10(3):245–267, 1999. 194

M. Suderman and M. Hallett. Tools for visually exploring biological networks. *Bioinformatics*, 23(20):2651–2659, 2007. doi: 10.1093/bioinformatics/btm401. URL `http://bioinformatics.oxfordjournals.org/cgi/content/abstract/23/20/2651`. 32

K. Sugiyama and K. Misue. Visualization of structural information: automatic drawing of compound digraphs. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(4):876–892, 1991. ISSN 0018-9472. doi: 10.1109/21.108304. 58, 60, 81, 141, 142

K. Sugiyama and K. Misue. Graph drawing by the magnetic spring model. *J. Vis. Lang. Comput.*, 6(3):217–231, 1995. 84

K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981. ISSN 0018-9472. doi: 10.1109/TSMC.1981.4308636. URL `http://dx.doi.org/10.1109/TSMC.1981.4308636`. 60, 64, 66, 71, 82, 98, 219

J. Z. Sun and C. Zhang. Advances on c-planarity testing of extrovert c-graphs. In H. R. Arabnia, Y. Mun, and P. L. Zhou, editors, *FCS*, pages 25–31. CSREA Press, 2008. ISBN 1-60132-066-3. 132

SUPERFAMILY. Superfamily is a database of structural and functional annotation for all proteins and genomes. `http://supfam.cs.bris.ac.uk/SUPERFAMILY/`. 22

I. E. Sutherland. Sketch pad a man-machine graphical communication system. In *DAC '64: Proceedings of the SHARE design automation workshop*, New York, NY, USA, 1964. ACM. doi: 10.1145/800265.810742. URL `http://dx.doi.org/10.1145/800265.810742`. 81

S. Takahashi, S. Matsuoka, K. Miyashita, H. Hosobe, and T. Kamada. A constraint-based approach for visualization and animation. *Constraints*, 3:61–86, 1998. ISSN 1383-7133. URL `http://dx.doi.org/10.1023/A:1009708715411`. 10.1023/A:1009708715411. 88, 97

R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987. 64, 68, 100

R. Tamassia. Constraints in graph drawing algorithms. *Constraints*, 3(1):87–120, 1998. ISSN 1383-7133. doi: http://dx.doi.org/10.1023/A:1009760732249. 81

R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, 18:61–79, January 1988. ISSN 0018-9472. doi: 10.1109/21.87055. URL `http://portal.acm.org/citation.cfm?id=46931.46937`. 67, 78

R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, June 1972. 177

J. J. Thomas and K. A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Ctr, 2005. ISBN 0769523234. URL `http://www.worldcat.org/isbn/0769523234`. 13

I. G. Tollis and M. Patrignani, editors. *Graph Drawing, 16th International Symposium, GD 2008, Heraklion, Crete, Greece, September 21-24, 2008. Revised Papers*, volume 5417 of *Lecture Notes in Computer Science*, 2009. Springer-Verlag. ISBN 978-3-642-00218-2. 237, 239, 243, 247, 249, 250

Tulip. Tulip framework. `http://tulip.labri.fr`. 39

D. Tunkelang. A practical approach to drawing undirected graphs. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994. 70

W. T. Tutte. How to draw a graph. *Proc Lond Math Soc*, 13:743–767, 1963. URL `http://www.ams.org/mathscinet-getitem?mr=28:1610`. 64, 69

W. T. Tutte. *Connectivity in graphs*, volume 15 of *Mathematical Expositions*. University of Toronto Press, 1966. 10

B. Tversky. Distortions in cognitive maps. *Geoforum*, 23(2):131 – 138, 1992. ISSN 0016-7185. doi: DOI:10.1016/0016-7185(92)90011-R. URL `http://www.sciencedirect.com/science/article/B6V68-466FGFY-1F/2/65df56cd6390c676bc0e38693d038cee`. 84, 185

UML. UML diagrams. `http://www.uml-diagrams.org/`, a. 44

UML. UML specifications, object management group (OMG). `http://www.omg.org/spec/UML/`, b. 2, 43

F. van Ham and B. Rogowitz. Perceptual organization in user-generated graph layouts. *IEEE Transactions on Visualization and Computer Graphics*, 14:1333–1339, November 2008. ISSN 1077-2626. doi: 10.1109/TVCG.2008.155. URL `http://portal.acm.org/citation.cfm?id=1477066.1477432`. 13

S. Vishveshwara, A. Ghosh, and P. Hansia. Intra and inter-molecular communications through protein structure network. *Curr Protein Pept Sci*, 10:146–160, 2009. 223

Visone. visone project web page. `http://visone.info`. 38

V. E. Waddle and A. Malhotra. An e log e line crossing algorithm for levelled graphs. In Kratochvíl [1999], pages 59–71. ISBN 3-540-66904-3. 219

K. Wagner. Bemerkungen zum vierfarbenproblem. In *Jahresbericht Deutscher Math. Verein*, volume 46, pages 26–32, 1936. 64

C. Walshaw. A multilevel algorithm for force-directed graph-drawing. *J. Graph Algorithms Appl.*, 7(3):253–285, 2003. 70

X. Wang and I. Miyamoto. Generating customized layouts. In Brandenburg [1996], pages 504–515. ISBN 3-540-60723-4. 85, 97

C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. ISBN 1558608192. 3, 12

C. Ware, H. Purchase, L. Colpoys, and M. McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1:103–110, June 2002. ISSN 1473-8716. doi: 10.1057/palgrave.ivs.9500013. URL `http://portal.acm.org/citation.cfm?id=942164.942167`. 62, 63

M. Wattenberg. Visual exploration of multivariate graphs. In R. E. Grinter, T. Rodden, P. M. Aoki, E. Cutrell, R. Jeffries, and G. M. Olson, editors, *CHI*, pages 811–819. ACM, 2006. ISBN 1-59593-372-7. 38

S. Wetzel, K. Klein, S. Renner, D. Rauh, T. I. Oprea, P. Mutzel, and H. Waldmann. Interactive exploration of chemical space with scaffold hunter. *Nat Chem Biol*, 5(8):581–583, 2009. 5, 210

C. E. Wheelock, A. M. Wheelock, S. Kawashima, D. Diez, M. Kanehisa, M. van Erk, R. Kleemann, J. Z. Haeggström, and S. Goto. Systems biology approaches and pathway tools for investigating cardiovascular disease. *Mol. BioSyst.*, 5:588–602, 2009. doi: 10.1039/B902356A. URL `http://dx.doi.org/10.1039/B902356A`. 20

H. D. White and K. W. McCain. Visualizing a discipline: An author co-citation analysis of information science, 1972-1995. *JASIS*, 49(4):327–355, 1998. 40

H. Whitney. Non-separable and planar graphs. *Transactions of the American Mathematical Society*, 34(2):339–362, 1932. 10

R. Wiese and M. Kaufmann. Adding constraints to an algorithm for orthogonal graph drawing. In S. Whitesides, editor, *Graph Drawing*, volume 1547 of *Lecture Notes in Computer Science*, pages 462–463. Springer Verlag, 1998. URL `http://dx.doi.org/10.1007/3-540-37623-2_47`. 10.1007/3-540-37623-2_47. 97

P. Willett, J. M. Barnard, and G. M. Downs. Chemical similarity searching. *Journal of Chemical Information and Computer Sciences*, 38(6):983–996, 1998. 35

K. Wong and D. Sun. On evaluating the layout of uml diagrams for program comprehension. *Software Quality Journal*, 14:233–259, 2006. ISSN 0963-9314. URL `http://dx.doi.org/10.1007/s11219-006-9218-2`. 10.1007/s11219-006-9218-2. 44

M. Wybrow, K. Marriott, and P. J. Stuckey. Incremental connector routing. In Healy and Nikolov [2006], pages 446–457. ISBN 3-540-31425-3. 195

M. Wybrow, K. Marriott, L. McIver, and P. J. Stuckey. Comparing usability of one-way and multi-way constraints for diagram editing. *ACM Trans. Comput.-Hum. Interact.*, 14(4), 2008. 90, 101

C.-H. Yeang and M. Vingron. A joint model of regulatory and metabolic networks. *BMC Bioinformatics*, 7:332, 2006. 31

K.-P. Yee, D. Fisher, R. Dhamija, and M. A. Hearst. Animated exploration of dynamic graphs with radial layout. In *INFOVIS*, pages 43–50, 2001. 196

J. S. Yi, Y.-A. Kang, J. Stasko, and J. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13:1224–1231, November 2007. ISSN 1077-2626. doi: http://dx.doi.org/10.1109/TVCG.2007.70515. URL `http://dx.doi.org/10.1109/TVCG.2007.70515`. 182

Q. Zhang, C. Zmasek, L. Dishaw, M. G. Mueller, Y. Ye, G. Litman, and A. Godzik. Novel genes dramatically alter regulatory network topology in amphioxus. *Genome Biology*, 9(8):R123, 2008. ISSN 1465-6906. doi: 10.1186/gb-2008-9-8-r123. URL `http://genomebiology.com/2008/9/8/R123`. 213

P. Zhou and Z. Shang. 2d molecular graphics: a flattened world of chemistry and biology. *Briefings in Bioinformatics*, 10(3):247–258, 2009. 224