10 Reasons to Get Interested in Graph Drawing

Carla Binucci[1], Ulrik Brandes[2], Tim Dwyer[3], Martin Gronemann[4], Reinhard von Hanxleden[5], Marc van Kreveld[6], Petra Mutzel[7(✉)], Marcus Schaefer[8], Falk Schreiber[9], and Bettina Speckmann[10]

[1] University of Perugia, Perugia, Italy
carla.binucci@unipg.it
[2] ETH Zurich, Zurich, Switzerland
ulrik.brandes@gess.ethz.ch
[3] Monash University, Melbourne, Australia
tim.dwyer@monash.edu
[4] University of Cologne, Cologne, Germany
gronemann@informatik.uni-koeln.de
[5] Kiel University, Kiel, Germany
rvh@informatik.uni-kiel.de
[6] Utrecht University, Utrecht, The Netherlands
m.j.vankreveld@uu.nl
[7] TU Dortmund University, Dortmund, Germany
petra.mutzel@cs.tu-dortmund.de
[8] DePaul University, Chicago, USA
MSchaefer@cdm.depaul.edu
[9] University of Konstanz, Konstanz, Germany
falk.schreiber@uni-konstanz.de
[10] TU Eindhoven, Eindhoven, The Netherlands
b.speckmann@tue.nl

Abstract. This is an invitation to the research area of graph drawing. It encompasses basic research such as graph theory, complexity theory, data structures, and graph algorithms as well as applied research such as software libraries, implementations, and applications. Application domains include areas within computer science (e. g., information visualization, software engineering, model-based design, automated cartography) as well as outside (e. g., molecular biology and the social sciences). A selection of results demonstrates the influence of graph drawing on other areas and vice versa.

Keywords: Graph drawing · Visualization · Complexity
Computational geometry · Software engineering

1 Introduction

The ultimate goal of graph drawing is to construct suitable visualizations of graphs and networks. While important contributions date back much further,

B. Steffen and G. Woeginger (Eds.): Computing and Software Science, LNCS 10000, pp. 85–104, 2019.
https://doi.org/10.1007/978-3-319-91908-9_6

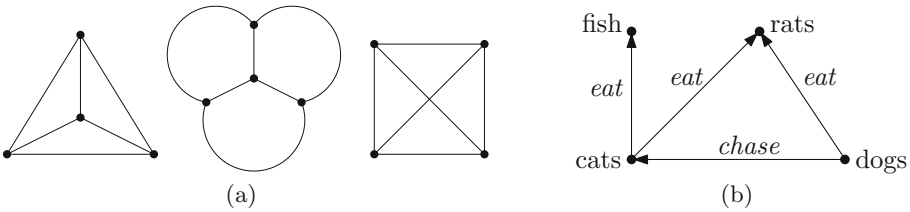


Fig. 1. (a) The graph K_4 in three different styles: planar straight-line, planar with circular arcs, and non-planar straight-line. (b) Graph with vertex and edge labels.

institutionalization began with an *International Work Meeting on Graph Drawing* in 1992. The first published proceedings appeared as *Lecture Notes in Computer Science* vol. 894 in 1994 and have appeared in this series ever since.[1] From the very beginning, the area has featured a unique combination of basic research in, for instance, topological graph theory, complexity, data structures, computational geometry, and optimization, research in various application domains, and practical research on implementations, tools, and usage.

We highlight ten characteristic topics of basic and applied research in graph drawing to incentivize readers to learn more about the area. A comprehensive overview is given in the *Handbook of Graph Drawing and Visualization* [80], and some open problems have been compiled by Brandenburg *et al.* [16].

2 Basic Research

The most common visual representation of a graph is a two-dimensional *drawing* in which points in the plane represent vertices and curves connecting them represent edges.

2.1 Computational Geometry

When drawing a graph we need to give the vertices coordinates and the edges shapes. Computational geometry is the field within algorithms research that is concerned with coordinates and shapes. Most of the aesthetic criteria that assess the quality of a drawing of a graph are geometric, and techniques from computational geometry can be used to compute them.

We call a graph *planar* if it can be drawn in the plane without edge crossings. Suppose we are given a planar straight-line drawing of a graph (see Fig. 1). Its *angular resolution* is the smallest angle in the drawing over any two edges incident to the same vertex. The *graph resolution* is the maximum ratio between the longest edge length and the shortest distance between distinct, non-incident features (two vertices, or a vertex and a non-incident edge). Both resolutions can be computed in linear time.

[1] See http://graphdrawing.org/ for a complete list.

The *area requirement* of a graph is the size of the integer grid needed to embed the graph so that all vertices lie on grid points. Commonly, the graph is planar and a planar drawing on the grid is required. The algorithm of de Fraysseix *et al.* [35] shows that any planar graph can be drawn planarly on the $2n-4$ by $n-2$ grid (see Fig. 2(b)). This bound was improved by Schnyder [71], who shows that an $n-2$ by $n-2$ grid suffices. It is also possible to use quality measures on faces. Since the "best" shape of a face is convex, one may wonder which planar graphs allow drawings where all faces are convex. Chrobak and Kant [24] showed that every triconnected planar graph allows a drawing where all bounded faces are convex. The vertices are chosen on an $n-2$ by $n-2$ grid.

The angular resolution of planar graph drawings can often be improved if one is willing to use curved edges; angular resolution must now be defined using tangents of curves at incident vertices. *Lombardi drawings* are plane drawings where all edges are circular and the angular resolution at every vertex is perfect [30]. Not all planar graphs admit a Lombardi drawing. Other variants from the straight-line edge style are edges with bends and thick edges. Especially the former is studied extensively in graph drawing.

For non-planar graphs, intersection angles of edges are important for readable, aesthetic graph drawings. This observation has led to the introduction of right-angle crossing drawings [29] and large angle crossing drawings [37] of non-planar graphs. How different drawing styles and aesthetic measures relate was investigated by Hoffmann *et al.* [44]. Figure 1(a) shows a K_4 drawing in three different styles, leading to a different optimal angular resolution in each style.

Often drawings also need labels, see Fig. 1(b). Automated label placement has been studied extensively in various research fields. To compute placements, text labels are usually represented by a rectangular bounding box. Labels should be placed close to the features they refer to, and they should not intersect each other, nor any other features. In this setting, label placement can be seen as an optimization problem related to packing; for an overview of results, see [54].

2.2 Graph Theory: Canonical Orderings

One of the most intuitive ways to draw a planar graph by hand is to add elements (vertices, edges, faces, etc.) of the graph in an incremental manner to an already existing drawing. This drawing usually satisfies certain properties that serve as an invariant during this process.

In 1988 de Fraysseix *et al.* [35] took this idea and introduced the so-called *canonical orderings* for maximal planar graphs (graphs to which no edge can be added without losing planarity). They used this order of the vertices in an algorithm to draw every maximal planar graph in a planar straight-line style on a grid of quadratic size (see Sect. 2.1 and Fig. 2). The canonical ordering as described in [35] requires the graph to be maximal planar. In case the input does not satisfy this constraint, one may augment it by simply triangulating it. This step, however, is not advisable for certain applications. A more general variant for triconnected planar graphs has been given by Kant [48]. His definition differs from the original one in that it uses an ordered partition of the vertices instead

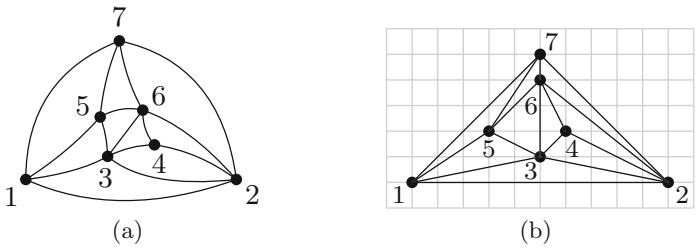


Fig. 2. (a) Example for a canonical ordering of a maximal planar graph and (b) incremental construction of a planar straight-line grid drawing using the algorithm in [35].

of a vertex ordering. A more detailed description of a linear-time algorithm to obtain such an ordering is given by Badent *et al.* [5]. They also show that a canonical ordering induces one in the dual of a triconnected planar graph.

Kant's definition has found numerous applications in graph drawing. Although the initial purpose was to draw planar graphs, it has been successfully applied to other graph-related problems. For example, Chiang *et al.* [23] use it to encode planar graphs with as few bits as possible. See [5] for an extensive list of applications.

Gronemann [39] suggested orderings for directed planar graphs based on *st*-orderings. This allows one to use techniques for undirected graphs to construct *upward planar drawings* (all arcs point upward). For undirected triconnected non-planar graphs, Schmidt [70] showed how to efficiently obtain a *Mondshein sequence*, a special non-separating ear decomposition similar to canonical orderings. With this result, Schmidt is able to improve the runtime to linear time for several algorithms, e. g., for finding independent spanning trees in triconnected graphs, which is the preprocessing step for querying internally disjoint paths.

2.3 Complexity: A Real Analogue of NP in Graph Drawing

In this section, we give some intuition for the fact that several problems in graph drawing with a geometric flavor like the rectilinear crossing number are computationally different from NP-complete problems like the crossing number.

The *existential theory of the reals*, $\exists\mathbb{R}$, is the set of all true, existential statements over the real numbers, such as $(\exists x, y)[xy = 1 \wedge x^2 + y^2 = 1]$, stating that the hyperbola intersects the unit circle, or, equivalently, the set of real satisfiable formulas like $[xy = 1 \wedge x^2 + y^2 = 1]$. $\exists\mathbb{R}$ is very expressive, particularly for graph drawing problems involving straight lines, convexity, or metric concepts. Take $\overline{\text{cr}}$, the *rectilinear crossing number*: Deciding whether a graph has a straight-line drawing with k crossings can be phrased in $\exists\mathbb{R}$. Since $\exists\mathbb{R}$ is decidable in polynomial space, we can compute $\overline{\text{cr}}$, at least in principle. A closer study reveals that many problems decidable via $\exists\mathbb{R}$ are computationally equivalent to it; this implies that solving them is likely hard, much harder than NP-complete problems. Similarly to NP, we can introduce a complexity class

$\exists\mathbb{R}$, the real satisfiability problem, as the set of problems which computationally reduce to ETR. Returning to the $\overline{\text{cr}}$-problem: Bienstock [9] showed that it is equivalent to ETR, so $\exists\mathbb{R}$-complete. Since ETR encodes NP-complete problems, this also implies that computing $\overline{\text{cr}}$ is NP-hard.

While ETR, like satisfiability for NP, can serve as a starting point to show $\exists\mathbb{R}$-completeness, there are problems closer to graph drawing that can fulfill this role. The two most fundamental ones are *stretchability of pseudoline arrangements*, deciding the question whether a pseudoline arrangement is isomorphic to a straight-line arrangement, a result due to Mnëv, and its projective dual, the realizability of a *chirotope* by a pointset. A promising third problem has been added to the list recently, realizability of an *allowable sequence* [45]. These three problems can serve as the starting point for reductions, like the Clique or Independent set problem for NP.

Intersection graphs, such as *string graphs* (Jordan arcs), and *interval graphs* (intervals on a line), can often be recognized in NP, but convexity seems to escalate the complexity to $\exists\mathbb{R}$. We know that recognizing intersection graphs of line segments (one of the oldest $\exists\mathbb{R}$-results, due to Kratochvíl and Matoušek [53]), rays, convex sets, disks and unit disks is $\exists\mathbb{R}$-complete. There are further $\exists\mathbb{R}$-complete problems in simultaneous graph drawing, visibility graphs, and metric problems such as unit distance and matchstick graphs, Delaunay triangulations, and problems related to angles and slopes. See [53,59] for a survey.

We conclude with some candidates for $\exists\mathbb{R}$-completeness: Does the rectilinear crossing number problem, $\overline{\text{cr}}(G) \leq k$, remain $\exists\mathbb{R}$-complete for fixed k? Is calculating the geometric thickness of a graph, or its maximum rectilinear crossing number $\exists\mathbb{R}$-complete? How hard is it to decide whether a graph has a straight-line drawing in which certain edges have to be free of crossings? For puzzle fans: How hard is it to tell whether a set of puzzle pieces can be placed into a given frame without overlapping (see Nagata's Arrow Puzzle)?

2.4 Data Structures: SPQR-Tree

Decomposition techniques often lead to efficient approaches for solving graph problems. The idea of using a decomposition in triconnected components goes back to MacLane (1937) and Tutte (1966) and has been used early in the graph algorithm literature (e. g., Bienstock and Monma [10]), but the methods became much easier using the data structure of SPQR-trees.

The data structure of SPQR-trees was suggested by Di Battista and Tamassia [26] in the context of graph drawing to represent the triconnected decomposition of a biconnected graph using series parts (S-nodes), parallel parts (P-nodes), and triconnected parts (R-nodes). Q-nodes denote single edges. Every node comes with a skeleton describing the whole graph with some parts contracted to an edge. For example, the skeleton of an S-node is a cycle, the skeleton of a P-node is a pair of vertices with some parallel edges, and the skeleton of an R-node is a triconnected component. The data structure combines these nodes in form of a tree (see Fig. 3). Since for planar graphs the skeletons are also planar, a combinatorial embedding of all the skeletons uniquely describes a planar embedding

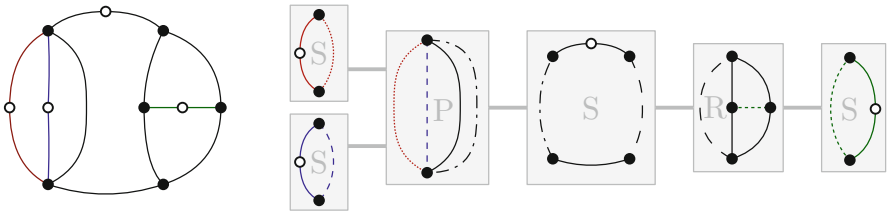


Fig. 3. (a) A biconnected graph and (b) its SPQR-tree with the skeletons (Q-nodes skipped). The edges in the skeletons represent either an original edge (solid) or a larger part of the graph (dashed, dotted).

of the whole graph and vice versa. Hence, SPQR-trees can be used to represent the set of all planar embeddings of a biconnected graph. This data structure can be computed in linear time and linear space [40].

SPQR-trees are applicable to problems that are easier to solve for triconnected graphs than for non-triconnected ones. This is particularly true for problems in which combinatorial embeddings play a crucial role. Another example are problems that can be solved in linear time for the class of series-parallel graphs. This data structure also works for problems in which divide-and-conquer methods work well.

SPQR-trees have been used heavily within the graph drawing community, e. g., to elegantly solve variations of planarity testing problems such as on-line, cluster or upward planarity testing, and to efficiently compute layouts (e. g., bend minimization and symmetric planar drawings). A survey on the SPQR tree data structure and its applications can be found in [63].

Outside graph drawing, SPQR-trees have been used for many different graph problems, e. g., for maintaining a minimum spanning tree and for solving triangulation problems. In computer-aided design they are important for solving layout decomposition problems in general multiple patterning lithography [87]. In business process management, the data structure has been used for developing process models and analyzing the control flow of business processes (e. g., [83]). SPQR-trees have also been used outside Computer Science: in electrical engineering for the generation of wave digital structures from reference circuits [34] and in theoretical physics for reducing Feynman integrals in perturbative quantum field theory [58].

3 Applications

3.1 Information Visualization

Information visualization is the research field concerned with all aspects of creating interactive visuals for *abstract* data. Abstract data are any data *without* inherent geometry: whether multivariate *tabular* data or—the chief concern of this article—relational data, where objects (which may have their own

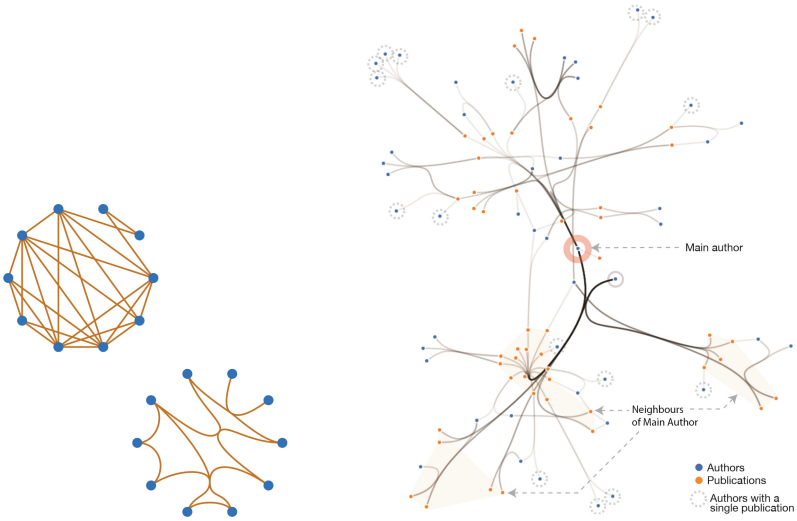


Fig. 4. Investigations of confluent drawing by a graph drawing paper and later by an information visualization paper. (a) Flat and outerplanar strict confluent drawings of the same graph by Eppstein *et al.* [33]. (b) Bach *et al.* [3] apply confluent drawing to a large authorship network.

attributes) are related to one another in various ways. The latter, of course, are networks or graphs. Network visualization has always been a core topic of information visualization. Papers on this topic presented at information visualization forums routinely cite—and are heavily inspired by—material originally presented at graph drawing forums, and vice-versa.

Speaking generally, the graph drawing community tends to be rigorous about developing efficient and correct algorithms and the theory to support these. In information visualization, the focus is more on applications and the human factors or usability of the methods. Just one such example is the idea of *confluent* drawings of graphs in which the edges are drawn in bundles to reduce clutter but in such a way that their connectivity remains clear (see Fig. 4). Confluent drawing was introduced in the graph drawing community first [28] primarily as a theoretical topic. More recently, the practical applications of this idea have been explored at InfoVis [3].

Another area where early work in graph drawing had significant impact upon information visualisation is *force-directed* layout. Graph drawers were the first to make this algorithm scale to large graphs with, for example, the Barnes-Hut cell opening criteria used in physics particle simulations by Tunkelang [82] and the first to make interactive, animated versions for online graph layout [31]. These ideas were later chosen for the force-directed layout implementation in D3, one of the most highly cited and influential InfoVis papers ever [15].

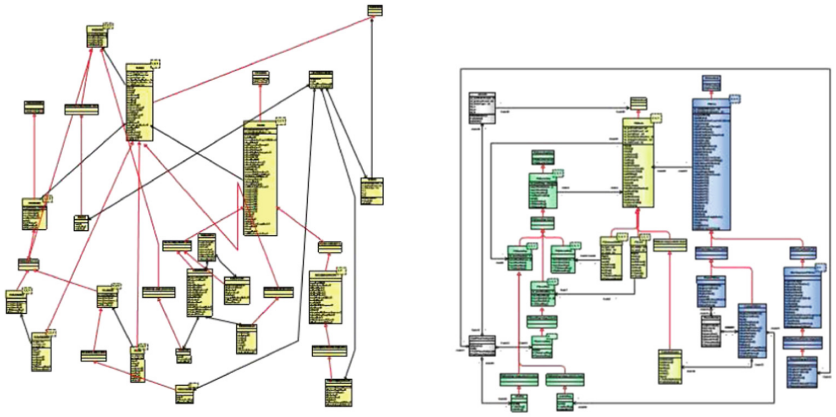
It is also interesting to consider the graph drawing approach to tree comparison, which has focused on crossing minimisation problems, e. g., Tanglegrams. By contrast, an early information visualization approach focused on interaction, e. g., Tree Juxtaposer [62]. Another major theme of tree visualization at InfoVis has been treemaps [76]. This design, developed by InfoVis researchers, inspired graph drawers to tackle the much harder problem of creating space-filling drawings of directed acyclic graphs at Graph Drawing [81]. In summary (and in keeping with the subject of trees), it is an extremely healthy cross-pollination that occurs between these two communities, helping both to grow and prosper.

3.2 Software Engineering

The field of software engineering concerns all the phases of the lifecycle of a software system: design, development, implementation, testing, and maintenance. Each of these phases may involve a large amount of data, thus requiring the use of visualization techniques to help software engineers in carrying out their job. Since the relationships and the interplay between data, objects, procedures, and architectural components of an architectural system are usually modeled as graphs, special attention has been devoted to the study of algorithms and user interfaces for the visualization of graphs in the scientific literature.

In the following we describe interconnections between software engineering and graph drawing. Early works that used graph drawing techniques focused on the automatic layout of Entity-Relationship diagrams [6] and data flow diagrams [7]. These papers are milestones since they are among the first applications of graph drawing to computer-aided software engineering and they devise a new strategy to incrementally build a graph layout. This strategy, called the *topology-shape-metrics (TSM)* approach, has been formally defined and made popular by a work of Tamassia [79] and it aims to compute a drawing of the graph in an *orthogonal* style (vertices are drawn as points or rectangles and edges are drawn as chains of horizontal and vertical segments).

The object-oriented programming paradigm became extremely popular in the late'90s and motivated the introduction of the *Unified Modeling Language (UML)*, a universal formalism intended to visually describe the architecture and the behavior of a software system at different levels of abstractions. In particular, *class diagrams* are among the most adopted types of UML diagrams. They are based on the use of graphs and are helpful in the design of a software architecture in terms of its classes (vertices of the graph) and their relationships (edges of the graph). These class diagrams required new graph drawing research. One of the main challenges for automatic visualization of a class diagram is to clearly show different types of relationships that such a diagram can have: some relationships (e. g., generalizations) correspond to oriented edges that describe the hierarchical structure (inheritance) of the classes, while other types of relationships correspond to non-oriented edges. Moreover, labels (both for the vertices and for the edges) [13,50] and clustering information (to model containment relations) [27,42] must be taken into account in the layout.



(a) (b)

Fig. 5. Two layouts of the same UML class diagram taken from [41]: (a) an industrial layout; (b) a layout based on the extended TSM approach (OGDF).

Several techniques have been proposed in the graph drawing literature to automatically visualize a class diagram. The layouts computed by the first commercial tools were mainly based on the well-known *layered* approach of Sugiyama *et al.* [78], without distinguishing between directed and undirected edges. According to this approach vertices are suitably distributed on different horizontal layers. Seemann [75] was the first to propose a modified version of the layered approach, considering separately directed and undirected edges.

The approaches in [32,41] proposed new drawing algorithms that exploit and extend the TSM approach in order to handle *mixed* graphs (i. e., graphs with both directed and undirected edges), vertices of prescribed size, and clusters of vertices. These algorithms produce significant improvements with respect to the layered approach (see Fig. 5) and their implementations are integrated in software libraries and systems, like OGDF and the yFiles library. Alternative techniques have been described for dealing with mixed graphs [12,14], vertices of prescribed size, and orthogonal drawings with prescribed clusters of vertices (see [80]).

We finally mention that some tools for software documentation integrate graph visualization facilities to automatically generate class diagrams of object-oriented software from annotated source code. Among them, Doxygen[2] is widely used and adopts the layered drawing algorithm available in the GraphViz library.

3.3 Model-Based Design

Model-based design (MBD), also referred to as *model-based development* or *model-driven engineering*, is a design methodology where some artefact, referred

[2] http://www.stack.nl/~dimitri/doxygen/.

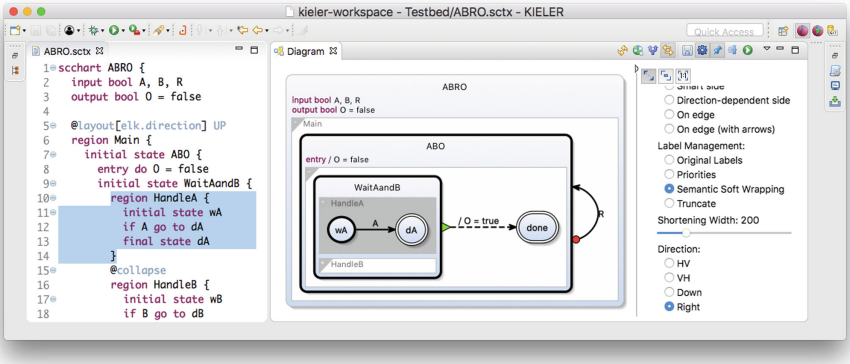


Fig. 6. An SCChart modeled with KIELER. The graphical view (center) is synthesized automatically from the textual ABRO.sctx model (left). *Layout directives* (starting with @) govern the filtering and drawing, e. g., region HandleA is collapsed. The view also helps to navigate in the model; here, the user has clicked in region HandleB, which selects the corresponding part in the text. The control panel (right) gives further options on layout and filtering, concerning for example the shortening of labels.

to as *system under development* (SUD), is created based on some model(s) of it. This model (or collection of models) is initially rather abstract, concentrating on *what* the SUD is supposed to do, and only in later—possibly automated—development stages it is specified *how* the SUD does what it does. The models tend to use a graphical instead of a textual syntax; as Schätz *et al.* put it, "Intuitively, model-based development means to use diagrams instead of code" [69]. As argued in this section, automated graph drawing is not as systematically employed in MBD as it could and should be.

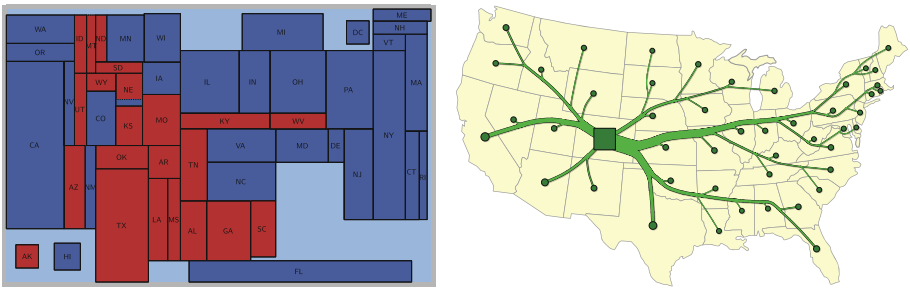
It is common practice especially in the development of cyber-physical systems to start with a graphical model of the SUD and often also its environment, and synthesize (textual) code for generating software or hardware from this model. There are numerous commercially successful tools that support this, such as Matlab/Simulink (from Mathworks), LabVIEW (National Instruments), ASCET (ETAS) or SCADE (Ansys/Esterel Technologies). The typical scenario is that the modeler manually creates a drawing (or *view*) of the model, using an initially empty drawing canvas and a palette from which graphical elements are dragged and dropped onto the canvas. This can be very time consuming; Petre quotes a developer: "I quite often spend an hour or two just moving boxes and wires around, with no change in functionality, to make it that much more comprehensible when I come back to it" [67]. When creating or changing a model, an estimated 30% of a user's time is spent on manual layout adjustments according to Klauske and Dziobek [51]. In particular programmers who are used to powerful text editors and integrated development environments (IDEs) such as Eclipse often find working with today's graphical editors rather cumbersome.

Ideally, one would like that modelers can focus their efforts on the models they work with, and do not have to spend significant time on mechanical drawing activities, just like today's circuit developers leave the place-and-route step typically to automation. This is also advocated in *modeling pragmatics*, which concerns all practical aspects of handling a model in its design process [36]. The separation of *model* and *view* is in fact a classic design principle in software development, known as model-view-controller pattern. Applied to MBD, this means that customized views should be constructed automatically from a model. This, however, requires automated graph drawing capabilities. One modeling tool that follows this approach is KIELER, shown in Fig. 6, which uses the Eclipse Layout Kernel[3] (ELK), an open-source collection of numerous layout algorithms implemented in Java. However, to adapt this approach into common practice, there is a range of obstacles to overcome, ranging from fundamental difficulties and technical problems (such as properly dealing with comments [74]) to psychological issues, concerning various stakeholders in different communities. For example, todays modelers are just accustomed to creating the layout manually, just like early circuit designers were used to do manual placement and routing. Even though there seems to be a pretty clear case for the usage of graph drawing techniques to improve modeler productivity, as argued above, there is little pressure on the tool vendors to provide good solutions. Sometimes, however, there is no way around this; for example, when the visual syntax changes significantly from one tool version to the next, old models must be migrated automatically to the next version [68]. Also, while modelers are often unhappy with automatic layout results applied to "their" finished models that they have hand-crafted before, they seem much more open to automatic layout if it has been applied from the very beginning. But still, mechanisms that let modelers guide the layout and *layout stability*, meaning that small changes in the model should not lead to abrupt changes in the overall drawing, are important issues to be addressed for increasing the acceptance of automated graph drawing in MBD practice.

3.4 Automated Cartography

Graph drawing and cartography both use a certain degree of abstraction when visualizing data. The graph drawing perspective has hence been used to address several questions from automated cartography. Consider, for example, an administrative map of the countries of Europe. Such a map can be viewed as a graph in two ways: (1) the boundaries of the countries can be considered edges, and the three-country points are the most prominent vertices, and (2) the adjacencies of countries can be represented by a graph, dual to the first view. Also the information shown on certain maps can be seen as graphs to be drawn. A prominent example are the weighted trees of flow maps. Below we describe the main map types that relate to graph drawing.

[3] http://www.eclipse.org/elk.



(a) (b)

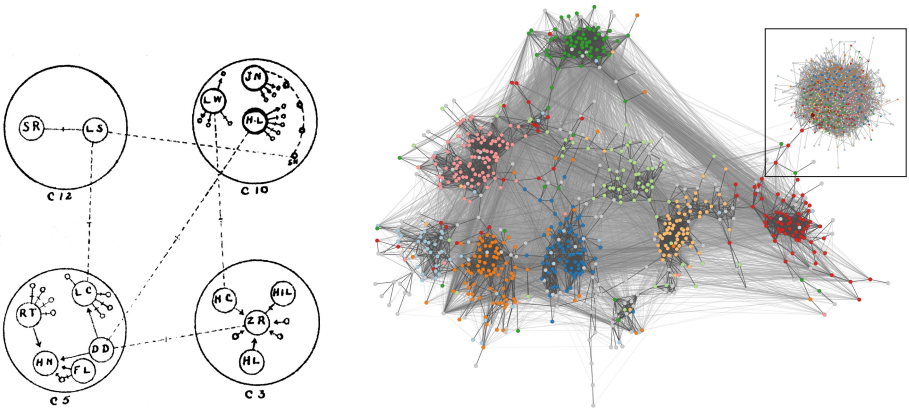
Fig. 7. (a) Rectangular cartogram of the 2008 US presidential election (from Buchin *et al.* [21]. (b) Flow map showing migration from Colorado (from Verbeek *et al.* [84]).

Cartograms show values for regions by shrinking and expanding those regions, so that the area of each region corresponds to the value represented, for example total population. Necessarily, cartograms show distorted regions.

The first algorithmic study of *rectangular cartograms*, where all regions are rectangles of specified sizes (Fig. 7(a)), is due to van Kreveld and Speckmann [55]; extensions and refinements were presented by Buchin *et al.* [21]. It is not always possible to realize the same rectangle adjacencies as the corresponding region adjacencies on a normal map. To overcome this, *rectilinear cartograms* were introduced, where regions can have more than four corners. De Berg *et al.* [8] showed that only constantly many corners per region are needed in rectilinear cartograms. Alam *et al.* [1] showed that eight corners is always enough. In *linear cartograms*, Euclidean distances between vertices represent values, such as travel time. Vertices must be placed correspondingly and the map will be distorted [11,47]. Alternatively, one can use distorted edges to represent travel time [19].

Flow maps show the movement of objects between geographic locations on a map using thick arrows (Fig. 7(b)). Edge bundling is often used to avoid visual clutter. Using a modification of Steiner trees, Buchin *et al.* [20] modelled this problem and gave an approximation algorithm, since a general formulation is NP-hard.

Schematic maps are commonly used for public transportation systems. Connections between major stations are drawn with polygonal lines that are highly abstracted: they have only a few segments with few orientations (horizontal, vertical, or slope $+1$ or -1). Cabello *et al.* [22] compute an order of the connections suitable for incremental placement, leading to an $O(n \log n)$ time algorithm. Neyer [64] views the problem as a line simplification problem and approximates each connection with the minimum number of segments in the specified orientations. Nöllenburg and Wolff [66] give an integer programming approach to the problem, respecting multiple constraints. Brandes and Wagner [17] draw connections between stations as circular arcs and address the visualization problem as a graph layout problem.



(a) manual layout [61] (b) density-adapted layout [65]

Fig. 8. Social networks of actors organized into (a) cottages (circles) and (b) dorms (colors). Layout is (a) manual taking known groups into account and (b) with a graph-drawing algorithm based on local density variation not knowing the clusters (inset shows result of straightforward force-directed layout).

3.5 Social Sciences

Graph drawing is relevant to much of the social sciences but its most direct association is with social structure and social relations. The analytic concept of *social networks* has been linked so closely with its representation as a graph that the use of related graph-theoretic techniques in any discipline is often considered an application of social network analysis.

Social networks in the strict sense consist of actors and the social ties that moderate their actions [43,85]. Variant types include affiliation networks (depending on context, represented as hypergraphs or bipartite graphs with a fixed bipartition), ego networks (represented with or without the defining focal actor who is in relationship with everyone else, and with or without relationships between the other actors), and longitudinal networks (given, for instance, as cross-sectional panel data, interval-censored aggregations, or relational events). Descriptive features include macro-level classifications such as being a core-periphery or small-world type network as well as structural properties such as cohesive groups, roles, and actor centralities. Statistical inference is often based on particular families of models for which there is a long history [38].

It was realized early on that visualization is not only for communication but that it can serve as a tool to explore the intricate and a-priori unknown patterns of complex webs of relationships [18]. The first known matrix representation of a graph of social relations goes back to the end of the 19th century [25] and Moreno's influential book [61] is full of hand-made graph drawings such as the one in Fig. 8(a).

While graphs associated with social relations often exhibit certain tendencies such as being sparse with one or more locally dense centers, low average

distance, and a skewed degree distribution, there is no guaranteed restriction to any particular class of graphs. Instead, layout problems are often associated with an analytic focus. A rule of thumb for effective visualization of social networks is that the aspect of interest defines layout constraints whereas the objective for the remaining degrees of freedom is to maximize readability. In this way, social networks provide a rich source of graph drawing problems, even if the resulting problems have often been addressed without this particular application in mind. Examples include (straight or radial) layered layout to depict actor centrality and status, clustered layout for (nested or flat) communities, and preprocessing techniques for skewed degree distributions.

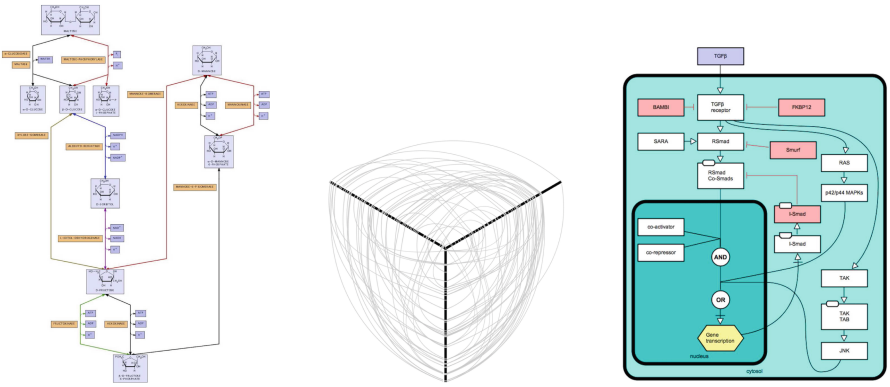
Actual use of graph drawing methods is limited, though. A lack of graphical standards and, more importantly, widely known and easy-to-use dedicated software tools hinders the routine practice of purposefully designed graphical illustrations that are prepared with the help of graph drawing algorithms. While the share of network visualization papers in the area of information visualization is increasing, the development of dedicated layout algorithms is lagging behind. Consequently, as in almost any applied area, the most widely used tools are relatively standard implementations of force-directed layout algorithms. Given the rich history of visualization in social network analysis, the variety of layout problems, and its increasing relevance due to the spread of online social networks, there is a lot to be gained by developing – and applying – more sophisticated layout algorithms. An example, a preprocessing technique to untangle small-world networks common in social media [65], is given in Fig. 8(b). Other heavily underexplored areas are network models, ensembles of networks, multilayer networks, and sequences of relational events.

3.6 Molecular Biology

Molecular biology is a subfield of biology which studies structure and function of cells at a molecular level. Cells are living objects composed of molecules such as DNA, proteins, and metabolites, that interact with each other in different ways. *Molecules* and their *interactions* play a central role, and structures based on these elements are commonly referred to as *biological networks*. Examples include gene regulatory and metabolic networks. In addition, there are further graphs derived from those elements such as phylogenetic trees and correlation networks. See [46] for an overview of networks in molecular biology.

These structures are often represented by multivariate networks which differ in both the semantics of vertices and edges as well as the data attached to vertices and edges. Examples are undirected graphs (e. g., for protein interaction networks), rooted trees (phylogenetic trees) and hyper-graphs (metabolic networks), often containing additional attributes attached to vertices and edges. Figure 9 shows some examples, Kohlbacher *et al.* present more information about multivariate networks in the life sciences in [52].

While manual drawings of tree-like information such as the tree of life appeared at least at the beginning of the 19th century [77], the earliest drawings of cellular networks are most likely of metabolic (sub)pathways in the early 20th



(a) Metabolic network (hyper-graph) visualized with an adapted layered approach [72].

(b) Regulatory network (directed graph) visualized with a hive plot layout [56].

(c) Signal transduction network (directed graph) visualized manually in the SBGN AF standard [60].

Fig. 9. Some biological networks and related layout methods.

century, for example, the glucose fermentation pathway proposed by Wohl in 1907 [86]. A huge number of biological networks have been drawn manually, and manual drawings are still common nowadays for illustrations in publications, in electronic systems such as the well-known KEGG database and so on.

When graph drawing algorithms became available, they were first used to compute visualizations for presentations (e. g., for networks derived from databases as in [49]), later employed to support the discovery process such as to investigate structure, connectivity, or hubs in such networks, and finally novel layout algorithms were developed tailored to specific networks (e. g., see Fig. 9) or—as generic algorithms—for different visualization tasks (e. g. [73]). Applications and specific adaptions of common graph drawing algorithms for the visualization of biological networks are detailed in [4]. Examples for specific layout methods motivated by biological questions or data characteristics are power-graph layout, which reduces the network complexity by explicitly representing re-occurring network motifs, and hive plot layout, which is a parallel coordinate layout of a graph with radially arranged axes, see also Fig. 9.

Standardised representations, ontologies and taxonomies are common in biological sciences, an early example is Linnaeus' taxonomy from 1735 [57]. Recent developments also include graphical standards: SBGN (Systems Biology Graphical Notation) covers the graphical representation of major networks and processes in molecular biology. The specifications of the three SBGN languages not only defines glyphs for vertices and edges, their syntax and semantics, but also contain rules and recommendations for a good layout of these networks.

Graph drawing is well established in molecular biology as method to visualize biological networks. Similar to other areas discussed earlier, molecular biology is not only a field of science which uses and applies graph drawing algorithms,

but also an interesting source of new problems in graph drawing. This field offers a broad range of layout problems for multivariate graphs and, given the increasing size, complexity and availability of the data, there is huge interest for better visualization (layout) and exploration methods. Some open problems in biological network visualization are presented by Albrecht *et al.* [2].

Acknowledgements. This work was supported by the DFG under the project *Compact Graph Drawing with Port Constraints* (DFG HA 4407/8-1 and MU 1129/9-1). Marc van Kreveld is supported by the Netherlands Organisation for Scientific Research (NWO) on project no. 612.001.651. Bettina Speckmann is supported by the Netherlands Organisation for Scientific Research (NWO) on project no. 639.023.208.

References

1. Alam, M., Biedl, T., Felsner, S., Kaufmann, M., Kobourov, S., Ueckerdt, T.: Computing cartograms with optimal complexity. Discret. Comput. Geom. **50**(3), 784–810 (2013)
2. Albrecht, M., Kerren, A., Klein, K., Kohlbacher, O., Mutzel, P., Paul, W., Schreiber, F., Wybrow, M.: On open problems in biological network visualization. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 256–267. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11805-0_25
3. Bach, B., Riche, N.H., Hurter, C., Marriott, K., Dwyer, T.: Towards unambiguous edge bundling: investigating confluent drawings for network visualization. IEEE Trans. Vis. Comput. Graph. **23**(1), 541–550 (2017)
4. Bachmaier, C., Brandes, U., Schreiber, F.: Biological networks. In: Handbook of Graph Drawing and Visualization, pp. 621–651. Chapman and Hall/CRC, Boco Raton (2014)
5. Badent, M., Brandes, U., Cornelsen, S.: More canonical ordering. J. Graph Algorithms Appl. **15**(1), 97–126 (2011)
6. Batini, C., Talamo, M., Tamassia, R.: Computer aided layout of entity relationship diagrams. J. Syst. Softw. **4**(2), 163–173 (1984)
7. Batini, C., Nardelli, E., Tamassia, R.: A layout algorithm for data flow diagrams. IEEE Trans. Softw. Eng. **12**(4), 538–546 (1986)
8. de Berg, M., Mumford, E., Speckmann, B.: On rectilinear duals for vertex-weighted plane graphs. Discret. Math. **309**(7), 1794–1812 (2009)
9. Bienstock, D.: Some provably hard crossing number problems. Discret. Comput. Geom. **6**(5), 443–459 (1991)
10. Bienstock, D., Monma, C.: On the complexity of embedding planar graphs to minimize certain distance measures. Algorithmica **5**, 93–109 (1990)
11. Bies, S., van Kreveld, M.: Time-space maps from triangulations. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 511–516. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36763-2_45
12. Binucci, C., Didimo, W.: Computing quasi-upward planar drawings of mixed graphs. Comput. J. **59**(1), 133–150 (2016)
13. Binucci, C., Didimo, W., Liotta, G., Nonato, M.: Orthogonal drawings of graphs with vertex and edge labels. Comput. Geom. **32**(2), 71–114 (2005)
14. Binucci, C., Didimo, W., Patrignani, M.: Upward and quasi-upward planarity testing of embedded mixed graphs. Theoret. Comput. Sci. **526**, 75–89 (2014)

15. Bostock, M., Ogievetsky, V., Heer, J.: D^3 data-driven documents. IEEE Trans. Vis. Comput. Graph. **17**(12), 2301–2309 (2011)
16. Brandenburg, F., Eppstein, D., Goodrich, M.T., Kobourov, S., Liotta, G., Mutzel, P.: Selected open problems in graph drawing. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 515–539. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24595-7_55
17. Brandes, U., Wagner, D.: Using graph layout to visualize train interconnection data. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 44–56. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-37623-2_4
18. Brandes, U., Freeman, L.C., Wagner, D.: Social networks. In: Tamassia, R. (ed.) Handbook of Graph Drawing and Visualization, pp. 805–839. Chapman and Hall/CRC, Boca Raton (2013)
19. Buchin, K., van Goethem, A., Hoffmann, M., van Kreveld, M., Speckmann, B.: Travel-time maps: linear cartograms with fixed vertex locations. Geograph. Inf. Sci. (GIScience) **2014**, 18–33 (2014)
20. Buchin, K., Speckmann, B., Verbeek, K.: Angle-restricted Steiner arborescences for flow map layout. Algorithmica **72**(2), 656–685 (2015)
21. Buchin, K., Speckmann, B., Verdonschot, S.: Evolution strategies for optimizing rectangular cartograms. GIScience **2012**, 29–42 (2012)
22. Cabello, S., de Berg, M., van Kreveld, M.: Schematization of networks. Comput. Geom. **30**(3), 223–228 (2005)
23. Chiang, Y.T., Lin, C.C., Lu, H.I.: Orderly spanning trees with applications to graph encoding and graph drawing. In: SODA 2001, pp. 506–515. SIAM (2001)
24. Chrobak, M., Kant, G.: Convex grid drawings of 3-connected planar graphs. Int. J. Comput. Geom. Appl. **7**(3), 211–223 (1997)
25. Delitsch, J.: Über Schülerfreundschaften in einer Volksschulklasse. Zeitschrift für Kinderforschung **5**(4), 150–163 (1900)
26. Di Battista, G., Tamassia, R.: On-line graph algorithms with SPQR-trees. In: Paterson, M.S. (ed.) ICALP 1990. LNCS, vol. 443, pp. 598–611. Springer, Heidelberg (1990). https://doi.org/10.1007/BFb0032061
27. Di Battista, G., Didimo, W., Marcandalli, A.: Planarization of clustered graphs. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 60–74. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45848-4_5
28. Dickerson, M., Eppstein, D., Goodrich, M.T., Meng, J.Y.: Confluent drawings: visualizing non-planar diagrams in a planar way. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 1–12. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24595-7_1
29. Didimo, W., Eades, P., Liotta, G.: Drawing graphs with right angle crossings. Theoret. Comput. Sci. **412**(39), 5156–5166 (2011)
30. Duncan, C.A., Eppstein, D., Goodrich, M.T., Kobourov, S.G., Nöllenburg, M.: Lombardi drawings of graphs. J. Graph Algorithms Appl. **16**(1), 85–108 (2012)
31. Eades, P., Cohen, R.F., Huang, M.L.: Online animated graph drawing for web navigation. In: Di Battista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 330–335. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63938-1_77
32. Eiglsperger, M., Gutwenger, C., Kaufmann, M., Kupke, J., Jünger, M., Leipert, S., Klein, K., Mutzel, P., Siebenhaller, M.: Automatic layout of UML class diagrams in orthogonal style. Inf. Visual. **3**(3), 189–208 (2004)
33. Eppstein, D., Holten, D., Löffler, M., Nöllenburg, M., Speckmann, B., Verbeek, K.: Strict confluent drawing. In: Wismath, S., Wolff, A. (eds.) GD 2013. LNCS, vol. 8242, pp. 352–363. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03841-4_31

34. Franken, D., Ochs, J., Ochs, K.: Generation of wave digital structures for networks containing multiport elements. Trans. Circuits Syst. **52**(3), 586–596 (2005)
35. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica **10**(1), 41–51 (1990)
36. Fuhrmann, H., von Hanxleden, R.: Taming graphical modeling. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010. LNCS, vol. 6394, pp. 196–210. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16145-2_14
37. Giacomo, E.D., Didimo, W., Liotta, G., Meijer, H.: Area, curve complexity, and crossing resolution of non-planar graph drawings. Theory Comput. Syst. **49**(3), 565–575 (2011)
38. Goldenberg, A., Zheng, A.X., Fienberg, S.E., Airoldi, E.M.: A survey of statistical network models. Found. Trends Mach. Learn. **2**(2), 129–233 (2010)
39. Gronemann, M.: Bitonic *st*-orderings for upward planar graphs. In: Hu, Y., Nöllenburg, M. (eds.) GD 2016. LNCS, vol. 9801, pp. 222–235. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50106-2_18
40. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44541-2_8
41. Gutwenger, C., Jünger, M., Klein, K., Kupke, J., Leipert, S., Mutzel, P.: A new approach for visualizing UML class diagrams. In: Diehl, S., Stasko, J.T., Spencer, S.N. (eds.) Symposium on Software Visualization 2003, pp. 179–188. ACM (2003)
42. Gutwenger, C., Jünger, M., Leipert, S., Mutzel, P., Percan, M., Weiskircher, R.: Advances in *C*-planarity testing of clustered graphs. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 220–236. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36151-0_21
43. Hennig, M., Brandes, U., Pfeffer, J., Mergel, I.: Studying Social Networks - A Guide to Empirical Research. Campus Frankfurt, New York (2012)
44. Hoffmann, M., van Kreveld, M.J., Kusters, V., Rote, G.: Quality ratios of measures for graph drawing styles. In: 26th Canadian Conference on Computational Geometry, CCCG (2014)
45. Hoffmann, U.: Intersection graphs and geometric objects in the plane. Ph.D. thesis, Technische Universität Berlin, Berlin (2016)
46. Junker, B.H., Schreiber, F.: Analysis of Biological Networks. Wiley Series on Bioinformatics, Computational Techniques and Engineering. Wiley, New York (2008)
47. Kaiser, C., Walsh, F., Farmer, C., Pozdnoukhov, A.: User-centric time-distance representation of road networks. GIScience **2010**, 85–99 (2010)
48. Kant, G.: Drawing planar graphs using the canonical ordering. Algorithmica **16**, 4–32 (1996)
49. Karp, P.D., Paley, S.M.: Automated drawing of metabolic pathways. In: Lim, H., Cantor, C., Bobbins, R. (eds.) International Conference on Bioinformatics and Genome Research, pp. 225–238 (1994)
50. Klau, G.W., Mutzel, P.: Combining graph labeling and compaction. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 27–37. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-46648-7_3
51. Klauske, L.K., Dziobek, C.: Improving modeling usability: Automated layout generation for Simulink. In: Proc. MathWorks Automotive Conference (2010)
52. Kohlbacher, O., Schreiber, F., Ward, M.O.: Multivariate networks in the life sciences. In: Kerren, A., Purchase, H.C., Ward, M.O. (eds.) Multivariate Network Visualization. LNCS, vol. 8380, pp. 61–73. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06793-3_4

53. Kratochvíl, J., Matoušek, J.: Intersection graphs of segments. J. Combin. Theory Ser. B **62**(2), 289–315 (1994)
54. van Kreveld, M.: Geographic information systems (Chap. 59). In: Goodmann, J., O'Rourke, J., Toth, C. (eds.) Handbook of Discrete and Computational Geometry, 3rd edn. Chapman & Hall/CRC, Boca Raton (2017)
55. van Kreveld, M., Speckmann, B.: On rectangular cartograms. Comput. Geom. **37**(3), 175–187 (2007)
56. Krzywinski, M., Birol, I., Jones, S.J., Marra, M.A.: Hive plots - rational approach to visualizing networks. Brief. Bioinform. **13**, 627–644 (2012)
57. Linnaei, C.: Species Plantarum. Holmiae (1735)
58. von Manteuffel, A., Studerus, C.: Reduze 2–distributed Feynman integral reduction. CoRR (2012)
59. Matousek, J.: Intersection graphs of segments and $\exists\mathbb{R}$. arXiv:1406.2636 (2014)
60. Mi, H., Schreiber, F., Moodie, S., Czauderna, T., Demir, E., Haw, R., Luna, A., Novère, N.L., Sorokin, A., Villéger, A.: Systems biology graphical notation: activity flow language level 1 version 1.2. J. Integr. Bioinform. **12**(2), e265 (2015)
61. Moreno, J.L.: Who Shall Survive? Foundations of Sociometry, Group Psychotherapy and Sociodrama. Beacon House, New York (1953). (First published in 1934)
62. Munzner, T., Guimbretière, F., Tasiran, S., Zhang, L., Zhou, Y.: TreeJuxtaposer: scalable tree comparison using focus+ context with guaranteed visibility. ACM Trans. Graph. (TOG) **22**(3), 453–462 (2003)
63. Mutzel, P.: The SPQR-tree data structure in graph drawing. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 34–46. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45061-0_4
64. Neyer, G.: Line simplification with restricted orientations. In: Dehne, F., Sack, J.-R., Gupta, A., Tamassia, R. (eds.) WADS 1999. LNCS, vol. 1663, pp. 13–24. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48447-7_2
65. Nocaj, A., Ortmann, M., Brandes, U.: Untangling the hairballs of multi-centered, small-world online social media networks. J. Graph Algorithms Appl. **19**(2), 595–618 (2016)
66. Nöllenburg, M., Wolff, A.: Drawing and labeling high-quality metro maps by mixed-integer programming. IEEE Trans. Vis. Comp. Graph. **17**(5), 626–641 (2011)
67. Petre, M.: Why looking isn't always seeing: readership skills and graphical programming. Commun. ACM **38**(6), 33–44 (1995)
68. Rüegg, U., Lakkundi, R., Prasad, A., Kodaganur, A., Schulze, C.D., von Hanxleden, R.: Incremental diagram layout for automated model migration. In: ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, MODELS 2016, pp. 185–195. ACM, New York (2016)
69. Schätz, B., Pretschner, A., Huber, F., Philipps, J.: Model-based development of embedded systems. In: Bruel, J.-M., Bellahsene, Z. (eds.) OOIS 2002. LNCS, vol. 2426, pp. 298–311. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46105-1_34
70. Schmidt, J.M.: The Mondshein sequence. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 967–978. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43948-7_80
71. Schnyder, W.: Embedding planar graphs on the grid. In: Proceedings of 1st Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 138–148 (1990)
72. Schreiber, F.: High quality visualization of biochemical pathways in BioPath. Silico Biol. **2**(2), 59–73 (2002)
73. Schreiber, F., Dwyer, T., Marriott, K., Wybrow, M.: A generic algorithm for layout of biological networks. BMC Bioinform. **10**, 375 (2009)

74. Schulze, C.D., von Hanxleden, R.: Automatic layout in the face of unattached comments. In: Proceedings of Symposium on Visual Languages and Human-Centric Computing (2014)
75. Seemann, J.: Extending the Sugiyama algorithm for drawing UML class diagrams: towards automatic layout of object-oriented software diagrams. In: DiBattista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 415–424. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63938-1_86
76. Shneiderman, B.: Tree visualization with tree-maps: 2-D space-filling approach. ACM Trans. Graph. **11**(1), 92–99 (1992)
77. Stevens, P.: Augustin Augier's "Arbre Botanique" (1801), a remarkable early botanical representation of the natural system. Taxon **32**, 203–211 (1983)
78. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst. Man. Cybern. **11**(2), 109–125 (1981)
79. Tamassia, R.: On embedding a graph in the grid with the minimum number of bends. SIAM J. Comput. **16**(3), 421–444 (1987)
80. Tamassia, R. (ed.): Handbook of Graph Drawing and Visualization. Chapman and Hall/CRC, Boca Raton (2013)
81. Tsiaras, V., Triantafilou, S., Tollis, I.G.: Treemaps for directed acyclic graphs. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 377–388. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77537-9_37
82. Tunkelang, D.: JIGGLE: Java interactive graph layout environment. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 413–422. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-37623-2_33
83. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data Knowl.Eng. **68**(9), 793–818 (2009)
84. Verbeek, K., Buchin, K., Speckmann, B.: Flow map layout via spiral trees. IEEE Trans. Vis. Comput. Graph. **17**(12), 2536–2544 (2011)
85. Wasserman, S., Faust, K.: Social Network Aanalysis. Methods and Applications. Cambridge University Press, Cambridge (1994)
86. Wohl, A.: Die neueren Ansichten über den chemischen Verlauf der Gärung. Biochemische Zeitschrift **5**, 45–64 (1907)
87. Zhang, Y., Luk, W.S., Zhou, H., Yan, C., Zeng, X.: Layout decomposition with pairwise coloring for multiple patterning lithography. In: Proceedings of International Conference on Computer-Aided Design, pp. 170–177. IEEE Press (2013)