Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 9, no. 1, pp. 7–18 (2005)

Drawing Graphs Within Graphs

Paul Holleis

Ludwig-Maximilians-Universität München, Germany
http://www.hcilab.org/paul/
paul@hcilab.org

Thomas Zimmermann

Universität des Saarlandes, Saarbrücken, Germany
http://www.st.cs.uni-sb.de/∼zimmerth/
zimmerth@cs.uni-sb.de

Daniel Gmach

Technische Universität München, Germany
http://www-db.in.tum.de/∼gmach/
daniel.gmach@in.tum.de

Abstract

The task of drawing subgraphs is often underestimated and they are simply emphasized using different colors or line styles. In this paper, we present an approach for drawing graphs within graphs that first produces a layout for the subgraphs thus increasing their locality. We introduce connection sets stressing relationships between several subgraphs. In a case study, we demonstrate how they can be used to visualize connections between many small network motifs.

This paper is based on a submission to the Graph Drawing Contest 2003 "Drawing Graphs within Graphs" [5]. The intent of the contest was to encourage research concerned with layouts of subgraphs.

Article Type	Communicated by	Submitted	Revised
regular paper	G. Liotta	January 2004	July 2005

1 Introduction

Graphs play an important role in many fields of research and economy, as well as in day-to-day life. They are often huge and complex structures that are difficult to understand. Subgraphs are a powerful tool for reducing complexity and emphasizing particular aspects. However, the handling of subgraphs is still not a prominent topic in graph drawing. Often subgraphs are not considered for the layout of the entire graph (even if they are known in advance), and only highlighted afterwards. For simple subgraphs this approach suffices but for complex subgraphs their structure is lost.

There are software packages that support the specification of subgraphs and consider them in one way or the other during the layout process, for instance the yEd tool [12]. However, most drawings with standard algorithms (like spring embedder methods, etc.) clutter the drawing with edges thus obscuring important information. In contrast, we build on a reduction of the number of edges, a sophisticated placement of subgraphs, and the possibility to choose different algorithms for different parts of the graph. This allows us to give a pleasing layout of the graph that includes good drawings for each subgraph and stresses the relationships between them. In our approach, we lay out all subgraphs before the entire graph. Thus, subgraphs are easier to spot as their locality is increased. For a single subgraph this is straightforward, therefore we concentrate on several subgraphs that are known in advance.

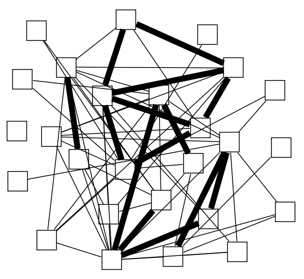
The remainder of this paper is organized as follows. In Section 2 we formalize our approach in general. Section 3 and 4 present a specialized layout algorithm for emphasizing connections between subgraphs. In Section 5, we apply the idea to many similar subgraphs. We close with a discussion of our approach in Sections 6 and 7.

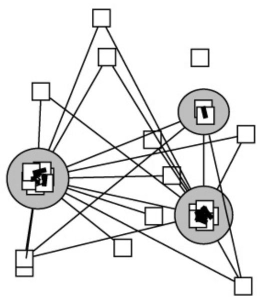
2 General Approach

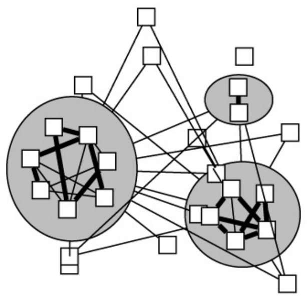
First, we present a generic framework for emphasizing multiple subgraphs within one graph, to be able to deal with different tasks and different graphs. Later we will specialize this framework and introduce applications for it.

Create a Summary Graph. In the first step, we create a graph in which each subgraph H_i is represented as a single vertex. We refer to this graph as the *summary graph* $G_S = (V_S, E_S)$. The mapping between the original graph G and G_S is obtained by two functions $f_V : V \to V_S$ and $f_E : E \to E_S$. Note, that f_V and f_E may also be used to combine vertices and edges that are not part of a subgraph.

Lay out the Subgraphs and the Summary Graph. After having created the summary graph, we first lay out the subgraphs and then the summary graph. All graphs are laid out independently. That means that different layout algorithms may be used. If possible, the layout algorithm for the summary graph should use information about the size of the subgraphs.


(a) Original graph.


(b) Summary graph.


(c) Subgraphs laid out.

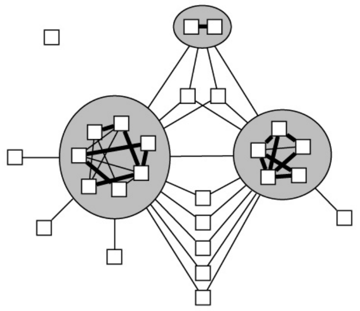

(d) Summary graph laid out.

Figure 1: Phases of the general approach.

Combine the Layouts of Subgraphs and Summary Graph. In the next step, we combine the layouts of the subgraphs and the summary graph to a layout of our representation of the original graph. This is done by replacing vertices that represent a subgraph by the layout of the respective subgraph. The mapping between the original graph and the summary graph can be restored using f_V^{-1} and f_E^{-1}. [1]

Optimize the Layout. The resulting layout is not optimal. In a final fine-tuning step we try to reduce the number of crossings and the required space of the layout.

Basically, our summary graphs are special cases of *X-Y-graphs* [2]. X denotes the structure of an X-Y-graph, and Y the property of some of its nodes. The particular restrictions on Y for a summary graph depend on each individual case.

We demonstrate the approach described above with one of the graphs of the Graph Drawing Contest 2003. The graph represents a social network that

[1]Note that f_V^{-1} and f_E^{-1} are relations and not functions.

evolved from an analysis of organizations involved in drug policy making. The network is an undirected graph where vertices represent organizations and the existence of informal communication chains is mirrored by edges between them. Since the collection of such data always implies uncertainty, *confirmed* relations are given precedence over others.

Figure 1(a) on the preceding page shows this graph, highlighting the *confirmed* edges using bold arcs. With some effort, one can see that the subgraph induced by *confirmed* relations consists of three connected components. However, neither the structure of the subgraph nor its relationship with the graph can be seen. That is why we decided to use the three components as three subgraphs. The graph is transformed into the summary graph in Figure 1(b).

In Figure 1(c) the *subgraphs* are laid out individually. For each, a spring embedder modeled after the Kamada-Kawai scheme is used for that [8]. An additional force from a circular boundary of each graph is applied to make them take up less space when surrounded by a circle as shown for instance in Figure 1(c). This can be done by temporarily adding a node in the center of the graph and connecting this node with all nodes of the graph.

Finally, the *summary graph* is laid out and all subgraphs are embedded in its layout in Figure 1(d). Note that the size of the subgraphs has been taken into account for the layout of the summary graph. This will be described in more detail later in Section 4.

This framework assumes that all subgraphs are disjoint. However, there are three possible solutions to cope with subgraphs that share nodes:

- Overlapping subgraphs can be *combined* into a new subgraph. This of course reduces the amount of information present in the choice of subgraphs.
- Vertices contained in several subgraphs can be *duplicated*, creating disjoint subgraphs. Edges incident to such nodes can either be duplicated as well or split into disjoint sets. This requires labeling those nodes in a way such that copies can be recognized.
- Subgraphs can be *partitioned* in overlapping and non-overlapping parts. Each part is then considered a separate subgraph.

Our approach works with multigraphs or self loops, provided that the algorithms used to draw the different parts of the layout can cope with those.

3 Connection Sets

Given multiple subgraphs, an interesting aspect is the relationship between them. We present a layout algorithm that emphasizes these relations by identifying *connection sets*. A connection set for some subgraphs contains all vertices that are reachable from each subgraph without visiting vertices of any other subgraph

Definition: Let $G = (V, E)$ be an undirected graph, and $H_1 = (V_1, E_1), \ldots,$ $H_n = (V_n, E_n)$ be subgraphs of G. The remainder sets are $V_R = V - \bigcup V_i$ and $E_R = E - \bigcup E_i$ with $i \in \{1, \ldots, n\}$. A *connection set* is defined as follows:

$$\begin{aligned} C_{\{i\}} &= \{\, v \mid \exists u \in V_i,\ \{w_1, \ldots, w_k, v\} \subset V_R, \\ &\qquad \exists\,\text{path } u \to w_1 \to \cdots \to w_k \to v \} \\ C_I &= \bigcap_{i \in I} C_{\{i\}} \end{aligned} \qquad \square$$

Note that the definition above allows vertices to be in more than one connection set. For example, if a vertex connects subgraphs H_1, H_2 and H_3 it is contained in seven connection sets $C_{\{1\}}$, $C_{\{2\}}$, $C_{\{3\}}$, $C_{\{1,2\}}$, $C_{\{1,3\}}$, $C_{\{2,3\}}$, and $C_{\{1,2,3\}}$. For layout purposes it serves better if such a vertex is contained in one set only. Therefore we define *exclusive connection sets* in which this vertex, e.g., would only be contained in the set $C^e_{\{1,2,3\}}$.

Definition: Let C_I be connection sets of an undirected graph G. We define an *exclusive connection set* as:

$$C^e_I = C_I - \bigcup_{i \in \{1,\ldots,n\},\ i \notin I} C_{\{i\}} \qquad \square$$

We can now use (exclusive) connection sets to create a layout that emphasizes subgraphs and their relationships to each other. For this we define the summary graph first.

Definition: Let $H_i = (V_i, E_i)$ with $i \in \{1, \ldots, n\}$ be n subgraphs of a graph $G = (V, E)$. We define a *summary graph based on connection sets* as $G_S = (V_S, E_S)$ with

$$\begin{aligned} V_S =& \{\, v_s \mid \exists v \in V \ \textit{with}\ v_s = f_V(v)\} \\ E_S =& \{\, e_s \mid \exists e \in E \ \textit{with}\ e_s = f_E(e)\} \end{aligned}$$

where

$$f_V(v) = \begin{cases} v_i & \text{if } v \in H_i \\ v & \text{else} \end{cases}$$

$$f_E\big(\{u_1, u_2\}\big) = \big\{f_V(u_1), f_V(u_2)\big\} \text{ for } u_1, u_2 \in V$$

$\square$

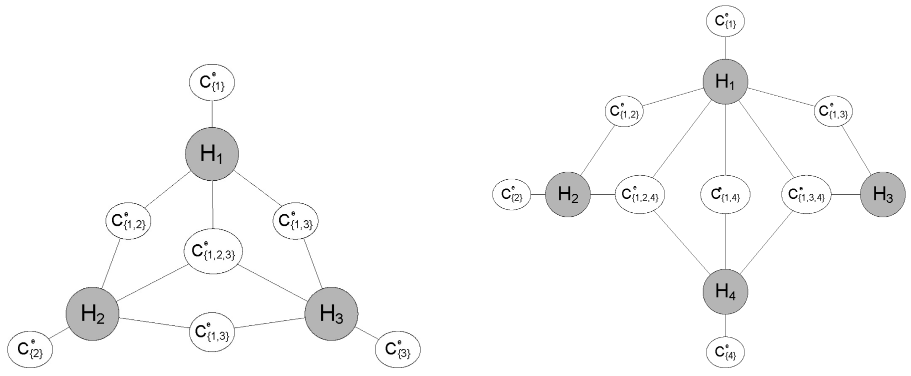
4 Layout of the Summary Graph

For the layout of the summary graph we assume that the subgraphs have been laid out in some way and surround those layouts with an ellipse.[2] These ellipses are placed on a circle, taking their size into account. For this, we calculate the (approximate) part of the perimeter that each ellipse will need on the circle.

[2]Other geometrical objects might fit better for specific layouts of the subgraphs but this would complicate their placement with respect to each other.

Then we place each ellipse and move the subgraphs along with their bounding ellipse. The connection sets C_I^e are put around and between the subgraphs, helping to minimize overlapping of vertices and edge crossings.

Figure 2(a) illustrates one possible layout for a graph with three subgraphs H_1, H_2 and H_3. For layouts including up to three subgraphs the placement of connections sets does not pose any problems. Although this approach does not seem to scale well with a higher number of subgraphs, it is unlikely that all 2^n connecting sets really show up in the final layout. For example in thin graphs, many connection sets will be empty; in Figure 2(b) we show such a layout for four subgraphs where *not all* connection sets exist. We use this layout later in the drawing of a biological network in Section 5.2.



(a) Three subgraphs. (b) Four subgraphs.

Figure 2: Placement of subgraphs and connection sets.

For dense graphs, we can concentrate on non-exclusive connection sets connecting one, two or all n subgraphs. This reduces the total number of possible connection sets to $\frac{n^2+n}{2} + 2$ and can then be drawn in a similar fashion to Figure 2(a). For many graphs these non-exclusive connection sets are sufficient.

5 Case Study: Network Motifs

This section applies the general approach presented in Section 2 and the connection sets of Section 3 to a larger graph and a high number of subgraphs.

5.1 The Challenge: A Biological Network

We will demonstrate our algorithm on a biological network representing the transcriptional regulation of *Escherichia coli.* The graph consists of 423 labeled vertices and 578 edges. It is shown in Figure 3(a), drawn using a standard spring embedder algorithm [3]. Basically, the vertices represent elements that interact with each other. These interactions are modeled by edges that carry information on the type of interaction. For a detailed description on this graph and its interpretation we refer to [9].

As subgraphs we will use *motifs*—introduced in [11]. A motif describes small subgraphs that need not necessarily be isomorphic to each other but all match a specific pattern. In biological research, the frequent recurrence of such motifs gives insight into the order of some events and helps to get an overall view of the system, collapsing information that may be spread throughout the whole network into a few nodes.

We will focus on the large connected component of the biological network and leave small components aside. Additionally, we will concentrate on the *single input module* motif (SIM) adopted from [9]. A SIM is a tree of height one where all leaves have exactly one directed in-edge each emerging from one and the same parent node. All nodes may have additional out-edges and self loops, the parent node also may have in-edges as well. The idea behind this motif is that several *elements* are controlled by one single element called a *transcription factor*. A search for this pattern in our component of the biological network identified 25 appearances of SIMs. They can be recognized in Figure 3(a) by their fan-outs.

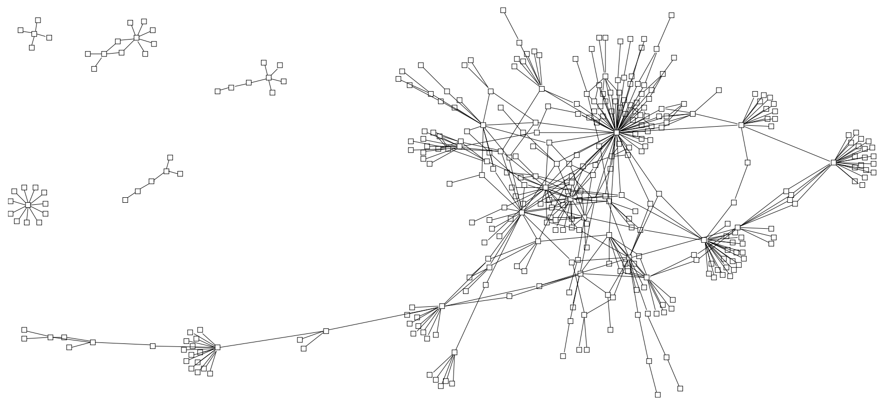
5.2 Layout of the Biological Network

We use the 25 identified SIMS as subgraphs for our summary graph. Since the biological network is quite large, we replace the individual subgraphs with a motif-specific geometric shape. This allows us to distinguish between original vertices and vertices representing motifs as well as between several motifs. For the SIM motif, a triangle has been chosen, representing the transcription factor on top that controls several elements below. Figure 3(b) shows a version of the summary graph laid out using the same spring embedder as before.

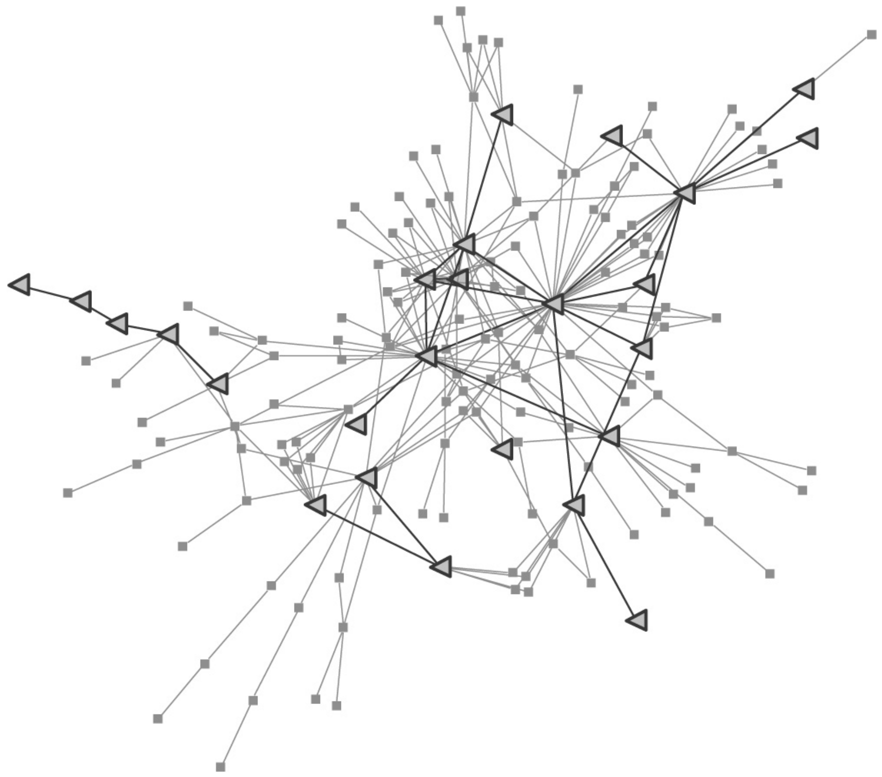
Applying the connection set method of Section 3 turns out not to be feasible. The number of possible connection sets can be up to 2^{25}. As mentioned before, the actual number of non-empty connection sets is much smaller: In our case there are 36 non-empty exclusive connections sets. Eleven of these have been connected to exactly one SIM, another eleven to exactly two SIMs and 13 to more than two SIMs. There are no vertices connected to all SIMs. It is still hardly possible to provide a neat layout for those 25 subgraphs and their connection sets. Therefore, we decided to pursue another direction to cope with such a large graph.

Instead of taking each single motif as a subgraph, we combine SIMs. As Figure 3(b) indicates, the SIMs induce a subgraph. This subgraph consists of four connected components. We will visualize the connections between these four components. The justification is that an analyst interested in the occurrence of motifs will very probably be interested in the connections between those subgraphs as well.

Figure 4 on page 15 shows one possible layout resulting from that scenario. This layout also points out one of the strengths of our approach: All those



(a) Network drawn with a spring embedder [4].



(b) Network drawn with a spring embedder [4] and highlighted SIMs.

Figure 3: Different layouts of a biological network.

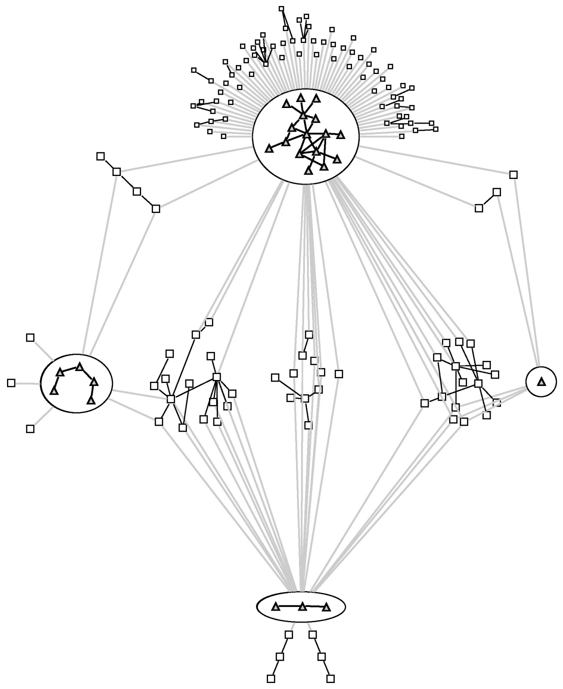


Figure 4: Layout of the biological network using connection sets.

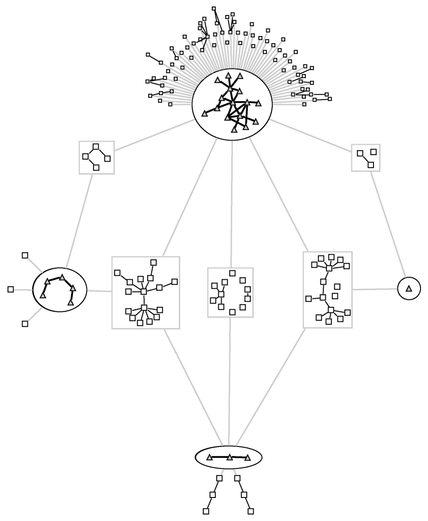


Figure 5: Optimized layout of the biological network using connection sets.

parts of the graph can be laid out using algorithms chosen individually for each particular instance. In fact, we applied several different methods to create the final layout which we present in Figure 4:

- Components up to four vertices have been placed equidistantly on a line.
- A *spring embedder* has been applied to the subgraph on top and the central connection sets.

  We implemented a spring embedder that does not change the position of nodes with a particular property. Thus, the summary nodes can be included in the layout process without getting moved. This untangles the connection sets and helps to ensure that nodes from the connection set that have edges to a summary node are place closer to (i.e., in sight with) this summary node, thus avoiding edges from crossing the connection set. We again used the trick to place one or more temporary nodes at a central point (connected to all nodes in the connection set) to prevent the graph from being stretched too much towards the summary nodes.
- Finally, for the connection set topping the large component, we chose a *pincushion* algorithm that works particularly well for thin graphs where most of the vertices are connected to one central vertex. In this case, there are 73 loosely connected vertices all sharing an edge with the summary node. For dense graphs we suggest using more sophisticated methods for ordering nodes or use splines and bends instead of straight lines.

  The recursive pincushion algorithm takes a central node (initially the summary node) and neighbor nodes (initially the complete connection set) as input. It places all neighbor nodes equidistantly on a segment (initially a semi-circle) around the central node.

  To reduce the number of edge crossings we use a heuristic that puts nodes that are connected in the same or later layers close together: From each node n we find all nodes that are in the same connected component and layer as n and place them next to n. Of course the central node is seen as a separator since otherwise all nodes would be connected to each other.

  Additionally, we place the nodes that are on the same layer alternately with three slightly different distances. This enables us to place more nodes side to side without overlapping.

  The algorithm is called recursively for each placed node n as central node and the unplaced nodes in the component of n as neighbors. The radius and the size of the segment decreases in each recursion step. Processed nodes are marked so that no node is used twice.

5.3 Optimization for Large Connecting Sets

For large graphs with many edges between connection sets and summary nodes, the notion of a summary graph can be extended: The connection sets are considered components and put into (or replaced by) a summary node themselves.

Edges incident to a vertex within a connection set are redirected to the corresponding summary node. Although this discards a fair amount of information, it considerably reduces the number of edges in the final layout and therefore increases readability.

Additionally, as the connection sets have lost many edges their structure is much more apparent and can be laid out more concisely. We used a standard spring embedder for these layouts. The optimized layout of our biological network is shown in Figure 5.

6 Limitations

There are situations where our approach produces undesirable results, because considering subgraphs for the layout of entire graphs is not always reasonable. As an example take a subway network as the graph, and a path between two places as the subgraph. Drawing the subgraph first, and the subway network afterwards is not a good idea. First, the vertices of the connection will probably be laid out on a single line or in a circle. And second, the viewer will lose orientation, because the layout does not match his or her mental map, which is influenced by the geographical distribution of the vertices. The mental map could be restored by enriching the subgraph with additional information about its environment or using graph animation.

7 Conclusion

We presented an approach for emphasizing subgraphs within graphs. Additionally, we described two applications for our approach: One for visualizing connections between subgraphs, and one for highlighting many similar subgraphs. In a final step we combined both techniques. We implemented the ideas presented in this paper and integrated them within a graph editor called Gravisto [1, 6]. The Gravisto platform has been developed at the University of Passau, Department of Computer Science.

A particular strength of our method is that the choice of algorithms used to find good layouts for each subgraph and connection set is entirely free for each application. Therefore we suggest to use frameworks like QUOGGLES [7] where all algorithms that produce the final layout are shown in the order of their execution. QUOGGLES allows to easily replace one layout algorithm with another without having to change any code at all.

Acknowledgments

We thank Prof. Dr. Franz J. Brandenburg, University of Passau, for the organization of the Graph Drawing Contest 2003 and for his constructive comments earlier versions of this paper. We thank the referees for their valuable comments.

References

[1] C. Bachmaier, F.-J. Brandenburg, M. Forster, P. Holleis, and M. Raitner. Gravisto: Graph visualization toolkit. In Pach [10], pages 502–503.

[2] F. J. Brandenburg. Graph clustering 1: Cycles of cliques. In *Proceedings of the Graph Drawing 1997*, volume 1353 of *Lecture Notes in Computer Science*, Berlin, Germany, 1997. Springer.

[3] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, Englewood Cliffs, N.J., 1999.

[4] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software–Practice and Experience*, 21(11):1129–1164, 1991.

[5] Graph Drawing Contest 2003. `http://www.infosun.fmi.uni-passau.de/br/lehrstuhl/GraphDrawing/GD2003/contest.html`.

[6] Gravisto Project Homepage. `http://www.gravisto.org/`.

[7] P. Holleis and F.-J. Brandenburg. QUOGGLES: Query on graphs - a graphical largely extensible system. In Pach [10], pages 465–470.

[8] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[9] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs in the transcriptional regulation network of Escherichia coli. *Nature Genetics*, 31:64–68, 2002.

[10] 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*. Springer, 2004.

[11] S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298:824–827, 2002.

[12] yEd Java Graph Editor. `http://www.yworks.com/en/products_yed_about.htm`.