Focused Animation of Dynamic Compound Graphs

Florian Reitz, Mathias Pohl and Stephan Diehl
University of Trier
reitzf@uni-trier.de, pohlm@uni-trier.de, diehl@uni-trier.de

Abstract

Many applications feature large hierarchic dynamic graphs that change over time. Often, these changes are more important than the graphs themselves. In our approach, areas of interests in dynamic graphs are detected based on user preferences. The user is guided from one area of interest to another in such a way that reduced contextual information is shown. To this end, dynamic graph layout mechanisms are extended by a preprocessing that decides what to show and a post processing that stages the animation based on spatial information.

1. Introduction

Compound graphs extend ordinary graphs by hierarchical information. Imagine a large company with several thousand employees. A company like this usually has a hierarchical organization. It is divided into departments and sub departments. This structure can be described by a *compound graph*. A *compound graph* is defined as $G = (V, E_I, E_A)$ where V denotes the *set of nodes*, $E_I \subseteq V \times V$ the *set of inclusion edges* and $E_A \subset V \times V$ the *set of adjacency edges*. The set of nodes V and the inclusion edges E_I form a tree. This tree is called the *hierarchy tree* of G. A compound graph is usually visualized as a node-link-diagram using the nested box metaphor (see Figure 1). In our company example the adjacency edges can model a communication network. For instance we can connect two nodes if the corresponding persons or departments exchanged e-mails with each other.

The structure of an organization typically changes over time. There are small changes like an employee moving to another department and large reorganizations that alter the company on the whole. Each time we look at the structure we can create a new compound graph from this snapshot. The chronological ordered sequence of these compound graphs $G = G_1, \ldots, G_n$ is called a *dynamic compound graph*. It describes the structural changes over time.

Analyzing a dynamic graph is difficult because it can contain a large and constantly changing structure. In some cases the differences between the graphs are more important than the graphs themselves. For instance, a study that aims at communication changes after a large restructuring may not have to consider substructures that remained unchanged.

In this paper we present a novel approach that helps the user to focus on the interesting parts of a dynamic graph during a graph animation. In Section 3 we describe the underlying layer-based visualization of large dynamic graphs. Then we discuss extensions to this algorithm that improve the visibility of changes. In Section 4 we describe a preprocessor that reduces the graph size based on areas of interest. Finaly we will see how the user's attention can be drawn to these areas in a way that all changes can be recognized.

2. Visualization of dynamic graphs

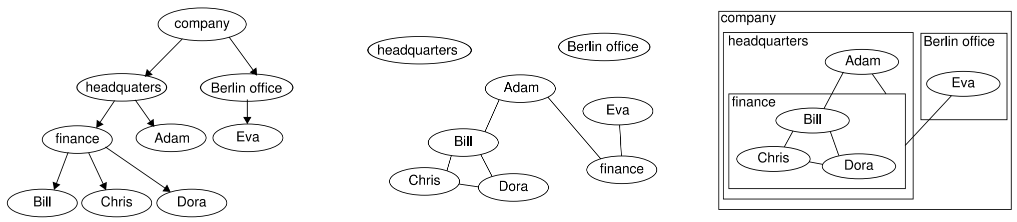
Dynamic graph visualization systems either work *online* or *offline*. When an online system draws a graph, its layout algorithm does at most know the layout of the previously drawn graph. In contrast, an offline system knows the whole sequence of graphs and can compute the layouts of all graphs in the sequence simultaneously.

Dynamic graphs that have no inclusion edges can be visualized using offline-dynamic graph layout algorithms such as *Foresighted Graphlayout* [1, 3]. Most algorithms try to *preserve the mental map* in order to make the animation readable. The term *mental map* refers to the abstract structural information a user forms by looking at the layout of a graph [4]. Changing layout between two graphs in a sequence causes the user to re-read the visualization completely in order to identify structural changes.

The use of compound graphs has become more important in recent years. For example, in social network analysis large networks are transformed in a so-called *block model* in order to cope with the huge amount of data. From a graph-theoretical point of view these block models are compound graphs.

The visualization of dynamic compound graphs is very challenging. Frishman and Tal presented an online-dynamic

978-0-7695-3733-7/09 $25.00 © 2009 IEEE
DOI 10.1109/IV.2009.24



(a) A companie's hierarchy tree (b) The communication network (c) Nested box visualization

Figure 1. A small example of a compound graph. The inclusion edges form a tree consisting of all nodes (a) while adjacency edges connect arbitrary nodes (b). The compound graph can then be visualized in a nested-box diagram (c).

layout algorithm for clustered graphs [2]. A clustered graph is a compound graph with a hierarchy of depth 1. In contrast, Pohl and Birke showed how to deal with dynamic compound graphs in order to explore their structure with their tool XLDN [5]. Their work is based on *Foresighted Graphlayout* and does not constrain the structural changes in the graph sequence. However, XLDN neither provides different layout styles nor automatic attention management.

3. Dynamic layer-based layout

By combining layer based approach of Sugiyama and the supertree concept of XLDN our algorithm is capable of arbitrary structural changes throughout a compound graph sequence. Different node and adjacency edge sets are possible as well as different hierarchy trees. To describe our algorithm we first have to explain some essential elements of Sugiyama's traditional layout algorithm.

3.1. Layer-based layout of static graphs

The static layer-based approach works in four steps:
Step 1: Nodes are assigned upon parallel layers such that most adjacency edges point into the same direction. For compound graphs the layer assignment is encoded in number sequences rather than numbers. Therefore the layer-assignment function clevel for a graph $G = (V, E_I, E_A)$ maps all nodes to a sequence of natural numbers:

$$\mathrm{clevel} : V \to \mathbb{N} \cup \mathbb{N}^2 \cup \mathbb{N}^3 \cup \ldots$$

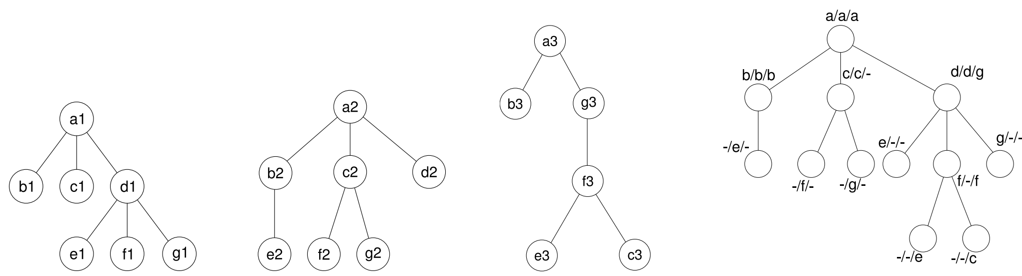
Such a sequence $\mathrm{clevel}(v)$ is called *compound level* of v. On these compound levels a *lexicographic ordering* is defined – i.e., $(1) < (1,1) < (1,2) < (2)$.

According to Sugiyama, a compound level assignment is considered valid if it satisfies the *inclusion condition* and the *down-arrow condition* [6]. The inclusion condition guarantees that compound-levels can later be mapped to the levels in nested boxes. The down-arrow condition guarantees that all edges to point from nodes with lower levels to nodes with higher levels.
Step 2: Edges are made *proper* which means that all edges should connect only nodes between subsequent layers. To transform non-proper edges they are split into a path of temporary dummy nodes.
Step 3: All nodes are ordered on their assigned layers to reduce edge crossings. This can be done using the barycenter heuristic proposed by Sugiyama and Misue.
Step 4: In the last step a final visualization of the graph is computed according to the compound levels and the respective orderings of all nodes. Dummy nodes are removed from the graph after this computation. Their positions serve as bends for the respective edges.

3.2. The supertree

To create layouts for dynamic compound graphs XLDN (like Foresighted Graphlayout) first computes a global layout for a union of all graphs in a sequence. This layout then serves as a template for the separate graphs and therefore a priori ensures a good preservation of the mental map. Afterwards some additional optimizations can be applied to the separate layouts as long as they still preserve the user's mental map. However, merging of compound graphs does not necessarily yield another compound graph. Instead, the set of inclusion edges may degenerate and no longer have a tree structure. To cope with this major problem the *supertree* concept was introduced. The idea behind it is to find a tree that contains all information of its base trees. To realize such a supertree a heuristic with quadratic runtime is used by XLDN.

The formal definition of the supertree is as follows: For a sequence $G_1, \ldots, G_n$ with $G_i = (V_i, E_{I_i}, E_{A_i})$ the tree $\widetilde{T} = (\widetilde{V}, \widetilde{E})$ is a tree if the following conditions hold: $\forall i \in \{1, \ldots, n\} : \exists \sigma_i : V_i \to \widetilde{V}$ such that σ_i is injective, $\forall v \in V_i : \sigma_i(v) \in \widetilde{V}$ and $\forall (v, w) \in E_{I_i} : (\sigma_i(v), \sigma_i(w)) \in \widetilde{E}$.



(a) Tree 1 (b) Tree 2 (c) Tree 3 (d) Supertree

Figure 2. A sequence of three trees and a possible supertree. The labels of the supertree nodes indicate the represented objects in the respective trees e.g. $d/d/g$ means that the supertree node represents d in tree 1, d in tree 2 and c in tree 3.

Such a tree always exists. In a trivial construction the roots of all trees are siblings beneath a new root node. However, this supertree would be a poor one since it would contain too many nodes. As shown later on, the number of nodes in a supertree should be minimal to reduce unused space in the visualization. An example for an efficient supertree is depicted in Figure 2.

The computation of a minimal supertree, i.e. a tree containing a minimal number of nodes is $\mathcal{NP}$-complete [5]. Therefore a heuristic for a reasonably small supertree was developed. This algorithm starts with the initial tree and tries to combine equal nodes of subsequent trees in a greedy way.

3.3 Dynamic graph layout

After the supertree of a compound graph sequence is computed it can be used for the computation of the global layer-based layout template. In a first step the adjacency edges of all graphs are added to the supertree to obtain a global compound graph for the sequence. This global compound graph is called *supergraph* and is defined as

$$\widetilde{G} = \left(\widetilde{V}, \widetilde{E_I}, \widetilde{E_A}\right)$$

where $\left(\widetilde{V}, \widetilde{E_I}\right)$ is the supertree and

$$\forall i \in \{1, \ldots, n\} : (v, w) \in E_{I_i} \Rightarrow (\sigma_i(v), \sigma_i(w)) \in \widetilde{E_A}.$$

With the same methods as in the static case a compound level assignment can be computed for the supertree. Afterwards all edges are made proper and the nodes are ordered to reduce the number of resulting edge crossings. The layout of the supergraph then is *induced* to the separate graphs in the sequence.

All nodes of the graphs are assigned to the compound levels of their representative in the supergraph. Furthermore they are ordered according to the global ordering in the supergraph. Since nodes in different graphs can vary their position inside the graph's hierarchy tree, these assignments are not necessarily the same for all graphs.

As a result the dynamic compound graph can be visualized as a sequence of static compound graphs whose layouts preserve the user's mental map. Due to the global layout template layout changes are only visible where structural changes occur. The sequence is finally shown in an animation to make it easier for the user to follow the changes.

4. Areas of interest

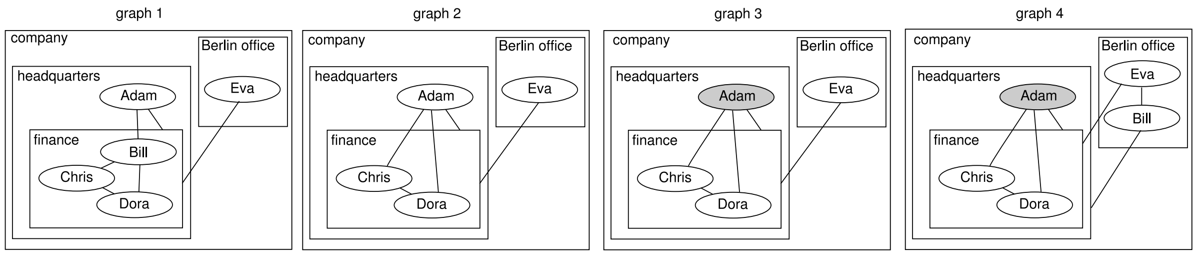
The layer based layout algorithm stabilizes the graph sequence so unchanged parts of the graph move as little as possible. This emphasizes changes to a certain extent but some graphs are still too large and the changes too scattered to see them. In this section we present a preprocessor that identifies areas of interest and turns the sequence of compound graphs into a sequence of focused graphs.

4.1. Importance Measure

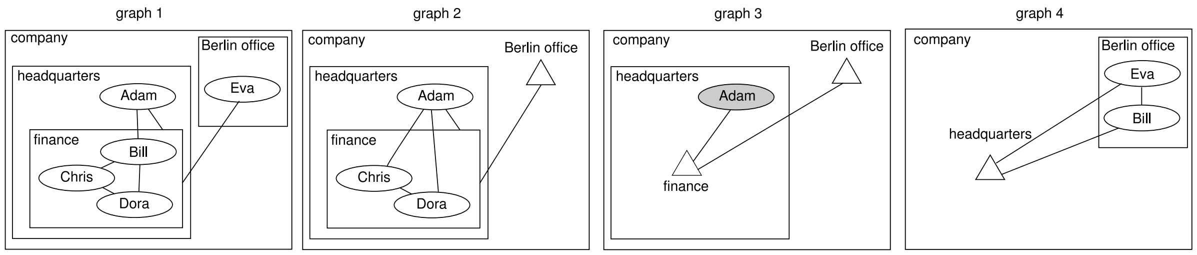
In real applications not only the structure of the graph changes but also the properties of the entities represented by the nodes. In this work we cover two types of changes: *structural* and *attribute related.*

Structural change: A structural change is an operation that transforms one compound graph into another. Structural changes consist of removing or adding nodes, adjacency edges and inclusion edges. For instance, a structural change might be that an employee moves from one department to another. Note, that structural changes can be identified without domain knowledge.

Attribute related change: For example an employee can have the attribute *on holiday* or *married* in some parts



(a) An example sequence without folding.



(b) The sequence after folding was applied. The first graph is completely unfolded.

Figure 3. A graph sequence with folding (b) and without (a). At first Bill leaves the finance department (graph 1). Without him, Adam has to communicate directly with Chris and Dora (graph 2). In graph 3 Adam gains the tag *promoted*. Finally Bill joins the Berlin department.

of the dynamic graph. For every dynamic graph a tag set T and a partial function $\tau : V \times \{G_1, \ldots, G_n\} \rightarrow \mathcal{P}(T)$ is defined that assigns these tags to nodes in certain graphs. Thus an attribute related change is adding or removing a tag from a node. Usually, the user is not equally interested in all types of changes. For each change type c an importance weight w_c is defined. The higher the value the more important the change. The number of change types is small so a manual definition of w_c is doable. Let $C = c_1, \ldots, c_m$ be the set of changes that affect node n in graph g. The local relevance function ϑ is defined as

$$\vartheta : V \times \{G_1, \ldots, G_n\} \mapsto \mathbb{N} \quad \vartheta(n, g) = \sum_{c_i \in C} w_{c_i}$$

There is no separate importance function for adjacency edges. All changes that apply to them become part of the change sets of their source and target node.

4.2. Focusing by Folding

Folding a node n in a compound graph produces a new compound graph in which all descendants of n are removed. Edges between the descendants of n are removed. Edges that connect descendants of n to other nodes are replaced by edges between n and these other nodes. Folding can be used to remove uninteresting parts of the graph. The folding of nodes should not rely on its local importance value ϑ only. In that case descendants with higher ϑ value would also be removed. To avoid this ϑ is propagated upwards in the inclusion tree. The *subtree relevance* $\gamma : V \times \{G_1, \ldots, G_n\} \rightarrow \mathbb{N}$ has to satisfy the following condition:

$$\forall (v, w) \in E_I \quad \forall g \in \{G_1, \ldots, G_n\} : \gamma(v, g) \geq \gamma(w, g)$$

If n is leaf in graph g $\gamma(n, g)$ is defined as $\vartheta(n, g)$ and $n \notin g \Rightarrow \gamma(n, g) = 0$.

For inner nodes γ can be defined in different ways:

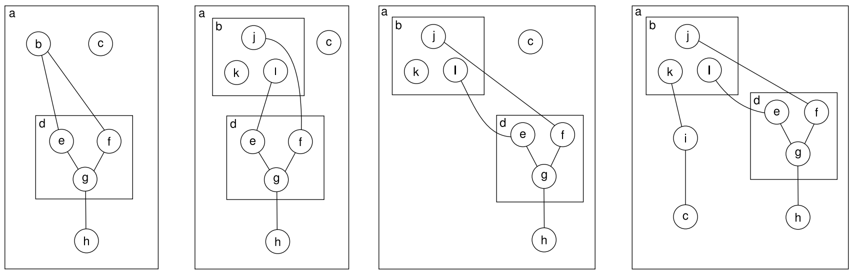
(a) $\gamma_{sum}(n, g) = \sum_{i=1}^{m} \gamma(c_i, g) + \vartheta(n, g)$

(b) $\gamma_{max}(n, g) = \max(\gamma(c_1, g), ..., \gamma(c_m, g), \vartheta(n, g))$

(c) $\gamma_\alpha(n, g) = \alpha \cdot +\gamma_{sum}(n, g) + (1 - \alpha) \cdot \gamma_{max}(n, g)$ for $\alpha \in [0, 1]$

where $c_1, \ldots, c_m$ are the children of n. Option (a) favors nodes with many children of lesser importance while (b) rewards single nodes with high relevance values. Option (c) is a compromise. Note, that option (c) is equivalent to option (a) if $\alpha = 1$ and equivalent to option (c) if $\alpha = 0$. Obviously, all options meet the folding condition. Figure 3 shows how folding works on the company example.

Assume a sequence of three graphs where node n is important in the first graph, unimportant in the second, and important again in the third graph. Focusing based only on γ would fold n in the second graph and unfold it again in



(a) Graph n (b) Step 1 (c) Step 2 (d) Graph $n+1$

Figure 4. A serialized animation that blends the drawing of graph n (a) into the drawing of graph $n+1$ (d). At first node b is inflated (a $\rightarrow$ b) then d and h move sideward (b $\rightarrow$ c). In the final step node c moves and h appears.

the third graph. The situation worsens if the importance of n changes periodically. The resulting flickering effect can reduce the quality of the animation. In such cases a balancing mechanism is applied that prevents the subtree relevance of a node from rising and falling too fast.

Let g_m be a compound graph and node $n \in g_m$. The *relaxed relevance* δ with lookbehind s and lookahead t is defined as

$$\delta(n, g_m) = \max_{-s \leq i \leq t} \{\gamma(n, g_{m+i}) \cdot f(|i|)\}$$

where $f(i)$ is a monotonically decreasing function with values from $[0, 1]$ and $f(0) = 1$. Function f denotes how this influence is reduced by distance. Thus, δ takes the s proceeding and t succeeding graphs into account.

Foldings are massive modifications of the structure of the graph and likely to destroy the mental map. For this reason in our approach folding is done prior to the layout algorithm. In this case the modifications can be accounted for in the generation of the super tree.

5. Attention by animation

A folding can reduce the size of a graph but the result can still be too large. In this section we present a way to make the user aware of the changes during the animation that blends one graph into another.

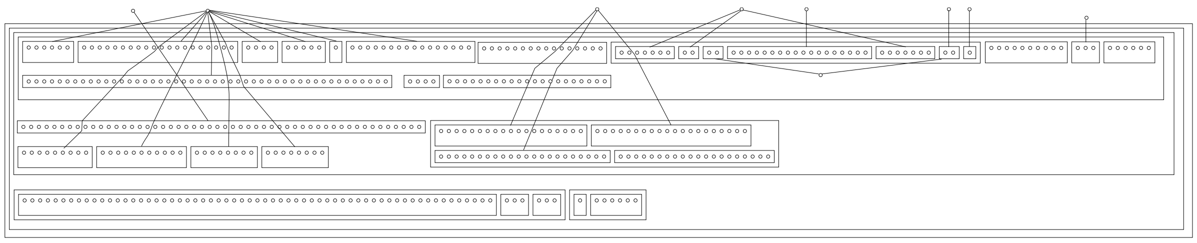
The human attention is not uniformly distributed over the user's field of sight. Objects that are located at the field perimeter are less likely to be recognized than those in the center (inattentional blindness). The same effect applies to slow changes for instance when a node fades in (change blindness). However, movement can be detected throughout the field of sight as long as it is not scattered all over the screen. (For more information on visual perception see [7]). In the dynamic layout, see Section 3.3, some of the changes in the underlying graph result in the movement of nodes. Since for attribute related changes there is no movement artificial movement is created.

Movement is easy to recognize as long as it is limited to a small area and the number of moving objects is small. The set on nodes that are going to move during the next animation is partitioned in *animation groups*. The animation is serialized in a way that the animation groups are moved one after another. More formal: let $L_i(v)$ be the exact position of node v in graph g_i and $dist_i(v, w)$ the euclidean distance of nodes v and w in g_i. If $v \notin g_i \vee w \notin g_i \Rightarrow dist_i(v, w) = 0$. For each animation group g we demand $max_{v,w \in g}(dist_i(v, w)) < t$ and $max_{v,w \in g}(dist_{i+1}(v, w)) < t$, where t is a predefined threshold. This conditions ensure that the group's nodes are close to each other during the whole animation. The size of each group should be limited so the user can follow the changes.

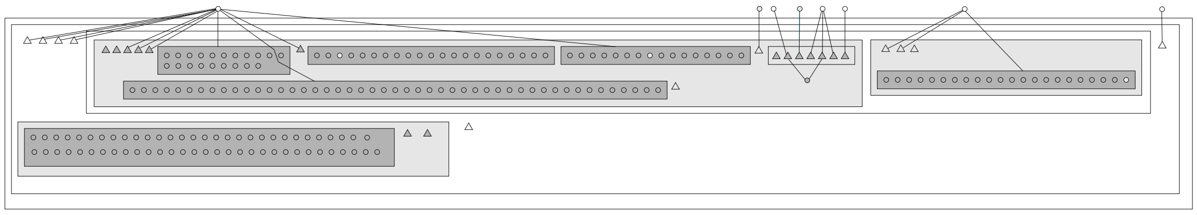
The partition is not arbitrary: if node v moves all its descendants must move as well. Furthermore, a node can not move to a place that is still occupied by another node from a subsequent animation group. Note, that in some cases it is not possible to compute partitions. Figure 4 shows the use of animation groups in a small example.

6. Case study

We implemented our approach as a Java application which reads graph and tag data from an XML file that can be easily generated. Figure 5 shows two images produced with this tool. The images show refactorings in the *jftp* software archive. The inclusion edges of the visualized compound graph represent the structure of the source code and the adjacency edges represent the *implements* and *extends* relations. As tags we use the names of refactorings that have been applied to the part of the source code represented by



(a) The unfolded compoud graph of *jftp*.



(b) The compound graph from (a) with folded nodes.

Figure 5. The compound graph representation in one stage of the *jftp* software archive.

the node. Every structural and attribute related change has the same importance weight. For focusing we used δ with γ_{sum}, lookbehind $s = 1$, lookahead $t = 1$ and $f(i) = \frac{1}{2^i}$.

The pictures show a massive restructuration at an early stage of the development. Three new packages were added (light gray) and some old code was moved there (dark gray). Triangles represent folded nodes. Note, that some folded nodes are colored too because the changes affected their sub tree but γ_{sum} did not exceed the folding threshold. The nodes above the outer rectangle represent external classes or interfaces like *LinkedList*. The class *Object* (the top of the Java hierarchy structure) is removed to make the image more comprehendible. Folded nodes are painted as triangles.

7. Conclusion

Starting from the hypothesis that the user is more interested in what changes that in what stays the same we developed a visualization technique which guides the user through a dynamic compound graph. As tools become available that guide users through a graph the question arises whether these tools are more efficient than those that leave the decision up to the user. We expect that future evaluation will show that this is the case although no automatic tool can fully compete with the user's intention.

Acknowledgements

We'd like to acknowledge Peter Weißgerber who supplied us with refactoring data for the case study.

References

[1] S. Diehl and C. Görg. Graphs, They Are Changing. In *Proc. of 10th Int. Symp. on Graphdrawing, GD*, volume 2528 of *LNCS*, pages 23–30. Springer, 2002.

[2] Yaniv Frishman and Ayellet Tal. Online dynamic graph drawing. In *EuroVis*, pages 75–82. Eurographics Association, 2007.

[3] C. Görg, M. Pohl, P. Birke, and S. Diehl. Dynamic Graph Drawing of Sequences of Orthogonal and Hierarchical Graphs. In *Proc. of 12th Int. Symp. on Graphdrawing, GD*, volume 3383 of *LNCS*, pages 228–238. Springer, 2004.

[4] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout Adjustment and the Mental Map. *J. Vis. Lang. Comput.*, 6(2):183–210, 1995.

[5] M. Pohl and P. Birke. Interactive exploration of large dynamic networks. In *Proc. of 10th Int. Conf. on Visual Information Systems, VISUAL*, LNCS 5188, pages 56–67, 2008.

[6] K. Sugiyama and K. Misue. Visualization of Structural Information: Automatic Drawing of Compound Digraphs. *IEEE Trans. on Systems, Man and Cybernetics*, 21(4):876–892, 1991.

[7] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, San Francisco, 2 edition, 2004.