Erschienen in: IEEE Transactions on Visualization and Computer Graphics. 2008, 14(6), pp. 1293-1300
https://doi.org/10.1109/TVCG.2008.130

Exploration of Networks Using Overview+Detail with Constraint-based Cooperative Layout

Tim Dwyer, Kim Marriott, Falk Schreiber, Peter J. Stuckey, Michael Woodward and Michael Wybrow

Abstract— A standard approach to large network visualization is to provide an overview of the network and a detailed view of a small component of the graph centred around a focal node. The user explores the network by changing the focal node in the detailed view or by changing the level of detail of a node or cluster. For scalability, fast force-based layout algorithms are used for the overview and the detailed view. However, using the same layout algorithm in both views is problematic since layout for the detailed view has different requirements to that in the overview. Here we present a model in which constrained graph layout algorithms are used for layout in the detailed view. This means the detailed view has high-quality layout including sophisticated edge routing and is customisable by the user who can add placement constraints on the layout. Scalability is still ensured since the slower layout techniques are only applied to the small subgraph shown in the detailed view. The main technical innovations are techniques to ensure that the overview and detailed view remain synchronized, and modifying constrained graph layout algorithms to support smooth, stable layout. The key innovation supporting stability are new dynamic graph layout algorithms that preserve the topology or structure of the network when the user changes the focus node or the level of detail by in situ semantic zooming. We have built a prototype tool and demonstrate its use in two application domains, UML class diagrams and biological networks.

1 Introduction

Scalability is a key issue for network visualisation paradigms. Examples of very large network datasets abound: biological pathways mapping out complex processes in living organisms; large social networks available through on-line resources like Facebook; or even the World-Wide Web itself. While it may be useful to look at small subnetworks in isolation, increasingly data is available to provide context in the form of a much larger neighbourhood. We believe an interactive system for exploring such data should provide flexible tools for interactively filtering and aggregating large networks to obtain a more manageable sub-network most relevant to a particular task; see e.g. [30]. However, both during and after the query and filtering process the most common way to visualise network connectivity is as a classic node-link diagram. For this representation to convey information as effectively as possible, good layout is crucial.

There are a number of techniques for creating high-quality layout for small networks (fewer than 100 nodes). However, these algorithms do not scale up to larger networks (see §2.2). For this reason, much recent work on network layout has focused on applying the popular force-directed layout approach to very large graphs with hundreds or thousands of nodes in reasonable time. While these techniques are fast and are designed to reveal the overall structure of the network, they do not cater for the layout aesthetics that are important in a detailed view such as non-overlapping nodes and high-quality edge routing.

In this paper we couple these two kinds of layout to provide an overview to show context—using a high-speed layout method—and a detailed view using a slower, higher-quality layout algorithm. As well as providing scalability, the use of two different layout engines reflects that these two views have quite different aesthetic requirements. Our aims—largely based on previous research on interactive visualization of networks—when developing our model and the associated visualization tool were: *Scalability*—support for interactive visualization of very large graphs with thousands of nodes; *Synchronization*—the overview and detailed layouts must remain synchronized; *High quality layout*—layout in the detailed view should satisfy widely accepted graph drawing aesthetics; *Customizable*—the user can impose application specific layout conventions and also so that they can create landmarks for better navigation; *Levels of Detail (LOD)*—where available in the network or in the textual or graphical content of nodes, LOD should be navigable through in-situ semantic zooming; *Stability*—the basic structure of the layout should remain stable as the detailed view moves about the network and also during changes of LOD; *Continuity*—there should be a smooth, continuous transition to new layouts during interaction.

Because of the desire to support high-quality, customizable layout in the detailed view, we decided to use a constrained graph layout method in combination with automatic connector routing. It supports non-overlapping node labels, nested sub-graphs (clusters), and high quality poly-line connector routing. Importantly for our application, it also allows the user to specify additional constraints such as downward pointing edges, distribution alignment and minimum separation constraints between nodes in the network. These allow the user to customize the presentation to their particular application specific style.

Our previous constraint-based network layout was based on a technique called *constrained stress majorization* [9]. However, after working with constrained stress majorization in an interactive context we found it had a number of limitations that were not easily overcome. Thus, in this paper we present a new optimisation method based on *gradient projection* and *Runge-Kutta integration*, which is more easily extensible to different goal functions, and a new path-based force model called *P-stress*. This new force model removes long range attractive forces between loosely connected parts of the network and adds forces which consider all the line segments in an *edge routing* and act to minimise the total path length of that routing which leads to better layout with poly-line connectors.

One of the major benefits of our new approach is that it preserves the topology of the network during layout. This means that while nodes and connectors can move if this leads to better layout, they cannot move through each other to change the inherent topological structure of the layout. This helps to keep the layout stable during online ex-

- *Tim Dwyer is a visiting researcher at Microsoft Research, E-mail: t-tdwyer@microsoft.com*
- *Kim Marriott and Michael Wybrow are with Monash University, Australia, E-mail: {Kim.Marriott,Michael.Wybrow}@infotech.monash.edu.au*
- *Falk Schreiber is with IPK-Gatersleben, Germany, E-mail: schreibe@ipk-gatersleben.de*
- *Peter J. Stuckey is with National ICT Australia and the University of Melbourne, Australia, E-mail: pjs@csse.unimelb.edu.au*
- *Michael Woodward is with the University of Melbourne, Australia, E-mail: wmg@csse.unimelb.edu.au*

ploration when the user changes the focus node or the level of detail (LOD) by in situ semantic zooming. A byproduct of using such topology preserving layout is that it allows natural handling of clusters since topology preservation means that cluster boundaries remain inviolate.

In order to handle very large networks in the overview we use a fast multi-pole method for layout which is the state-of-the-art for simulating large N-body problems; of which the incremental force-directed network layout problem is effectively an instance. It is not possible for this layout method to support the full gamut of constraints provided in the detailed view without making it impractically slow. Instead, we show that simple *fixed-position* constraints are sufficient to require the contextual view to reflect the layout modifications applied at the detailed level. As the user—or *users* in a collaborative environment—explore the larger network their local layout refinements (either automatic or through directed constraint inferencing) are gradually reflected in the holistic layout of the overview.

2 Background

2.1 Interactive exploration of large networks

An early formal description of the *on-line graph drawing* problem is given by Eades *et al.* [12]. They describe a system in which users navigate through a large graph by selecting a focal node. This focal node is centred on the display and a subgraph of a predefined graph-theoretic radius is expanded around that node. They used a basic Fruchterman-Reingold [18] force-directed approach for layout where the iterations in the layout process are used as key-frames for animation as the subgraph is updated following a change of focal node. This general mode of on-line graph navigation has been revisited on numerous occasions. It has, for example been extended to hierarchical navigation of clustered graphs [13]; used in a 3D visualisation of WWW browsing [5]; updated with a more scalable Barnes-Hut force-model and applied to social network navigation [25]; and it has also appeared in a commercial tool for navigating Google and Amazon search results.[1]

There has been considerable research into the underlying interaction issues that need to be considered in a general on-line network navigation system and which have guided the development of our tool.

Overview+detailed view: Graph viewing systems such as yEd[2] and Cytoscape[3] make use of an overview window to show the entire network and also a main window with a detailed view of part of the network. Unlike our system, the detailed view is simply a zoomed in view of the overview. This means that quality of the detailed view is restricted by the need to ensure scalability of the layout method for the overview. Furthermore, the network layout in these systems is static, and is not updated in response to user interaction or to optimise the part of the graph that is visible in the detailed display. The model we explore in this paper supports such local optimisations while reflecting all such local changes in the overview display in a synchronized, coordinated way.

Continuity: The on-line graph drawing paradigm seeks to make layout locally optimal for a small focal sub-graph, but when changing layout dynamically in response to user navigation events there are many lessons to be learned from related tasks in the information visualisation design space. Animation is known to be very useful in supporting the user's mental map through transitions of display focus [3].

Stability: Plaisant *et al.* [31] investigated interactive exploration of large tree structures. While finding readable layout for trees is much easier than general graphs, their results concerning interaction remain applicable. In particular, they found in their user study that users were more easily able to find already visited nodes if the layout surrounding those nodes remained consistent from visit to visit. Thus, when navigating to a new focal node the part of the visible subgraph that remains on the screen should not be radically altered. Further, when re-displaying a previously visited node, its position relative to the structure around it should be recognisable [31].

There has been considerable interest in developing techniques for stable graph layout that preserve the user's mental model of the graph [28]. These techniques are quite specialized to the underlying layout algorithms. The standard approach for supporting stability in force-directed approaches is to simply add a "stay force" on each node so that it does not move unnecessarily, e.g. [17]. In the case of orthogonal graph layout, stability is gained by trying to preserve the current bend points and angles, e.g. [6]. This has the effect of preserving the layout topology. Finally, North and Woodhill [29] have given algorithms for preserving the current horizontal and vertical ordering between nodes in dynamic Sugiyama-style layered layout. Our approach is the first that we are aware of to base stability on topology preservation in a force-directed style layout. It has the advantage over stay forces that the layout is better able to adjust to changes while still preserving the original structure.

Customized landmarks: Ware [38] gives extensive guidelines for navigation of large "abstract data spaces". His focus is on navigation in 3D environments but the guidelines drawn from numerous studies on wayfinding also apply to extended 2D environments, for example: "...cognitive spatial maps form easily and rapidly in environments where the viewer can see everything at once... The practical application of this is that overviews should be provided wherever possible in extended spatial information spaces." (p.331). Ware also discusses the importance of providing landmarks to support wayfinding tasks.

In-situ zooming: Distortion-oriented techniques [27] are well accepted as providing in-situ zooming in information displays. So called "fisheye" techniques [19] were quickly extended to graph visualization [34]. More recent interest in this area has concentrated on *semantic zooming* of information by means of dynamic expansion of clustered graphs [20, 37]. While these methods show promise, both may introduce edge crossings as various levels of detail are exposed or elided. Thus these methods potentially violate the fisheye or "rubber-sheet" metaphor since they are not guaranteed to preserve topology. In this paper we explore a new method which guarantees preservation of topology of the surrounding network layout when individual nodes or subgraphs are expanded to show more detail.

2.2 Layout of large and small networks

We note that there is a divide between graph layout methods which work well for small and simple networks[4] and those which are able to produce layouts for networks with hundreds or thousands of nodes in reasonable time. A very popular method for layout of small and simple directed graphs is the Sugiyama layered layout method [35]. This method is able to produce layouts with mostly downward pointing directed edges and no overlap between nodes and edges. There has also been some effort to ensure that Sugiyama methods can run very fast for large graphs [15]. However, regardless of run-time, it has been shown that when applied to larger graphs this method is poor at showing the large-scale structure of the network [8]. Another family of layout methods which produce reasonably high quality layout for small and simple graphs and which have received much attention from graph drawing researchers is *orthogonal* layout [14]. Orthogonal layout is also able to produce layouts without overlap between nodes and edges and to produce layouts with minimal crossings between edges. Again, however, the layout for larger graphs often obscures overall structure, e.g. see Fig. 1.

Constrained force-directed layout has been introduced as a method for obtaining high-quality layout with non-overlapping nodes and edges and other constraints for capturing various application specific drawing conventions, with the advantages of force-directed layout such as more uniform edge lengths [9]. Unfortunately, the run-time scales poorly in the number of constraints.

On the other hand there has been much effort to develop fast layout methods for very large graphs. Hachul and Juenger give a good survey and comparison of the fastest techniques in [23]. To briefly summarize, they found that Eigen-projection methods are fastest but perform

[1] http://www.touchgraph.com/
[2] http://www.yfiles.com/producs/yed/
[3] http://www.cytoscape.org/

[4] What is meant by "small and simple" is very dependent on the application and network structure, but, for example, let us assume we mean networks with around 50 nodes and most nodes having degree less than around 10

(a) Force-directed (b) Layered (c) Orthogonal

Fig. 1. Three popular methods for "high-quality" layout of relatively small graphs applied to a "Sierpinski triangle" graph with 123 nodes and 243 edges. Produced using the y-Ed software with default settings for the three layout styles. Moderate size examples where the combinatorial layout methods do not show large scale structure are easy to find.

poorly for treelike structures, while multiscale force-directed methods give a reasonable compromise between speed and quality.

Our approach attempts to overcome the different requirements for large scale layout (showing overall graph structure and speed) and small scale layout (high quality layout where every node and edge is easily visible), by marrying the two different approaches in an overview+detail display. Various choices could be made for the two layout methods but we are guided in our choice by the requirements for interactive navigation outlined in the previous section.

3 Basic Visualization Model

Based on the classic overview+detail visualization model, the core innovation in our network exploration paradigm is that we use a slower, higher quality layout algorithm for layout of a small subset of the graph called the *primary graph* (which includes the sub-graph displayed in the detailed view) and a high-speed layout method for the remainder of the graph which is called the *secondary graph*. The user navigates through the network by repeatedly selecting a focal node. Nodes from a neighbourhood around the focal node are added to the primary graph until a threshold number of nodes is reached. The high-quality layout is updated and the focal node is centred in the detailed view. Updating of the high-quality layout is done in isolation from the overall layout and only considers the nodes in the primary graph, although the starting point for the layout is the position of the nodes in the secondary graph. After this the layout for the secondary graph is updated using the fast network layout technique. This takes account of the positions of the nodes in the primary graph layout but is not allowed to move them.

Of course the primary graph is not allowed to become too large (e.g. > 50 nodes) since this would defeat the aim of using fast layout techniques for most of the graph. We restrict its size by removing the nodes that are the furthest in graph-theoretic terms from the current focus node and were the last to be focus nodes. In order to ensure stability of layout, we cache the nodes' associated constraints and the routing for their associated edges and cluster boundaries. These are restored if the nodes come back into the primary graph.

We have elected to use a constrained graph layout method in combination with automatic poly-line connector routing for layout in the detailed view. This is a generic layout technique that provides high quality layout and also allows the viewer to tailor the layout in the detailed view by imposing placement constraints on the layout. The relationship is maintained in subsequent interaction until the author explicitly removes it. The great advantage of placement constraints over explicit positioning of nodes is that while they allow the author to control the layout, they still allow the layout to adjust sensibly to future interaction such as LOD changes.

A significant benefit of allowing constraints to be placed on the layout is that the user can use these to improve navigation through the network by for instance aligning nodes in an important metabolic pathway, or orthogonalising the layout and so creating landmarks to guide their subsequent exploration [38]. In order to facilitate this, our tool provides two high-level styling tools that generate placement constraints designed to make the layout more memorable by highlighting the largest cycles in a directed graph and by orthogonalising the layout.

Our visualization tool allows the user to change the LOD shown in an individual node, i.e. resizing the node label. They can also change the LOD in the network by choosing a cluster, i.e. a hierarchical collection of nodes, to be expanded or collapsed in the detailed view.

Example sessions with the tool are shown in Figures 4 and 5. In the next two sections we detail the layout algorithms used in our tool.

4 Detailed Display

Our system utilizes so called *constrained graph layout* algorithms for layout of the primary graph [24, 8, 9]. These are related to the more common force-directed graph layout algorithms. Like force-directed methods, they find a layout minimizing a goal function such as stress. However, unlike force directed methods, constrained graph layout algorithms allow the goal to be minimized subject to placement constraints on the allowed layouts. To do so, they use sophisticated constrained optimization techniques to ensure that the generated layouts satisfy the constraints *exactly*. However, these layout algorithms were not designed to support interactive visualization. We had to extend the algorithms to support stability of layout, user driven changes to the LOD, and continuity.

4.1 Network Diagrams

A *network diagram* (V,E,C) is a drawing of a graph with nodes V, a set of possibly directed edges $E \subseteq V \times V$, and a set of node clusters C. Each node $v \in V$ has a rectangular bounding box with width w_v and height h_v. A cluster of nodes has a *boundary* which is a sequence of distinct node corners except that the first and last element is the same. This defines a closed region called the *cluster region*.

A *route* for an edge $(u,v) \in E$ is a sequence of line segments starting from the centre of u, 0 or more corners of nodes around which the edge is routed and finishing at the centre of v.

A *layout* for a network diagram is a triple (X,R,B) giving a position $X_v \equiv (x_v, y_v)$ for each node $v \in V$, a route R_e for each edge $e \in E$, and a boundary B_c for each cluster $c \in C$.

We impose a number of automatically generated *refinement constraints* on the layout to ensure it is "tidy." There is a non-overlap constraint between each pair of basic graphic shapes. There is a membership constraint on each node cluster: node v is in cluster c iff v is in the cluster region of c. The last refinement constraint is that *paths*, i.e. edge routes and cluster boundaries, are not allowed to pass through nodes. In addition the author can impose *placement constraints* on the layout by using placement tools (see §4.2).

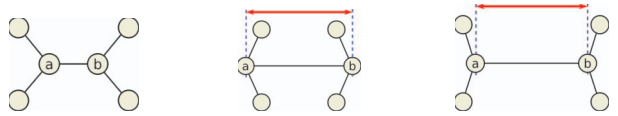
Constrained graph layout methods use a goal function to measure the quality of a layout. We use a new goal function we call *P-stress* (for path-stress). Given a layout (X,R,B), its *P-stress* is

$$\sum_{i<j} w_{ij}((d_{ij} - ||X_i - X_j||)^+)^2 + \sum_{p \in P} w_p((||p|| - d_p)^+)^2 \qquad (1)$$

where $w_p = \frac{1}{d_p^2}$, z^+ is z if $z \geq 0$ and 0 otherwise, and P is the set of paths $\{R_e | e \in E\} \cup \{B_c | c \in C\}$ in the network.

The first component of *P-stress* is a modification of the stress function which penalizes nodes that are too close together. Unlike the stress function, nodes i and j that are more than their ideal distance d_{ij} apart are not penalized, thus eliminating long range attraction which can cause problems in highly constrained problems (see Fig. 2). In this regard *P-stress* has more in common with a Fruchterman-Reingold force model [18] than the *stress* model.

The second component of *P-stress* tries to make the length of each path p in the network no more than its ideal length d_p. This will also have the effect of straightening edges and making clusters more compact and circular in shape, see Fig. **??**. The ideal length of an edge is a user defined parameter while the desired boundary length of cluster c is $2\sqrt{\pi \sum_{v \in c} w_v h_v}$ (i.e. proportional to the perimeter of the circle with area equal to the combined area of the constituent nodes). The development of the *P-stress* model was in part motivated by a previous attempt at adding edge straightening forces to the stress majorization



(a) Unconstrained *stress* (b) Constrained *stress* (c) Constrained *P-stress* or *P-stress*

Fig. 2. When a separation constraint is added between nodes a and b stress majorization gives undesirable attractive forces between not immediately connected nodes (b). By contrast *P-stress* leads to a layout that is less surprising (c).

method, see [10]. In practice, however, that method was not reliably convergent and since it involved complex management of an excessive number of dummy nodes—one for every potential bend point on an edge—it scaled poorly.

4.2 Placement tools

The user interface provides standard *placement tools* on horizontal or vertical positions of nodes: e.g. alignment and equally spaced distribution, minimum distance *separation constraints*, an *anchor* tool that allows the user to fix the current position of a selected object or set of objects. These are similar to the constraints provided in diagramming tools such as GLIDE [33] and Microsoft Visio. Placement constraints are created by selecting objects and invoking a placement tool. These constraints have a visual representation and objects can be added to or removed from them by direct interaction. Placement constraints are *persistent*, meaning that if nodes involved in a constraint leave the primary graph, the constrained positions will be preserved (i.e. the positions fixed) in the secondary graph, and the constraint will be reinstated when the nodes return to the primary graph.

The visualization tool also provides two higher-level *styling tools* which automatically generate placement constraints. These constraints behave like author specified placement constraints and so the author is free to modify the layout by removing some or all of them. Both tools work on a user selected sub-graph of the detailed view.

The first styling tool "orthogonalizes" the layout by adding vertical and horizontal alignment constraints by a greedy traversal of the selection. It is worth pointing out that orthogonal layout has been shown to improve comprehensibility of UML class diagrams [32]. An example is shown in Fig. 4.

The second styling tool, *flow*, makes the selected directed edges downward pointing with a separation constraint between the y-positions of the ends of each edge. Cycles are handled using a heuristic to identify an approximately maximal cycle in each strongly connected (directed) component, and places the nodes in this cycle on the boundary of a rectangle using alignment constraints. This process is repeated until no more strongly connected components exist.[5]

The choice of styling tools is not exhaustive. We expect that different applications will require different styling tools to capture particular layout conventions. However, so long as the styling tools generate placement constraints, it is straightforward to extend our visualization tool to support them. Using constraints to model layout style is one of the reasons our tool is very flexible.

4.3 Topology-preserving layout

Constrained graph layout techniques have not been previously used for interactive visualization. In such a dynamic application we need layout algorithms that ensure stability of layout so that interaction does not destroy the user's mental model of the network. However, what is meant by stability? As discussed in §2, in previous systems stability has been modelled by (a) trying to keep the positions of nodes unchanged, (b) by keeping the relative horizontal and vertical position of nodes unchanged or (c) preserving the topology of the layout. It is simple to modify constrained graph layout to support approaches (a) and (b). However, both of these approaches severely restrict how the layout

[5]This is a very different approach to the *largest acyclic subgraph* strategy used to break cycles in Sugiyama methods. We prefer this approach because it emphasises cyclic components rather than disguising them by arbitrarily reversing edges.

(a) Initial placement (b) After minimising *P-stress*

Fig. 3. Reducing *P-stress* by shortening edge routings improves the layout while preserving the topology of the initial layout.

can adjust to changes such as expanding a hierarchical sub-graph. We believe that approach (c), preserving the topology of the layout during interaction with a weak preference that nodes do not move unnecessarily, is preferable. Such topology preserving layout fits well with *P-stress* minimization and has a simple, readily understood physical metaphor: The poly-line connectors and cluster boundaries act like rubber-bands, trying to shrink in length and hence straighten. Like physical rubber bands, the connectors and cluster boundaries are impervious in that nodes and other connectors cannot pass through them.

We have developed a novel algorithm for preserving topology of poly-line connectors during layout. Due to space limitations we give only a very brief outline of the algorithm. Full details can be found in [11]. Assume that our initial layout is (X,R,B) and that we have a new position for the nodes X'. Assume that X and X' satisfy the non-overlap and placement constraints and so does any linear interpolation between them. Then to find new edge routing R' and boundary routing B' for node placement X', conceptually, we move the nodes smoothly between X and X', updating the routing as we go. The two changes that can occur are: (a) two consecutive segments (v_1,v_2), (v_2,v_3) on a route straighten and merge into a single segment when the segments become co-linear, and (b) a segment (v_1,v_2) splits into two segments (v_1,v) and (v,v_2) when a node corner v "runs" into the segment.

It is impossible for nodes to go through edges, and so it is impossible for edges to go through edges and so introduce crossings, and it is also impossible for nodes or edges to go through a cluster boundary. Thus, the new layout (X',R',B') preserves the topology of the original layout. And, as long as the initial layout is feasible, the cluster membership constraint and the requirement that edges do not intersect graphic objects will remain satisfied.

Another operation we need is to *repair* an edge route or cluster boundary given that the path may have become invalid because it now passes through a graphic object or could be shortened by straightening and merging some adjacent segments in the path. However, as much as possible we want to preserve the current path. In the case of an edge route we do this by finding a new route for the path and tracing the old path, effectively threading the path through the objects to the destination object. At all stages the connector acts like a rubber band, fitting snugly around objects on the route. The tool uses the connector routing library described in [39] to dynamically route from the start object to the cursor location while preserving as much of the previous route as possible. More exactly, the last bend in the route is removed from the route whenever the bend angle around the node becomes 180° or more, and routing proceeds from the preceding bend.

4.4 Basic layout algorithm

The goal of the constrained graph layout engine is to find a layout that minimizes *P-stress* and which satisfies the refinement and placement constraints. We utilize a three stage algorithm.

(1) A position for the nodes satisfying the placement and refinement constraints is found by projecting the current position of the nodes X^0 on to the placement constraints and then using a greedy heuristic to satisfy the non-overlap constraints and cluster containment constraints. We use the approach detailed in [9].

(2) Edge routing is performed using the incremental poly-line connector routing library [39] to compute poly-line routes for each edge, which minimise edge length and amount of bend. Cluster boundaries are obtained by using the convex hull of the objects in the cluster.

(3) The layout is optimized by iteratively improving the current layout by using a (non-linear) gradient projection approach. This preserves the topology of the initial layout.

Graph	$\|V\|$	$\|E\|$	FM^3	FM^2 Adj.	$\|V'\|$	Avg. FR
tahoe-small	36	51	0.04	0.01	20	19.1
tahoe-large	266	373	0.40	0.13	20	17.3
tahoe-large					40	16.6
biological	432	481	0.72	0.23	20	19.3
biological					40	18.4

Table 1. Timings (in seconds) for the overview and detail layout during interaction on a 1.83 GHz MacBook Pro. FM^3 is the Fast Multilevel Multipole method used for initial overview layout, FM^2 Adj. is the Fast Multipole relayout method with fixed position constraints, and Avg. FR is the average frames (iterations) per second during full, topology preserving constraint layout of the detailed view of the primary graph with node set V'.

It is step (3) of the algorithm which is most novel. Its effect is illustrated in Fig. **??**. It uses a non-linear gradient projection approach [4] to iteratively improves the current layout (X,B,R). It works by choosing a descent direction d and step-size α in which to improve the current node position. However, the new position, $X^d = X + \alpha d$, for the nodes may violate the constraints. This is corrected by projecting X^d on to the feasible region.[6]

One of the keys to the efficiency of our algorithm is that placement constraints are examples of so-called *separation constraints* in a single dimension.[7] Since these only involve variables from a single dimension, it is correct to project on to each dimension independently. There are a number of algorithms for projection on to separation constraints. We use the algorithm given in [9] which is based on an active set method (like the related Simplex algorithm). Non-overlap constraints are handled by using a heuristic to generate a separation constraint which ensures non-overlap. This choice of separation constraint is dynamically updated during the iterative optimization so as to allow objects to move around each other.

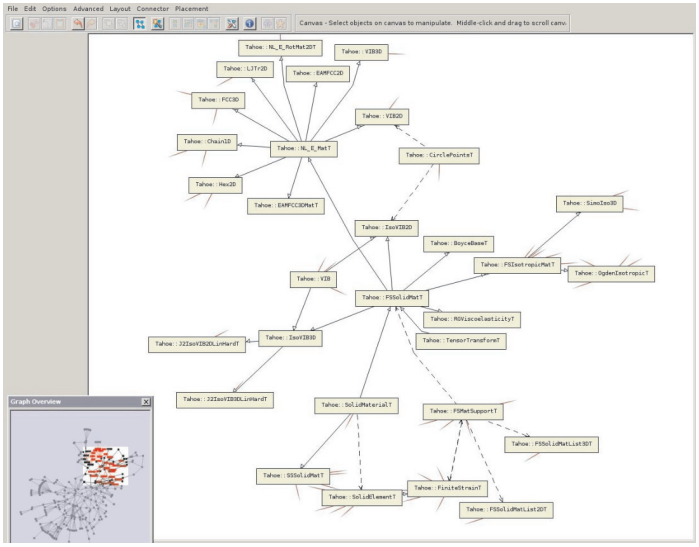
As part of the projection step we update the connector routing R and cluster boundaries B to be in accord with the projected node placement X^d while preserving the original topology. Edge routing correction works as follows. Each node v is moved horizontally/vertically from the initial feasible solution X^0 for which the routing is correct, towards X^d. The current horizontal/vertical position of v is given by $(1-\alpha)X_v^0 + \alpha X_v^d$ where the interpolation factor $\alpha \in [0,1]$ indicates how far along the path to their optimal solution the nodes are. Initially $\alpha = 0$. We then increase α until any further increase will cause one of the edge routing events (split or merge) to occur. We perform the event appropriately straightening or bending an edge or boundary segment and update the edge events. This continues until $\alpha = 1$, in which case we have found the updated routing.

4.5 Smooth transitions

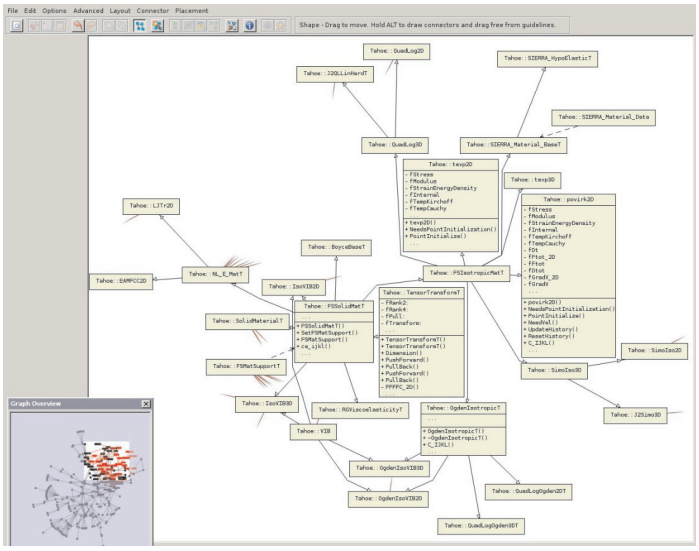
Smooth transitions as the layout is modified in response to changes in the primary graph are essential in supporting a user's mental map as they navigate through the larger secondary graph. One possible approach would be to simply wait for the algorithm to find the local minimum and then use a simple linear interpolation from the current node position to the new position. However, this has two limitations. First, it means that there is no feedback until the layout engine has finished computing the new layout. While layout is reasonably fast, it still means there is a noticeable lag during interaction. Second, Friedrich and Eades [16] suggest that when animating changes in graph layout, intermediate frames that satisfy accepted layout aesthetic criteria are preferable to a simple linear interpolation of node positions. A compelling example is that in some cases such linear interpolation can momentarily collapse a diagram to a point before expanding back to the target layout.

[6] The projection of a point d on to constraints S is the closest feasible point to d. That is, the projection of d on to S is the vector y minimizing $\sum_{i=1}^{n}(y_i - d_i)^2$ subject to S.

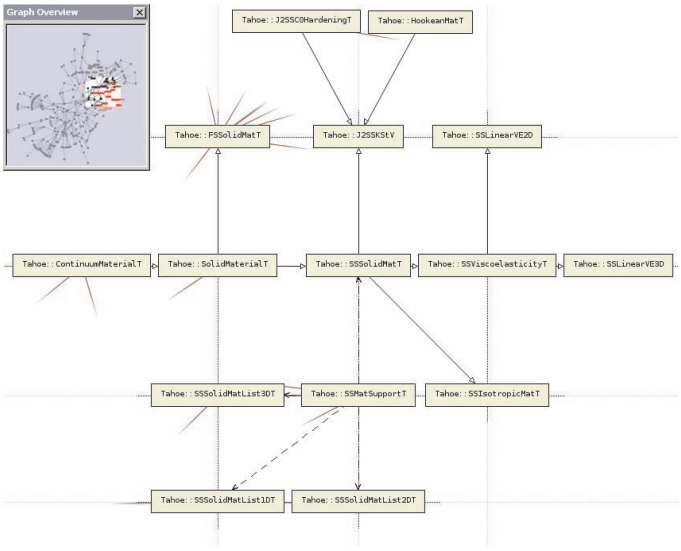
[7] Separation constraints have the form $u + d \leq v$ or $u + d = v$ where u and v are variables representing horizontal or vertical position of a node and d is a constant giving the minimum separation required between u and v.



(a) Here the class *FSSolidMatT* is the initial focus



(b) The user changes focus to *FSIsotropicMatT* and zooms in to some of its neighbours to see specific methods and attributes. Note that the topology of the common subgraph between this and the previous neighbourhood is preserved.



(c) Here orthogonalization constraints have been added to a neighbourhood around the class *SSSolidMatT*

Fig. 4. Navigating a large UML collaboration diagram, from the Tahoe Development Server project (`http://tahoe.ca.sandia.gov/`) The diagram contains 267 classes and 373 relationships between classes.

At each iteration our gradient-projection-based constraint layout method seeks to improve the layout by moving nodes along a gradient related feasible descent vector d. A useful property of layout methods using such numerical optimisation is that they can easily be animated by redrawing at each iteration of the layout. With careful selection of step-size a reasonably smooth animation can be obtained of the layout converging on a local-minimum. We choose the step size from a second degree Taylor expansion of the *P-stress* function about the current set of positions x combined with application of the Armijo-rule[4] to guarantee monotonic decrease of *P-stress*. Unfortunately, monotonic decrease in the cost function is not enough to guarantee that there will be no apparent vibration of nodes as the layout converges. High degree nodes can be particularly problematic in this regard. To alleviate this problem we also apply a fourth-order Runge-Kutta smoothing—essentially choosing each d as a weighted average of 4 (feasible) descent vectors in a neighbourhood from the starting x, see [36, pp. 653–658]. In practice we find that this method leads to smooth animation without having to resort to the "fudge" factors or ad-hoc cooling schemes often applied in force-directed graph layout. Principled techniques from numerical optimisation theory are often eschewed in graph-layout implementations as either too costly or complex, and there is an overhead in computing second derivative information. However, since we are less interested in making our detailed layout method scale to very large graphs, and more interested in obtaining high quality layout and smooth animation we feel that such techniques are well justified in this instance. Table 1 gives an indication of the frame rates and total times to convergence achieved for various size primary graphs.

4.6 Layout modification

We now describe how the layout is modified during user interaction. All interactions trigger the same basic events:

(a) Modify the primary graph if necessary,

(b) Find a new layout satisfying the placement and topological constraints that changes the topology of the current layout as little as possible

(c) Use step (3) of the layout algorithm (§4.4) to optimize the layout while preserving its topology.

(d) Center the detailed view on the focus node.

Steps (c) and (d) are the same for all interactions, so we concentrate on describing steps (a) and (b).

Changing the focus of the detailed view to a new node v. Compute the new nodes V', edges E' and clusters C' that need to be added to the primary graph to ensure v and the nodes it is connected to are in the primary graph. A position X' for the new nodes satisfying the placement and non-overlap constraints is found by projecting the current position of the nodes in the secondary graph on to the placement constraints and then using a greedy heuristic to satisfy the non-overlap and containment constraints. Edge and boundary routes are computed for E' and C' using the incremental poly-line connector routing library if they have not been previously computed, or else if old routes exist because the edge or cluster has previously been in the primary graph the route is repaired.

Changing the LOD of a single node v. This means resizing the node v. The primary graph remains unchanged. Let (X,R,B) be the current layout. The new layout (X',R',B') is computed as follows. A new position X' satisfying the placement and separation constraints generated from the non-overlap constraints is found by projecting the current position X on to these constraints. Updated edge and boundary routes R' and B' are computed from R and B using topology preserving interpolation from X to X'.

Collapse a cluster c of nodes into a single node v_c. We record the edge routing and boundary routing for c—it will be used if the cluster is re-expanded. The primary graph is modified so as to collapse the cluster by removing internal edges and nodes, adding the new node v_c and collapsing edges in and out of the cluster to edges to v_c. The new layout is the same as the old layout except that the internal edges, boundary and nodes of c are no longer placed or routed, the node v_c is placed at the center of where c used to be, and each edge to v_c uses a repaired route of one of the original edges it was based on.

Expand a node v_c representing a cluster c of nodes. We remove v_c and add the internal edges and nodes in c to the primary graph. A position for the nodes satisfying the placement and non-overlap constraints is found by projecting the current position of the nodes on to the placement constraints and then using a greedy heuristic to satisfy the non-overlap constraints. The new edge and boundary routes for the original edges and cluster boundaries are computed using topology preserving interpolation from the old node positions to the new. Edge routes for internal edges in c and boundary of c are computed using the incremental poly-line connector routing library and convex hull unless routing information has previously been recorded for them in which case this is repaired.

Adding a placement constraint. The user can customize and improve the layout by adding placement constraints either manually or by using a styling tool. A new position X' for the nodes in the primary graph is computed by projecting the current position of the nodes X on to the new set of placement constraints and then using a greedy heuristic to satisfy the non-overlap constraints. The edge and cluster routes are repaired.

Removing a placement constraint. This simply removes the constraint.

Direct manipulation. The user can also improve the layout by repositioning a node in the detailed view using direct manipulation. The layout engine is fast enough to provide "live" feedback. Thus the tool is really a kind of collaborative graph-layout tool in which the user can interact with the optimisation engine to improve the layout and escape local minima by providing user hints [7]. By default topology is preserved during direct manipulation. However, if an "alt" key is held down during the manipulation then the user can reposition the node anywhere they like, modifying the topology and breaking any placement constraints. In this case, new routes for the edges from the node are continuously recomputed during the manipulation and other edges and boundaries repaired if the node involved in the direct manipulation is placed on top of them.

5 Overview Display

There are three key requirements for our overview display, it must be:
Fast: layout for either the whole secondary graph or at least a large enough neighbourhood to give context must be completed in only a second or so, even for thousands of nodes;
Able to support fixed position constraints: the positions of the nodes in the primary graph must be fixed in the positions obtained from the detailed layout algorithm, they should influence the layout of the secondary graph but should not be changed;
Incremental: it should take as input a set of starting positions for all nodes in the secondary graph and fixed position constraints for the nodes in the primary graph and find a new layout satisfying the constraints while only moving the unconstrained nodes as little as necessary to beautify the overall layout.

There are now a number of fast layout methods for large graphs as surveyed in §2.2. Unfortunately the second and third requirements eliminate Eigen-projection methods such as ACE [26] and the decomposition methods such as SPF [1]. Fortunately, the above requirements are not dissimilar from those found in N-body physics simulations for which a whole family of methods are well known. Fast multi-pole methods—in which long range forces are approximated for clusters of particles allowing all repulsive forces to be computed in $O(n \log n)$ for n particles—represent the state of the art in this regard. Hachul and Juenger incorporated a multilevel scheme for graphs with a fast multi-pole method in [22]. Unfortunately, it is not clear how such a multilevel scheme (in which a recursive coarsening of the graph is obtained and the layout applied to each level of granularity separately) can be made incremental. Thus, for all but the initial layout of the secondary graph we use a standard fast multi-pole method without the multilevel scheme of Hachul's so called FM^3 algorithm. Fixed position constraints are achieved by disregarding the forces on such nodes when computing the descent vector. Timings for initial and incremental layout of large graphs using the FM^3 and standard fast multi-pole

methods respectively, are given in Table 1.

6 Case Studies

6.1 UML Class diagrams

UML Class diagrams can be quite large with hundreds of classes for a mature software project. They are naturally clustered either by module defining the classes or by aspect. The feature that differentiates UML Class diagrams from most other diagrams is the large amount of information that can be presented in each node. A full Class node may contain very detailed declarations of class attributes and methods, which may require substantial space to display. Simply displaying each Class node at full detail means that very little of the class structure can be viewed in the detailed view, so an essential requirement for browsing Class diagrams is the ability to semantically zoom nodes in situ, smoothly moving from simply the name of the Class to the full detailed definition. Because the size change can be so dramatic, topology preservation is very useful in maintaining the mental map of the diagram when performing semantic zooming. Orthogonalization of selected neighbourhoods (§4.2) also provides landmarks to aid the mental map. Fig. 4 gives an example use-case for UML diagram exploration.

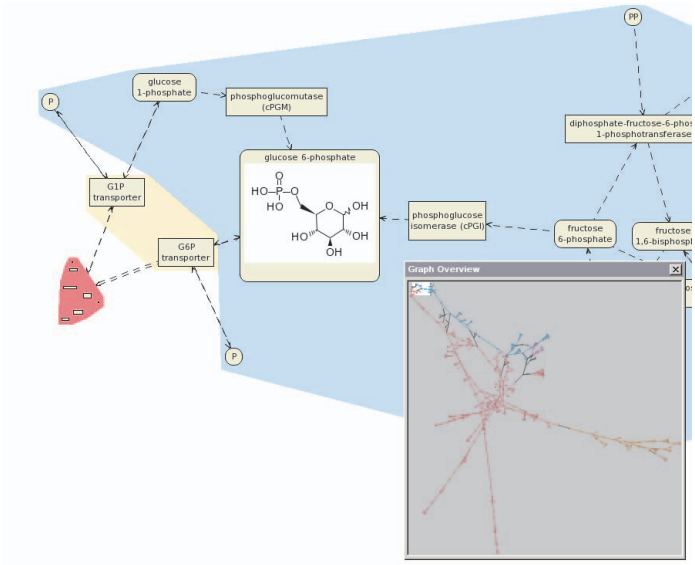
6.2 Biological pathways

Biological networks are used to represent processes in biological systems and to capture important dependencies between biological entities. Due to the steady growth of knowledge in the life sciences such networks are increasingly large and complex with hundreds and thousands of elements. They can be clustered, for example, by functional properties (e. g. different protein classes in protein interaction networks), by spatial information (e. g. cellular compartments to which parts of the network belong to), or user given hierarchies (e. g. the classification of metabolism into groups such as energy metabolism, amino acids synthesis and so on). The features that make biological networks interesting are different additional information which can be present at nodes and edges (such as structural formulas, experimental data) and their diverse visualization requirements [2]. We demonstrate the investigation of metabolic networks which have a complex visual appearance and provide LOD: in each reaction metabolic network substances (nodes) are often divided into main and co-substances; and nodes can also be clustered either by cellular compartments or by user given hierarchies. The network in Fig. 5 shows parts of the carbohydrate metabolism (glycolysis, gluconeogenesis) and several amino acid synthesis pathways derived from the MetaCrop database [21]. Figures 5(a,b,c) show navigation around a large network with 432 nodes and 481 edges, note the ability to expand nodes and clusters to better show detailed node graphics or hidden subgraphs. Fig. 5(c) shows a region of the network where the user has expanded several nodes of interest and applied a number of placement constraints. Important paths in the network such as the backbone of particular pathways can be aligned horizontally or vertically. Using the presented framework users are able to smoothly investigate metabolic networks from abstract overview diagrams (where each pathway is represented by one node) to details of specific reactions (where many details are shown for a few nodes) and to produce high quality, user guided layout ready for publication.

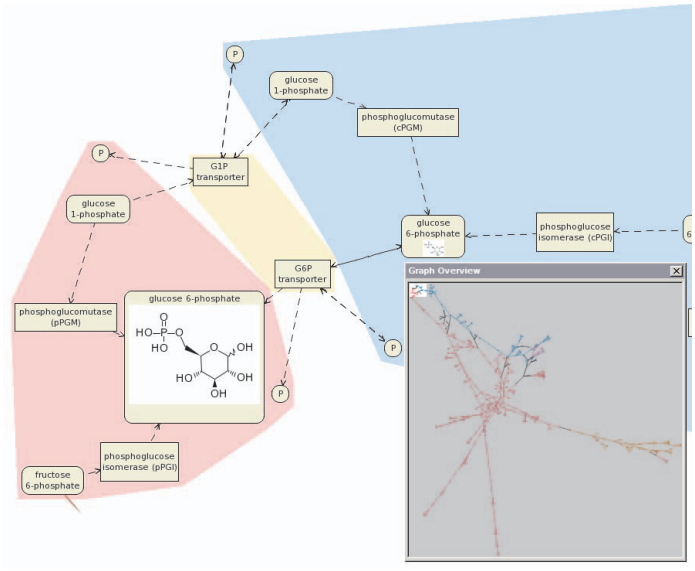
7 Discussion and Further Work

We have presented a new model for interactive exploration of large networks and demonstrated its use for visualization of UML class diagrams and biological networks. Our strategy for combining overview and detail displays is novel as is our approach to providing stable incremental layout via topology preservation. We have built a prototype network browser that demonstrates the viability of our model. However, our experience with the prototype has revealed several limitations.

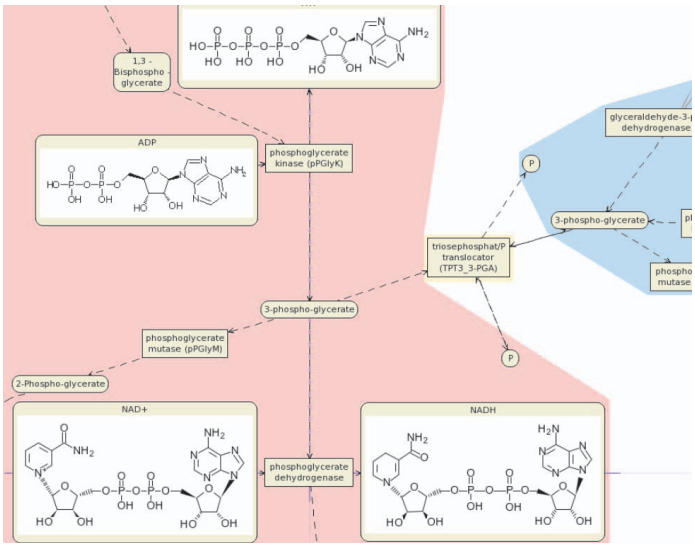
The first is the handling of large clusters. Currently, information about clusters is ignored in the overview, which shows the fully expanded network laid out as an unclustered graph. Even when the user explicitly collapses a cluster in the detailed view, the cluster remains



(a)



(b)



(c)

Fig. 5. Navigation of a metabolic pathway network with 432 nodes and 481 edges; and final precise layout with placement of a detailed subgraph.

expanded in the overview. We are investigating whether it would be better for cluster LOD to remain synchronized between the two views.

The second limitation is the treatment of nodes that have been removed from the primary graph because it has grown too large. Currently the position of these nodes remains fixed so as to ensure any constraints between these nodes remain satisfied. This makes the layout somewhat inflexible: it might be better to allow the nodes to move if the layout quality degrades. We also plan to explore if it is possible to extend the multi-pole algorithm to support penalty-based approximate satisfaction of constraints while still keeping efficiency.

The third limitation we have identified is edge routing in dense graphs. Currently, multiple edges may be routed for part of their length along the same path. We are exploring schemes for offsetting such edges so that their paths are not ambiguous, but this further complicates the algorithms and may increase running times.

A final potential limitation is how to choose which nodes to include in the detailed view if the focal node has a very high degree. This seems an inherent limitation of the focal node-based network exploration model. However, it has not been an issue in our case studies.

There are also a number of other avenues for future work. As described in our literature survey a number of different approaches to mental-map preserving exploration of graphs now exist. A comparative study of the various methods to determine the best approach for various applications would be useful. Further work could also explore the use of different layout algorithms, particularly for the detailed view. Our particular model for constraint-based layout may be a useful way to provide interactive refinement of combinatorial techniques such as Sugiyama and orthogonal layout, using these as *styling tools* to generate constraints, rather than for finding absolute positions.

REFERENCES

[1] D. Archambault, T. Munzner, and D. Auber. Smashing peacocks further: Drawing quasi-trees from biconnected components. *Trans. on Visualization and Computer Graphics*, 12(5):13–820, 2006.

[2] C. Bachmaier, U. Brandes, and F. Schreiber. *Handbook of Graph Drawing and Visualization*, chapter Biological Networks. Chapman & Hall/CRC Press, 2008, to appear.

[3] B. B. Bederson and A. Boltman. Does animation help users build mental maps of spatial information. In *Proc. 1999 IEEE Symp. on Information Visualization*, pages 28–35. IEEE, 1999.

[4] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

[5] U. Brandes, V. Kääb, A. Löh, D. Wagner, and T. Willhalm. Dynamic WWW structures in 3D. *Graph Algorithms and Applications*, 4(3):183–191, 2000.

[6] S. S. Bridgeman, J. Fanto, A. Garg, R. Tamassia, and L. Vismara. InteractiveGiotto: An algorithm for interactive orthogonal graph drawing. In *GD 1997: Revised Papers from the 5th Int. Symp. on Graph Drawing*, pages 303–308. Springer, 1998.

[7] H. A. D. do Nascimento and P. Eades. User hints for directed graph drawing. In *Revised Papers from the 9th Int. Symp. on Graph Drawing (GD '01)*, pages 205–219. Springer, 2002.

[8] T. Dwyer, Y. Koren, and K. Marriott. Drawing directed graphs using quadratic programming. *IEEE Trans. on Visualization and Computer Graphics*, 12(4):536–548, 2006.

[9] T. Dwyer, Y. Koren, and K. Marriott. IPSep-CoLa: An incremental procedure for separation constraint layout of graphs. *IEEE Trans. on Visualization and Computer Graphics*, 12(5):821–828, 2006.

[10] T. Dwyer, K. Marriott, and M. Wybrow. Integrating edge routing into force-directed layout. In *Proc. 14th Intl. Symp. Graph Drawing (GD '06)*, volume 4372 of *LNCS*, pages 8–19. Springer, 2007.

[11] T. Dwyer, K. Marriott, and M. Wybrow. Topology preserving constrained graph layout. Technical Report 227, Monash University, 2008. http://www.csse.monash.edu.au/publications/2008/tr-2008-227-full.pdf.

[12] P. Eades, R. F. Cohen, and M. L. Huang. Online animated graph drawing for web navigation. In *Proc. 5th Int. Symp. on Graph Drawing (GD'97)*, pages 330–335, 1997.

[13] P. Eades and M. L. Huang. Navigating clustered graphs using force-directed methods. *Graph Algorithms and Applications: Special Issue on Selected Papers from 1998 Symp. Graph Drawing*, 4(3):157–181, 2000.

[14] M. Eiglsperger, S. P. Fekete, and G. W. Klau. *Orthogonal Graph Drawing*, pages 121–171. Springer, 2001.

[15] M. Eiglsperger, M. Siebenhaller, and M. Kaufmann. An efficient implementation of Sugiyama's algorithm for layered graph drawing. In *Proc. 12th Int. Symp. on Graph Drawing (GD'04)*, volume 3383 of *LNCS*, pages 155–166, 2004.

[16] C. Friedrich and P. Eades. Graph drawing in motion. *Graph Algorithms and Applications*, 6(3):353–370, 2002.

[17] Y. Frishman and A. Tal. Online dynamic graph drawing. In *Eurographics/IEEE-VGTC Symp. on Visualization*. Eurographics Association, 2007.

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

[19] G. W. Furnas. Generalized fisheye views. In *Proc. of CHI'86*, pages 16–23. ACM Press, 1986.

[20] E. Gansner, Y. Koren, and S. North. Topological fisheye views for visualizing large graphs. In *INFOVIS '04: Proc. IEEE Symp. on Information Visualization*, pages 175–182. IEEE, 2004.

[21] E. Grafahrend-Belau, S. Weise, D. Koschützki, U. Scholz, B. H. Junker, and F. Schreiber. MetaCrop—a detailed database of crop plant metabolism. *Nucleic Acids Research*, 36:D954–958, 2008.

[22] S. Hachul and M. Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Proc. 12th Int. Symp. on Graph Drawing (GD'04)*, volume 3383 of *LNCS*, pages 285–295. Springer, 2004.

[23] S. Hachul and M. Jünger. An experimental comparison of fast algorithms for drawing general large graphs. In *Proc. 13th Int. Symp. on Graph Drawing (GD'05)*, LNCS, pages 235–250. Springer, 2005.

[24] W. He and K. Marriott. Constrained graph layout. *Constraints*, 3:289–314, 1998.

[25] J. Heer and D. Boyd. Vizster: Visualizing online social networks. In *INFOVIS '05: Proc. 2005 IEEE Symp. on Information Visualization*, page 5. IEEE, 2005.

[26] Y. Koren, L. Carmel, and D. Harel. Ace: A fast multiscale eigenvectors computation for drawing huge graphs. In *INFOVIS '02: Proc. IEEE Symp. on Information Visualization*, page 137. IEEE, 2002.

[27] Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Trans. on Computer-Human Interaction*, 1(2):126–160, 1994.

[28] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages and Computing*, 6(2):183–210, 1995.

[29] S. C. North and G. Woodhull. Online hierarchical graph drawing. In *GD '01: Revised Papers from the 9th Int. Symp. on Graph Drawing*, pages 232–246. Springer, 2002.

[30] A. Perer and B. Shneiderman. Balancing systematic and flexible exploration of social networks. *Trans. on Visualization and Computer Graphics*, 12(5), 2006.

[31] C. Plaisant, J. Grosjean, and B. B. Bederson. Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *INFOVIS*, pages 57–, 2002.

[32] H. Purchase, J. Allder, and D. Carrington. Graph layout aesthetics in UML diagrams: User preferences. *Journal of Graph Algorithms and Applications*, 6(3):255–279, 2002.

[33] K. Ryall, J. Marks, and S. Shieber. An interactive constraint-based system for drawing graphs. In *Proc. 10th Annual ACM Symp. on User Interface Software and Technology*, pages 97–104. ACM Press, 1997.

[34] M. Sarkar and M. H. Brown. Graphical fisheye views of graphs. In *Human Factors in Computing Systems, CHI'92 Conference Proc.: Striking A Balance*, pages 83–91. ACM Press, 1992.

[35] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. on Systems, Man and Cybernetics (SMC)*, 11(2):109–125, 1981.

[36] M. Tenenbaum and H. Pollard. *Ordinary Differential Equations*. Dover, 3rd edition, 1985.

[37] F. van Ham and J. J. van Wijk. Interactive visualization of small world graphs. In *INFOVIS '04: Proc. IEEE Symp. on Information Visualization*, pages 199–206. IEEE, 2004.

[38] C. Ware. Interacting with visualizations. In *Information Visualization: Perception for Design*, chapter 10, pages 317–350. Elsevier, 2nd edition, 2004.

[39] M. Wybrow, K. Marriott, and P. J. Stuckey. Incremental connector routing. In *Proc. 13th Int. Symp. on Graph Drawing (GD'05)*, volume 3843 of *LNCS*, pages 446–457. Springer, 2006.