Electronic Communications of the EASST
Volume 13 (2008)

Proceedings of the Second International Workshop on Layout of (Software) Engineering Diagrams (LED 2008)

Interactive, Constraint-based Layout of Engineering Diagrams

Tim Dwyer, Kim Marriott and Michael Wybrow

8 pages

Guest Editors: Andrew Fish, Harald Störrle
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Interactive, Constraint-based Layout of Engineering Diagrams

Tim Dwyer[1], **Kim Marriott**[1] **and Michael Wybrow**[1]

[1]Clayton School of Information Technology, Monash University, Australia
{Tim.Dwyer,Kim.Marriott,Michael.Wybrow}@infotech.monash.edu.au

Abstract: Many engineering disciplines require visualisation of networks. Constrained graph layout is a powerful new approach to network layout that allows the user to impose a wide variety application-specific placement constraints—such as downwards pointing directed edges, alignment of nodes, cluster containment and non-overlapping nodes and clusters—on the layout. We have recently developed an efficient algorithm for topology-preserving constrained graph layout. This underpins two dynamic graph layout applications we have developed: a network diagram authoring tool, Dunnart, and a network diagram browser. In this paper we provide an overview of topology-preserving constrained graph layout and illustrate how Dunnart and the network diagram browser can be applied to engineering diagram authoring and visualisation.

Keywords: constraint-based layout, diagram authoring, diagram exploration

1 Introduction

Many engineering disciplines require visualisation of networks. For example, software and process engineers need to understand the complex networks of relationships between system components. A wide variety of graph layout algorithms have been developed to aid such visualisation [DETT99]. However, many of these algorithms are designed to draw simple, idealised mathematical graphs. This significantly limits their usefulness since, in many applications, the networks have much more complex structure and, consequently, more constraints on their layout. Such constraints include, for instance, requiring directed connections to be represented by arrows that point downward, grouping of particular nodes into clusters, large labels on nodes and edges, alignment of selected nodes, and an ordering on nodes perhaps reflecting an underlying physical or temporal ordering. Standard graph layout techniques for handling these application-specific layout requirements are complex and are brittle in the sense that each technique can handle only a particular kind of layout constraint.

We have developed a new approach to network layout that provides a generic, robust framework that handles the layout constraints arising in a wide variety of applications. The approach, *constrained graph layout* [HM98, DKM06a, DKM06b], generalises the popular force-directed model for graph layout. Like force-directed methods, these techniques find a layout minimising a goal function such as the standard *stress* goal function which tries to place all pairs of nodes their ideal (graph-theoretic) distance apart. However, unlike force-directed methods, constrained graph layout algorithms allow the goal to be minimised subject to so-called *separation constraints* on the nodes in each dimension [DKM06b]. These have the form $u + d \leq v$ or $u + d = v$ where u and v are variables representing horizontal or vertical position and d is a constant giv-

ing the minimum separation required between u and v. Although seemingly a very restricted kind of linear constraint, separation constraints are expressive enough to handle a wide variety of application-specific layout constraints. These include:

Directed edges We can ensure that node v is placed above (or, to the left of) node u if there is a directed edge from v to u.

Alignment or distribution E.g. placing selected nodes on different horizontal layers.

Bands By adding dummy variables we can ensure that nodes are placed in vertical or horizontal bands, as defined in [DK05].

Fixed position A node's position can be fixed in any axis.

Containment We can ensure that selected nodes lie in a rectangular region, for instance within the boundaries of page, window or a cluster dynamically sized to fit its contents.

Orthogonal ordering between nodes We can ensure that nodes are to the left/right or above/below other nodes.

Non-overlap of nodes and/or clusters By dynamically generating separation constraints we can ensure that nodes do not overlap each other and, in combination with containment constraints, that clusters do not overlap.

While constrained graph layout techniques were originally employed for static layout they are also suited to dynamic layout. A core requirement of dynamic graph layout is stability of layout during changes to the graph so as to preserve the user's mental model of the graph. One natural requirement to achieve this is that, as far as possible, the topology of the current layout is preserved during layout changes. We have recently developed an efficient algorithm for *topology-preserving* constrained graph layout [DMWb]. In addition to handling separation constraints, this algorithm can improve a layout while preserving the original topology of the layout.

Topology-preserving constrained graph layout underpins two dynamic graph layout applications we have developed. The first is a network diagram authoring tool, Dunnart, which uses the algorithm to provide continuous layout adjustment during user interaction [DMWa]. The second is a network diagram browser which uses the algorithm to update the layout of a detailed view of part of the network as the user changes the focus node or collapses or expands node clusters [DMS+]. In this paper we provide an overview of topology-preserving constrained graph layout and of these two applications, and illustrate how they can be applied to engineering diagram authoring and visualisation.

2 Topology-preserving constraint-based graph layout

In this section we briefly review topology-preserving constrained graph layout. For more details the reader is referred to [DMWb]. A *network diagram* (V,E,C) is a graph with nodes V, a set of edges $E \subseteq V \times V$, and a set of node clusters $C \subseteq \wp V$. These reflect the three kinds of objects provided in the diagramming tool. The nodes correspond to the basic graphic shapes, such as rectangles, ellipses, etc. Each node $v \in V$ is assumed to be rectangular with a fixed width w_v and height h_v and to have four corners tl_v, tr_v, bl_v, br_v. Node clusters correspond to container shapes: a cluster $c \in C$ is simply the set of nodes in a particular container shape. Container shapes automatically change shape to snugly fit their component shapes. Note that for simplicity we do

not allow nested clusters—this would be relatively easy to add. Each cluster c has a *boundary* B_c which is a sequence of distinct node corners except that the first and last element are the same. This defines a closed region called the *cluster region*. Consecutive pairs of elements in the boundary are called *boundary segments*. The third kind of objects in a network diagram are the edges, which model connectors attached to a start and an end object. Edges may have an arrow head at one end, in which case they are said to be "directed". A *route* R_e for an edge $e \equiv (u,v) \in E$ is a sequence of node corners and nodes s.t. the first element is u, the last element is v and the other elements are distinct node corners. Consecutive pairs of elements in the route are called *edge segments*.

A *layout* for a network diagram is a pair (X,P) which consists of a position $X_v \equiv (x_v, y_v)$ for each node $v \in V$ and the set of *paths*, i.e. edge routes and cluster boundaries, $P \equiv \{R_e | e \in E\} \cup \{B_c | c \in C\}$ in the network.

Constrained graph layout allows constraints on the placement of nodes. These are required to be separation constraints in a single dimension. The layout must also satisfy various *refinement constraints* to ensure that 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: if node v is in cluster c then v must be fully contained in the cluster region of c and if v is not in cluster c then v must not intersect the cluster region (although it can be on the boundary). The last refinement constraint is that edges are not allowed to pass through nodes—every path $p \in P$ is *valid* and *tight* where a valid path is one in which no segment passes *through* a node and a tight path is one in which the path "wraps" tightly around each node corner in the path.

Topology-preserving constrained graph layout uses the *P-stress* (for path-stress) goal function to measure the quality of a layout. Given a layout (X,P), its *P-stress* is

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

where d_{ij} is the graph theoretic distance between nodes i and j, d_p is the ideal length of path p, $w_p = \frac{1}{d_p^2}$ and z^+ is z if $z \geq 0$ and 0 otherwise.

The first component of *P-stress* is a modification of the stress function which penalises nodes that are too close together. However, unlike the stress function, nodes that are more than their ideal distance apart are not penalised, thus eliminating long range attraction since this can cause issues in highly constrained problems. 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 has the effect of straightening edges and making clusters more compact and circular in shape. The ideal length of an edge is a user-defined parameter while the desired boundary length of cluster c is $2\sqrt{\pi \sum_{v \in c} w_v w_h}$ (i.e. the ideal length is proportional to the perimeter of the circle of the same area as that of the constituent nodes).

The following algorithm finds a layout that minimises *P-stress* and satisfies the layout constraints.

(1) A position for the nodes satisfying the layout constraints is found by projecting [1] the current position of the nodes X^0 on to the placement constraints and then using a greedy heuristic

[1] 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 minimising $\sum_{i=1}^{n}(y_i - d_i)^2$ subject to S.

to satisfy the non-overlap constraints and cluster containment constraints (modeled using a rectangular box) [DKM06b].

(2) Edge routing is performed using an incremental poly-line connector routing algorithm to compute poly-line routes for each edge, which minimise edge length and amount of bend [WMS06]. The cluster boundary is obtained using the convex hull of the cluster.

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

Note that unlike force-directed layout, constrained graph layout techniques ensure that the generated layouts really do satisfy all of the layout constraints (unless the constraints are infeasible).

Also note that topology-preserving minimisation of *P-stress* has a simple physical metaphor: the paths, i.e. poly-line connectors and cluster boundaries, act like rubber-bands, trying to shrink in length and hence, in the case of connectors, straighten. And, just like physical rubber bands, the paths are impervious and do not allow nodes and other paths to pass through them.

The above layout algorithm is used in a network diagram authoring tool and network browsing tool that we have developed. We now describe these.

3 Diagram authoring

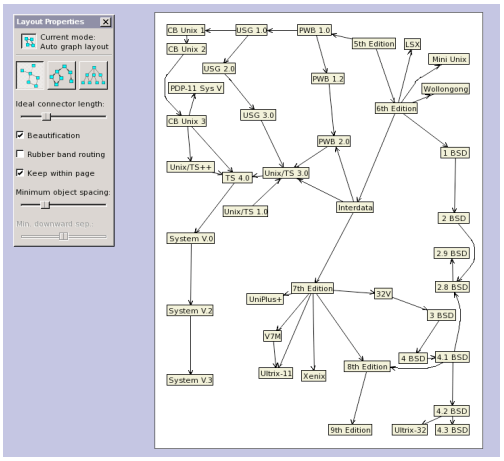
Although some general purpose diagramming tools, such as Microsoft Visio[2] and Omnigraffle,[3] provide automatic graph layout, the integration of graph layout into these tools is quite unsatisfactory. Similar concerns apply to the network layout tool yEd.[4] The issue is that these tools use static graph layout algorithms which are not well-matched to the inherently interactive nature of diagramming tools. They provide only "once off" graph layout and allow little flexibility for the author to tailor the resulting layout by, say, requiring that certain nodes are aligned.

We believe that a better model for integrating automatic graph layout into diagramming tools is *continuous network layout*. In this model the graph-layout engine runs continuously to improve the layout in response to user interaction. The author uses placement constraints, such as alignment and distribution, to tailor the layout style and can guide the layout by repositioning diagram components or rerouting connectors. Importantly, layout should be fast enough to immediately show the effect of changes made to the diagram by the author. Thus, continuous network layout requires efficient dynamic graph layout techniques that support placement constraints.

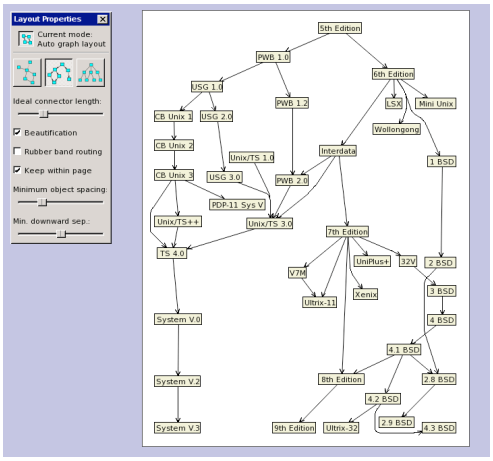
Continuous network layout was introduced in GLIDE [RMS97]. However, the spring-based layout algorithm used by GLIDE was not robust or powerful enough to truly support the model. We have developed a new network diagram authoring tool, Dunnart [DMWa],[5] that provides continuous network layout and uses a topology-preserving constrained graph layout algorithm.

Dunnart supports a variety of different layout styles, arbitrary clusters of nodes, and placement tools such as alignment, distribution and separation. Dunnart's layout engine continuously adjusts the layout in response to user interaction, ensuring that the diagram remains "tidy," i.e. without overlapping objects, while still maintaining the layout style and user-imposed place-

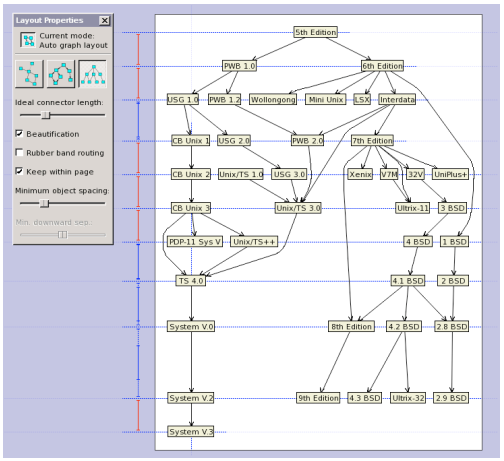
[2] "Layout Assistant for Visio", Tom Sawyer Soft., http://www.tomsawyer.com/lav/
[3] "Omnigraffle", The Omni Group, http://www.omnigroup.com/omnigraffle/
[4] "yEd", yWorks, http://www.yworks.com/products/yed/
[5] "Dunnart", http://www.csse.monash.edu.au/~mwybrow/dunnart/



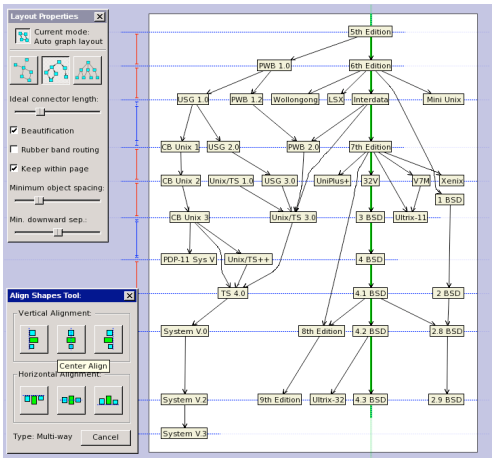
(a) *Organic* layout style. The controls on the left allow parameters such as default edge length to be adjusted with a slider. As the slider is moved the layout updates immediately to show the results.



(b) The author tries a different layout style: *flow*. This adds style constraints requiring that directed edges be downward pointing.



(c) Unhappy with the result, the author now tries the *layered* layout style. This forces objects to be arranged on layers. It generates a set of horizontal alignment constraints with separation constraints between them. The separation constraints keep the layers in order and enforces a minimum spacing (adjustable by the user) between layers.



(d) The author now fine-tunes this layout. First, they draw attention to a particular path through the diagram by vertically aligning some objects. Following this action, they manually change the ordering of several nodes within their layers so that the diagram better fits within the page boundary.

Figure 1: Screenshots illustrating interaction and the various styles of network layout possible with Dunnart. The "Unix family tree" network is based on a Graphviz example diagram: `http://www.graphviz.org/Gallery/directed/unix.html`

ment constraints. Layout adjustment occurs in real-time providing immediate feedback about the effect of user changes.

Figure 1 illustrates the use of Dunnart. See [DMWa] for a detailed description of Dunnart's features and behaviour.

4 Network diagram visualisation

Another tool we have developed is a network visualisation tool [DMS+]. This is based on the classic overview+detail visualisation model. The core innovation in the tool is that we use topology-preserving constrained graph layout to generate a high quality layout for the sub-graph displayed in the detailed view and a high-speed layout method for the remainder of the graph.

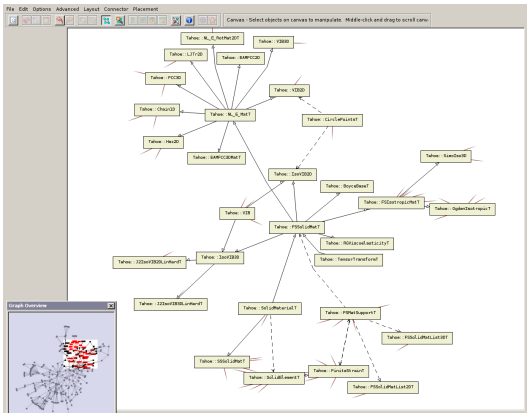
The user navigates through the network by repeatedly selecting a focal node to centre in the detailed view. The tool then displays the surrounding neighbourhood (up to a maximum of forty nodes) within the detailed view. The tool allows the user to change the level of detail shown in an individual node, as demonstrated with the UML Class nodes in Figure 2. They can also change the level of detail in the network by choosing a cluster, i.e. a hierarchical collection of nodes, to be expanded or collapsed in the detailed view.

The tool also allows the viewer to tailor the layout in the detailed view by imposing placement constraints on the layout. These allow the author to control the layout without having to explicitly position objects. The relationship is maintained in subsequent interaction until the author explicitly removes it, including being remembered and restored when the affected nodes leave and return to the detailed view. 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 [War04]. 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.

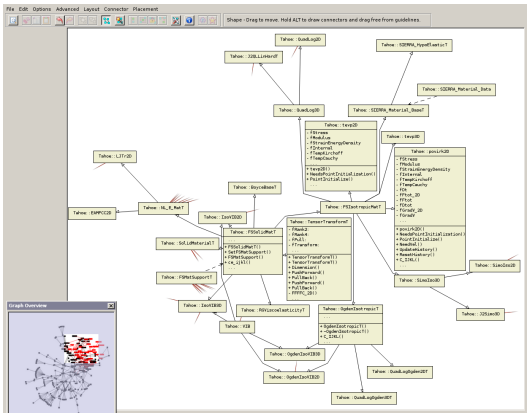
An example session with the network visualisation tool is shown in Figure 2. See [DMS+] for a detailed description of the tool's features and behaviour.

5 Conclusion

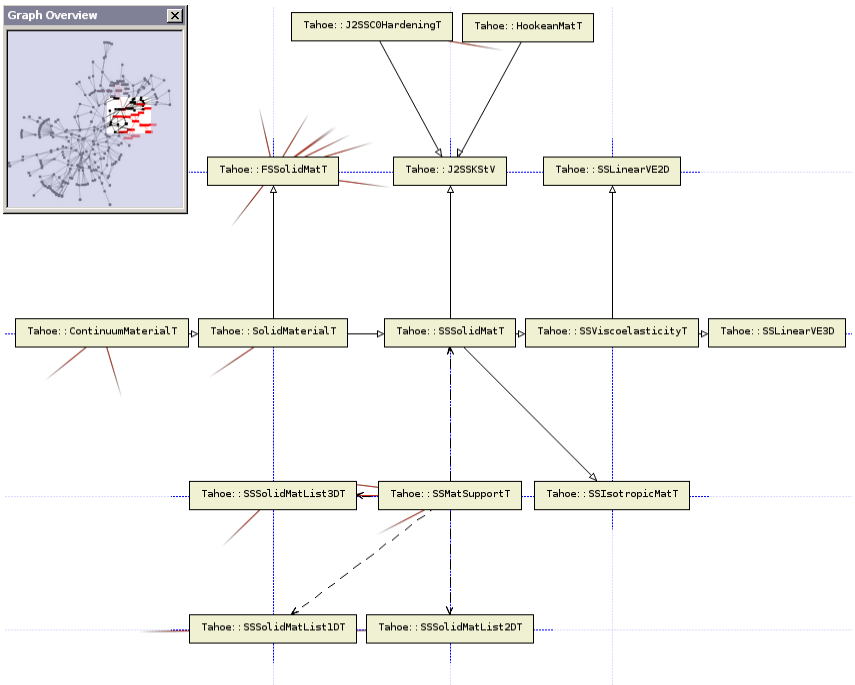
We have described two new interactive tools: one for authoring network diagrams and one for visualising large network diagrams. Both tools utilise constrained graph layout, allowing the user to impose placement constraints on the diagram to capture a wide variety of application-specific layout constraints. Furthermore, the dynamic network layout in these tools is stable and, where reasonable, the topology of the current layout is preserved during user interaction. We believe that these tools demonstrate that constrained graph layout provides an ideal basis for layout of network diagrams occurring in engineering applications.



(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 orthogonalisation constraints have been added to a neighbourhood around the class *SSSolidMatT*

Figure 2: Exploring 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. This figure is taken from [DMS+].

Bibliography

[DETT99] G. Di Battista, P. Eades, R. Tamassia, I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, Inc., 1999.

[DK05] T. Dwyer, Y. Koren. Dig-CoLa: Directed Graph Layout through Constrained Energy Minimization. In *Proceedings of the IEEE Symposium on Information Visualization (Infovis'05)*. Pp. 65–72. IEEE, 2005.

[DKM06a] T. Dwyer, Y. Koren, K. Marriott. Drawing Directed Graphs Using Quadratic Programming. *IEEE Transactions on Visualization and Computer Graphics* 12(4):536–548, 2006.

[DKM06b] T. Dwyer, Y. Koren, K. Marriott. IPSep-CoLa: An incremental procedure for separation constraint layout of graphs. *IEEE Transactions on Visualization and Computer Graphics* 12(5):821–828, 2006.

[DMS+] T. Dwyer, K. Marriott, F. Schreiber, P. J. Stuckey, M. Woodward, M. Wybrow. Exploration of networks using overview+detail with constraint-based cooperative layout. *IEEE Transactions on Visualization and Computer Graphics (InfoVis 2008)*. To appear 2008.

[DMWa] T. Dwyer, K. Marriott, M. Wybrow. Dunnart: A Constraint-based Network Diagram Authoring Tool. In *Proc. 16th Int. Symp. on Graph Drawing (GD'08)*. To appear 2008.

[DMWb] T. Dwyer, K. Marriott, M. Wybrow. Topology Preserving Constrained Graph Layout. In *Proc. 16th Int. Symp. on Graph Drawing (GD'08)*. To appear 2008.

[HM98] W. He, K. Marriott. Constrained Graph Layout. *Constraints* 3:289–314, 1998.

[RMS97] K. Ryall, J. Marks, S. M. Shieber. An Interactive Constraint-Based System for Drawing Graphs. In *ACM Symposium on User Interface Software and Technology*. Pp. 97–104. 1997.

[War04] C. Ware. Interacting with visualizations. In *Information Visualization: Perception for Design*. Chapter 10, pp. 317–350. Elsevier, 2nd edition, 2004.

[WMS06] M. Wybrow, K. Marriott, P. J. Stuckey. Incremental Connector Routing. In *Proc. 13th Int. Symp. on Graph Drawing (GD'05)*. LNCS 3843, pp. 446–457. Springer, 2006.