Dunnart: A Constraint-based Network Diagram Authoring Tool

Tim Dwyer, Kim Marriott, and Michael Wybrow

Clayton School of Information Technology, Monash University, 3800, Australia
{Tim.Dwyer,Kim.Marriott,Michael.Wybrow}@infotech.monash.edu.au

Abstract. We present a new network diagram authoring tool, Dunnart, that provides *continuous network layout*. It continuously adjusts the layout in response to user interaction, while still maintaining the layout style and, where reasonable, the current layout topology. The diagram 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. The key to the flexibility of our approach is the use of topology-preserving constrained graph layout.

1 Introduction

Producing well laid out network diagrams is not easy and extremely tedious for any but the simplest networks. While automatic graph layout algorithms can provide high-quality layout [4], in many situations users would like the ability to interactively control and fine-tune the layout with similar flexibility to that provided in standard diagram authoring tools. Although some general purpose diagramming tools, such as Microsoft Visio[1] and Omnigraffle,[2] provide automatic graph layout, the integration of graph layout into these tools is quite unsatisfactory. Similar concerns apply to the network layout tool yEd.[3] 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 allow the diagram author to immediately see the effect of their changes. Thus, continuous network layout requires efficient dynamic graph layout techniques that support placement constraints.

[1] "Layout Assistant for Visio", Tom Sawyer Soft., http://www.tomsawyer.com/lav/
[2] "Omnigraffle", The Omni Group, http://www.omnigroup.com/omnigraffle/
[3] "yEd", yWorks, http://www.yworks.com/products/yed/

Continuous network layout was introduced in GLIDE [13]. However, the spring-based layout algorithm used by GLIDE was not robust or powerful enough to truly support the model. Here we present a new network diagram authoring tool, Dunnart,[4] that provides continuous network layout and which uses a recently developed topology preserving constrained graph layout algorithm [7]. This provides considerably more robust and powerful automatic layout than is possible with unconstrained optimisation techniques such as those underlying GLIDE.

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" by, for instance, removing object overlap, while still maintaining the layout style and user imposed placement constraints. Figure 1 illustrates the use of Dunnart.

One of the most interesting innovations in Dunnart is a simple, readily understood physical metaphor for *layout adjustment*: 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. This means that layout adjustment preserves the general structure of the network drawing, i.e. its *topology*, and so changes are smooth and predictable. Changes to the topology only occur as the result of explicit direction by the author and for common user editing actions, such as moving objects during direct manipulation or resizing a node, the diagram topology is preserved.

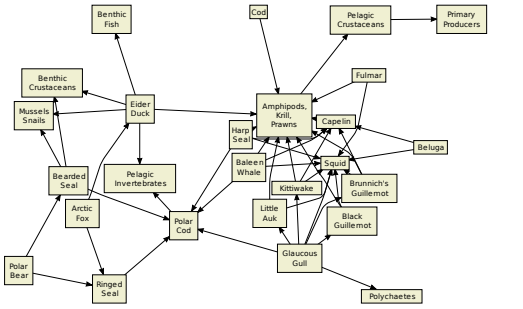
Usability concerns have guided the design of Dunnart from its beginning and we have carefully considered the design of the constraint-based placement tools including how to provide adequate feedback about constraint interaction (especially in the case of inconsistency) and how to ensure that the diagram is not too cluttered by visual representation of constraints (See Fig. 2). One important factor improving usability is that layout adjustment occurs in real-time, providing immediate feedback about the effect of user changes.

2 Related Work

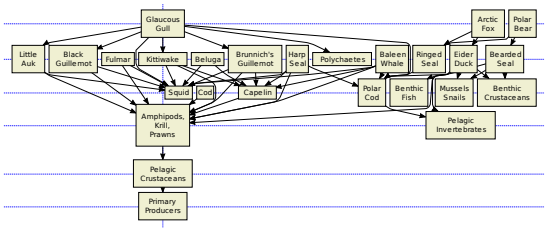
Our work brings together research into constraint-based diagramming editors and research into graph drawing. Since the very infancy of diagram authoring tools there has been interest in allowing the author to specify persistent layout relationships on the diagram components, e.g. [14, 12, 8]. Previous systems have explored constraint solving techniques and user interaction for constraint-based placement tools. However, apart from GLIDE [13], these were not designed for network diagrams and none provided automatic network layout in the sense that we are discussing.

GLIDE was the first constraint-based diagramming tool explicitly designed for network diagrams. It introduced continuous network layout and provided

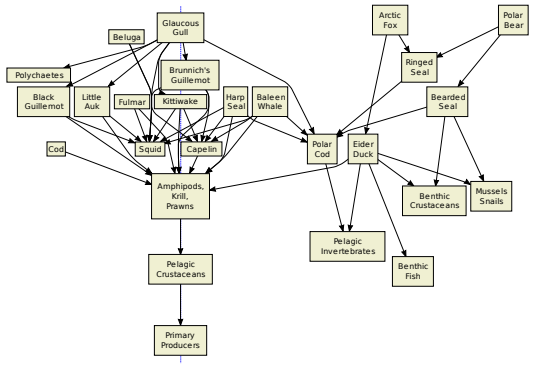
[4] Dunnart, `http://www.dunnart.org/`



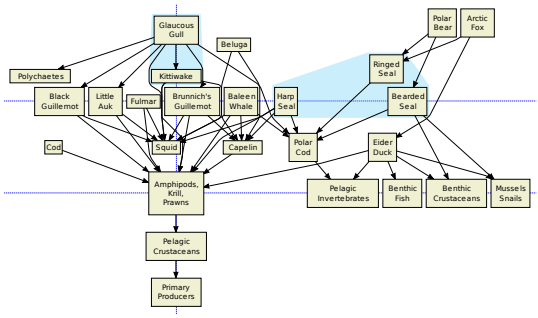
(a) Organic layout style



(b) Layered layout style



(c) Flow layout style



(d) Final layout

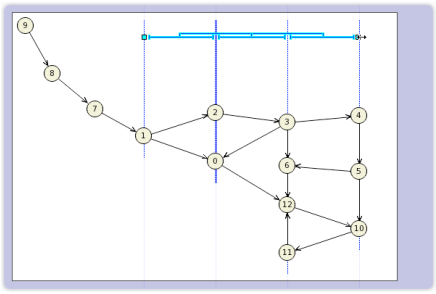
(a) The author initially calls the structural layout tool with *organic layout* style. Note how various automatic refinement constraints such as non-overlap of nodes keep the layout tidy.

(b) The author tries the *layered* layout style. This generates a set of horizontal alignment constraints with separation constraints between them. The separation constraints keep the layers in order and enforce a minimum spacing (adjustable by the user) between layers. The author has added a vertical alignment constraint to several nodes to improve the layout.

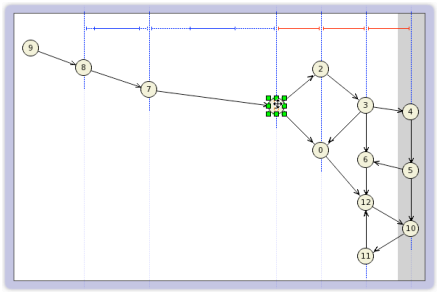
(c) Unhappy with the result, the author now tries the *flow* layout style. This generates style constraints requiring that directed edges be downward pointing. The minimum separation between nodes connected by directed edges can be adjusted using a slider. Note that the vertical alignment constraint was maintained when switching layout styles.

(d) Reasonably happy, the author now fine-tunes the layout. They first add two new horizontal alignment constraints. They then cluster together three seal species and two types of gull. As a result the position of several nodes and edges are automatically updated to remove overlap with the clusters. Finally, the author repositions the "Beluga" and "Arctic Fox" nodes to improve the clarity of the diagram.

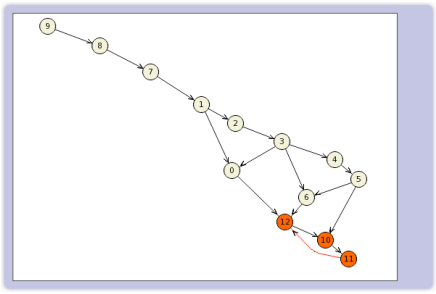
Fig. 1. Example of interactive network layout with Dunnart. Network data from the Many Eyes "Arctic food chain" visualisation, `http://www.many-eyes.com/`.



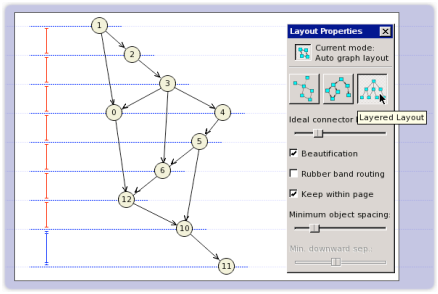
(a) Guidelines are bold in the regions between attached objects and are significantly faded outside those regions. This helps to reduce clutter while still allowing the user to attach objects to guidelines. The four vertical alignment constraints (represented by the guidelines) are involved in a horizontal distribution constraint which requires them to be equally spaced. The user is currently adjusting the distribution spacing (about the highlighted guideline) by dragging a handle on right-side of the distribution indicator.



(b) The six alignment constraints have horizontal separation constraints between them. The minimum separation distance may be adjusted by dragging a handle on the separation indicator. Note that three constraints are currently active (highlighted in red) while the other two have some slack. The grey band on the right edge of the page shows that a page-containment constraint is not satisfied. This results from the user indirectly pushing a shape outside the page while dragging another shape.



(c) Flow layout has been enabled, constraining all directed edges to point downward. A cycle of directed edges causes a constraint conflict. Dunnart drops one of the conflicting constraints, and highlights this as well as the set of affected nodes.



(d) The widgets in the "Layout Properties" window control the structural layout setting. They also allow the user to adjust just the ideal edge length, and toggle generation of non-overlap and page-boundary constraints.

Fig. 2. Screenshots showing the visual representations of constraints in Dunnart.

high-level placement constraints (called VOFs) which the author could add to control the layout and which the layout engine endeavoured to satisfy during subsequent changes to the layout. However GLIDE had two serious limitations. The first was a lack of robustness. GLIDE used springs to approximately enforce layout constraints which effectively meant the constraints were solved by minimising a goal function that contains an error term for each constraint. In the case of conflicting forces, such as, for example, alignment of nodes in a network

with springy connectors, the so called "constraints" would simply not be satisfied, or worse, the whole system could become unstable and not converge to a local minimum. The second limitation was that GLIDE provided little automatic network layout. While it did allow the user to manually impose VOFs to control edge length, when used in combination with other user-specified VOFs this led to conflicting forces and unsatisfied constraints.

Our techniques for network layout draw upon recent research into graph drawing. One relevant area of research is *dynamic graph layout* [2] which focuses on stable re-layout of changing graphs, or interactive navigation of large graphs [10]. Most such systems are based on unconstrained force-directed layout in which the forces between nodes are modified in response to user interaction. However, in these systems the level of user control over the layout is very limited (i.e., alignment or distribution of nodes is not supported) and, because of the underlying optimisation techniques, it would be difficult to provide more.

Our work is also related to collaborative graph-layout tools in which the user can interact with the optimisation engine to improve the layout and escape local minima by providing user hints [11], such as repositioning a node. This is also true in the continuous network layout model, since user interaction can guide the layout engine away from undesired local minima. The fundamental difference is that Dunnart is a generic network diagramming tool, while collaborative graph layout is intended to allow the user to improve the layout obtained with a single specialised layout engine. Thus, user hints are quite restrictive and depend on the underlying layout algorithm. For example, the systems of [11, 1] are built on top of a layered graph-drawing algorithm for directed graphs, while the Giotto system [3] is built on top of an orthogonal graph layout engine.

Dunnart is based upon so called *constrained graph layout* algorithms which perform graph layout subject to various kinds of layout constraints [9, 5]. It uses a recent algorithm for topology preserving constrained graph layout [7] designed for dynamic graph layout. This has previously been used for interactive visualisation of large networks [6]. Here we demonstrate its usefulness in a new application area: authoring.

3 Background: Constrained Graph Layout

In this section we briefly review the algorithm for topology preserving constrained graph layout. It is described more fully in [7]. The algorithm works on *network diagrams*. These can contain: basic graphic shapes, such as rectangles and ellipses, which are treated as rectangular nodes in the diagram; connectors, which form the edges in the diagram and may be directed; and container shapes, which contain a set of nodes and so specify node clusters in the diagram.

A *layout* for a network diagram gives a position for each node in the diagram and a route for the *paths*, i.e. edge routes and cluster boundaries, in the network.

Constrained graph layout allows constraints on the placement of nodes. These are required to be *separation constraints* in a single dimension.[5] The layout must also satisfy various *refinement constraints* to ensure that it is "tidy." The refinement constraints are:

- no two nodes overlap;
- the nodes inside the region defined by the boundary of each cluster are exactly the nodes in the cluster;
- every path 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. *P-stress* modifies the standard *stress* function to penalise nodes that are too close together, but not nodes that are more than their ideal distance apart, thus eliminating long range attraction which can cause issues in highly constrained problems. *P-stress* also tries to make the length of each path in the network no more than its ideal length. This has the effect of straightening edges and making clusters more compact and circular in shape.

The basic algorithm to find a layout that minimises *P-stress* and which satisfies the layout constraints is:

(1) Find a position for the nodes satisfying the layout constraints 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 and cluster containment constraints (modeled using a rectangular box).

(2) Perform edge routing using an incremental poly-line connector routing algorithm [15] to compute poly-line routes for each edge, which minimise edge length and amount of bend. The cluster boundary is obtained using the convex hull of the cluster.

(3) Optimise the layout by iteratively improving the current layout using gradient projection to reduce *P-stress*. This preserves the topology of the initial layout.

As noted previously, 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).

4 Dunnart

Dunnart is intended to be a generic diagramming tool that supports most diagram types, including network diagrams. The original motivation for Dunnart was to explore usability issues in constraint-based diagramming tools. Thus, usability has been a focus of its design from the beginning. Feedback from its

[5] 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.

use—for constructing a wide variety of diagrams including UML diagrams and biological networks—has greatly improved the interface design. We now look at its more novel aspects.

A primary usability consideration was when and how much the layout engine should change the layout in response to user interaction. Typically, when first constructing a network diagram, the user will try different layout styles and, for each style, wants the tool to automatically find a good layout. Then, once the basic layout and style is chosen, the user will fine-tune the layout. During fine-tuning, it is important that changes made by the layout engine are predictable and controllable by the author. To support these two use cases, Dunnart provides two kinds of network layout: *structural layout* and *layout adjustment*. We now look at these.

4.1 Structural layout

Dunnart provides a structural layout tool which is free to completely rearrange the layout so long as the user-specified placement constraints remain satisfied. It is explicitly invoked by the author to re-layout the network. The other function of the structural layout tool is to impose a layout style on the diagram. Dunnart currently provides three layout styles: organic, flow and layered (shown in Fig. 1). It could be extended with other layout styles. The only requirement is that the aesthetic constraints imposed by the style must be able to be modelled using separation constraints so that the layout aesthetic can be maintained in subsequent interaction.

Organic layout is the most basic style since it does not impose any style constraints. It simply calls the constrained graph layout algorithm sketched in Sect. 3. Flow-style layout is the same except that the tool adds style constraints ensuring that the start node of each directed edge is above its end node.

Structural layout can also use external graph layout algorithms to find a layout and determine the style constraints. As an example of this, structural layout with the layered style uses the Graphviz[6] library implementation of the Sugiyama algorithm. This determines a layer for each object in the network, the ordering of objects on each layer and a routing for connectors through the layers which minimises crossings. An alignment placement constraint is generated for each layer and a separation constraint between each pair of layers keeps them a minimum distance apart and preserves the layer order. Currently, existing placement constraints are initially ignored in this style and only imposed in the subsequent layout adjustment step.

Style constraints behave like author specified placement constraints. Thus, the author is free to modify the layout by removing style constraints. Using constraints to model layout style is one of the reasons Dunnart is very flexible. It means that, unlike most previous diagramming tools, layout styles are not brittle and the author is free to tailor the layout style by adding placement constraints

[6] Graphviz, AT&T Research, `http://www.graphviz.org/`

Table 1. Computation of new feasible layout after common kinds of user interaction. Note that this step is always followed by topology-preserving layout optimisation.

Add graphic object: Node repair followed by edge routing repair.
Delete graphic object: Edge routing repair.
Add connector: Automatically or manually route connector.
Delete connector: Nothing—layout remains feasible.
Add/modify cluster: Node repair followed by edge routing repair.
Delete cluster: Nothing—layout remains feasible.
Cut/Copy (to clipboard): Copy nodes to clipboard and perform edge routing repair. If cutting, delete graphic objects and connectors.
Paste (from clipboard): Add nodes to canvas and perform node repair. Then perform edge routing repair (based on connector routing in clipboard for pasted connectors).
Add a placement constraint: Node repair followed by edge routing repair. However, nodes that have moved too far because of the placement behave as if cut and pasted.
Delete a placement constraint: Nothing—layout remains feasible.

to the diagram before calling the structural layout tool, or by subsequently modifying the placement and style constraints.

4.2 Layout Adjustment

The second kind of automatic layout provided in Dunnart is called *layout adjustment*. This supports fine-tuning of the layout and runs continuously during interaction. Changes made by the layout engine during layout adjustment need to be predictable and (reasonably) continuous. Consequently, we believe layout adjustment should preserve the topology of the starting layout as far as possible.

We now describe how the layout is updated after the main kinds of user interaction provided in Dunnart. For most interactions this has two steps. First, find a new feasible layout satisfying the placement, style and refinement constraints that changes the topology of the current layout as little as possible. Second, perform step (3) of the layout algorithm (Sect. 3) to optimise the layout while preserving its topology. Table 1 gives details of how the new feasible layout is found for different kinds of user interaction. We make use of two techniques.

The first is *node position repair*. This is done using step (1) of the layout algorithm (Sect. 3) to compute new position for the nodes which satisfies the placement and style constraints as well as the non-overlap and cluster containment constraints.

The second technique, which we call *rubber-banding*, is for repairing edge routes. The issue is that the route may have become invalid because it now passes through a graphic object or is no longer tight and so should be shortened by straightening and merging some adjacent segments. As much as possible we want to preserve the current route. Rubber-banding finds a new edge route by tracing the original connector path—object corner by object corner—until the destination object is reached. At all stages the connector acts like a rubber-band, fitting snugly around objects encountered so far on the route. The rubber-banding implementation uses the connector routing algorithm to dynamically route from the start object to the current object corner while preserving as

Table 2. Implementation of user actions providing live feedback during manipulation.

Dragging objects: Simply add terms to the goal function for each node v being manipulated of form $(y_v - y_d)^2 + (x_v - x_d)^2$ where (x_d, y_d) is the new desired position of node v.
Horizontal resizing of a node: The node to be resized is internally replaced by two artificial nodes which correspond to the left and right boundary edges of the original node's bounding box. Separation constraints couched in terms of these nodes are generated to maintain non-overlap between the bounding box and the other nodes. The width is changed by dragging the two artificial nodes to the required width, and updating the appearance of the node.
Vertical resizing of a node: Analogous to horizontal resizing.
Simultaneous vertical and horizontal resizing of a node: Achieved by resizing in small horizontal and vertical increments.
Tuning of goal function: The user can use sliders to change parameters of the goal function, such as the desired edge length.

much of the previous route as possible. More exactly, the last vertex in the route is removed from the route whenever the bend angle around the vertex becomes 180° or more, and routing proceeds from the preceding vertex.

Rubber-banding is also used for manual specification of connector routes. Connectors are typically created by specifying their start and end object, in which case automatic connector routing is used to determine a shortest-path route. However, the author is also free to specify the *topological* route of a connector. The author starts from an object and then threads the connector through the objects to the destination object with rubber-banding computing the route to the current cursor location. This is shown in Fig. 3.

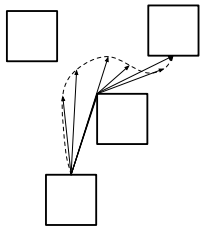


Fig. 3. Manual connector routing. The author "threads" the endpoint of the connector between the objects to specify the topological route for the connector.

The remaining user interactions are kinds of *direct manipulation* of the diagram. A strength of Dunnart is that the layout engine is fast enough to provide "live" feedback during direct manipulation. With live feedback, all objects and connectors in the diagram have their position and routing updated immediately in response to user manipulation. Direct manipulation is guaranteed not to change the topology of the layout. Details of the process—essentially achieved by performing step (3) of the layout algorithm with a modified P-stress goal function—are given in Table 2.

Clearly topology-preservation means that when dragging objects the author cannot move objects through connectors or other objects, since this changes the topology. This may make it difficult or impossible for the author to achieve their objective of, say, snapping an object to an alignment guideline because the alignment guideline keeps moving away from the object being dragged. For this reason, Dunnart allows the author to temporarily escape from continuous layout adjustment during object dragging by depressing a modifier key. This suspends any current layout activity and causes those objects not being directly

manipulated to maintain their current position. The user is now free to move objects through connectors and other objects or to add or remove an object from a container shape. This allows the user to quickly and easily modify the topology of the diagram.

Depressing the modifier key also breaks the selected objects free from placement and style constraints involving non-selected objects. Dunnart treats this as if the objects have been cut and pasted into their new location. The only difference is that connectors between the manipulated objects and non-manipulated objects are treated as new, automatically routed connectors.

4.3 Understanding constraints

Placement constraints are the primary method for the author to tailor the layout without having to explicitly position objects. The placement tool sets up a persistent relationship that is maintained in subsequent interaction until the author explicitly removes it rather than a once-off position adjustment. Dunnart provides standard placement tools: horizontal and vertical alignment and distribution, horizontal and vertical separation (sequencing) that keeps objects a minimum distance apart horizontally or vertically while preserving their relative ordering, and an "anchor" tool that allows the user to fix the current position of a selected object or set of objects.

Like most constraint-based diagramming tools, there is a graphical representation for each placement relation in the diagram. A potential usability issue for constraint-based layout tools that utilise such visual representations is that they clutter the diagram. To reduce clutter we have chosen to use an explicit visual representation only for user-created placement constraints and some style constraints but not for refinement constraints since the objects themselves and their behaviour during manipulation provide sufficient feedback. To further reduce clutter, the visual representation for constraints is by default very faded, leaving the actual diagram components clearly visible (see Fig. 2).

Another well-known usability issue of constraint-based layout tools is that users can find it difficult to understand interaction between the constraints. Immediate feedback during direct manipulation helps this considerably since it allows the author to quickly notice unexpected interaction between the objects being manipulated and other parts of the diagram. As a more sophisticated way to understand constraint interaction, Dunnart also provides a query tool dubbed "Information Mode." This tool finds the path of constraints between two objects and illustrates this to the user by highlighting the relevant constraint indicators.

The extreme kind of unexpected interaction between constraints is when the author tries to perform an action which will give rise to inconsistent constraints. For instance: the author may try to add a downward pointing connector which creates a cycle of downward edges; try to apply a placement tool which gives rise to an inconsistent constraint; or use the modifier key to move an object to an infeasible position. To allow the author to understand the problem, Dunnart highlights the placement and style constraints and objects associated with the subset of separation constraints causing the inconsistency.

Table 3. Indicative running times on an average (Dual Core 2GHz) PC for various sized randomly generated directed networks with flow style. For each graph we give the number of nodes and edges. Note that the number of separation constraints imposing downward edges is $|E|$. We give the time to find (a) a feasible layout after adding a new alignment constraint; and (b) the average rate of layout updates during dragging of a random node and the time for the layout to converge following the movement.

(a) Feasibility

$\|V\|$	$\|E\|$	Feasibility repair (seconds)
59	62	0.19
105	117	0.84
156	167	1.96
230	276	5.16

(b) Direct manipulation

$\|V\|$	$\|E\|$	Layout frame rate (frames/sec)	Time to converge (seconds)
59	62	15.83	0.94
105	117	11.72	1.75
156	167	8.59	4.50
230	276	2.21	7.26

5 Performance

One of the most important requirements of Dunnart is that the layout algorithms are fast enough for interactive layout. Table 3(a) lists for network diagrams of various sizes the time taken to complete node position and edge routing repair after the addition of a new alignment constraint. Up to a few seconds are required to layout networks of around 250 nodes. We have found that the dominating cost of this process is finding the initial connector routing.

Perhaps more interesting, is the speed of topology-preserving layout adjustment, especially during direct manipulation. Table 3(b) shows the average number of layout updates per second while the user drags a random node slowly to the four corners of the screen and back to the centre. It also shows the time taken for the layout to converge, once the user has stopped dragging the object. As expected, because the layout optimisation algorithm generally starts from a solution close to the optimal solution it converges quite quickly, allowing real-time feedback during manipulation of graphs with up to 100–150 nodes. It is worth noting that layout occurs in separate thread so that Dunnart is still responsive while layout adjustment is taking place. Furthermore, layout adjustment typically finds a near optimal solution very rapidly, and the majority of time is spent moving nodes only very slightly.

6 Conclusion

We have described Dunnart, a new network diagram authoring tool that provides powerful automatic graph layout, yet still allows the user total layout flexibility. Topology preserving constrained graph layout provides predictable behaviour during editing and allows the author to use placement constraints to control and improve the layout.

The underlying graph layout engine is fast enough to provide live update of the layout during direct manipulation for networks with up to 100 nodes.

This is more than sufficient for the kind of diagrams that are typically created with interactive authoring tools. For larger networks we believe that a combination of fast layout techniques (for an overview layout) and topology preserving constrained graph layout (for the detailed view) is the right approach [6].

There are a number of extensions to Dunnart that we intend to investigate. One is orthogonal connector routing. We want to explore further use of Dunnart in particular application areas, such as biological networks and concept maps.

References

1. Böhringer, K.F., Paulisch, F.N.: Using constraints to achieve stability in automatic graph layout algorithms. In: CHI'90: Proceedings of the SIGCHI conference on Human Factors in Computing Systems. pp. 43–51. ACM Press (1990)
2. Brandes, U., Wagner, D.: A bayesian paradigm for dynamic graph layout. In: GD 1997. LNCS, vol. 1353, pp. 236–247. Springer (1998)
3. Bridgeman, S.S., Fanto, J., Garg, A., Tamassia, R., Vismara, L.: InteractiveGiotto: An algorithm for interactive orthogonal graph drawing. In: GD 1997. LNCS, vol. 1353, pp. 303–308. Springer (1998)
4. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall, Inc. (1999)
5. Dwyer, T., Koren, Y., Marriott, K.: IPSep-CoLa: An incremental procedure for separation constraint layout of graphs. IEEE Transactions on Visualization and Computer Graphics 12(5), 821–828 (2006)
6. Dwyer, T., Marriott, K., Schreiber, F., Stuckey, P.J., Woodward, M., Wybrow, M.: Exploration of networks using overview+detail with constraint-based cooperative layout. IEEE Transactions on Visualization and Computer Graphics (InfoVis 2008) To appear 2008
7. Dwyer, T., Marriott, K., Wybrow, M.: Topology preserving constrained graph layout. In: GD 2008. LNCS, Springer, to appear 2009
8. Gleicher, M.: Briar: A constraint-based drawing program. In: CHI'92: Proceedings of the SIGCHI conference on Human Factors in Computing Systems. pp. 661–662. ACM Press, New York (1992)
9. He, W., Marriott, K.: Constrained graph layout. Constraints 3, 289–314 (1998)
10. Huang, M.L., Eades, P., Lai, W.: Online visualization and navigation of global web structures. The International Journal of Software Engineering and Knowledge Engineering 13(1), 27–52 (2003)
11. do Nascimento, H.A.D., Eades, P.: User hints for directed graph drawing. In: GD 2001. pp. 205–219. Springer, London (2002)
12. Nelson, G.: Juno, a constraint-based graphics system. In: SIG-GRAPH 85 Conference Proceedings. ACM Press (1985)
13. Ryall, K., Marks, J., Shieber, S.M.: An interactive constraint-based system for drawing graphs. In: ACM Symposium on User Interface Software and Technology. pp. 97–104 (1997)
14. Sutherland, I.E.: Sketchpad: A Man-Machine Graphical Communication System. Ph.D. thesis, Massachusetts Institute of Technology (1963)
15. Wybrow, M., Marriott, K., Stuckey, P.J.: Incremental connector routing. In: GD 2005. LNCS, vol. 3843, pp. 446–457. Springer (2006)