Topology Preserving Constrained Graph Layout

Tim Dwyer, Kim Marriott, and Michael Wybrow

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

Abstract. Constrained graph layout is a recent generalisation of force-directed graph layout which allows constraints on node placement. We give a constrained graph layout algorithm that takes an initial feasible layout and improves it while preserving the topology of the initial layout. The algorithm supports poly-line connectors and clusters. During layout the connectors and cluster boundaries act like impervious rubber-bands which try to shrink in length. The intended application for our algorithm is dynamic graph layout, but it can also be used to improve layouts generated by other graph layout techniques.

1 Introduction

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 to preserve the topology of the current layout during layout changes. While topology preservation has been used for dynamic layout based on orthogonal graph layout, its use in force-directed approaches to dynamic layout is much less common.

Constrained graph layout [12,3,4] is a recent generalisation of the 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 placement constraints on the nodes. In this paper we detail a constrained graph layout algorithm that preserves the topology of the initial layout. The primary motivation for our development of this algorithm was to support dynamic layout but it can also be used to improve layouts generated by other graph layout techniques such as planarisation techniques [11].

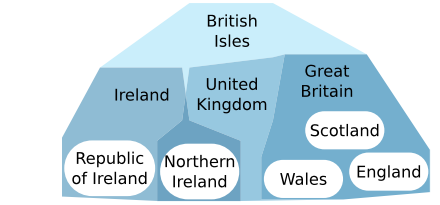
Our algorithm supports network diagrams with poly-line connectors and arbitrary node clusters. It ensures that the nodes do not overlap and that additional constraints on the layout—such as alignment and downward pointing edges—remain satisfied. During layout optimisation the *paths*, i.e poly-line connectors and cluster boundaries, act like rubber-bands, trying to shrink in length and hence, in the case of connectors, straighten. Like physical rubber bands, the paths are impervious and do not allow nodes and other

I.G. Tollis and M. Patrignani (Eds.): GD 2008, LNCS 5417, pp. 230–241, 2009.

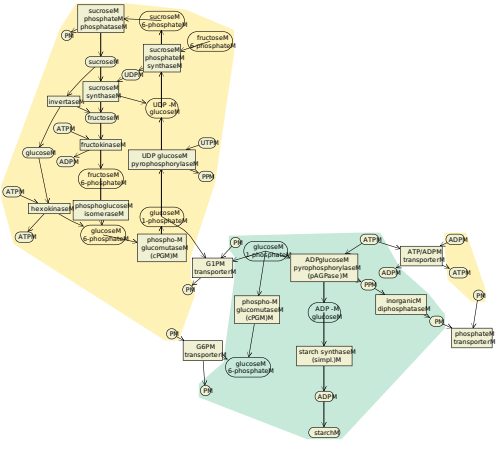
paths to pass through them. Thus, the initial layout topology is preserved. Figure 1 shows example layouts obtained with our algorithm.

Extending constrained graph layout to handle topology preservation is conceptually quite natural since topology preservation can be regarded as a kind of constraint. However, it was not possible to straightforwardly extend existing constrained graph layout algorithms to preserve topology. One issue is that previous algorithms were based on functional majorization whose use relied on particular properties of the stress goal function.

The main technical innovations in our new algorithm are fourfold. First, we utilise a new goal function, *P-stress*, that encodes the rubber band metaphor, measuring the stretch of paths as well as trying to place objects a minimum distance apart. Importantly, the *P-stress* is *bend-point invariant* in the sense that merging two consecutive collinear segments in a path does not change the value of the goal function. This aids convergence since it means that the goal function behaves continuously as paths change during optimisation. Second, we utilise gradient projection rather than functional majorization. This approach is generic in the choice of goal function and so can be used to minimise *P-stress*. Third, we give a novel algorithm for updating paths in a layout given that nodes are moved in a single dimension. This maintains the relative order of nodes and paths in that dimension and so preserves the initial topology. The final innovation is our uniform treatment of connector routes and cluster boundaries as impervious paths. This allows our algorithm to handle arbitrary clusters.



(a) Euler diagram



(b) Metabolic pathway

Fig. 1. Example layouts obtained with the topology preserving constrained graph layout algorithm. In the metabolic pathway, three vertical alignment constraints have been added to improve the layout.

The algorithm for topology preserving constrained graph layout given here 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 [5]. 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 [6]. The contribution of this paper is to detail the algorithm.

2 Related Work

There has been considerable interest in developing techniques for stable graph layout that preserve the user's mental model of the graph [14]. These techniques are quite specialised 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. [9]. Stable dynamic layout has also been studied for orthogonal graph layout, e.g. [2]. There, stability is preserved by trying to preserve the current bend points and angles. This has the effect of preserving the layout topology. Finally, in the case of Sugiyama-style layered layout stability is achieved by preserving the current horizontal and vertical ordering between nodes, e.g. [15]. 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.

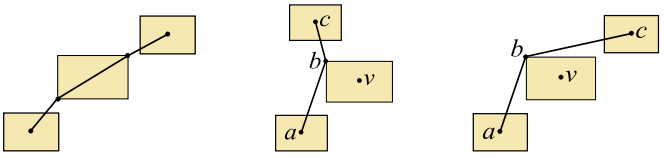
Orthogonal graph layout algorithms typically feature a refinement step that attempts to shorten edges while preserving edge crossing topology [8]. However, the approach is very specific to orthogonal drawings. Another method, [1], used a force directed approach but only handled abstract graphs with point nodes and straight-line edges. Most closely related is our earlier extension to constrained stress majorization that preserves layout topology while trying to straighten bends in poly-line connectors [7]. This works by introducing dummy nodes in each connector at all possible bend points and adding constraints to ensure a minimum separation between objects and bend-points. Unfortunately, our experience with this algorithm was that straightening bends sometimes meant that connector length was increased and that the algorithm did not scale to moderately sized networks because of the large number of dummy nodes. Even worse it did not always converge because the goal function was not bend-point invariant. The algorithm given here is considerably simpler, convergent and faster.

3 Problem Definition

A graph $G = (V, E, C)$ consists of a set of nodes V, a set of edges $E \subseteq V \times V$, and a set of node clusters $C \subseteq \wp V$. We let $width(v)$ and $height(v)$ give the width and height of the bounding rectangle, r_v, of each node $v \in V$.

A 2-D drawing of a graph is specified by a tuple (x, y, P) where (x_v, y_v) gives the centre position for each node $v \in V$ and P is a set of *paths* specifying the edge routings and cluster boundaries. A path is a piecewise linear path through a sequence of points $p_1, \ldots, p_k$ where each point is either the center or one of the corners of a node's bounding rectangle and represented by a pair (v, i) where $v \in V$ and $i \in \{Centre, TL, TR, BL, BR\})$. In the case of a path giving the routing for an edge $e = (s, t) \in E$, p_1 is the centre of node s and p_k the centre of node t while the other points are node bounding rectangle corners. In the case that the path is for a cluster boundary, all points must correspond to node bounding rectangle corners and $p_1 = p_k$.

Separation constraints are inequality or equality constraints over pairs of position variables in either the horizontal or vertical axes of the drawing, e.g. for a pair of nodes $u, v \in V$ we might define a separation constraint over their $x-$positions: $x_u + g \leq x_v$ where g specifies a minimum spacing between them.



(a) Invalid (b) Not tight (c) Feasible

Fig. 2. Example of incorrect (a,b) and correct (c) paths

A *feasible* drawing of a graph (see Fig. 2) is one in which:

- all separation constraints are satisfied;
- no two node rectangles overlap;
- the nodes inside the region defined by the boundary of each cluster c are exactly the nodes in c;
- every path $p \in P$ is *valid* and *tight*.

A *valid* path is one in which no segment passes *through* a node rectangle, except the first and last segments in a path corresponding to an edge which must terminate at the centre of rectangles as specified above. A *tight* path is one where every bend (described by three consecutive points a, b, c in the path) is wrapped around the rectangle r_v associated with the bend point $b = (v, i)$. That is, the points a, b, c in order must constitute a turn in the same direction as the points a, b, v in order, and the points b, c, v must also constitute a turn in the same direction.

A common strategy for finding aesthetically pleasing drawings of graphs is to define a cost function over the positions of the nodes and then to minimise this cost function by adjusting these positions. In our case we are also interested in the lengths of paths. Therefore, we use a novel cost function *P-stress* which also takes the paths P of the layout into consideration:

$$\sum_{u<v\in V} w_{uv}\left((d_{uv} - ||(x_u, y_u), (x_v, y_v)||)^+\right)^2 + \sum_{p\in P} w_p\left((||p|| - L_p)^+\right)^2$$

where $(z)^+$ is z if $z \geq 0$ and 0 otherwise and $w_p = \frac{1}{L_p^2}$, $w_{uv} = \frac{1}{d_{uv}^2}$.

The first component of *P-stress* is a modification of the *stress* function used in the stress majorization [10] and Kamada and Kawai [13] layout methods. This considers the ideal distance d_{uv} between each pair of nodes which is proportional to the graph theoretic distance, i.e. shortest path, between the nodes. 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 L_p. The ideal length of the route for an edge e is simply a fixed constant while the desired length of the boundary for cluster c is $2\sqrt{\pi \sum_{v\in c} width(v)height(v)}$ (i.e. the ideal length is proportional to the perimeter of the circle of the same area as that of the constituent nodes). This second component is purely attractive, otherwise minimising *P-stress* could potentially *increase* bends.

Note that *P-stress* is *bend-point invariant* in the sense that merging two consecutive collinear segments in a path does not change the *P-stress* of layout since the overall path length does not change. This is important for convergence of the layout algorithm.

4 Minimising P-Stress Using Gradient Projection

Our layout problem is, therefore, given a feasible layout for a graph to find a new layout that is feasible, has the same topology as the original layout, and which locally minimises *P-stress*. In this section we give an algorithm to do this. An example of its operation is shown in Fig. 3.

Our algorithm works by alternately adjusting horizontal and vertical positions of all nodes to incrementally reduce *P-stress*. This makes the computation of the new positions considerably simpler than if both dimensions were considered together. Constrained stress majorization [4] also uses a similar approach to reduce stress. However, the useful Cauchy-Schwarz based expansion of the stress function into horizontal and vertical quadratic forms which strictly (upper-)bound the goal function, is no longer easily derived for *P-stress*. Instead, at each iteration we use a quadratic approximation based on the second order Taylor series expansion of *P-stress* around the current horizontal position x and compute a descent vector $-g$ and step size α from the first and second derivatives of this quadratic to compute a new position d for the horizontal position variables. We then use the function *project-x* to project d onto the horizontal constraints necessary to avoid overlap and to preserve topology and any other user specified separation constraints, C_x, on the horizontal variables. Next we perform an analogous operation to compute a new position for the vertical position variables y. The high-level algorithm is thus:

procedure *gradient-projection-x*(x, y, P, C)
  $g \leftarrow \nabla_x$*P-stress*(x, y, P)
  $H \leftarrow \nabla^2_x$*P-stress*(x, y, P)
  $\alpha \leftarrow \frac{g^T g}{g^T H g}$
  $d \leftarrow x - \alpha g$
return *project-x*(x, y, P, d, C)

procedure *improve*(x, y, P, C_x, C_y)
  $(x', y', P') \leftarrow (x, y, P)$
  repeat
    $(x, P'') \leftarrow$ *gradient-projection-x*(x, y, P, C_x)
    $(y, P) \leftarrow$ *gradient-projection-y*(x, y, P'', C_y)
  until $|$*P-stress*$(x', y', P') -$ *P-stress*$(x, y, P)|$ sufficiently small
return (x, y, P)

Before giving details of projection we must make precise what we mean by topology preservation. Considering just the horizontal case, since the vertical is symmetrical, we say that a horizontal adjustment of the nodes from feasible layout L to feasible L' is *topology preserving* if no node or line segment moves through another node or line segment. More exactly, let M and M' be the layouts obtained from L and L', respectively, by infinitesimally reducing the height of each node's bounding rectangle and

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

Fig. 3. Example of how our layout algorithm improves the network layout by reducing *P-stress* (which shortens edge routes) while preserving the topology of the initial layout

appropriately modifying the paths. This means that rectangles whose top and bottom were aligned in the original layout now have a infinitesimal vertical separation between them. Then for any height h we must have that scanning left to right along the horizontal line $y = h$ encounters exactly the same sequence of edges, clusters and nodes in both M and M' where an edge is encountered whenever the line intersects a path segment for the edge, a cluster is encountered whenever the line intersects a path segment for its boundary and a node is encountered when the line intersects the node's bounding rectangle.

5 Topology Preserving Projection

The heart of the layout algorithm are the procedures *project-x* and *project-y* which perform a projection operation in the specified axis. We shall focus on *project-x*: procedure *project-y* is symmetric. The call *project-x*(x, y, P, d, C) returns a new x position and paths (x', P') s.t layout (x', y, P') is feasible and preserves the topology of (x, y, P) while ensuring x' is as close as possible to the desired position d. It has three main steps:

(1) Generate separation constraints C^{no} to ensure non-overlap of nodes and topology constraints TC to ensure topology preservation.
(2) Project d on to $SC = C \cup C^{no}$ giving $\bar{x}$. This is achieved by solving the quadratic program:
$$\min_{x} \sum_{vinV} (x_v - d_v)^2 \textit{ subject to } SC$$
(3) Update the path routing P to give P' by moving the nodes smoothly from x to $\bar{x}$ appropriately adjusting the paths as the nodes move in order to satisfy the topology constraints TC.

In Step 2 of *project-x* we solve the quadratic program using the incremental active-set procedure *solveQPSC* given in [4]. Like most active-set methods it is difficult to prove that this has polynomial running time, but in practice it is very fast, as indicated by our experimental results. We now look at Steps 1 and 3 in more detail.

Non-overlap and topological constraints are generated for a horizontal move using a top-to-bottom scan of the drawing. At each step we keep the list of currently open

node bounding rectangles and path line segments. To do so we process the vertical opening and closings of each rectangle OR, CR and line segment OS, CS of the given routing in order from top to bottom and, when two such events occur at the same vertical position, then with precedence:

- OS before CS so that horizontal segments are handled properly
- CR before OR to avoid unnecessary non-overlap constraints (assuming no zero height rectangles)
- CS before OR, CR before OS, OS before OR, and CR before CS to ensure all possible segment/rectangle interactions are considered.

For each rectangle opening (i.e. the top of each rectangle) we add to C^{no} a separation constraint between the rectangle and its immediate left and right neighbours in the list of open rectangles at that y-position (the scan position). Each separation constraint has the form $x_u + s \leq x_v$ over the x positions of nodes u and v and preserves the relative horizontal ordering of u and v and prevents the nodes from overlapping, where $s = (width(u) + width(v))/2$.

The scan also generates *topology* constraints between nodes and paths which ensure that the paths remain tight and valid. There are two types of topology constraints: *straight* constraints—between a node w and a path segment uv which ensures that the path remains valid, i.e. the node does not overlap the path segment and *bend* constraints associated with a bend point between two consecutive line segments uv and vw which ensures that the path remains tight around the bend point v.

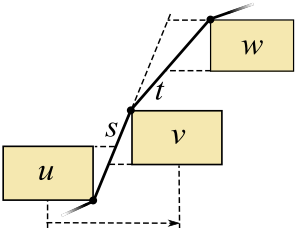


Fig. 4. Constraints generated during a vertical scan. There is one separation constraint $x_u + \frac{1}{2}(width(u) + width(v)) \leq x_v$ to prevent overlap, three bend constraints (the construction for the constraint ensuring the path remains tight around v is shown) and three straight constraints at the places where the segments s and t may potentially bend.

Both kinds of topology constraint give rise to a linear inequality over the three variables corresponding to u, v and w enforcing that the rectangle r_w associated with node w must be to the right or left of a line between the corners of two nodes u and v. We write this in the standard form $x_w + g \oplus x_u + p(x_v - x_u)$ where $\oplus$ is either $\leq$ or $\geq$. For straight constraints $0 < p \leq 1$ while for bend constraints $p > 1$. For instance, in the case of the bend constraint enforcing that the path remains tight around the bend point v in Fig. 4 we have that

$$x_{w^{TL}} \geq x_{u^{BR}} + \frac{y_{w^{TL}} - y_{u^{BR}}}{y_{v^{TL}} - y_{u^{BR}}}(x_{v^{TL}} - x_{u^{BR}})$$

where $x_{w^{TL}} = x_w - width(w)/2$ etc. This can be rewritten into the standard form. The procedures for creating each type of constraint is given in Fig. 5.

If $|P|$ denotes the number of path segments, the worst case complexity of Step 1 of *project-x* is $O(|V|(|P| + \log|V|))$ and up to $O(|V|)$ non-overlap constraints and $O(|P||V|)$ topological constraints can be generated.

We now consider Step 3 of *project-x*. This is performed by procedure *move* (Fig. 6). This updates the paths by moving the nodes horizontally from the initial feasible solution

procedure *createStraightConstraint*(s, w, y, TC)
  % for segment $s = uv$ and node w at scan pos y
  $p \leftarrow (y - y_u)/(y_v - y_u)$
  $x_p \leftarrow x_u + p(x_v - x_u)$
  $leftOf \leftarrow x_w < x_p$
  $corner \leftarrow$ **if** $y < y_w$ **then if** *leftOf* **then** BR **else** BL
    else if *leftOf* **then** TR **else** TL
  $offset(w) \leftarrow width(w)/2$ (-ve if *leftOf*)
  $g \leftarrow offset(u) + p(offset(v) - offset(u)) - offset(w)$
  $TC \leftarrow TC \cup \{TopologyConstraint(straight, u, v, w, p, g, leftOf)\}$
procedure *createBendConstraint*(b, TC)
  % for bend point $b = (v, i)$, between segments ab and bc
  if i is the centre of v **then return**
  if existing bend constraint t on b **then** remove t
  $leftOf \leftarrow i \in \{TR, BR\}$
  if $|y_a - y_b| > |y_b - y_c|$ **then**
    $p \leftarrow (y_c - y_a)/(y_b - y_a)$
    $g \leftarrow offset(a) + p(offset(b) - offset(a)) - offset(c)$
    $t \leftarrow TopologyConstraint(bend, a, b, c, p, g, leftOf)$
  else
    $p \leftarrow (y_a - y_c)/(y_b - y_c)$
    $g \leftarrow offset(c) + p(offset(b) - offset(c)) - offset(a)$
    $t \leftarrow TopologyConstraint(bend, c, b, a, p, g, leftOf)$
  $TC \leftarrow TC \cup \{t\}$

Fig. 5. The procedures for creating straight constraints and bend constraints are used in both the initial scan to set up topology constraints and by the procedure *satisfy* (Fig. 6). The function *TopologyConstraint* creates a constraint of the form $x_w + g \leq x_u + p(x_v - x_u)$ if leftOf (or $\geq$ otherwise).

x for which the routing is correct towards $\bar{x}$ detecting violated topology constraints as they move. A violated *bend* constraint indicates that consecutive segments have become aligned and can be replaced with a single segment. A violated *straight* constraint indicates that a single segment needs be split into two new segments with a new bend point.

The maximum horizontal move γ that can be made along the line $x = a + \gamma(b - a)$ from a to b without violating topology constraint t is determined by solving the linear equation associated with the constraint. For example, if t is the constraint $x_w + g \leq x_u + p(x_v - x_u)$ then the maximum safe move is obtained by substituting $x_i = a_i + \gamma(b_i - a_i)$ for each node i and solving for γ:

$$\gamma = \frac{\alpha}{\beta} = \frac{a_w - g - a_u + p(a_u - a_v)}{b_u - a_u + p(a_u - b_u + b_v - a_v) + a_w - b_w}$$

The iterative process of finding the next such constraint and updating the paths P is accomplished in the *move* procedure, Fig. 6.

Note that the *satisfy* procedure shown in Fig. 6, which satisfies a topology constraint by either merging or splitting segments, must transfer or replace other bend and straight

procedure $satisfy(t, TC, P)$
  $TC \leftarrow TC \setminus \{t\}$
  if t is a *bend constraint* over points a, b, c **then**
    % $b = (v, i)$ is the bend point of t
    replace segments ab and bc in P with new segment ac
    $createStraightConstraint(ac, v, b_y)$
  else % t is a *straight constraint* over u, v, w
    replace segment uw in P with segments uv and vw
    transfer straight constraints on uw to either uv or vw
    $createBendConstraint(u)$
    $createBendConstraint(v)$
procedure $move(x, \bar{x}, TC, P)$
  repeat
    $\alpha \leftarrow \beta \leftarrow 1$
    $t* \leftarrow$ None
    for $t \in TC$
      % t is a Topology Constraint over u, v, w with constants p, g
      $a \leftarrow a_w - g - a_u + p(a_u - a_v)$
      $b \leftarrow b_u - a_u + p(a_u - b_u + b_v - a_v) + a_w - b_w$
      if $a\beta < \alpha b$ **then**
        $\alpha \leftarrow a, \beta \leftarrow b, t* \leftarrow t$
    $x \leftarrow x + \frac{\alpha}{\beta}(\bar{x} - x)$
    if $t* \neq$ None **then** $satisfy(t, TC, P)$
  until $\frac{\alpha}{\beta} = 1$

Fig. 6. The procedure $satisfy(t, TC, P)$ satisfies a topology constraint $t \in TC$ that is at equality, by modifying P with a *valid* and *tight* system of segments. Procedure $move(x, \bar{x}, TC, P)$ updates the path P by moving nodes in one dimension from position x to $\bar{x}$ to satisfy the topology constraints TC.

constraints associated with the affected segments. The detail is not shown, but an example of the difficult edge case of a horizontal path segment is shown in Fig. 7.

The *move* procedure used for updating the paths to preserve validity and topology can also be thought of as a kind of active-set process, and as such it is difficult to prove that it is polynomial. Again, however, please see our results section for actual running times which indicate that running times scale fairly well with the number of topology constraints generated. Note that the number of bend constraints is exactly the number of bend points in P, and the number of straight constraints—while the worst case is $O(|P||V|)$—is limited by only generating constraints for segments which are *visible* in the axis of movement from a given rectangle open/close.

Theorem 1. *Let (x, y, P) be a feasible layout with respect to the separation constraints C_x and C_y in the x and y dimensions, respectively. Then* project-x(x, y, P, d, C_x) *returns a new x position and paths (x', P') s.t layout (x', y, P') is feasible and preserves the topology of (x, y, P) while ensuring x' is as close as possible to the desired position d.*

Proof. (Sketch) Any feasible and topology preserving layout must satisfy $SC = C_x \cup C^{no}$. Step 2 ensures that x' is the projection of d onto SC, so it is the closest node

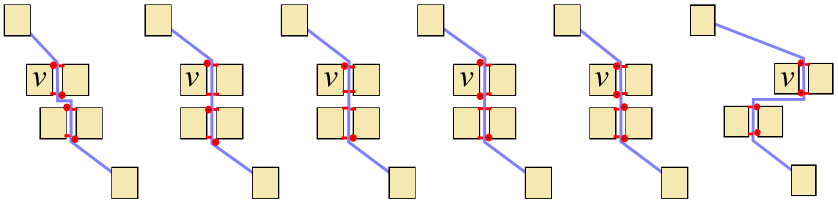


Fig. 7. The result of each iteration of *move* is shown for a path with a horizontal segment. The iterations progress from left to right. The node v is required to move to the right relative to the other nodes. The four central nodes are shown slightly separated for clarity, but we assume that the boundaries of these nodes are actually touching—hence creating, initially, a horizontal segment. The small circles represent bend points, while the '-'s represent straight topology constraints. Note that, to properly preserve topology as the segments are split to satisfy a straight constraint, the remaining straight constraints must be transferred to the correct sub-segments.

position that satisfies SC. Furthermore, one can prove by induction that the *satisfy* procedure returns updated paths P' that are topology preserving, tight and valid.

6 Finding a Feasible Topology

We can apply our topology preserving layout adjustment to a layout obtained by any graph drawing algorithm, assuming the generated layout is feasible as defined in §3. Although not the primary focus of this paper we have also developed an algorithm to find an initial feasible layout. This has two main steps:

(1) Perform standard stress majorization to find an initial position for the nodes. A position for the nodes satisfying the constraints is found by projecting this position on to the user specified separation constraints and then using a greedy heuristic to satisfy the non-overlap constraints and cluster containment constraints. We use the approach sketched in [4].

(2) Edge routing is performed using the incremental poly-line connector routing library `libavoid` [16] to compute poly-line routes for each edge, which minimise edge length and amount of bend. An initial cluster boundary is obtained by taking the convex hull of the nodes in the cluster.

We note that the edge routing library has been extended to handle clusters and finds routes for edges that do not unnecessarily pass through clusters. It also performs "nudging" on the final routes to separate paths with shared sub-routes.

7 Experimental Results

Table 1 gives some indicative run-times on various size graphs for finding an initial layout using the two-step algorithm given above, then using the topology-preserving constrained graph layout algorithm to find a locally optimal layout. The topology-preserving constrained graph layout algorithm[1] is quite fast with less than two seconds

[1] Implemented as part of the Adaptagrams project. `http://adaptagrams.sf.net/`

Table 1. Indicative running times for layout on an average (1GHz) PC for various size randomly generated directed networks with constraints imposing downward pointing edges. All times are in seconds.

$\|V\|$	$\|E\|$	Feasible layout		Optimise	Total
		Step 1	Step 2		
49	51	0.08	0.11	0.06	0.17
93	105	0.22	0.50	0.24	0.74
128	144	0.51	1.02	0.55	1.57
144	156	0.92	1.31	0.45	1.76
169	195	0.83	1.97	0.82	2.79
199	238	1.31	2.94	1.45	4.39
343	487	2.65	13.94	1.89	15.83

For each graph we give the number of nodes and edges. The number of separation constraints imposing downward edges is $|E|$. We give the time to find an initial feasible layout (Step 1 and Step 2) from a random starting configuration; and then to optimise the result using the topology preserving constrained graph layout algorithm. Optimisation algorithms were set to terminate when the change in P-stress or stress was $< 10^{-5}$.

required to layout networks of around 350 nodes. We have found that the main cost for each iteration is computation of the descent vector and step size. We also note that our experience with the algorithm in interactive applications is that it provides real-time updating of layout for graphs with up to 100 nodes.

Computing an initial layout is more expensive, and the dominating cost in finding the initial layout is finding the initial connector routing.

8 Conclusion

We have presented a constrained graph layout algorithm that preserves the topology of the initial layout. It supports network diagrams with poly-line connectors and arbitrary node clusters. It ensures that nodes do not overlap and that additional placement constraints on the layout remain satisfied. The algorithm is fast enough to support real-time layout of networks with up to 100 nodes in two dynamic graph layout applications we have developed: a network diagram authoring tool and a network diagram browser. While the primary motivation for our development of the algorithm was to support dynamic layout it can also be used to improve layouts generated by other graph layout techniques.

One of the strengths of the algorithm is that it can be straightforwardly modified to work with other goal function, so long as the second derivative is computable and the goal function is bend-point invariant. We plan to explore other goal functions. We also plan to explore generalising the algorithm to handle arbitrary linear constraints, not only separation constraints. As part of this we plan to modify the algorithm to perform minimization in both dimensions at once, rather than separately.

References

1. Bertault, F.: A force-directed algorithm that preserves edge crossing properties. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 351–358. Springer, Heidelberg (1999)
2. Bridgeman, S.S., Fanto, J., Garg, A., Tamassia, R., Vismara, L.: InteractiveGiotto: An algorithm for interactive orthogonal graph drawing. In: DiBattista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 303–308. Springer, Heidelberg (1997)

3. Dwyer, T., Koren, Y., Marriott, K.: Drawing directed graphs using quadratic programming. IEEE Transactions on Visualization and Computer Graphics 12(4), 536–548 (2006)
4. 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)
5. Dwyer, T., Marriott, K., Wybrow, M.: Dunnart: A constraint-based network diagram authoring tool. In: GD 2008. LNCS, vol. 5417. Springer, Heidelberg (to appear, 2009)
6. Dwyer, T., Marriott, K., 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.: Integrating edge routing into force-directed layout. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 8–19. Springer, Heidelberg (2007)
8. Eiglsperger, M., Fekete, S.P., Klau, G.W.: Drawing Graphs: Methods and Models, chap. Orthogonal graph drawing, pp. 121–171. Springer, London (2001)
9. Frishman, Y., Tal, A.: Online dynamic graph drawing. In: Eurographics/IEEE-VGTC Symp. on Visualization. Eurographics Association (2007)
10. Gansner, E., Koren, Y., North, S.: Graph drawing by stress majorization. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 239–250. Springer, Heidelberg (2005)
11. Gutwenger, C., Mutzel, P., Weiskircher, R.: Inserting an edge into a planar graph. In: SODA 2001: Proc. of the 12th Annual ACM-SIAM Symp. on Discrete Algorithms, pp. 246–255. Society for Industrial and Applied Mathematics (2001)
12. He, W., Marriott, K.: Constrained graph layout. Constraints 3, 289–314 (1998)
13. Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Information Processing Letters 31, 7–15 (1989)
14. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. Journal of Visual Languages and Computing 6(2), 183–210 (1995)
15. North, S.C., Woodhull, G.: Online hierarchical graph drawing. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 232–246. Springer, Heidelberg (2002)
16. Wybrow, M., Marriott, K., Stuckey, P.J.: Incremental connector routing. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 446–457. Springer, Heidelberg (2006)