Eurographics/ IEEE-VGTC Symposium on Visualization 2009
H.-C. Hege, I. Hotz, and T. Munzner
(Guest Editors)

Volume 28 (2009), Number 3

Scalable, Versatile and Simple Constrained Graph Layout

Tim Dwyer[†] [1]

[1] Microsoft Research, Redmond, USA

Abstract

We describe a new technique for graph layout subject to constraints. Compared to previous techniques the proposed method is much faster and scalable to much larger graphs. For a graph with n nodes, m edges and c constraints it computes incremental layout in time $O(n \log n + m + c)$ per iteration. Also, it supports a much more powerful class of constraint: inequalities or equalities over the Euclidean distance between nodes. We demonstrate the power of this technique by application to a number of diagramming conventions which previous constrained graph layout methods could not support. Further, the constraint-satisfaction method—inspired by recent work in position-based dynamics—is far simpler to implement than previous methods.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Display algorithms—Optimization [G.1.6]: Constrained optimization—

1. Introduction

Automatic layout of network (or graph) node-link diagrams is usually tackled as an optimization problem. Some objective function is defined over the positions of nodes (and possibly over the routing of edges) based on various aesthetic criteria, such as the optimal length of edges or the number of crossings between them. A configuration of nodes and edges that corresponds to a minimum in this objective function should therefore correspond to an “optimal” layout, at least with respect to the aesthetic criteria that the goal function was designed to capture. There are two fairly obvious limitations of such optimization. First, many commonly considered graph drawing aesthetics—such as minimizing crossings—give rise to NP-hard optimization problems, so non-optimal heuristics must be used to give a “good enough" solution. Second, algorithm designers cannot possibly anticipate all the possible layout requirements of end users. Aesthetics which may seem reasonable for abstract graphs are often less important to users than their own application-specific drawing conventions.

Recent work has recast the layout problem as one of continuous, incremental layout subject to user-defined constraints over node positions. This approach addresses the two problems above by: (1) providing users with interactive control over the layout so that they can guide it out of obvious local minima (corresponding to suboptimal layout); and (2) allowing users to achieve layout customized for their specific application or diagram. Dwyer et al. [DKM06] use quadratic programming techniques in the context of force-directed layout to support a simple class of equality or inequality constraints over pairs of either x- or y-position variables:

$$x_i + a \leq x_j, \quad y_i + b \leq y_j \tag{1}$$

These mean, respectively, that node i is required to be at least a (units) to the left of node j and at least b below y_j. Although simple, these so-called “separation constraints” can be combined into systems of more high-level user-defined *placement* constraints or automatically generated *style* constraints. Examples of such placement and style constraints [DMW09a] are:

- Horizontal or vertical alignment of nodes
- Non-overlapping rectangular node boundaries
- Containment of nodes within a page boundary
- Variable-sized hierarchical rectangular group boundaries
- Downward pointing directed edges

There are two problems with this technique however:

1. The approach treats the horizontal and vertical axes separately, so that the separation constraints are over pairs of x- or y-position variables exclusively. Arbitrary linear

† t-tdwyer@microsoft.com

constraints combining x- and y-position variables are not possible, nor are non-linear constraints such as circles.

2. The approach does not scale well to large graphs with thousands of nodes or constraints, with both the optimization and constraint satisfaction steps requiring at least quadratic time complexity.

In some respects these limitations are due to mathematical rigor. All the methods in the separation constraint layout family are provably convergent to stable local minima. However, it is not clear that such rigor is necessary simply to obtain an aesthetically appealing layout. On the other hand, it is clear that users have diagramming conventions that are not always neatly captured by this limited class of constraint and that the networks they want to visualize may easily have more than just a hundred or so nodes.

A parallel field that has made great advances in recent years by eschewing mathematical rigor in favor of very fast (if more ad-hoc) methods, is computer game character animation. For example, position-based dynamics approaches [MHHR06] use a very simple scheme of iteratively solving simple one-degree of freedom constraints (at each step clobbering the result of any previous constraint satisfaction), and by a miracle routinely attributed to either Jacobi or Gauss-Seidel the method usually converges to a stable state in very few iterations. Note that Gauss-Seidel and Jacobi methods are usually associated with unconstrained optimization. As of this writing the authors are unaware of formal proofs of correctness or convergence for so called cyclic-coordinate relaxation methods for the class of constraints often used in character animation.

In this paper we introduce a method for constrained layout based on a state-of-the-art force-directed layout approach combined with a simple constraint relaxation scheme inspired by position-based dynamics methods. This allows us to perform fast, scalable layout subject to a new, more general, class of constraint. The new class of constraint—a separation constraint over Euclidean distance or distance projected on an arbitrary direction vector—is able to reproduce all of the placement and style constraints described by earlier work and is also able to provide a host of new placement constraints. In this paper we demonstrate three novel types of constraint:

- unoriented or fixed (arbitrary) orientation linear alignment;
- circular constraints, e.g. for drawing cycles in directed graphs;
- non-overlap constraints for nodes or clusters with boundaries that are circular, rectangular, capsule-shaped or arbitrary convex hulls.

2. A simple and versatile class of layout constraint

The basic building block of our constraint layout system is a very simple class of constraint over the Euclidean distance between the 2D positions $\mathbf{p}$ and $\mathbf{q}$ of two nodes:

$$|\mathbf{p}-\mathbf{q}| = d \tag{2}$$

We can also handle inequality constraints ($\leq$,$\geq$) which specify a minimum or maximum distance allowed between the two nodes. As reviewed in Section 6 such constraints appear in cloth simulation and rigid body dynamics, such as computer game animation. We also allow distance constraints oriented with some arbitrary unit-length direction vector $\mathbf{v}$ of the form:

$$|(\mathbf{p}-\mathbf{q})\cdot\mathbf{v}|(=,\leq,\geq)d \tag{3}$$

3. Performing layout subject to constraints

Our basic layout method is a force-directed approach using a fast-multipole [Lau07] approximation of long-range repulsive forces. This can be viewed as a kind of gradient-descent optimization where at each iteration nodes are moved according to a descent vector based on the gradient of the potential energy function. After unconstrained movement in the descent direction constraints must be satisfied using a *projection* operation. Before explaining the method we use to project against constraints of the form (2), we briefly review how constraints of the form (1) were handled by earlier constrained graph layout techniques.

In [DMW09b] *gradient projection* is used for layout subject to simple horizontal or vertical separation constraints like (1). From a given starting configuration (x,y), the gradient of the goal function f with respect to the x-position variables is found giving a descent vector $-\nabla f_x$. An unconstrained step in the descent direction is taken $x' = x - \alpha\nabla f_x$ with the step size α chosen to ensure $f(x') \leq f(x)$. A projection step is then applied to x', i.e. the solution of a least squares problem subject to the *horizontal* separation constraints, to obtain positions $\bar{x'}$ that are *feasible* with respect to the constraints. It is important that $\bar{x'}$ corresponds to the smallest possible move from x' required to satisfy the constraints. This ensures that $f(\bar{x'}) \leq f(x)$ (assuming x was also feasible) and therefore that the layout converges. This process is then repeated for the y-position variables and *vertical* separation constraints, and repeated again, alternating between axes, until some convergence criteria are met. The simple structure of the horizontal or vertical separation constraints—and the fact that constraints on separate axes do not interact—means that efficient techniques can be used to solve the constrained least squares problem over each axis exactly in worst case $O((n+c)^2)$ for n variables and c constraints.

In performing gradient-projection optimization subject to the new class of Euclidean distance constraint (2), we abandon the axis-separation approach, and at the start of each iteration, take an unconstrained step in a descent direction

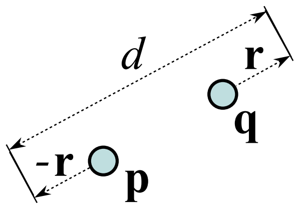


Figure 1: *Projection of the constraint $|\boldsymbol{p}-\boldsymbol{q}| = d$. The smallest position delta required to satisfy the constraint is $\boldsymbol{r}$.*

in 2D coordinates and then satisfy the constraints via projection. Unfortunately, projection of a system of Euclidean distance constraints over 2D coordinates is much more difficult to solve exactly than the 1D projection over separation constraints described above. The square root required to compute the norm is non-linear and the boundary of many such constraints taken together may not be convex. However, a very simple, naïve and yet effective heuristic is proposed by Jakobsen [Jak01] for performing computer-game skeletal animation of characters, where the "bones" are modeled by constraints like (1). The two ends of a single constraint are moved minimally (projected) to satisfy the constraints. That is, the smallest $\mathbf{r}$ is found such that:

$$|(\mathbf{p}-\mathbf{r})-(\mathbf{q}+\mathbf{r})| = d$$

It is easy to see (from Fig. 1) that the smallest such $\mathbf{r}$ must lie along the line $\mathbf{pq}$, and therefore:

$$\mathbf{r} = \frac{(d-|\mathbf{p}-\mathbf{q}|)\mathbf{pq}}{2|\mathbf{p}-\mathbf{q}|} \tag{4}$$

Each constraint in the system is projected in turn, cycling m times. For inequality constraints, we need only project if the inequality is violated at the current position. Oriented constraints of the form (3) are projected in the same way, except that direction of projection is $\mathbf{v}$ instead of $\mathbf{pq}$. Surprisingly, although solving each constraint by projection can potentially undo progress made by an earlier projection, these methods seem to converge quite reliably after cycling over all constraints relatively few times (see Fig. 2), e.g. $m = 10$ gives quite rigid constraint satisfaction.

In the context of position-based dynamics, Müller et al. [MHHR06] couch stable satisfaction of distance constraints such as (2) in terms of conservation of linear and angular momentum. However, the process above can also be viewed as a kind of gradient projection to optimize a goal function subject to constraints. That is, starting from a feasible configuration (x,y) with potential energy $f(x,y)$, we take a step in the steepest descent direction $-\alpha\nabla f(x,y)$ to obtain (x',y'). The stepsize α is found through the simple trust-region heuristic of Hu [Hu05]. We then attempt to find a feasible $(\bar{x'},\bar{y'})$ as close as possible to (x',y'). For convergence we require that $f(\bar{x'},\bar{y'}) \le f(x,y)$. To strictly prove convergence

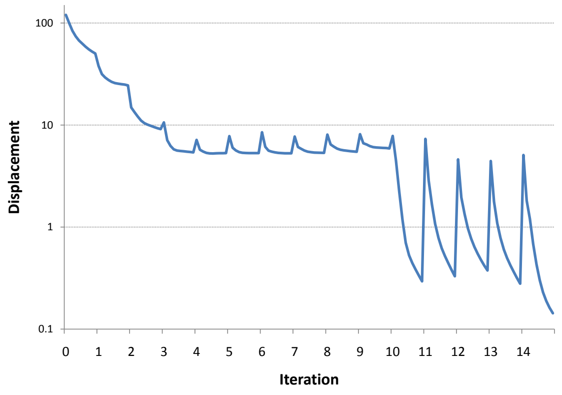


Figure 2: *The total displacement of nodes due to projection of all constraints for the first 15 iterations of layout for the citric acid cycle graph (Fig. 4), with $m = 10$ iterations of cyclically satisfying constraints for each layout iteration. Each "spike" corresponds to the increase in error on constraints after taking an unconstrained step in a descent direction of the energy function which is rapidly reduced as the constraints are projected. Note that the units of displacement are proportional to the ideal edge length and that the Displacement axis is log-scale. For the first 10 iterations the circular cycle constraint is rotating, hence the "step-down" in displacement once it is oriented such that the edges connecting it to the rest of the graph are close to their ideal lengths.*

we need to show that the method always makes progress towards a solution where the KKT conditions [NW06, 321] are met. Further analysis of convergence is beyond the scope of this paper, but the plot of displacement versus constraint projection iteration in Figure 2 gives an indication of the behavior of the iterative relaxation method in practice.

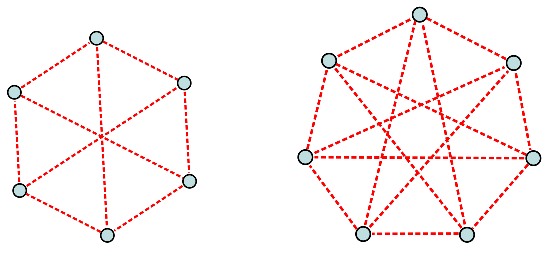
In summary, layout subject to constraints proceeds as follows:

1. compute steepest descent direction $-\nabla f(x,y)$
2. compute trust-region step-size α
3. move all nodes by $-\alpha\nabla f(x,y)$
4. repeat m constraint-satisfaction iterations:
   - project each constraint
5. repeat from Step 1, until convergence or maximum number of layout iterations.

4. Applications of Distance Constraints in Graph Layout

4.1. Fixed Position Constraints

In interactive applications with dynamic layout it is very useful to allow the users to "pin" or fix the positions of particular nodes. When manipulating nodes directly with the mouse the node should be positioned exactly at the position of the



(a) Even number of nodes (b) Odd number of nodes

Figure 3: *Construction of distance constraints to position nodes equidistantly around a circle.*

mouse cursor. Providing such fixed position constraints in conventional force-directed layout approaches is trivial. We simply disregard any forces on nodes that are fixed. However, when such nodes are involved in constraints we want the projection operation to be applied only to the free end of the constraint and we want the position found for the free end to completely satisfy the constraint. To achieve this (and also to facilitate cluster overlap resolution, see 4.5) we use a weighted form of (4):

$$\mathbf{r}_p = \frac{w_q(d - |\mathbf{p}-\mathbf{q}|)\mathbf{pq}}{(w_p + w_q)|\mathbf{p}-\mathbf{q}|} \tag{5}$$

where $\mathbf{r}_p$ is the displacement of node p due to the projection of the constraint constraint $|\mathbf{p}-\mathbf{q}| = d$ and where w_p and w_q are the weights of nodes p and q respectively. The displacement of q is computed symmetrically. By default all nodes have unit weight. If a node's position is fixed then we assign it a very large weight (several orders of magnitude larger than that of unfixed nodes), and hence its displacement as computed by (5) will be negligible.

4.2. Oriented Directed Edges

In [DKM06] separation constraints over y-position variables are used to draw directed acyclic graphs with strictly downward pointing edges. A constraint of the form: $y_i + g_{ij} \leq y_j$ is defined over each directed edge from node j to node i so that node j must be at least g_{ij} above node i.

We can achieve the same type of vertical constraint using oriented distance constraints of the form (3), taking $v = (0,1)$. Edges oriented left-to-right or at any other angle are just as easy.

4.3. Circular cycles

The most commonly used technique for drawing directed graphs is the layered layout method of Sugiyama [STT81]. It handles graphs with directed cycles by trying to find a minimal set of edges to reverse in order to make the graph acyclic. Unfortunately, this tends to hide the cycles, often resulting in drawings with very long upward pointing edges.

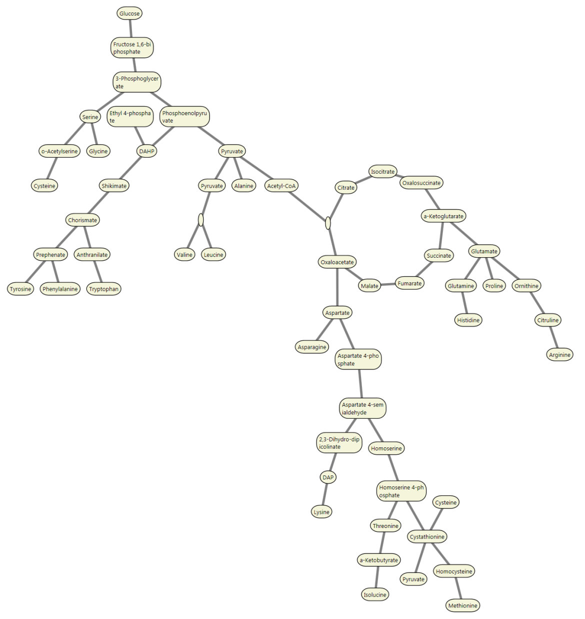


Figure 4: *A metabolic pathway using constraints of the form (3) to require directed edges (not involved in a cycle) to point downwards and using a wheel constraint to make the cycle circular.*

As discussed in [DMS*08], when drawing directed graphs with oriented distance constraints, we can simply leave edges involved in cycles unconstrained so that an orientation is not forced on the edge where there is no real precedence in the graph structure. However, cycles in process charts or biological pathways may be very important features and diagram authors may want to highlight them. In many applications the convention for drawing such cycles is to arrange them equidistantly around the perimeter of a circle.

Such an arrangement is easy to construct with a system of rigid distance constraints as in Figure 3. We use a "wheel" type arrangement. The constraints on the outer "rim" keep the nodes spaced equidistantly, and the "struts" keep the wheel rigid. Note that the required lengths for each constraint must be computed exactly from the chord lengths of the inscribed circle or the system will be unsatisfiable resulting, for example, in a perpetual motion machine. For a circle of n nodes, if n is even $n + \frac{n}{2}$ constraints are required (see Fig. 3(a)) and if n is odd then $3n$ constraints are required (see Fig. 3(b)).

Figure 4 shows a metabolic pathway showing conversion of citrate in the citric acid cycle into α-KG in mitochondria (see `www.uky.edu/~dhild/biochem/24/lect24.html`).

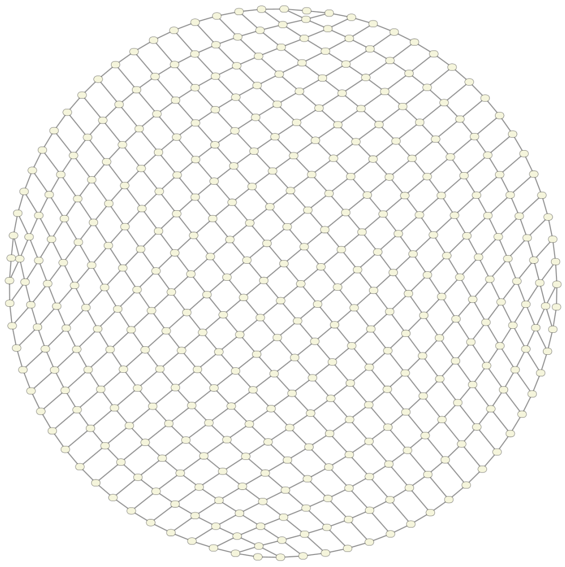


Figure 5: *A 400 node lattice with the outside nodes constrained to a circle.*

4.4. Non-overlap of Simple Node Boundaries

Two nodes p and q with circular boundaries of radii r_p and r_q can be prevented from overlapping with a constraint of the form:

$$|\mathbf{p}-\mathbf{q}| \geq r_p + r_q \tag{6}$$

Rectangular node boundaries of dimensions $\text{width}_p \times \text{height}_p, \text{width}_q \times \text{height}_q$ can be prevented from overlapping by generating either a horizontally or vertically aligned distance constraint, ie:

$$|(\mathbf{p}-\mathbf{q})\cdot(1,0)| \geq (\text{width}_p + \text{width}_q)/2$$

or:

$$|(\mathbf{p}-\mathbf{q})\cdot(0,1)| \geq (\text{height}_p + \text{height}_q)/2$$

depending on which requires smaller displacement. Rounded corner rectangles such as those used in Fig. 4 can be handled by a straightforward combination of the above.

When finding an initial layout such non-overlap constraints would not be applied since they may prevent the graph from "untangling". Rather, they would be used in a separate refinement phase. Assuming there are fewer than $O(n)$ overlaps left after the initial layout (which is reasonable if the ideal edge length is proportional to the average node size), a minimal but sufficient set of non-overlap constraints can be generated for each layout iteration in $O(n \log n)$ time using an efficient region query data structure such as a quad- or KD-tree. In practice, the same KD-tree data structure used in the multipole-force approximation [Lau07] can be used to detect overlapping node boundaries.

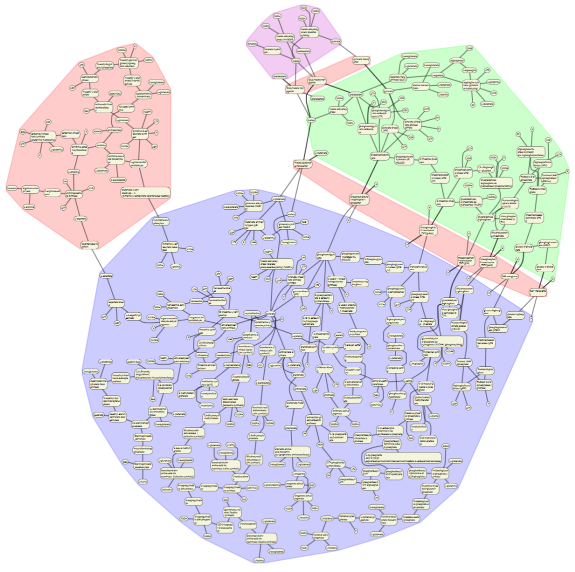


Figure 6: *A large biological pathway with non-overlapping convex hull boundaries around clusters. The clusters indicate functional partitions, such as parts of the carbohydrate metabolism (glycolysis, gluconeogenesis) and several amino acid synthesis pathways derived from the MetaCrop database. Network courtesy Falk Schreiber.*

4.5. Non-overlap of Convex Polygonal Node and Cluster Boundaries

The general problem of projecting two polygonal boundaries to resolve overlap is akin to finding the *minimal penetration depth* which arises in collision detection in rigid-body simulations and computer games [DHKS93]. The Minkowski Difference $A-B$ of two polygons A and B is the Minkowski Sum of A and $-B$ (the reflection of B across an arbitrary origin, e.g. one of the vertices of B). Intuitively, $A-B$ is built by "wrapping" $-B$ around the boundary of A. The minimal penetration depth is then the distance from the origin of B to the boundary of $A-B$. The minimal penetration depth vector **mpd** between two convex polygonal boundaries with s and t vertices respectively, can be found from their Minkowski Difference in $O(s+t)$ time. Once the Minkowski Difference for a pair of shapes has been computed it can be cached such that subsequent queries to find **pv** take $O(log(s+t))$ time. This gives us a non-overlap constraint:

$$|(\mathbf{p}-\mathbf{q})\cdot\frac{\mathbf{mpd}}{|\mathbf{mpd}|}| \geq |\mathbf{mpd}| \tag{7}$$

Such polygonal boundaries are useful for clustered graph layout. We compute the convex hull of a cluster of nodes

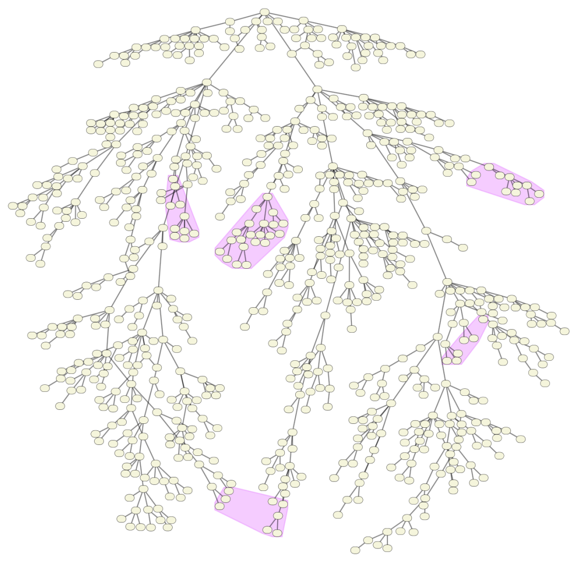


Figure 7: *A randomly generated tree with 768 nodes and several small clusters. Fixed orientation constraints are used to require the edges to point downwards and constraints are generated to prevent overlap between the convex-hulls of clusters.*

after each layout iteration and generate non-overlap constraints like (7) between cluster boundaries and the boundaries of nodes (and other clusters) external to the cluster. When projecting apart two clusters or a node and a cluster using (5) we take the weight of a cluster to be the number of nodes it contains. The displacement of the cluster hull due to projection is then applied to each of its constituents.

5. Results and Discussion

The method described in this paper was implemented in C# and run on a standard desktop PC with an Intel Core2 2.4GhZ processor and 4 GB of RAM.

We give a summary of layout time for a range of graphs of various sizes in Table 1. The graphs range from practical biological examples (such as Figures 4 and 6) to completely artificial examples (such as Figures 5, 7 and 8) invented to better show the various types of constraints discussed in Section 4. In practice we perform layout in three stages. First we compute an initial unconstrained layout using the fast Pivot Multidimensional Scaling method which was demonstrated to provide excellent unfolding and initial layout for general force-directed layout by Brandes and Pich in [BP09]. Then we ran 50 iterations of layout with constraints. Finally, we ran 10 iterations of the constrained-layout with an additional set of non-overlap constraints generated at each iteration, as described in Section 4.4.

Instead of running a fixed number of iterations one could also stop after convergence criteria, such as the step-size falling below some small threshold, are reached. In practice however, we find the fixed number of iterations provides reasonably satisfactory results and for the timings it gives a better indication of the running time per iteration.

Following the recommendations of [MHHR06] and [Jak01] and also our own experiments (see Fig. 2) we typically project across all constraints a small, fixed number of times each iteration, e.g. in our experiments we did this 10 times per iteration. Note that this does not remove all error, and for example, large "wheel constraints" as described in Section 4.3 may appear a little "squashable" when the user is allowed direct interaction (dragging of nodes) as the layout is running. One can dial-up the apparent stiffness by increasing the number of times the constraints are projected.

To give an indication of the relative performance of the new method compared to previous constraint layout methods, the layout in Figure 8 took 4.09 seconds including an initial layout with Pivot MDS (not including layout adjustment with rectangular non-overlap constraints which took a further 4.31 seconds). The same graph, with the same set of constraints (but not non-overlap constraints), took 19.97 seconds using a scaled gradient projection method (implemented in native C++ as opposed to managed C#) as reported in [DM08].

In our experiments, by far the slowest part of layout was the final step with non-overlap constraints for convex hull clusters, particularly when the clusters are large. Figure 6 for example, with clusters of up to 277 nodes took more than double the amount of time required by Figure 7 which had more nodes but smaller clusters. We have not experimented a great deal with optimization of this part of the layout, but many optimizations are possible. For example, caching of Minkowski sums for pairs of hulls and lazy update of convex hulls and the KD-tree used for region queries only after significant movement.

6. Related Work

An early effort to give users more control over automatic graph-layout through a constraint definition language was by Böhringer and Paulisch [BP90]. They augmented the popular Sugiyama [STT81] framework for layered drawing of directed graphs with user-specified constraints over the relative horizontal and vertical positions of nodes. Of course the Sugiyama layout scheme already has quite rigid constraints (e.g. downward pointing edges, strict node layering) and any control ceded to the user must be within this framework.

Another fairly popular combinatorial optimization approach to graph drawing is the orthogonal layout style. Like most other layout algorithms, this approach was conceived as a batch process (from abstract graph definition to layout with no user intervention), but orthogonal layout methods have also been retrofitted to support some degree of user

				Time			
Graph	n	m	c	Pivot MDS	With Constraints	Without Overlap	Total
Citrate Cycle (Fig. 4)	57	83	49	0.16	0.62	0.65	1.43
Circular Grid (Fig. 5)	400	760	114	0.68	0.71	NA	1.39
1138Bus (Fig. 8)	1138	1458	1458	1.22	2.87	4.31	8.40
Tree with Clusters (Fig. 7)	758	757	757	0.69	1.67	6.69	9.05
Large biological (Fig. 6)	432	481	NA	0.72	NA	15.7	16.4

Table 1: *Running times for the examples shown in figures, where n is the number of nodes, m is the number of edges and c is the number of constraints. Times are in seconds.*

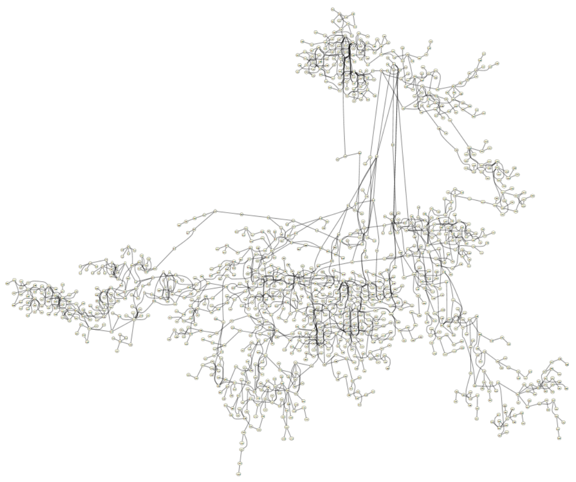


Figure 8: *The Bus1138 graph from [DKM06] with downward pointing edge constraints and non-overlapping rectangular node boundaries. The graph has 1,138 nodes, 1,458 edges and therefore 1,458 oriented distance constraints. Layout subject to constraints took 4.09 seconds. Layout adjustment subject to rectangular non-overlap constraints then took 4.31 seconds.*

control [BEKW02, EFK00]. Again, however, the degree of user control is limited by the strict constraints imposed by the layout model.

By contrast, the very popular force-directed layout approaches—which treat layout as optimization over a continuous goal function—are incremental by nature since the goal function is defined over all possible starting configurations. Ryal et al. [RMS97] made an early attempt at incorporating user defined placement constraints into force-directed layout. However, their system modeled constraints as springs which could not be strictly enforced without introducing stiffness and instability into their local descent solver. He and Marriott [HM98] augmented the Kamada-Kawai [KK89] spring layout method with constraints using quadratic programming techniques. However, their technique did not use efficient solver techniques and hence was only demonstrated with trivially small graphs.

Dwyer et al. [DKM06] demonstrated that the numerically stable stress-majorization approach could be made to support user defined constraints by, at each iteration, treating the majorizing functions as quadratic programs. Since successive quadratic programs in this iterative process could be very similar, any solver used needs to take advantage of the previous solution in order to provide timely layout. For simple two-variable inequality or equality constraints over pairs of x- or y-position variables they were able to solve these quadratic programs very quickly using a gradient-projection approach and their own active-set solver for the projection problem. In [DMW09b] Dwyer et al. proposed a new objective function and optimization method which worked better with highly constrained graphs and which could be more easily extended with new objective terms (for example, to optimize over the length of *routed* edges rather than straight edges). However, the axis separation and the underlying solver technique and hence the class of constraints that can be supported, is the same as their earlier work. Despite this limitation the utility of the approach and the potential of constraint-based layout was demonstrated in the context of online graph navigation in [DMS*08] and in an interactive diagram authoring tool in [DMW09a].

Highly related to force-directed graph layout techniques are particle-based physics simulations. Indeed, many advances in making force-directed layout scale to larger graphs have been adapted from techniques in use by physicists. The most recent of these is the elegant fast-multipole method [HJ05] for approximating long-range repulsive forces due to a group of particles (or in graph-layout, nodes).

This paper also represents an adaptation of techniques used in physical simulations to graph layout and is influenced in particular by the position-based dynamics approaches to skeletal animation in games, introduced by Jakobsen [Jak01] and described more formally by Müller et al. [MHHR06]. Merrick and Dwyer [MD04] and Murray et al. [MMT04] discussed some early experimentation with this kind of skeletal animation in the context of 3D graph layout but the constraints were used to provide a rigid spanning tree, an application that did not turn out to be particularly useful. This paper is the first to recognize the potential flexibility and scalability of a general framework for 2D graph layout built on Euclidean distance constraints.

7. Further Work

In Section 4 we describe several ways that systems of Euclidean distance constraints can be used in graph layout applications. However, many more are possible. The ad-hoc cyclical projection of constraints could conceivable be applied to other types of constraints where the projection operation is relatively straightforward for a single constraint but would be difficult to achieve optimally for all constraints. For example, alignment of nodes to an unoriented line could be achieved by perpendicular least-squares regression. Another example may be circular constraints with unfixed radius, where the projection operation would involve a least squares fit of radii from the barycenter.

If the network of constraints can be divided into vertex-disjoint groups then projection of these constraints can be completed in parallel. Therefore, efficient identification and management of such disjoint sets of constraints should make the method highly amenable to GPU or cloud-computing acceleration.

7.1. Acknowledgements

The implementation of the Minkowski minimum penetration depth algorithm was provided by Lev Nachmanson. The large clustered biological network shown in Figure 6 was provided by Falk Schreiber. Additional thanks to George Robertson and Ted Hart for the comments and suggestions.

References

[BEKW02] BRANDES U., EIGLSPERGER M., KAUFMANN M., WAGNER D.: Sketch-driven orthogonal graph drawing. In *Proc. of the 10th Intl. Symp. on Graph Drawing (GD'02)* (2002), vol. 2528 of *LNCS*, Springer, pp. 1–11.

[BP90] BÖRINGER K., PAULISCH F. N.: Using constraints to achieve stability in automatic graph layout algorithms. In *Proc. of ACM Conf. on Human Factors in Computing Systems* (1990), ACM, pp. 43–51.

[BP09] BRANDES U., PICH C.: An experimental study on distance-based graph drawing. In *Proc. 16th Intl. Symp. Graph Drawing (GD'08)* (2009), vol. 5417, Springer, pp. 218–229.

[DHKS93] DOBKIN D., HERSHBERGER J., KIRKPATRICK D., SURI S.: Computing the intersection-depth of polyhedra. *Algorithmica 9* (1993), 518–533.

[DKM06] 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 (2006), 821–828.

[DM08] DWYER T., MARRIOTT K.: Constrained stress majorization using diagonally scaled gradient projection. In *Proc. 15th Intl. Symp. Graph Drawing (GD '07)* (2008), vol. 4875 of *LNCS*, Springer.

[DMS*08] 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 14*, 6 (2008), 1293–1300.

[DMW09a] DWYER T., MARRIOTT K., WYBROW M.: Dunnart: A constraint-based network diagram authoring tool. In *Proc. 16th Intl. Symp. Graph Drawing (GD'08)* (2009), vol. 5417 of *LNCS*, Springer, pp. 420–431.

[DMW09b] DWYER T., MARRIOTT K., WYBROW M.: Topology preserving constrained graph layout. In *Proc. 16th Intl. Symp. Graph Drawing (GD'08)* (2009), vol. 5417 of *LNCS*, Springer, pp. 230–241.

[EFK00] EIGLSPERGER M., FÖSSMEIER U., KAUFMANN M.: Orthogonal graph drawing with constraints. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2000), Society for Industrial and Applied Mathematics, pp. 3–11.

[HJ05] HACHUL S., JÜNGER M.: An experimental comparison of fast algorithms for drawing general large graphs. In *Proc. 13th Intl. Symp. on Graph Drawing (GD'05)* (2005), vol. 3843 of *LNCS*, Springer, pp. 235–250.

[HM98] HE W., MARRIOTT K.: Constrained graph layout. *Constraints 3* (1998), 289–314.

[Hu05] HU Y.: Efficient and high quality force-directed graph drawing. *The Mathematica Journal 10*, 1 (2005), 37–71.

[Jak01] JAKOBSEN T.: Advanced character physics. In *San Jose Games Developers' Conference* (2001), www.gamasutra.com.

[KK89] KAMADA T., KAWAI S.: An algorithm for drawing general undirected graphs. *Inf. Process. Lett. 31*, 1 (1989), 7–15.

[Lau07] LAUTHER U.: Multipole-based force approximation revisited - a simple but fast implementation using a dynamized enclosing-circle-enhanced k-d-tree. In *Proc. 14th Intl. Symp. on Graph Drawing (GD'06)* (2007), vol. 4372 of *LNCS*, pp. 20–29.

[MD04] MERRICK D., DWYER T.: Skeletal animation for the exploration of graphs. In *Australian Symp. on Information Visualisation 2004* (2004), vol. 35 of *CRPIT*, ACS, pp. 61–70.

[MHHR06] MÜLLER M., HEIDELBERGER B., HENNIX M., RATCLIFF J.: Position based dynamics. In *Proc. of Virtual Reality Interactions and Physical Simulations (VRIPhys)* (2006), pp. 71–80.

[MMT04] MURRAY C., MERRICK D., TAKATSUKA M.: Graph interaction through force-based skeletal animation. In *Australian Symp. on Information Visualisation 2004* (2004), vol. 35 of *CRPIT*, ACS, pp. 81–90.

[NW06] NOCEDAL J., WRIGHT S. J.: *Numerical Optimization*, 2 ed. Springer Series in Operations Research. Springer, 2006.

[RMS97] RYALL K., MARKS J., SHIEBER S. M.: An interactive constraint-based system for drawing graphs. In *ACM Symposium on User Interface Software and Technology* (1997), pp. 97–104.

[STT81] SUGIYAMA K., TAGAWA S., TODA M.: Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics 11*, 2 (1981), 109–125.