Stress Majorization with Orthogonal Ordering Constraints

Tim Dwyer[1], Yehuda Koren[2], and Kim Marriott[1]

[1] School of Comp. Science & Soft. Eng., Monash University, Australia
{tdwyer,marriott}@mail.csse.monash.edu.au
[2] AT&T — Research yehuda@research.att.com

Abstract. The adoption of the stress-majorization method from multi-dimensional scaling into graph layout has provided an improved mathematical basis and better convergence properties for so-called "force-directed placement" techniques. In this paper we give an algorithm for augmenting such stress-majorization techniques with orthogonal ordering constraints and we demonstrate several graph-drawing applications where this class of constraints can be very useful.

Keywords: graph layout, constrained optimization, separation constraints

1 Introduction

The family of graph drawing algorithms that attempt to find an embedding of a graph that minimizes some continuous goal function, are variously known as *spring-embedder* or *force-directed placement* algorithms. A popular algorithm in this family has been that of Kamada and Kawai [9] in which squared differences between ideal distances for pairs of nodes and their Euclidean distance in the embedding is minimized. Gansner et al. [6] recently revisited this method and suggested using *functional majorization* — an optimization technique from the field of multidimensional scaling. Functional majorization iteratively improves the drawing by considering a sequence of quadratic forms that bound the stress function from above. They showed that it had distinct advantages over the original algorithm of Kamada and Kawai; particularly, a strictly monotonic decrease in stress and that it could achieve lower values of the cost function in the same running time.

A useful property of the majorization approach is that each iteration involves minimizing a convenient quadratic function. Gansner et al. [6] mentioned that this allows using any available equation solver. In this paper we take advantage of this property, and show how it helps in handling ordering constraints on the nodes. The quadratic nature of the function we minimize in each iteration allows us to efficiently add such linear constraints. In fact, minimizing linearly constrained quadratic functions is known as *quadratic programming*, which is an efficiently solvable problem [13]. However, we have found that general quadratic programming solvers will significantly slow down the stress majorization process. Therefore, we suggest a solver which is crafted especially for our problem, utilizing its unique nature. This solver can deal with ordering constraints without significantly increasing the running time of the layout process. We

also demonstrate the utility of imposing this class of constraints — which we call *orthogonal ordering* constraints — to applications such as network layout reflecting the relative positions of an underlying set of coordinates and directed graph drawing.

2 Background

We recently introduced the idea of using stress majorization coupled with standard quadratic programming techniques for drawing directed graphs [5]. In the so-called DIG-COLA[3] technique, nodes in the digraph were partitioned into layers based on their hierarchical level and constraints were introduced in the vertical dimension to keep these layers separated. Compared to standard hierarchical graph drawing methods the DIG-COLA algorithm was shown to produce layouts with a much better distribution of edge lengths and for large, dense graphs it was able to find layouts with fewer edge crossings. However, a commercial quadratic programming solver was used to minimize the quadratic forms subject to constraints. This generic approach meant that layout for graphs with hundreds or thousands of nodes could take some minutes to perform.

Another case where orthogonal ordering constraints are useful is when we want to improve the readability of a given layout without significantly changing it. Misue et al. [10] discussed the importance of preserving a user's "mental map" when adjusting graph layouts. One of their models for the mental map focused on preserving *orthogonal ordering* of the nodes in a layout — the relative above/below, left/right positions of the nodes.

The potential for constraint-based, force-directed graph layout was explored by Ryall et al. [11], however their implementation did not use true constraint solving techniques. Rather, they added stiff springs to a standard force-directed model to keep user-selected parts of the diagram roughly spaced as desired. True constraint solving techniques for graph drawing were explored by He and Marriott in [7], where a Kamada-Kawai-based method was extended with an active-set constraint solving technique to provide separation constraints. However, only small examples of fewer than 20 nodes were tested and the scalability of the technique was not tested.

3 Problem formulation

The general goal function, known as *the stress function*, which we seek to minimize is described by

$$\sum_{i<j} w_{ij}(||X_i - X_j|| - d_{ij})^2$$

where for each pair of nodes i and j, d_{ij} gives an ideal separation between i and j (usually their graph-theoretical distance), $w_{ij} = d_{ij}^{-2}$ is used as a normalization constant and X is a $n \times d$ matrix of positions for all nodes, where d is the dimensionality of the drawing and n is the number of nodes.

[3] Directed Graphs with Constraint-based Layout

Majorization minimizes this stress function by iteratively minimizing quadratic forms that approximate and bound it from above. Due to its central role in this work, we provide the essential details of the method. Recall that w_{ij} are the normalization constants in the stress function. We use the $n \times n$ matrix A, defined by

$$A_{i,j} = \begin{cases} -w_{ij} & i \neq j \\ \sum_{k \neq i} w_{ik} & i = j \end{cases} . \tag{1}$$

In addition, given an $n \times d$ coordinate matrix Z, we define the $n \times n$ matrix A^Z by

$$A^Z_{i,j} = \begin{cases} -w_{ij} \cdot d_{ij} \cdot \mathrm{inv}(\|Z_i - Z_j\|) & i \neq j \\ -\sum_{k \neq i} A^Z_{i,k} & i = j \end{cases} , \tag{2}$$

where $\mathrm{inv}(x) = 1/x$ when $x \neq 0$ and 0 otherwise.

It can be shown (see [6]) that the stress function is bounded from above by the quadratic form $F^Z(X)$ defined as

$$F^Z(X) = \sum_{i<j} w_{ij} d^2_{ij} + \sum_{a=1}^{d} \left(\left(X^{(a)}\right)^T A X^{(a)} - 2 \left(X^{(a)}\right)^T A^Z Z^{(a)} \right). \tag{3}$$

Here, $X^{(a)}$ denotes the a-th column of matrix X. Thus, we have

$$\mathrm{stress}(X) \leqslant F^Z(X) \tag{4}$$

with equality when $Z = X$.

We differentiate by X and find that the global minima of $F^Z(X)$ are given by solving

$$AX = A^Z Z \tag{5}$$

This leads to the following iterative optimization process. Given some layout $X(t)$, we compute a layout $X(t+1)$ so that $\mathrm{stress}(X(t+1)) < \mathrm{stress}(X(t))$. We use the function $F^{X(t)}(X)$ which satisfies $F^{X(t)}(X(t)) = \mathrm{stress}(X(t))$. Then, we take $X(t+1)$ as the minimizer of $F^{X(t)}(X)$ by solving (5).

Note that it would be equivalent to consider in each iteration d independent optimization problems, one problem for each axis. Hence the a-th axis of the drawing is determined by minimizing

$$x^T A x - 2 x^T A^Z Z^{(a)} \tag{6}$$

Henceforth, we will work, w.l.o.g., with this 1-D layout formulation as it allows a more convenient notation.

So far we have described the usual, unconstrained stress majorization. In this work we consider a case where we have additional ordering constraints on each axis. Each node i is assigned a level of index $1 \leq lev[i] \leq m$ and variable placement must respect this level. Thus, instead of minimizing (6), we would take the a-th axis of the drawing as the solution of

$$\begin{aligned} \min_x \quad & x^T A x - 2 x^T A^Z Z^{(a)} \\ \text{subject to: } & lev[i] < lev[j] \Rightarrow x_i \leq x_j \\ & \text{for all } i, j \in \{1, \ldots, n\} \end{aligned} \tag{7}$$

For brevity henceforth we will replace $2A^Z Z^{(a)}$ with $b \in \mathbb{R}^n$, so the target function is merely $f(x) = x^T Ax - x^T b$. We call this the Quadratic Programming with Orthogonal Constraints (QPOC) problem.

It is easy to show that A is positive semi-definite, so the problem has only global minima. Such a quadratic programming problem can be solved in a polynomial time [13]. However, our experiments show that generic quadratic-programming solvers are much slower than solving an unconstrained problem. To accelerate computation we can utilize two special characteristics of the problem:

1. During the majorization process, we iteratively solve closely related quadratic programs: The constraints and the matrix A are not changed between iterations, while only the vector b is changed. Therefore, the solution of the previous iteration is still a feasible solution for current iteration (satisfying all constraints). Moreover, this previous solution is probably very close to the new optimal solution (e.g., consider that in most iterations the coordinates are only slightly changed). However, such initialization, called "warm-start", is fundamentally not trivial for the barrier (or interior-point) methods used by most commercial solvers.
2. Our constraints are very simple as each of them involve only two variables, being of the form $x_i \leq x_j$. This allows a simple mechanism for guaranteeing the feasibility of the solution.

In the next section we describe an algorithm for solving the QPOC problem.

4 Algorithm

We give an iterative *gradient-projection* algorithm (see Bertsekas [1]) for finding a solution to a QPOC Problem. The algorithm, *solve_QPOC*, is shown in Figure 1. The first step is to decrease $f(x) = x^T Ax + x^T b$, by moving x in the direction of steepest descent, i.e. if the gradient is $g = \nabla f(x) = Ax + b$ this direction is $-g$. While we are guaranteed that — with appropriate selection of step-size s — the energy is decreased by this first step, the new positions may violate the ordering constraints. We correct this by calling the *project* procedure which returns the closest point $\bar{x}$ to x which satisfies the ordering constraints, i.e. it projects x on to the feasible region. Finally, we calculate a vector d from our initial position $\hat{x}$ to $\bar{x}$ and we ensure monotonic decrease in stress when moving in this direction by computing a second stepsize $\alpha = \arg\min_{\alpha \in [0,1]} f(x + \alpha d)$ which minimizes stress in this interval.

The procedure *project* is the main technical innovation in this paper. The main difficulty in implementing gradient-projection methods is the need to efficiently project on to the feasible region. Because of the simple nature of the orthogonal ordering constraints we can do this in $O(mn + n \log n)$ time where m is the number of levels and n the number of variables. The *project* procedure (Figure 2) iteratively changes the positions till all constraints are satisifed. In iteration k all constraints involving nodes up to the $(k+1)$-th level are imposed. More technically, it starts by finding an ordering of the nodes q such that $a = q[i], b = q[i+1]$ implies either $lev[a] < lev[b]$ or ($lev[a] = lev[b]$ and $x_a \leqslant x_b$). For convenience we also keep an array $1 < p_1, \ldots p_m = n + 1$ of indices for the start of each partition excluding the first

procedure *solve_QPOC(A, b, lev)*
  $k \leftarrow 0, x \leftarrow$ *initial_soln*()
  repeat
    $g \leftarrow 2Ax + b$
    $s \leftarrow \frac{g^T g}{g^T A g}$
    $\hat{x} \leftarrow x$
    $\bar{x} \leftarrow$*project*$(\hat{x} - sg,$ *lev*$)$
    $d \leftarrow \bar{x} - \hat{x}$
    $\alpha \leftarrow \max(\frac{g^T d}{d^T A d}, 1)$
    $x \leftarrow \hat{x} + \alpha d$
  until $\|\hat{x} - x\|$ sufficiently small
  return x

Fig. 1. Algorithm to find an optimal solution to a QPOC problem with variables $x_1, \ldots, x_n$, symmetric positive-semidefinite matrix A, vector b and $1 \leq lev[i] \leq m + 1$ gives the level for each node i.

(for convenience p_m was set to $n + 1$). When considering partition k, which contains the nodes $above_k = \{u | p_k \leq q[u] < p_{k+1}\}$, we ensure that none of these nodes are assigned positions lower than that of $below_k = \{l | 1 \leq q[l] < p_k\}$. To achieve this we create a minimal set $U_k \subseteq \{j | 1 \leq q[j] < p_{k+1}\}$ that includes nodes violating this condition. To impose the constraints we force all nodes of U_k to lie on a single point $posnU_k$. Since we want to minimize the quadratic function, we take this point as the average of all positions in U_k. The set U_k is minimal in that it does not necessarily include all nodes violating the boundary condition for k, but only the minimal number that need to be moved to $posnU_k$ such that this condition may be satisfied. The following lemma captures this.

Lemma 1. *During execution of* project$(x,$lev$)$ *after finishing the k^{th} iteration in which U_k and its associated $posnU_k$ are computed*

$$posnU_k = \frac{\sum_{i \in U_k} x_i}{|U_k|} \tag{8}$$

and

$$U_k = \{l \in below_k \mid x_l > posnU_k\} \cup \{u \in above_k \mid x_u < posnU_k\} \tag{9}$$

where the position for x_i is its value before the start of the iteration.

Proof. Equation (8) follows directly from the algorithm and is invariant throughout the loop incrementally building U_k (since whenever U_k is expanded $posnU_k$ is recalculated).

The post-condition (9) implies that U_k includes all nodes that violate the internal constraints among $1, \ldots, p_k - 1$ and $p_k, \ldots, p_{k+1} - 1$. Proof is as follows. The levels are examined in order. When examining level k all nodes in $below_k$ must be sorted by position in q (either by the initial precondition for q or since they have been assigned to

```
procedure project(x,lev)
  q ← {1 ≤ i ≤ n} sorted by (x_i, lev[i])
  p ← indices to start of each level in q
    s.t. p_1 < ... < p_{m-1} < p_m = n + 1
    and lev[q[p_k]] = lev[q[p_k − 1]] + 1, 1 ≤ k < m
  for 1 ≤ k < m do
    % below_k = {l|1 ≤ q[l] < p_k}, above_k = {u|p_k ≤ q[u] < p_{k+1}}
    % Find U_k = {q[i]|il < i < iu} ⊆ below_k ∪ above_k
    maxiu ← p_{k+1} − 1
    l ← q[p_k − 1], u ← q[p_k]
    sum ← x_l + x_u, w ← 2
    iu ← p_k + 1, il ← p_k − 2
    if x_l > x_u then
      repeat
        finished ← true
        u ← q[iu]
        posnU_k ← sum/w
        if iu ≤ maxiu and x_u < posnU_k then
          iu ← iu + 1, w ← w + 1
          sum ← sum + x_u
          finished ← false
        end if
        l ← q[il]
        if il ≥ 1 and x_l > posnU_k then
          il ← il − 1, w ← w + 1
          sum ← sum + x_l
          finished ← false
        end if
      until finished
      for il < i < iu do
        j ← q[i]
        x_j ← posnU_k
      end for
    end if
  end for
return x
```

Fig. 2. Algorithm to project variables to the closest position in the feasible region, $1 \leq lev[i] \leq m$ gives the level for each node i.

a position $posnU_l, l < k$). The precondition for q also ensures that nodes in $above_k$ are sorted by position.

If there is overlap between the tail of $below_k$ and the head of $above_k$ we place these in U_k and set $posnU_k$. We then iteratively examine the successive elements of $below_k$ (from the tail) and $above_k$ (from the head) and add them to U_k until no further overlap is found between these elements and $posnU_k$.

By construction the only elements $l \in below_k$ not placed in U_k are those for which $x_l \leq posnU_k$ (otherwise the loop would not terminate). Dually, for any element $u \in above_k$ not placed in U_k we have that $x_u \geq posnU_k$. Thus

$$U_k \supseteq \{1 \leq q[i] < p_k \mid x_i > posnU_k\} \cup \{p_k \leq i < p_{k+1} \mid x_i < posnU_k\}$$

We now show containment by induction. We prove for $U_k \cap below_k$, while the proof for $U_k \cap above_k$ is analogous. The base case follows from the fact that at the moment we add some $l \in below_k$, it must hold that $x_l > posnU_k$. Now, if later we add $l' \in below_k$, then since $below_k$ is ordered by position, $x_{l'} \leq x_l$. By hypothesis,

$x_l > posnU_k$ and since the new $posnU_k$ is the weighted average of x'_l and $posnU_k$, we still have $x_l > posnU_k$. If later we add $u \in above_k$, then since we are adding u we must have $x_u < posnU_k$. Now by hypothesis, $x_l > posnU_k$ and so $x_l > x_u$. Thus as for the previous case $x_l > posnU_k$.

□

Corollary 1. *During execution of* project*(*x*,*lev*) after finishing the* k^{th} *iteration in which* U_k *and its associated* $posnU_k$ *are computed*

$$posnU_k = \frac{\sum_{i \in U_k} x_i}{|U_k|} \tag{10}$$

where the position of x_i *is the input position.*

Proof. Notice that unlike Equation (8), the x_i's refer now to the *input* positions, rather than to their values before the current iteration. This makes a difference when we find that $posnU_k < posnU_l, l < k$ and therefore $U_k \supset U_l$ and $posnU_k$ will be calculated from $posnU_l$ for those nodes in U_l rather than their original positions. In this case (10) still holds as

$$\begin{aligned} posnU_k &= \frac{1}{|U_k|}\left(|U_l| posnU_l + \sum_{i \in U_k \setminus U_l} x_i\right) \\ &= \frac{1}{|U_k|}\left(|U_l|(\frac{1}{|U_l|}\sum_{j \in U_l} x_j) + \sum_{i \in U_k \setminus U_l} x_i\right) = \frac{1}{|U_k|}\sum_{i \in U_k} x_i \end{aligned}$$

□

We now show that this results in a valid gradient-projection method.

Lemma 2. *If the result of the call* project*(*x^0*,*lev*) is* x *then* x *is the closest point to* x^0 *satisfying the ordering constraints defined by lev.*

Proof. (Sketch) We must prove that x minimizes $F(x) = \sum_{i=1}^{n}(x_i - x_i^0)^2$ subject to satisfying the ordering constraints. It follows from the construction that x satisfies the ordering constraints. Proving optimality is more difficult. Let $u_1, \ldots, u_{m-1}$ be new variables, one for each partition k. We set values to the new variables by setting u_k to be $max\{x_i \mid lev[i] = k\}$.

Recall that if we are minimizing a function F with a set of convex equalities C over variables X, then we can associate a variable λ_c called the Lagrange multiplier with each $c \in C$. Given a solution x we have that this is a minimal solution iff there exist values for the Lagrange multipliers satisfying

$$\frac{\partial F}{\partial x} = \sum_{c \in C} \lambda_c \frac{\partial c}{\partial x} \tag{11}$$

for each variable $x \in X$. Furthermore, if we also allow inequalities then the above statement continues to hold as long as $\lambda_c \geq 0$ for all inequalities c of form $c(x) \geq 0$. By definition an inequality c which is not active, i.e., $c(x) > 0$ has $\lambda_c = 0$. These are known as the Karush-Kuhn-Tucker conditions; see [1].

We now prove that x minimizes $F(x)$ subject to, for $k = 1, \ldots, m-1$:

$u_{k-1} \leq u_k$ if $k > 1$
$x_i \leq u_k$ for all i s.t. $lev[i] = k$
$x_i \geq u_k$ for all i s.t. $lev[i] = k + 1$

These constraints are equivalent to the ordering constraints.

We show optimality by giving values for all λ_c satisfying Equation (11). An inequality $x_i \leq u_k$ or $x_i \geq u_k$ is active if $i \in U_k \setminus U_{k-1}$. Note that we can have $U_k \subseteq U_{k+1}$, in which case we must be careful to make the right constraint active so as to ensure that each x_i will be involved in no more than one active constraint. For a constraint c of form $x_i \geq u_k$ we set $\lambda_c = \frac{\partial F}{\partial x_i}$ and for c of form $x_i \leq u_k$ we set $\lambda_c = -\frac{\partial F}{\partial x_i}$. The constraint c of form $u_k \leq u_{k+1}$ is active if $U_k \subseteq U_{k+1}$. We set $\lambda_c = -\sum_{i \in U_k} \frac{\partial F}{\partial x_i}$. For all other inequalities c we set $\lambda_c = 0$. We give an extended formal proof of this lemma in [4].

We can now prove the correctness of *solve_QPOC*:

Theorem 1. *solve_QPOC converges to an optimal solution to the input QPOC Problem.*

Proof. Lemma 2 ensures that *solve_QPOC* is a gradient projection method. We now show that a more general proof of convergence for gradient projection methods holds for our specific stepsize calculations. First consider a variant of *solve_QPOC* in which s is always 1 — note that for both constant s and the calculation of s used in Figure 1 the method is equivalent to standard steepest-descent in the case when no active constraints are encountered. With constant $s = 1$ the computation of α implements a Limited Minimization Rule and so from [1, Proposition 2.3.1] every limit point of *solve_QPOC* is a stationary point. Since the original problem is convex any stationary point is an optimal solution. Now consider our computation of s. To ensure convergence we must prove that if $s^k \to 0$ where s^k is the value of s in the k^{th} iteration then the limit point of *solve_QPOC* is a stationary point. But since the computation of s^k is also an example of the Limited Minimization Rule on the unconstrained problem, $s^k \to 0$ only if the limit point of *solve_QPOC* is a stationary point for the unconstrained problem, in which case it must also be a limit point of the constrained problem.

□

4.1 Running time

The second part of the algorithm, satisfying the constraints, can be performed in $O(mn + n \log n)$ time. However each complete iteration is dominated by computing the desired positions which takes $O(n^2)$ time. This is of course the inherent complexity of the stress function that contains $O(n^2)$ terms. (In fact, this is the same as the complexity of an iteration of the conjugate-gradient method, which is used in the unconstrained majorization algorithm.) In practice only few (5-30) iterations are required to return the optimal solution depending on the threshold on $||x - \hat{x}||$. Running times for graphs with various sizes and with varying numbers of boundaries m are given in Table 1. We compare results for those obtained with the *solve_QPOC* algorithm implemented in C and the Mosek interior-point quadratic programming solver [14]. Tests were conducted

on a 2GHz P4-M notebook PC. As expected, since both solvers return the optimal or near optimal solution, the resulting drawings look identical. However, the dedicated *solve_QPOC* algorithm significantly outperformed the generic solver. The final "stress" value is given as a rough measure of relative quality. Note that this is the final stress value after being monotonically reduced by a number of iterations of the functional-majorization method. Sample graphs were obtained from the Matrix Market [2] (Such as *1138bus* as shown in Figure 4) and some graphs based on geographic coordinates which are shown in Figures 5 and 6.

			Solve_QPOC		Mosek	
graph	#nodes (n)	#levels (m)	Time	Stress	Time	Stress
1138bus	1138	231	4.53	74343	209	74374
nos4	100	34	0.14	216.5	2.75	216.8
nos5	468	256	2.172	8517.3	13.0	8614.6
dwa512	512	14	1.23	22464	37.7	22464
dwb512	512	19	1.57	15707	90.8	16418
NSW Rail	312	54/76 (x/y-axis)	4.92	2288	18.6	2274.5
Backbone	2603	2373/1805 (x/y-axis)	55.8	1246960	> 1000	

Table 1. A comparison of results obtained for arranging various graphs with *solve_QPOC* and the *Mosek* interior point method. Times are measured by seconds.

5 Applications

5.1 Directed graph drawing

The method and motivation for drawing directed graphs by constrained majorization is discussed at length in [5]. Generally, a digraph can be said to induce a hierarchical structure on its nodes based on the precedence relationships defined by its directed edges. Consequently, an appropriate depiction of a digraph allocates the y-axis to showing this hierarchy. Thus, if node i precedes node j in the hierarchy, then i will be drawn above j on the y-axis; see, e.g., Sugiyama et al. [12]. This usually leads to the majority of directed edges pointing downwards, thereby showing a clear flow from top to bottom. There are a few possibilities for computing the hierarchical ordering of the nodes. We base our ordering on the "optimal arrangement" suggested by Carmel et al. [3]. Then, we compute the 2-D layout that minimizes the stress, while the y-coordinates of the nodes must obey their hierarchical ordering.

It was shown that this method produces drawings with much more uniform edge lengths making connectivity in large graphs more visible than in drawings produced by standard hierarchical graph drawing techniques.

We reproduce some example graphs drawn in this style and compare performance of our *solve_QPOC* algorithm with that of the solver previously used. Figure 3 illustrates the concept with a small directed graph containing a cycle. Note that since all nodes in the cycle are in the same hierarchical level they are drawn within the same band. Figure 4 shows a much larger example from the matrix market collection [2].

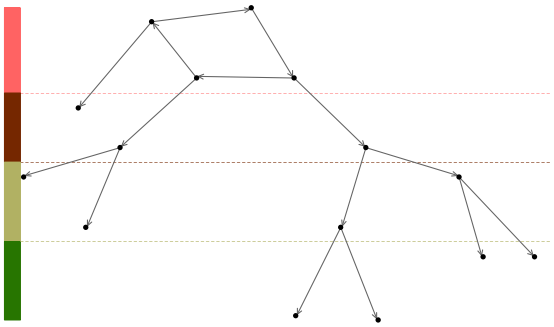


Fig. 3. A directed graph arranged using orthogonal ordering constraints in just the vertical dimension to preserve layering. The color bars on the left side indicate the layer-bands and the faint horizontal lines indicate the boundaries between these layers.

Fig. 4. The *1138bus* graph (1138 nodes, 1458 edges) from the Matrix market collection[2], displayed as a directed graph.

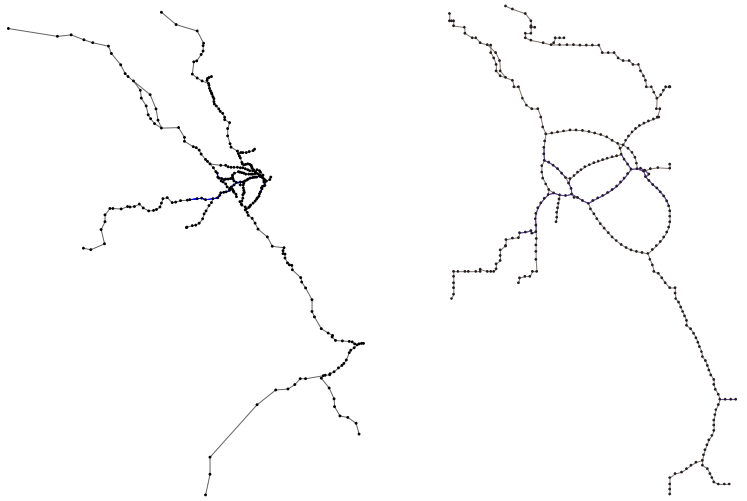
5.2 Layouts preserving the orthogonal ordering

Sometimes a graph has meaningful coordinates. These might be natural physical coordinates associated with the nodes, or just a given layout with which the user is familiar. We want to improve the readability of the given layout while keeping its overall structure, thus preserving the user's mental map and/or natural properties of the layout. A way to achieve these goals is to minimize the stress of the graph, while preserving the original vertical and horizontal ordering of the nodes. These can be achieved by our algorithm. We provide here two examples of refining layouts with meaningful physical coordinates.

The first example involves automatic production of rail network maps. This problem has been tackled as a graph drawing problem by Hong et al. [8]. To produce print quality drawings the authors seek to satisfy quite complex aesthetic requirements such as effective labelling, edges strictly aligned to axes or diagonals and no induced crossings. However, as illustrated in Figure 5, simple orthogonal ordering also goes a long

way to improving these diagrams. Note that the underlying geographic relationships are still evident while paths have been straightened and complex sections enlarged.

The second example is an internet backbone network as shown in Figure 6. The layout based on original coordinates contains very dense areas. However, readability is vastly improved by minimizing the stress, while original orthogonal order is preserved.



(a) Actual geographic positions (b) Ordering preserving layout

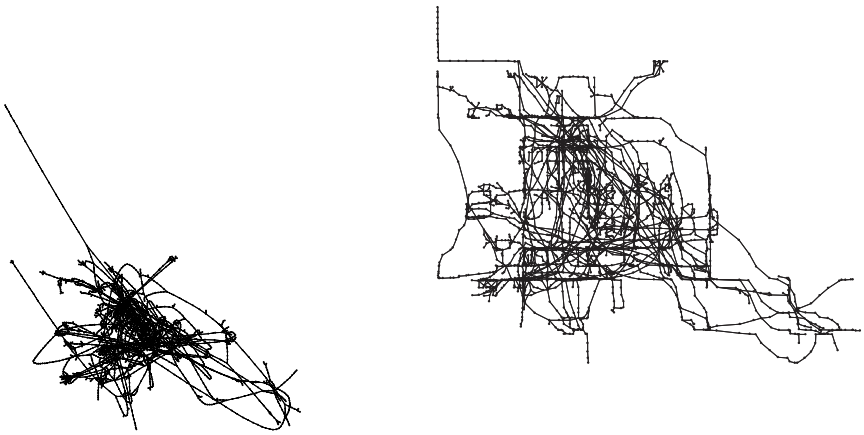
Fig. 5. The New South Wales rail network (312 nodes, 322 edges) shown with actual geographic positions (left) and then refined using stress minimization with orthogonal ordering constraints (right)

6 Conclusion and Further Work

We have demonstrated some applications of orthogonal-ordering constraints and that stress majorization can efficiently deal with such constraints. We are currently working on extending the algorithm to work for general separation constraints that may have many more applications, including clustered graph drawing — where we want to separate different clusters — and also cases where we want to restrict portions of the graph to specific rectangular regions. An obvious extension is to allow a wider variety of linear constraints. This would allow restricting portions of the graph to specific convex regions. However solving more general linear constraints requires a more sophisticated algorithm. Active-set techniques [13] may prove promising in this area.

7 Acknowledgements

Thanks to Damian Merrick for the NSW rail network data and members of the Adaptive Diagrams group at Monash University for their advice and support.



(a) Actual geographic positions (b) Ordering preserving layout

Fig. 6. A backbone network (2603 nodes, 2931 edges). Left picture is based on the actual geographic coordinates while the right picture is based on ordering-preserving constrained stress minimization

References

1. D. P. Bertsekas, *Nonlinear Programming*, Athena Scientific, 2^{nd} Edition (1999).
2. R. Boisvert, R. Pozo, K. Remington, R. Barrett and J. Dongarra, "The Matrix Market: A web resource for test matrix collections", in *Quality of Numerical Software, Assessment and Enhancement*, Chapman Hall (1997) 125–137.
3. L. Carmel, D. Harel and Y. Koren, "Combining Hierarchy and Energy for Drawing Directed Graphs", *IEEE Trans. Visualization and Computer Graphics* **10** (2004) 46–57.
4. T. Dwyer, Y. Koren and K. Marriott, "Stress Majorization with Orthogonal Ordering Constraints", Technical Report 2005/175, Monash University School of Computer Science and Software Engineering (2005). Available from `www.csse.monash.edu.au/~tdwyer`
5. T. Dwyer and Y. Koren, "DIG-COLA: Directed Graph Layout through Constrained Energy Minimization", *IEEE Symposium on Information Visualization (Infovis'05)* (To appear 2005).
6. E. Gansner, Y. Koren and S. North, "Graph Drawing by Stress Majorization", *Proc. 12th Int. Symp. Graph Drawing (GD'04)*, LNCS 3383, Springer Verlag (2004) 239–250.
7. W. He and K. Marriott, "Constrained Graph Layout", *Constraints* **3** (1998) 289–314.
8. S. Hong, D. Merrick and H. Nascimento, "The metro map layout problem", *Proc. 12th Int. Symp. Graph Drawing (GD'04)*, LNCS 3383, Springer Verlag (2004) 482–491.
9. T. Kamada and S. Kawai, "An Algorithm for Drawing General Undirected Graphs", *Information Processing Letters* **31** (1989) 7–15.
10. K. Misue, P. Eades, W. Lai, and K. Sugiyama, "Layout Adjustment and the Mental Map", *Journal of Visual Languages and Computing* **6** (1995) 183–210.
11. K. Ryall, J. Marks and S. M. Shieber, "An Interactive Constraint-Based System for Drawing Graphs", *ACM Symposium on User Interface Software and Technology* (1997) 97–104.
12. K. Sugiyama, S. Tagawa and M. Toda, "Methods for Visual Understanding of Hierarchical Systems", *IEEE Trans. Systems, Man, and Cybernetics* **11** (1981) 109–125.
13. J. Nocedal, S. Wright, *Numerical Optimization*, Springer (1999).
14. Mosek Optimization Toolkit V3.2 `www.mosek.com`.