

Constrained graph layout by stress majorization and gradient projection

Tim Dwyer^{a,*}, Yehuda Koren^b, Kim Marriott^a

^a Clayton School of Information Technology, Monash University, Australia

^b AT&T — Research, United States

Received 31 January 2006; accepted 27 December 2007

Available online 4 March 2008

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 explore algorithms for augmenting such stress-majorization techniques with simple linear constraints using gradient-projection optimization techniques. Our main focus is a particularly simple class of constraints called “orthogonal-ordering constraints” but we also discuss how gradient-projection methods may be extended to solve more general linear “separation constraints”. In addition, we demonstrate several graph-drawing applications where these types of constraints can be very useful.

© 2008 Elsevier B.V. All rights reserved.

Keywords: Graph drawing; Constraints; Stress majorization; Gradient projection

1. Introduction

The problem of embedding graphs in a two- or three-dimensional space falls into an uncomfortable gray area between mathematical disciplines. Graph theory is usually introduced as an area of discrete math where the existence or absence of an edge between a pair of nodes is described by a binary value. However, many applications require a real-valued weight to be assigned to edges and the space available for embedding graphs is certainly continuous. As a result, while many algorithms for graph drawing are combinatorial by nature, there are a number of algorithms that attempt to find an embedding of a graph that minimizes some continuous goal function. These are variously known as *spring-embedder* or *force-directed placement* algorithms. A popular algorithm in this family has been that of Kamada and Kawai [20] in which the sum of squared differences between ideal distances for pairs of nodes and their Euclidean distance in the embedding is minimized. Gansner et al. [16] recently revisited this method using *functional majorization* — an optimization technique used to solve a similar placement problem in the field of multidimensional scaling [3]. Functional majorization iteratively improves the drawing by considering a sequence of quadratic forms that bound the goal (or *stress*) function from above. They showed that this process has distinct advantages over the original algorithm of Kamada and Kawai; particularly, a strictly monotonic decrease in stress and that a lower value for the cost function is achieved in the same running time.

* Corresponding author.

E-mail address: Tim.Dwyer@infotech.monash.edu.au (T. Dwyer).

A useful property of the majorization approach is that each iteration involves minimizing a convenient quadratic function. In [7], in the context of drawing directed graphs, we discussed how this quadratic function could be replaced by a quadratic program (QP) with a particular class of simple linear constraints – called *orthogonal ordering constraints* – to restrict node placement. Then, in [10], we introduced an efficient algorithm based on the gradient-projection method for efficiently solving these QPs. Here, we extend the topics discussed in [10]. Particularly, we discuss the gradient-projection algorithm in more detail and discuss how it may be extended to more general *separation* constraints. Other types of constraints open up more possibilities for graph layout. We demonstrate this with another method for drawing directed graphs based on separation constraints rather than orthogonal-ordering constraints. This new method is found to produce lower stress drawings but the more complicated constraints are found to require more processing time.

2. Background

We recently introduced the idea of using stress majorization coupled with standard quadratic programming techniques for drawing directed graphs [7]. In the so-called DIG-COLA¹ 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 QP 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. [22] 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. [24], 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. Such an approximation of constraints by stiff springs has appeared a number of times in various force-directed layout applications, e.g. [4, 13]. True constraint solving techniques for graph drawing were explored by He and Marriott in [17,18], where a Kamada–Kawai-based method was extended with an active-set constraint solving technique to provide arbitrary linear constraints. However, only small examples of fewer than 20 nodes were tested and the scalability of the technique was not examined.

3. Problem formulation

3.1. Stress function

The general goal or *stress* function that we seek to minimize is given by

$$\text{stress}(X) = \sum_{i < j} w_{ij} (\|X_i - X_j\| - d_{ij})^2,$$

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 $n \times r$ matrix of positions for all nodes, where r is the dimensionality of the drawing and n is the number of nodes.

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

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

¹ Directed graphs with constraint-based layout.

Matrix A is sometimes called the weighted graph Laplacian.

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

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

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

It can be shown (see [16]) that the stress function is bounded from above by the quadratic form $F^Z(X)$ defined as

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

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

$$\text{stress}(X) \leq F^Z(X) \tag{4}$$

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 $\text{stress}(X(t + 1)) < \text{stress}(X(t))$. We use the function $F^{X(t)}(X)$ which satisfies $F^{X(t)}(X(t)) = \text{stress}(X(t))$. Then, we take $X(t + 1)$ as the minimizer of $F^{X(t)}(X)$ by solving (5).

Note that it is 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

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

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

3.2. Orthogonal-ordering constraints

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 $1 \leq \text{lev}[i] \leq m$ 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

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

The constant G in the constraints specifies an arbitrary “gap” that may be required between levels. Note that the level constraints define a partial ordering on the variables in x and that defining a level for each variable ensures a simple order. We discuss how the gap G and the different types of ordering are applied in Section 5. For brevity henceforth we will replace $2A^Z Z^{(a)}$ with $b \in \mathbb{R}^n$, so the target function is merely $f(x) = x^T A x - x^T b$. We call the problem of solving (7) the Quadratic Programming with Ordering Constraints (QPOC) problem.

It is easy to show that A is positive semi-definite, so the problem has only global minima. Such a QP problem can be solved in a polynomial time [23]. However, our experiments show that generic QP 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 QPs: 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 the current iteration (satisfying all constraints). Moreover, this previous solution is

```

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

```

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

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 $x_i + G \leq x_j$. 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. Gradient projection algorithm

We give an iterative *gradient-projection* algorithm (see Bertsekas [2]) for finding a solution to a QPOC Problem. The algorithm, *solve_QPOC*, is shown in Fig. 1. The first step is to decrease $f(x) = x^T A x + x^T b$, 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 a monotonic decrease in stress when moving in this direction by computing a second stepsize $\alpha = \arg \min_{\alpha \in [0,1]} f(\hat{x} + \alpha d)$ which minimizes the stress in this interval.

While the algorithm given in Fig. 1 describes a fairly standard gradient-projection approach, the procedure *project* is the part of the algorithm specific to our particular QP. 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 (as follows) in $O(mn + n \log n)$ time where m is the number of levels and n the number of variables.

The projection operation also requires solving a QP of the form

$$\begin{aligned}
 \min_x \quad & \sum_{i=1}^n (x_i - p_i)^2 \\
 \text{subject to:} \quad & lev[i] < lev[j] \Rightarrow x_i + G \leq x_j \\
 & \text{for all } i, j \in \{1, \dots, n\},
 \end{aligned} \tag{8}$$

where $p = \hat{x} - sg$ as defined in the gradient-projection step. It is also worth noting that this latter QP is related to the QP that is central to the problem of *isotonic regression* [1] — a type of statistical technique that involves finding a least squares fit of a monotonically increasing or decreasing curve to a set of data where there is some expected order. It is not surprising therefore that our projection algorithm has similarities to the algorithms used in isotonic regression. It differs from these techniques, however, in that our partial ordering is different to the simple or tree orderings usually considered in isotonic regression and that a gap G can be specified to separate levels.

The *project* procedure (see Figs. 2 and 3) iteratively adjusts the positions until all constraints are satisfied. In iteration k all constraints involving nodes up to the $(k + 1)$ -th level are imposed. More precisely, it starts by finding

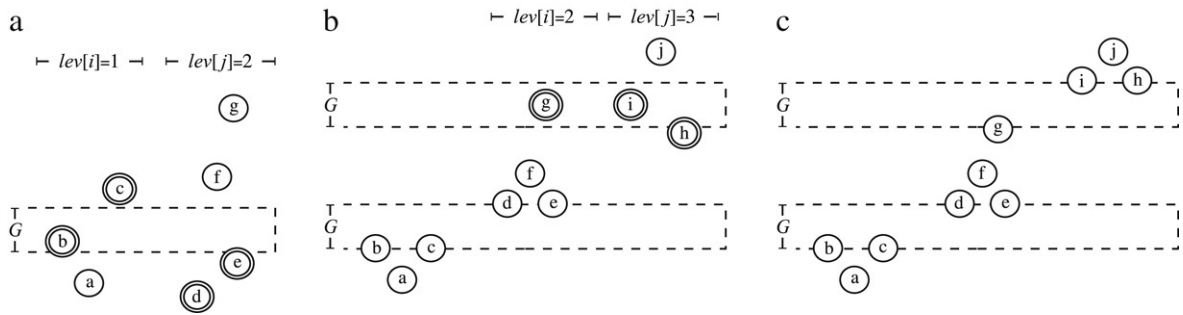


Fig. 2. An example of the *project* procedure. In (a) we consider the boundary between nodes i where $lev[i] = 1$ and nodes j where $lev[j] = 2$. Alphabetical ordering of the node labels corresponds to the order of their references in the sorted array q . Therefore, the array $p = [1, 4, 8]$ and for $k = 1$ the first nodes to be considered for inclusion in U_1 are $c(l = 3)$ and $d(u = 4)$. Nodes to be placed in U_1 in order to satisfy the constraints $x_i \leq x_j + G$ are highlighted. In (b) the nodes in U_1 have been moved to satisfy the level constraint and we show the next level to be considered. In (c) all level constraints have been satisfied. Note that in practice nodes are added to the set U_k one at a time and $posn U_k$ recalculated after each addition in order for U_k to be the minimal set required to satisfy the k th level constraint.

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 $x_a \leq x_b)$. For convenience we also keep an array $q[1] = p_0 < p_1 < \dots < p_m = n + 1$ of indices for the start of each partition excluding the first (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 compare against nodes in lower levels $below_k = \{l | 1 \leq q[l] < p_k\}$ and ensure that $x_u \geq x_l + G$ holds for all l and u . 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 position all nodes of U_k relative to a single point $posn U_k$. Since setting the derivative of Eq. (8) to 0 gives us the minimum, we take this point as the average of all positions (adjusted by required offsets which are a multiple of G) 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 $posn U_k$ 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 $posn U_k$ are computed*

$$posn U_k = \frac{1}{|U_k|} \left(\sum_{i \in U_k} x_i - G \cdot lev[i] \right) \tag{9}$$

and

$$U_k = \{l \in below_k \mid x_l > posn U_k\} \cup \{u \in above_k \mid x_u < posn U_k\}, \tag{10}$$

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

Proof. Eq. (9) follows directly from the algorithm and is invariant throughout the loop incrementally building U_k (since whenever U_k is expanded $posn U_k$ is recalculated).

The post-condition (10) implies that U_k includes all nodes that violate the internal constraints among $1, \dots, p_k - 1$ and $p_k, \dots, p_{k+1} - 1$. Without loss of generality we give the proof for the case when $G = 0$ (for $G \neq 0$ the situation is similar but more tedious to describe since positions calculated for nodes in U_k may be offset from $posn U_k$ rather than exactly $posn U_k$). 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 a position $posn U_l, l < k$). 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 $posn U_k$. 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 $posn U_k$.

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

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

```

procedure project( $x, lev$ )
   $q \leftarrow$  array of  $\{1 \leq i \leq n\}$  sorted by  $(lev[i], x_i)$ 
   $p \leftarrow$  array of indices indicating the start of each level
   $q[1] = p_0 < p_1 < \dots < p_m = n + 1$  (start of 2nd level is  $p_1$ )
  and  $lev[q[p_k]] = lev[q[p_k - 1]] + 1, 1 \leq k < m$ 
  for  $1 \leq k < m$  do
     $nextu \leftarrow p_k, nextl \leftarrow nextu - 1$ 
     $u \leftarrow q[nextu], l \leftarrow q[nextl]$ 
    if  $x_l + G > x_u$  then
      %  $below_k = \{l | 1 \leq q[l] < p_k\}, above_k = \{u | p_k \leq q[u] < p_{(k+1)}\}$ 
      % Find  $U_k = \{q[i] | 0 \leq nextl < i < nextu \leq n + 1\} \subseteq below_k \cup above_k$ 
       $sum \leftarrow x_l + x_u - G(lev[l] + lev[u]), w \leftarrow 2$ 
       $nextu \leftarrow nextu + 1, nextl \leftarrow nextl - 1$ 
      repeat
         $finished \leftarrow true$ 
         $posnU_k \leftarrow \frac{sum}{w}$ 
        if  $nextu < p_{(k+1)}$  then
           $u \leftarrow q[nextu]$ 
           $pos \leftarrow x_u - G \cdot lev[u]$ 
          if  $pos < posnU_k$  then
            % Add  $u$  to  $U_k$ 
             $nextu \leftarrow nextu + 1, w \leftarrow w + 1$ 
             $sum \leftarrow sum + pos$ 
             $finished \leftarrow false$ 
          end if
        end if
        if  $nextl \geq 1$  then
           $l \leftarrow q[nextl]$ 
           $pos \leftarrow x_l - G \cdot lev[l]$ 
          if  $pos > posnU_k$  then
            % Add  $l$  to  $U_k$ 
             $nextl \leftarrow nextl - 1, w \leftarrow w + 1$ 
             $sum \leftarrow sum + pos$ 
             $finished \leftarrow false$ 
          end if
        end if
      until  $finished$ 
      for  $nextl < i < nextu$  do
         $j \leftarrow q[i]$ 
         $x_j \leftarrow posnU_k + G \cdot lev[j]$ 
      end for
    end if
  end for
return  $x$ 

```

Fig. 3. Algorithm to project variables to the closest feasible position. The level for each node i is given by $1 \leq lev[i] \leq m$. An arbitrary gap G may be given to keep levels separated.

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 $x_l > posn U_k$. Now, if later we add $l' \in below_k$, then since $below_k$ is ordered by position, $x_{l'} \leq x_l$. By hypothesis, $x_l > posn U_k$ and since the new $posn U_k$ is the weighted average of $x_{l'}$ and $posn U_k$, we still have $x_l > posn U_k$. If later we add $u \in above_k$,

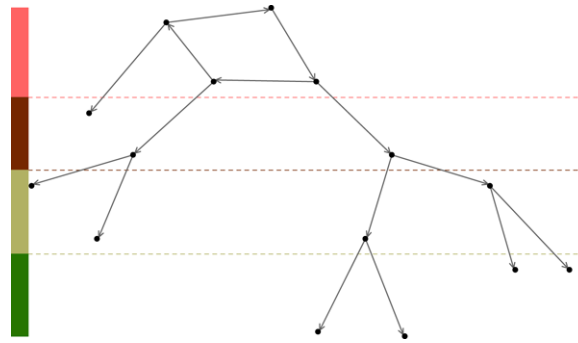


Fig. 4. A directed graph arranged using orthogonal ordering constraints in just the vertical dimension to preserve layering. The color bars on the left side indicate the layer-bands and the faint horizontal lines indicate the boundaries between these layers.

then since we are adding u we must have $x_u < \text{posn } U_k$. Now by hypothesis, $x_l > \text{posn } U_k$ and so $x_l > x_u$. Thus as for the previous case $x_l > \text{posn } U_k$. \square

Corollary 2. During execution of project (x, lev) after finishing the k th iteration in which U_k and its associated $\text{posn } U_k$ are computed

$$\text{posn } U_k = \frac{1}{|U_k|} \sum_{i \in U_k} (x_i - G \cdot \text{lev}[i]), \tag{11}$$

where the position of x_i is the input position (before the start of the projection operation).

Proof. Notice that unlike Eq. (9), 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 $\text{posn } U_k < \text{posn } U_l, l < k$ and therefore $U_k \supset U_l$ and $\text{posn } U_k$ will be calculated from $\text{posn } U_l$ for those nodes in U_l rather than their original positions. In this case (11) still holds as

$$\begin{aligned} \text{posn } U_k &= \frac{1}{|U_k|} \left(|U_l| \text{posn } U_l + \sum_{i \in U_k \setminus U_l} (x_i - G \cdot \text{lev}[i]) \right) \\ &= \frac{1}{|U_k|} \left(|U_l| \left(\frac{1}{|U_l|} \sum_{j \in U_l} (x_j - G \cdot \text{lev}[j]) \right) + \sum_{i \in U_k \setminus U_l} (x_i - G \cdot \text{lev}[i]) \right) \\ &= \frac{1}{|U_k|} \sum_{i \in U_k} (x_i - G \cdot \text{lev}[i]). \quad \square \end{aligned}$$

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

Lemma 3. 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, \dots, u_{m-1} be new variables, one for each partition k . We set values to the new variables by setting u_k to be $\max\{x_i \mid \text{lev}[i] = k\}$.

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 if there exist values for the Lagrange multipliers satisfying

$$\frac{\partial F}{\partial x} = \sum_{c \in C} \lambda_c \frac{\partial c}{\partial x} \tag{12}$$

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 [2].

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

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

These constraints are equivalent to the ordering constraints.

For clarity we assume, w.l.o.g, that $G = 0$. We show optimality by giving values for all λ_c satisfying Eq. (12). 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 first show that these satisfy Eq. (12). Consider some x_i . If x_i does not occur in an active constraint then we must show $\frac{\partial F}{\partial x_i} = 0$. Now

$$\frac{\partial F}{\partial x_i} = 2(x_i - x_i^0).$$

Since x_i does not occur in an active constraint we have $x_i = x_i^0$ and so this is trivially true.

Now consider the case when x_i occurs in an active constraint c of form $x_i \geq u_k$, i.e., $x_i - u_k \geq 0$. By construction x_i occurs in no other active constraints so we must show that $\frac{\partial F}{\partial x_i} = \lambda_c$ since $\frac{\partial c}{\partial x_i} = 1$. But this follows from the definition of λ_c . The case when x_i occurs in an active constraint c of form $x_i \leq u_k$ is dual.

Now consider the variable u_k . We must show that $\sum_{c \in C} \lambda_c \frac{\partial c}{\partial u_k} = 0$ since $\frac{\partial F}{\partial u_k} = 0$. Substituting for $\sum_{c \in C} \lambda_c \frac{\partial c}{\partial u_k}$ we have

$$\lambda_{u_{k-1} \leq u_k} - \lambda_{u_k \leq u_{k+1}} + \sum_{c \in C_{\leq}} \lambda_c - \sum_{c \in C_{\geq}} \lambda_c = 0, \tag{13}$$

where C_{\geq} is the set of active constraints of form $x_i \geq u_k$ and C_{\leq} the set of active constraints of form $x_i \leq u_k$.

For each constraint c of form $x_i \geq u_k \in C_{\geq}$, $\lambda_c = \frac{\partial F}{\partial x_i}$ and for each constraint c of form $x_i \leq u_k \in C_{\leq}$, $\lambda_c = -\frac{\partial F}{\partial x_i}$. Let us denote the respective sets of nodes by $V_{\geq} = \{i \mid x_i \geq u_k \text{ is active}\}$ and $V_{\leq} = \{i \mid x_i \leq u_k \text{ is active}\}$. Note that $V_{\geq} \cup V_{\leq} = U_k \setminus U_{k-1}$ and that V_{\geq} , V_{\leq} and U_{k-1} are disjoint. If $U_{k-1} \subseteq U_k$, $u_{k-1} \leq u_k$ is active and $\lambda_{u_{k-1} \leq u_k} = -\sum_{i \in U_{k-1}} \frac{\partial F}{\partial x_i}$. Thus

$$\lambda_{u_{k-1} \leq u_k} + \sum_{c \in C_{\leq}} \lambda_c - \sum_{c \in C_{\geq}} \lambda_c = -\sum_{i \in U_k} \frac{\partial F}{\partial x_i}.$$

On the other hand, if $U_{k-1} \not\subseteq U_k$, $u_{k-1} \leq u_k$ is not active and so $\lambda_{u_{k-1} \leq u_k} = 0$. Thus since $V_{\geq} \cup V_{\leq} = U_k$, again we have that

$$\lambda_{u_{k-1} \leq u_k} + \sum_{c \in C_{\leq}} \lambda_c - \sum_{c \in G} \lambda_c = -\sum_{i \in U_k} \frac{\partial F}{\partial x_i}.$$

Thus Eq. (13) holds if

$$-\sum_{i \in U_k} \frac{\partial F}{\partial x_i} - \lambda_{u_k \leq u_{k+1}} = 0. \tag{14}$$

There are two cases to consider. If $U_k \subseteq U_{k+1}$ then $u_k \leq u_{k+1}$ is active and by construction $\lambda_{u_k \leq u_{k+1}} = -\sum_{i \in U_k} \frac{\partial F}{\partial x_i}$. Thus (14) trivially holds. If $U_k \not\subseteq U_{k+1}$ then $u_k \leq u_{k+1}$ is not active and by construction $\lambda_{u_k \leq u_{k+1}} = 0$. Thus Eq. (14) holds if

$$\sum_{i \in U_k} \frac{\partial F}{\partial x_i} = 0 \tag{15}$$

and this is true if

$$\sum_{i \in U_k} 2(x_i - x_i^0) = 0.$$

But this follows from (11) since we have that for each maximal boundary k , i.e. k s.t. $U_k \not\subseteq U_{k+1}$, $\sum_{i \in U_k} x_i = \sum_{i \in U_k} x_i^0$.

We must now prove that for each active inequality c that $\lambda_c \geq 0$. Consider an active constraint c of form $x_i \geq u_k$. By construction

$$\lambda_c = \frac{\partial F}{\partial x_i} = 2(x_i - x_i^0).$$

For c to be active we have that $x_i^0 \leq x_i$, and so $\lambda_c \geq 0$. The case for an active constraint c of form $x_i \leq u_k$ is symmetric.

Now consider an active constraint c of form $u_k \leq u_{k+1}$. By construction

$$\lambda_c = - \sum_{i \in U_k} \frac{\partial F}{\partial x_i}.$$

We have seen that (11) implies that if x_i is placed at $posnU_k$ for all $i \in U_k$, then $\sum_{i \in U_k} \frac{\partial F}{\partial x_i} = 0$. Now, in case that $U_k \subset U_{k+1}$ for some $i \in U_k$ we may have $x_i \neq posnU_k$. In this case, it follows from (10) that if $U_k \subseteq U_{k+1}$, $posnU_k > posnU_{k+1}$. Thus, if $i \in U_k$ then $posnU_k \geq x_i$ and we get

$$\sum_{i \in U_k} \frac{\partial F}{\partial x_i} = \sum_{i \in U_k} 2(x_i - x_i^0) = 0 \leq 0$$

and so $\lambda_c \geq 0$. \square

We can now prove the correctness of *solve_QPOC*:

Theorem 4. *Solve_QPOC converges to an optimal solution to the input QPOC Problem.*

Proof. Lemma 3 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 Fig. 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 [2, 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 $s^k \rightarrow 0$ 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, $s^k \rightarrow 0$ 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. \square

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 [21]. 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

Table 1

A comparison of results obtained for arranging various graphs with *solve_QPOC* and the *Mosek* interior point method

Graph	#nodes (n)	#levels (m)	Solve_QPOC		Mosek	
			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	2.17	8517.3	13.0	8614.6
dwa512	512	14	1.23	22,464	37.7	22,464
dwb512	512	19	1.57	15,707	90.8	16,418
NSW rail	312	54/76 (x/y -axis)	4.92	2288	18.6	2274.5
Backbone	2603	2373/1805 (x/y -axis)	55.8	1,246,960	>1000	

Times are measured in seconds.



Fig. 5. The 1138bus graph (1138 nodes, 1458 edges) from the Matrix market collection [5], displayed as a directed graph.

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 [5] (Such as 1138bus as shown in Fig. 5) and some graphs based on geographic coordinates which are shown in Figs. 6 and 7.

5. Applications

5.1. Directed graph drawing

The method and motivation for drawing directed graphs by constrained majorization is discussed at length in [7]. 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. [25]. 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. [6]. 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 in [7] that this method produces drawings with 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. Fig. 4 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. Fig. 5 shows a much larger example from the matrix market collection [5].

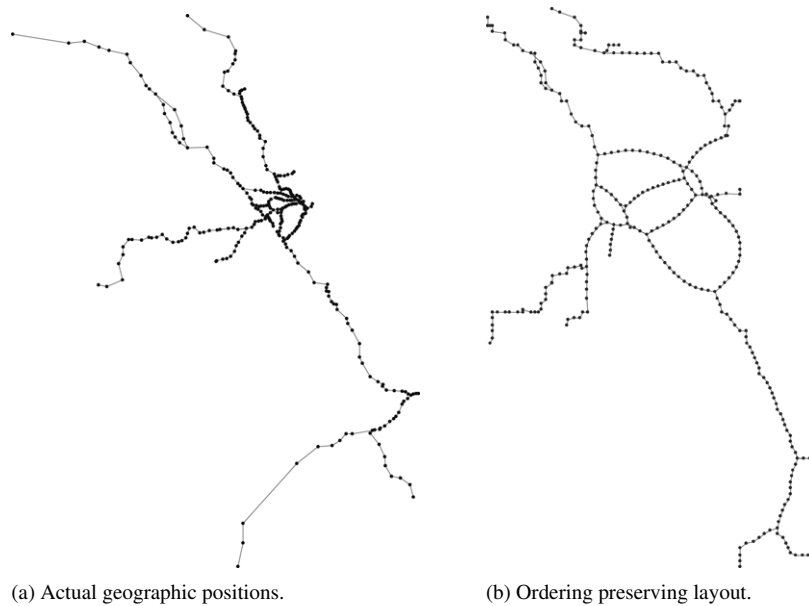


Fig. 6. The New South Wales rail network (312 nodes, 322 edges) shown with actual geographic positions (left) and then refined using stress minimization with orthogonal ordering constraints (right).

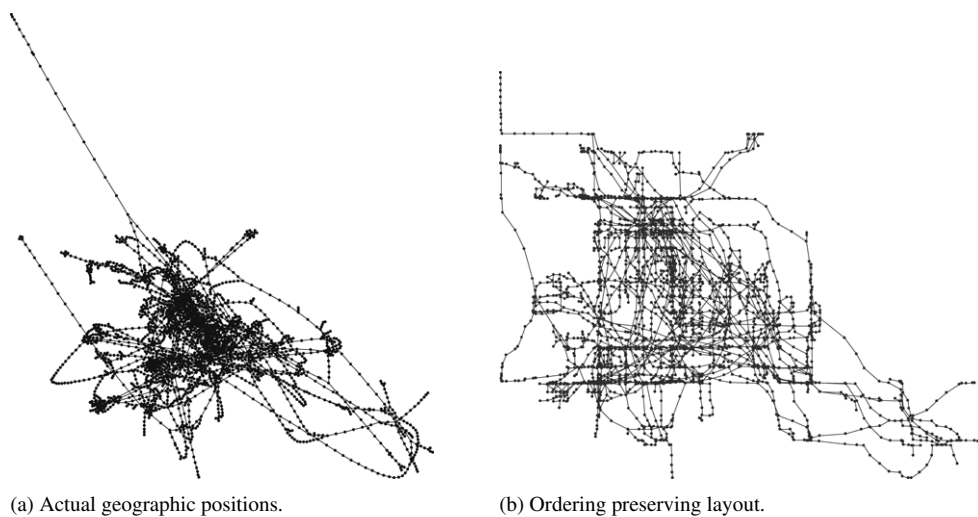


Fig. 7. A backbone network (2603 nodes, 2931 edges). Left picture is based on the actual geographic coordinates while the right picture is based on ordering-preserving constrained stress minimization.

5.2. Layouts preserving the orthogonal ordering

Sometimes a graph has meaningful coordinates associated with nodes. 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. [19]. 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 Fig. 6, 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 Fig. 7. The layout based on the original coordinates contains very dense areas. However, readability is vastly improved by minimizing the stress, while the original orthogonal order is preserved.

6. Extensions

6.1. Other algorithms

The gradient-projection method that we have detailed is not the only – or necessarily the fastest – method for solving the Quadratic Programming with Ordering Constraints (QPOC) problem. The algorithm we have given updates the values of all variables in each iteration, computing the desired location for each variable using the values from the previous iteration. A natural modification to this approach is in each iteration to cycle through the variables and compute the desired location for a variable from the current values of the other variables and then project on this single variable. In other words, to use a coordinate descent method [2]. The difference between the two approaches is analogous to that between the parallel update used in Fruchterman and Reingold [15] and the sequential update used in Kamada and Kawai [20]. We are currently investigating this approach [11].

Another approach we plan to investigate is a specialised interior point method. Efficient warm-start of interior point methods is still an open area of research – see for example Yildirim et al. [26] – but the simple nature of our constraints and the positive-semidefinite goal function, combined with the fact that at each iteration we begin from a feasible point, may make this particular problem amenable to warm start.

6.2. More expressive constraints

The simple level constraints that we have described so far can be solved almost as fast as solving the unconstrained quadratic form and – as we have shown – can be very useful in certain applications. However, an advantage of the gradient projection method is that since the constraints are entirely handled in the projection sub-problem, the algorithm can be relatively easily modified to handle more general convex linear constraints. The caveat is that an efficient method is required to solve the least squares regression QP in the projection step. In this section we briefly show how this may be accomplished for another class of constraints.

We call linear inequality constraints of the form $x_i + s_{ij} \leq x_j$ where s_{ij} is constant, *separation constraints* — since s_{ij} specifies a minimum margin required between x_i and x_j . Actually, the orthogonal-ordering constraints given in (7) are defined in terms of separation constraints, but they represent a very specific case where all variables in the same level share exactly the same constraints. In this section we consider QPs involving general (satisfiable) separation constraints over any subset of variables.

As an example of applying separation constraints to layout by constrained majorization we revisit the problem of arranging a directed graph. If the digraph is acyclic, we can simply define a separation constraint for each edge (u, v) to require that u be positioned above v . Certainly, this strategy cannot apply for cyclic digraphs, where such per-edge constraint will not be satisfiable. Hence, cyclic digraphs should be transformed into acyclic ones to ensure satisfiability of the constraints. Such transformations are well-known and standard in digraph drawing algorithms based on Sugiyama et al. [25]. Usually they are based on heuristics for reversing the direction of the minimal number of edges such that the resulting digraph will be acyclic. In the examples that follow we use the greedy heuristic suggested by Eades and Lin [14].

Fig. 8 shows two digraphs arranged using separation constraints. Note that in the small digraph shown in Fig. 8(a) the choice of which edge in the cycle to reverse is arbitrary depending on the precise heuristic used. This could be considered an artifact. However, a possible benefit of using per-edge separation constraints rather than level constraints is that various weakly connected subgraphs are ‘more free’ to move past one another vertically — the result is that more nearly ideal edge lengths are possible leading to a solution with lower stress and potentially far fewer edge crossings. Another benefit is that layouts using order-preserving constraints tend to produce drawings in which the nodes are layered while separation constraints do not. The issue is that these layers – while visually noticeable – do

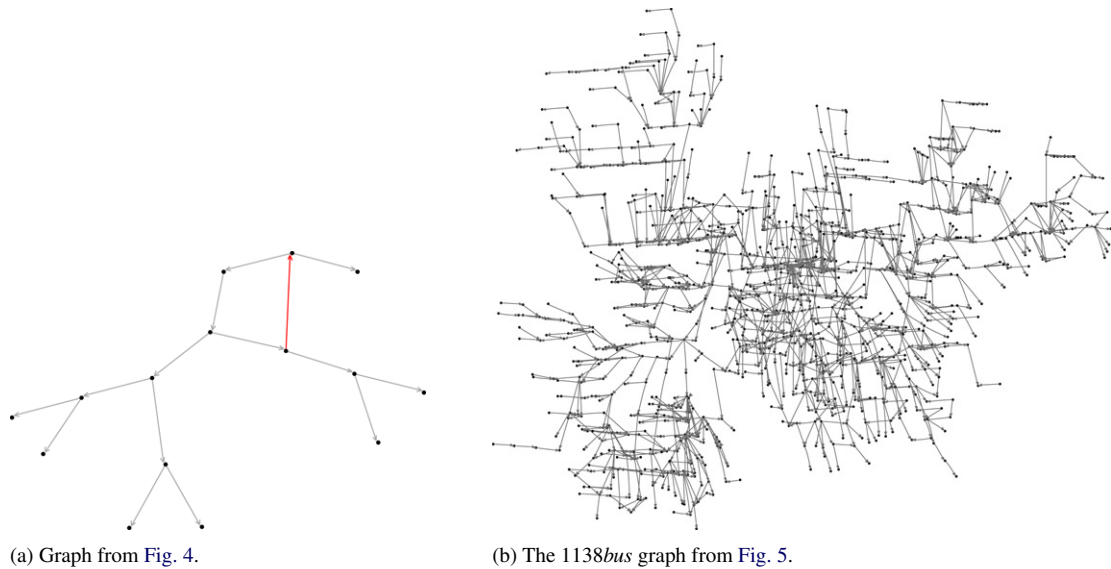


Fig. 8. Graphs from earlier figures arranged with a separation constraint for each edge to ensure all edges are downward pointing rather than level constraints as used previously.

not necessarily reflect anything important about the graph structure. This is evident in Fig. 8(b) where it is easier to make out weakly connected subgraphs than in the drawing of the same graph produced using level-constraints in Fig. 5. Note particularly, that the stress of this solution is 48,114 (compared with 74,434 for that in Fig. 5) and the number of crossings is 2367 (compared with 6374 in Fig. 5).

In [8] we defined the problem of adjusting a graph layout so that all overlaps between rectangular nodes are removed as the solution to a QP involving separation constraints. In fact that QP was the same type of sum-of-least-squares problem that is required in the projection step of a constrained majorization algorithm involving separation constraints. That is, the problem involved finding a solution that minimized the sum of squared displacements of all nodes — in the same way that in the projection step of gradient projection we need to displace each variable by as little as possible in order to satisfy the constraints.

The paper presented two algorithms to perform projection of separation constraints. The algorithm *solve_VPSC* performs an exact projection while the algorithm *satisfy_VPSC* performs an approximate projection but is considerably faster. As discussed in [9] *satisfy_VPSC* has worst case $O(n^2)$ runtime and typically $O(n \log n)$ time so the overall complexity of the stress-majorization is still comparable to the unconstrained case. In our example we have used *satisfy_VPSC*. For example Fig. 8(b) (with 1138 nodes, 1458 edges and therefore 1458 separation constraints) took 143 s to generate using this method. We believe that substantial time savings may be achieved by modifying these more complicated projection algorithms to make them incremental, preserving datastructures from one call to the next. Since submission of this manuscript we have made some progress in this regard, see [12].

7. Conclusions and further work

We have shown that a combination of stress majorization with appropriate quadratic programming techniques can be used for solving certain constrained layout problems. In particular, we found that orthogonal ordering constraints can be addressed without a significant increase of running time. The gradient-projection method that we have investigated with some rigour is significantly more efficient, in the context of stress majorization, compared with the standard interior-point methods used in most solvers.

The types of simple constraints and their application to graph drawing as discussed here are just representative of the possibilities afforded by this general approach to constrained force-directed graph drawing. We intend to further investigate these ideas and will try to find efficient ways to incorporate more general types of constraints into force-directed layout.

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.

References

- [1] R.E. Barlow, D.J. Bartholomew, J.M. Bremner, H.D. Brunk, *Statistical Inference under Order Restrictions*, John Wiley and Sons, London, 1972.
- [2] D.P. Bertsekas, *Nonlinear Programming*, 2nd edn, Athena Scientific, 1999.
- [3] I. Borg, P.J.F. Groenen, *Modern Multidimensional Scaling: Theory and Applications*, 2nd edn, Springer, 2005.
- [4] U. Brandes, D. Wagner, A bayesian paradigm for dynamic graph layout, in: Proc. 5th Int. Symp. Graph Drawing, GD'97, in: LNCS, vol. 1353, Springer, 1997, pp. 236–247.
- [5] R. Boisvert, R. Pozo, K. Remington, R. Barrett, J. Dongarra, The Matrix market: A web resource for test matrix collections, in: *Quality of Numerical Software, Assessment and Enhancement*, Chapman Hall, 1997, pp. 125–137.
- [6] L. Carmel, D. Harel, Y. Koren, Combining hierarchy and energy for drawing directed graphs, *IEEE Trans. Visualization Comput. Graphics* 10 (2004) 46–57.
- [7] T. Dwyer, Y. Koren, Dig-CoLa: Directed graph layout through constrained energy minimization, *IEEE Symposium on Information Visualization, Infovis'05*, 2005, pp. 65–72.
- [8] T. Dwyer, K. Marriott, P.J. Stuckey, Fast node overlap removal, in: Proc. 13th Int. Symp. Graph Drawing, GD'05, Springer, 2006, pp. 153–164.
- [9] T. Dwyer, K. Marriott, P.J. Stuckey, Fast node overlap removal (Addendum), Technical Report, Monash University. <http://www.csse.monash.edu.au/~tdwyer/FNORAddendum.pdf>.
- [10] T. Dwyer, Y. Koren, K. Marriott, Stress majorization with orthogonal ordering constraints, in: Proc. 13th Int. Symp. Graph Drawing, GD'05, Springer, 2006, pp. 141–152.
- [11] T. Dwyer, Y. Koren, K. Marriott, Drawing directed graphs using quadratic programming, *IEEE Trans. Visualization Comput. Graphics* 12 (4) (2006) 536–548.
- [12] T. Dwyer, Y. Koren, K. Marriott, IPSep-CoLa: an incremental procedure for separation constraint layout of graphs, *IEEE Trans. Visualization Comput. Graphics* 12 (5) (2006) 821–828.
- [13] P. Eades, M.L. Huang, Navigating clustered graphs using force-directed methods, *J. Graph Algorithms Appl.* 4 (12) (2000) 157–181.
- [14] P. Eades, X. Lin, A new heuristic for the feedback arc set problem, *Australian J. Combin.* 12 (1995) 15–26.
- [15] T. Fruchterman, E.M. Reingold, Graph drawing by force-directed placement, *Software — Practice Experience* 21 (1991) 1129–1164.
- [16] E. Gansner, Y. Koren, S. North, Graph drawing by stress majorization, in: Proc. 12th Int. Symp. Graph Drawing, GD'04, in: LNCS, vol. 3383, Springer, 2004, pp. 239–250.
- [17] W. He, K. Marriott, Constrained graph layout, in: Proc. 3rd Int. Symp. Graph Drawing, GD'96, in: LNCS, vol. 1190, Springer, 1996, pp. 217–232.
- [18] W. He, K. Marriott, Constrained graph layout, *Constraints* 3 (1998) 289–314.
- [19] S. Hong, D. Merrick, H. Nascimento, The metro map layout problem, in: Proc. 12th Int. Symp. Graph Drawing, GD'04, in: LNCS, vol. 3383, Springer, 2004, pp. 482–491.
- [20] T. Kamada, S. Kawai, An algorithm for drawing general undirected graphs, *Inform. Process. Lett.* 31 (1989) 7–15.
- [21] Mosek Optimization Toolkit V3.2. <http://www.mosek.com>.
- [22] K. Misue, P. Eades, W. Lai, K. Sugiyama, Layout adjustment and the mental map, *J. Visual Languages Comput.* 6 (1995) 183–210.
- [23] J. Nocedal, S. Wright, *Numerical Optimization*, Springer, 1999.
- [24] K. Ryall, J. Marks, S.M. Shieber, An interactive constraint-based system for drawing graphs, *ACM Sympos. User Interface Software Technol.* (1997) 97–104.
- [25] K. Sugiyama, S. Tagawa, M. Toda, Methods for visual understanding of hierarchical systems, *IEEE Trans. Syst. Man Cybernet.* 11 (1981) 109–125.
- [26] E.A. Yildirim, S.J. Wright, Warm-start strategies in interior-point methods for linear programming, *SIAM J. Optim.* 12 (3) (2002) 782–810.