Constrained Stress Majorization Using Diagonally Scaled Gradient Projection

Tim Dwyer and Kim Marriott

Clayton School of Information Technology,
Monash University, Clayton, Victoria 3800, Australia
{tdwyer,marriott}@infotech.monash.edu.au

Abstract. Constrained stress majorization is a promising new technique for integrating application specific layout constraints into force-directed graph layout. We significantly improve the speed and convergence properties of the constrained stress-majorization technique for graph layout by employing a diagonal scaling of the stress function. Diagonal scaling requires the active-set quadratic programming solver used in the projection step to be extended to handle separation constraints with scaled variables, i.e. of the form $s_i y_i + g_{ij} \leq s_j y_j$. The changes, although relatively small, are quite subtle and explained in detail.

Keywords: constraints, graph layout.

1 Introduction

Researchers and practitioners in various fields have been arranging diagrams automatically using physical "mass-and-spring" models since at least 1965 [1]. Typically, the objective of such *force-directed* techniques is to minimize the difference between actual and ideal separation of nodes [2], for example:

$$stress(X) = \sum_{i<j} w_{ij}(||X_i - X_j|| - d_{ij})^2 \tag{1}$$

where w_{ij} is $\frac{1}{{d_{ij}}^2}$, X_i gives the placement in two or more dimensions of the i^{th} node and d_{ij} is the ideal distance between nodes i and j based on the graph path length between them.

Recently, the force-directed model has been extended to allow *separation constraints* of the form $u + g \leq v$, enforcing a minimum gap g between the positions u and v of pairs of objects in either the x or y dimensions in the drawing [4]. The basic idea is to modify the iterative step in *stress majorization* [5, Ch. 8] to solve a one-dimensional quadratic objective subject to the separation constraints for that dimension. Separation constraints allow a wide variety of aesthetic requirements—such as downward-pointing edges in directed graphs, alignment or distribution of nodes, placement of nodes in horizontal or vertical bands, non-overlap of nodes, orthogonal ordering between nodes, containment of nodes in clusters, containment in a page, and edge straightening without

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 219–230, 2007.

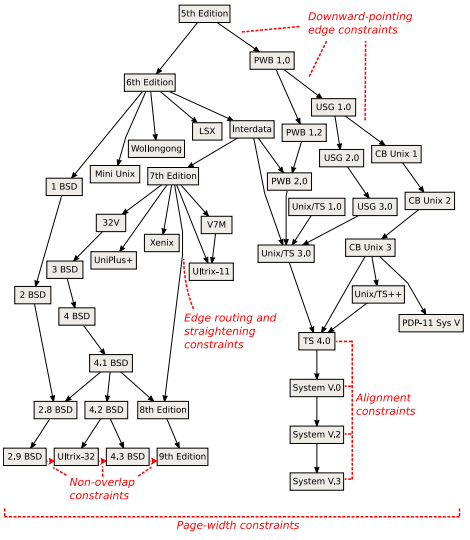


Fig. 1. Drawing of a directed graph illustrating the flexibility of constrained stress majorization. Separation constraints encode the aesthetic requirements that: (1) directed edges point downwards; (2) selected nodes are horizontally or vertically aligned; (3) the drawing fits within the page boundaries; and (4) nodes do not overlap edges or other nodes. The "history of unix" graph data is from `http://www.graphviz.org` and this drawing originally appeared in [3].

introduction of additional edge crossings–to be integrated into force-directed layout [4]. Thus, *constrained stress majorization* provides an extremely flexible basis for handling application specific layout conventions and requirements.

In majorization the value of the stress function (1) is reduced by alternately minimizing quadratic functions in the horizontal and vertical axes that bound the stress functions. These quadratic functions have the form:

$$f(x) \equiv \frac{1}{2}x^T Qx + x^T b \tag{2}$$

where, for a graph with n nodes, x is the n dimensional vector of node positions in the current dimension; the $n \times n$ Hessian matrix Q is the graph Laplacian (see below); and the linear term b is computed before processing each axis based on the difference between ideal separation of nodes and their actual separation at the current placement (for details see [6]). Constrained stress majorization extends this by additionally requiring that the solution returned satisfies the separation constraints for that dimension.

In [4] we gave a specialized gradient-projection-based method for solving this particular kind of quadratic program (QP) which was significantly faster than standard QP algorithms. However, gradient projection (GP) based methods, like other iterative optimization methods based on steepest descent, can display poor convergence when working with badly conditioned Hessian matrices. A standard

technique to improve convergence is to scale the variables so that the diagonal entries of the scaled Hessian matrix are all equal. This works particularly well if the Hessian, with entries Q_{ij}, is diagonally dominant, i.e. $|Q_{ii}| \geq \sum_{j \neq i} |Q_{ij}|$, which the graph Laplacian is by its definition:

$$Q_{ij} = \begin{cases} -w_{ij} & i \neq j \\ \sum_{k \neq i} w_{ij} & i = j \end{cases}$$

The main contribution of this paper is to demonstrate that using such diagonal scaling with GP is nearly twice as fast as the original unscaled GP algorithm and, even more importantly, the rate of convergence is more robust. The main technical difficulty is the need to modify the projection step to handle constraints of the form $s_i x_i + g \leq s_j x_j$ where s_i and s_j are the positive scaling factors for x_i and x_j, respectively. We detail the necessary modifications to the projection algorithm. Although these modifications are quite subtle, they make little difference to the implementation difficulty. Thus, there seems no reason to use the original unscaled GP algorithm in preference to the GP algorithm with diagonal scaling presented here. Another contribution of the paper is to provide more details of the gradient projection algorithm presented in [4].

2 Diagonally-Scaled Gradient Projection

The core step in constrained stress majorization is to solve a quadratic program with an objective of the form given in Equation 2 subject to some separation constraints $c \in C$ on the variables where each separation constraint c is of form $x_i + g \leq x_j$ where g is the minimum gap between the variables x_i and x_j. We call this the Quadratic Programming with Separation Constraints (QPSC) problem.

Previously we gave an iterative *gradient-projection* algorithm for solving a QPSC problem [4]. This works by first decreasing $f(x)$, by moving x in the direction of steepest descent, i.e. opposite to the gradient $\nabla f(x) = Qx + b$. While this guarantees that—with appropriate selection of step-size α—the stress is decreased by this first step, the new positions may violate the constraints. This is corrected by applying the function *project*, which returns the closest point $\bar{x}$ to x which satisfies the separation constraints, i.e. it projects x on to the feasible region. A vector p from the initial position $\hat{x}$ to $\bar{x}$ is then computed and the algorithm ensures monotonic decrease in stress when moving in this direction by computing a second stepsize $\beta = \arg\min_{\beta \in [0,1]} f(x + \beta p)$ which minimizes stress in this interval.

Unfortunately, GP-based methods, like other iterative methods based on steepest descent, can display poor convergence when working with poorly conditioned Hessian matrices. One remedy is to perform a linear scaling on the problem. The basic idea is to use an $n \times n$ scaling matrix S and transform the problem into one on new variables y s.t. $x = Sy$.

If we choose $S = Q^{-1} = (\nabla^2 f(x))^{-1}$ then steepest descent on the transformed problem is equivalent to performing Newton's method on the original problem.

Thus, at least in the unconstrained problem convergence will be quadratic. However, computing the inverse of Q is quite expensive and it also means that scaling of the separation constraints results in full-fledged linear constraints, so that the projection operation becomes considerably more complex and expensive.

Thus, an approach which approximates Q^{-1} is often used in practice [7]. Specifically, we choose S to be a diagonal matrix such that the diagonal entries in S^TQS are all 1, i.e. $S_{ii} = \frac{1}{\sqrt{Q_{ii}}}$ and for $i \neq j$, $S_{ij} = 0$. This is called diagonal scaling. We refer below to the diagonal entries in S as $s_i = S_{ii}$. Note that for all i, $s_i > 0$ and, clearly, S is very quick to compute.

It is straightforward to change the main gradient-projection routine, *solve_QPSC*, from [4] to use diagonal scaling. The modified routine is given in Fig. 2.

The chief difficulty is modifying the projection routine *project* called by *solve_QPSC*. We have that $x_i = s_i y_i$ so a separation constraint of form $x_i + g \leq x_j$ becomes, in the scaled space, $s_i y_i + g \leq s_j y_j$. We call such linear inequalities *positively scaled separation constraints*.

After computing an unconstrained descent direction the scaled GP algorithm calls *project* to find the nearest point to $d = \hat{y} - \alpha g$ satisfying the positively scaled separation constraints C'. That is, it must solve:

$$\begin{array}{l}\text{minimize } F(y) = \sum_{i=1}^{n}(y_i - d_i)^2 \\ \text{subject to positively scaled separation constraints } C'\end{array}$$

In [4] we described an *active-set* algorithm for incrementally finding a solution to the projection problem subject to (unscaled) separation constraints. Here we extend this to handle positively scaled separation constraints. Although the changes are minor, they are quite subtle. The complete algorithm is given in Fig. 2. Note that if c is a positively scaled separation constraint of form $su + g \leq tv$ we refer to u, v, s, t and g by lv_c, rv_c, ls_c, rs_c, gap_c, respectively.

The method works by building up *blocks* of variables spanned by a tree of *active* (or set at equality) constraints. At any point in time the block to which a variable y_i belongs is given by blk_i. If a block has k variables the tree of active constraints has $k - 1$ linear equations so variable elimination can be used to eliminate all but one variable and the position of all other variables is a linear function of that single unknown *reference variable*. This contrasts to the unscaled case in which the variables are simple offsets from the reference variable and are not scaled.

For each block B the algorithm keeps: the set of variables V_B in the block; the set of active constraints C_B; the current position Y_B of the block's reference variable; and the scaling factor S_B for the reference variable. For each variable y_i in block $B = blk_i$ we have a variable dependent scaling factor a_i and offset b_i giving its position relative to Y_B, i.e. it is an invariant that $y_i = a_i Y_{blk_i} + b_i$. As we shall see it is also an invariant that $a_i = \frac{S_{blk_i}}{s_i}$.

Each block B is placed at the position minimizing $F = \sum_{i \in V_B}(y_i - d_i)^2$ subject to the active constraints C_B. Now,

$$\frac{\partial F}{\partial Y_B} = \sum_{i \in V_B} \frac{\partial y_i}{\partial Y_B} \frac{\partial F}{\partial y_i} = \sum_{i \in V_B} a_i \frac{\partial F}{\partial y_i} = \sum_{i \in V_B} a_i(2(y_i - d_i)) = \sum_{i \in V_B} 2a_i(a_i Y_B + b_i - d_i)$$

The minimum occurs when $\frac{\partial F}{\partial Y_B} = 0$ so the optimum value is $Y_B = \frac{AD_B - AB_B}{A2_B}$ where $AD_B = \sum_{i \in V_B} a_i d_i$, $AB_B = \sum_{i \in V_B} a_i b_i$, and $A2_B = \sum_{i \in V_B} a_i^2$.

Initially, each variable y_i is placed in its own block B_i where it is the block's reference variable. This is done in the procedure *init_blocks* called at the start of *solve_QPSC*. After this the blocks persist between the calls to *project* and are incrementally modified in the routine *project*.

The function *project*(C, d) works as follows. First the routine *split_blocks* updates the position of each block B to reflect the changed value of d. The routine then splits the block if this will allow the solution to be improved. The procedure *split_block* is straightforward. The only point to note is that we define $left(c, B)$ to be the variables in V_B connected to the variable lv_c by constraints in $C_B \setminus \{c\}$ and we define $right(c, B)$ symmetrically.

Determining where and when to split a block is a little more difficult. It is formalized in terms of Lagrange multipliers. Recall that if we are minimizing function F with a set of convex equalities C over variables y, then we can associate a variable λ_c called the Lagrange multiplier with each $c \in C$. Given a configuration y^* feasible with respect to C we have that y^* is a locally minimal solution iff there exist values for the Lagrange multipliers satisfying for each y_i that

$$\frac{\partial F}{\partial y_i}(y^*) = \sum_{c \in C} \lambda_c \frac{\partial c}{\partial y_i}(y^*) \tag{3}$$

Furthermore, if we also allow inequalities, the above statement continues to hold as long as $\lambda_c \geq 0$ for all inequalities c of form $t \geq 0$. By definition an inequality c which is not active has $\lambda_c = 0$. Thus we need to split a block at active constraint c if $\lambda_c < 0$ since this tells us that by moving the two sub-blocks apart we can improve the solution.

One key to the efficiency of the projection algorithm is that the Lagrange multipliers can be computed efficiently for the active constraints in a block in linear time using the procedure *comp_dfdv*. The justification for this is the following lemma which is proved in [8]:

Lemma 1. *Let y^* place all blocks at their optimum position. If c is an active constraint in block B then*

$$\lambda_c \quad = \quad - \sum_{k \in left(c,B)} \frac{1}{s_k} \frac{\partial F}{\partial y_k}(y_k^*) \quad = \quad \sum_{k \in right(c,B)} \frac{1}{s_k} \frac{\partial F}{\partial y_k}(y_k^*)$$

Of course, after moving the blocks to their new location and perhaps splitting some blocks, there is no guarantee that the placement satisfies all of the constraints. Thus, after splitting the procedure *project* repeatedly modifies the blocks until a feasible solution is reached. The constraints are processed in decreasing order of violation until no more violated constraints are found and therefore a feasible solution has been obtained.

procedure *solve_QPSC*($Q,\ b,\ C,\ x$)
$s \leftarrow (\frac{1}{\sqrt{Q_{11}}}, \frac{1}{\sqrt{Q_{22}}}, \ldots, \frac{1}{\sqrt{Q_{nn}}})$
$S \leftarrow n \times n$ diagonal matrix with $S_{ii} = s_i$
global $y \leftarrow Sx$
init_blocks()
$Q' \leftarrow S^T Q S$
$b' \leftarrow Sb$
$C' \leftarrow \{s_i y_i + g \leq s_j y_j | (x_i + g \leq x_j) \in C\}$
repeat
  $g \leftarrow Q'y + b'$
  $\alpha \leftarrow \frac{g^T g}{g^T Q' g}$
  $\hat{y} \leftarrow y$
  $d \leftarrow \hat{y} - \alpha g$
  $nosplit \leftarrow project(C', d)$
  $\bar{y} \leftarrow y$ (y modified by *project*)
  $p \leftarrow \hat{y} - \bar{y}$
  $\beta \leftarrow \min(\frac{g^T d}{d^T Q' p}, 1)$
  $y \leftarrow \hat{y} - \beta p$
until $\|\hat{y}, y\|$ sufficiently small and $nosplit$
return $S^{-1}y$

function *project*(C, d)
$nosplit \leftarrow split\_blocks(d)$
$c \leftarrow \max_{c \in C}\ violation(c)$
while $violation(c) \geq 0$ **do**
  if $blk_{lv_c} \neq blk_{rv_c}$ **then**
    merge_block(c)
  else *expand_block*(c)
  $c \leftarrow \max_{c \in C}\ violation(c)$
return $nosplit$

procedure *init_blocks*()
for $i = 1, ..., n$ **do**
  let B_i be a new block s.t.
  $V_{B_i} \leftarrow \{i\}$
  $Y_{B_i} \leftarrow y_i$
  $S_{B_i} \leftarrow s_i$
  $AD_{B_i} \leftarrow y_i$
  $A2_{B_i} \leftarrow 1$
  $AB_{B_i} \leftarrow 0$
  $C_{B_i} \leftarrow \emptyset$
  $a_i \leftarrow 1$
  $b_i \leftarrow 0$
  $blk_i \leftarrow B_i$
return

procedure *split_blocks*(d)
$nosplit \leftarrow true$
for each active block B **do**
  $AD_B \leftarrow \sum_{i \in V_B} a_i d_i$
  $AB_B \leftarrow \sum_{i \in V_B} a_i b_i$
  $A2_B \leftarrow \sum_{i \in V_B} a_i^2$
  $Y_B \leftarrow \frac{AD_B - AB_B}{A2_B}$
  for $i \in V_B$ **do**
    $y_i \leftarrow a_i Y_B + b_i$
  for each $c \in C_B$ **do** $\lambda_c \leftarrow 0$
  choose $v \in V_B$
  $comp\_dfdv(v, C_B, NULL)$
  $sc \leftarrow \min_{c \in C_B} \lambda_c$
  if $\lambda_c \geq 0$ **then break**
  $nosplit \leftarrow false$
  $split\_block(c)$
return $nosplit$

function *violation*(c) =
let $c \equiv s_i y_i + g \leq s_j y_j$ in
$s_j y_j - (s_i y_i + g)$

procedure *merge_block*(c)
let $c \equiv s_i y_i + g \leq s_j y_j$
$LB \leftarrow blk_i$
$RB \leftarrow blk_j$
for $k \in V_{RB}$ **do**
  $blk_k \leftarrow LB$
  $a_k \leftarrow S_{LB}/s_k$
  $b_k \leftarrow b_k + g$
  $AB_{LB} \leftarrow AB_{LB} + a_k b_k / s_k$
  $AD_{LB} \leftarrow AD_{LB} + a_k d_k$
  $A2_{LB} \leftarrow AD_{LB} + a_k a_k$
$Y_{LB} \leftarrow \frac{AD_{LB} - AB_{LB}}{A2_{LB}}$
$C_{LB} \leftarrow C_{LB} \cup C_{RB} \cup \{c\}$
$V_{LB} \leftarrow V_{LB} \cup V_{RB}$
for $i \in V_{LB}$ **do**
  $y_i \leftarrow a_i Y_B + b_i$
return

procedure *expand_block*($\tilde{c}$)
$B \leftarrow blk_{lv_{\tilde{c}}}$
for each $c \in C_B$ **do** $\lambda_c \leftarrow 0$
$comp\_dfdv(lv_{\tilde{c}}, C_B, NULL)$
$[v_1, ..., v_k] := comp\_path(lv_{\tilde{c}}, rv_{\tilde{c}}, C_B)$
$ps \leftarrow \{c \in C_B \mid \exists j \text{ s.t. } lc_c = v_j \text{ and } rc_c = v_{j+1}\}$
if $ps = \emptyset$ **then error** % constraints unsatisfiable
$sc \leftarrow \min_{c \in ps} \lambda_c$
$split\_block(sc)$
$merge\_block(\tilde{c})$
return

procedure *split_block*(c)
$B \leftarrow blk_{lv_c}$
let RB be a new block s.t.
$S_{RB} \leftarrow S_B$
$V_{RB} \leftarrow left(c, B)$
$C_{RB} \leftarrow \{c' \in C_B \mid lv_{c'}, rv_{c'} \in V_{RB}\}$
for $i \in V_{RB}$ **do** $blk_i \leftarrow RB$
$AD_{RB} \leftarrow \sum_{i \in V_{RB}} a_i d_i$
$AB_{RB} \leftarrow \sum_{i \in V_{RB}} a_i b_i$
$A2_{RB} \leftarrow \sum_{i \in V_{RB}} a_i^2$
$Y_{RB} \leftarrow \frac{AD_{RB} - AB_{RB}}{A2_{RB}}$
for $i \in V_{RB}$ **do** $y_i \leftarrow a_i Y_{RB} + b_i$
let LB be a new block s.t.
symmetric construction to RB
return

function *comp_dfdv*($i, AC, \tilde{c}$)
$dfdv \leftarrow \frac{2}{s_i}(y_i - d_i)$
for each $c \in AC$ s.t. $i = lv_c$ and $c \neq \tilde{c}$ **do**
  $\lambda_c \leftarrow comp\_dfdv(rv_c, AC, c)$
  $dfdv \leftarrow dfdv + \lambda_c$
for each $c \in AC$ s.t. $i = rv_c$ and $c \neq \tilde{c}$ **do**
  $\lambda_c \leftarrow -comp\_dfdv(lv_c, AC, c)$
  $dfdv \leftarrow dfdv - \lambda_c$
return $dfdv$

Fig. 2. Diagonal scaling Gradient-Projection-based algorithm to find an optimal solution to a QPSC problem with variables $x_1, \ldots, x_n$, symmetric positive-semidefinite matrix Q, vector b and separation constraints C over the variables

If a constraint c is violated there are two cases. Either the variables in c, lv_c and rv_c, belong to different blocks, in which case *merge_block* is used to merge the two blocks, or else lv_c and rv_c, belong to the same block, in which case *expand_block* is used to modify the block.

The code for *merge_block* is relatively straightforward. If the merge is because of the violated constraint $c \equiv s_i y_i + g \leq s_j y_j$ then it merges the block $RB = blk_j$ into block $LB = blk_i$ (the direction is arbitrary and in practice we always move variables from the smaller to the larger block). The reference variable Y_{LB} becomes the reference variable of the new block. Now, rewriting the active version of c, $s_j y_j = s_i y_i + g$, in terms of Y_{LB} and Y_{RB} gives $s_j(a_j Y_{RB} + b_j) = s_i(a_i Y_{LB} + b_i) + g$. Thus,

$$Y_{RB} = \frac{s_i a_i}{s_j a_j} Y_{LB} + \frac{s_i b_i - s_j b_j + g}{s_j a_j} = \frac{S_{LB}}{S_{RB}} Y_{LB} + \frac{s_i b_i - s_j b_j + g}{S_{RB}}.$$

Taking $a = \frac{S_{LB}}{S_{RB}}$ and $b = \frac{s_i b_i - s_j b_j + g}{S_{RB}}$, we can express the variables of RB in terms of the reference variable $Y_B = Y_{LB}$ as:

$$y_k = a_k Y_{RB} + b_k = a_k(aY_{LB} + b) + b_k = a'_k Y_{LB} + b'_k$$

where

$$a'_k = (a_k a) = \frac{S_{RB}}{s_k}\frac{S_{LB}}{S_{RB}} = \frac{S_{LB}}{s_k} = \frac{S_B}{s_k}$$

and $b'_k = a_k b + b_k$.

The procedure *expand_block*$(b, \tilde{c})$ is probably the most complex part of the algorithm. It deals with a case where a previously constructed block now causes a constraint $\tilde{c}$ between two variables in the block to be violated. To fix this we must identify where to split the current block and then rejoin the sub-blocks using $\tilde{c}$, in effect expanding the block to remove the violation by choosing a different spanning tree of active constraints for the block. To do so, the algorithm computes the best constraint *sc* in the active set on which to split based on its Lagrange multiplier, λ_c. The intuition for this is that the value of λ_c gives the rate of increase of the goal function as a function of c_{gap}. Thus, the smaller the value of λ_c the better it is to split the block at that constraint. However, not all constraints in the active set are valid points for splitting. Clearly we must choose a constraint that is on the path between the variables $lv_{\tilde{c}}$ and $rv_{\tilde{c}}$. The call to the function *comp_path* returns the list of variables $[v_1, ..., v_k]$ on this path. Furthermore, to be a valid split point the constraint c must be oriented in the same direction as $\tilde{c}$, i.e. for some j, $lc_c = v_j$ and $rc_c = v_{j+1}$. If there are no such constraints then the constraints (and the original separation constraints) are infeasible so the algorithm terminates with an error. Otherwise, the split constraint *sc* is simply the valid split constraint with the least Lagrange multiplier. The remainder of *expand_block* uses *split_block* to split the block by removing *sc* from the active set C_B and then uses *merge_block* to rejoin the two sub-blocks with constraint $\tilde{c}$.

Clearly, *project* will only terminate if either no constraints are violated, or *expand_block* terminates with an error. We show that if *expand_block* gives rise

to an error then the original separation constraints are unsatisfiable. It gives rise to an error if there is a scaled constraint $\tilde{c}$ of form $s'_1 v_1 + g \leq s'_n v_n$ and a path of active constraints from v_1 to v_n of form

$$s'_2 v_2 + g_1 \leq s'_1 v_1, s'_3 v_3 + g_2 \leq s'_2 v_2, ..., s'_n v_n + g_{n-1} \leq s'_{n-1} v_{n-1}$$

since the orientation of the constraints is opposite that of $\tilde{c}$. Thus, a consequence of the path constraints is that $s'_n v_n + g' \leq s'_1 v_1$ where $g' = \sum_{i=1}^{n-1} g_i$. The current placement of v_1 and v_n satisfies $s'_n v_n + g' = s'_1 v_1$ but does not satisfy $s'_1 v_1 + g \leq s'_n v_n$. Thus $s'_n v_n + g' = s'_1 v_1$ and $s'_1 v_1 + g > s'_n v_n$ and so $s'_n v_n + g' + g > s'_n v_n$. Thus, $g + g' > 0$ and so the original scaled constraints are unsatisfiable since $s'_n v_n + g' \leq s'_1 v_1$ and $g + g' > 0$ implies $s'_1 v_1 + g > s'_n v_n$ which contradicts $s'_1 v_1 + g \leq s'_n v_n$. This also means the original separation constraints are unsatisfiable since we have in the unscaled space $x'_1 + g \leq x'_n$ and $x'_1 + g' \geq x'_n$.

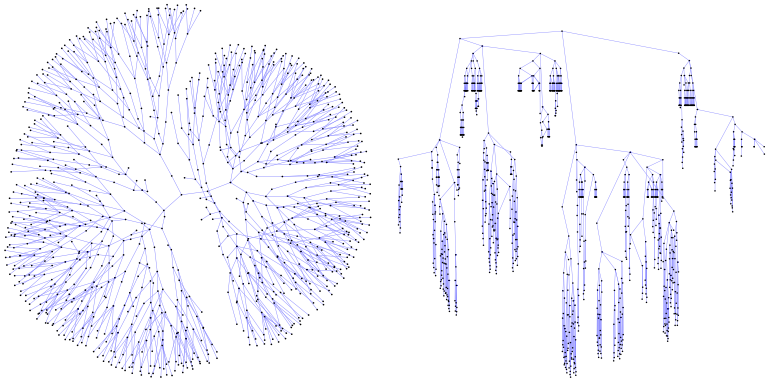
Thus, *project* always returns a feasible solution if one exists. The feasible solution is optimal in the case that *nosplit* is true and the solution has not changed. Thus although the call to *project* is not initially guaranteed to return the optimal solution it will converge towards it. Using this it is relatively straightforward to show that *solve_QPSC* converges towards the optimal solution.

Unfortunately, as for the unscaled gradient projection algorithm, we have yet to provide a formal proof of termination of the *project* function, though we conjecture that it does always terminate. The potential problem is that a constraint may be violated, added to the active set, then removed from the active set due to block expansion, and then re-violated because of other changes to the block. Note that we have tried thousands of very different examples and have never encountered non-termination.

Another potential source of non-termination, which arises in most active set approaches, is that it may be possible for the algorithm to cycle by removing a constraint because of splitting, and then be forced to add the constraint back again. This can only occur if the original problem contains constraints that are redundant in the sense that the set of equality constraints corresponding to the separation constraints C, namely $\{u + a = v \mid (u + a \leq v) \in C\}$, contains redundant constraints. We could remove such redundant separation constraints in a pre-processing step by adding ϵ^i to the gap for the i^{th} separation constraint or else use a variant of lexico-graphic ordering to resolve which constraint to make active in the case of equal violation. We can then show that cycling cannot occur. In practice however we have never found a case of cycling.

3 Results

To investigate the effect of diagonally scaled gradient projection on running time and convergence of constrained stress-majorization layout, we compared it against a number of other optimization methods for various graphs with a range of degree distributions. Table 3 gives results in terms of running times and numbers of iterations for a selection of graphs all of size around $|V| = 1000$. The optimization methods tested were:



(a) Unconstrained (b) Constrained

Fig. 3. A randomly generated tree as used in our tests, with 1071 nodes of varying degree, drawn with and without constraints. The vertical constraints enforce downward-pointing edges while the horizontal constraints are simply generated by an in-order traversal of the tree.

CG. Unconstrained conjugate gradient (as recommended by [6] for (unconstrained) functional majorization).

Int. Pnt. A commercially available QP solver based on the interior point method (Mosek[1]).

Unscaled GP. Gradient projection without scaling.

Scaled GP. Gradient projection with scaling.

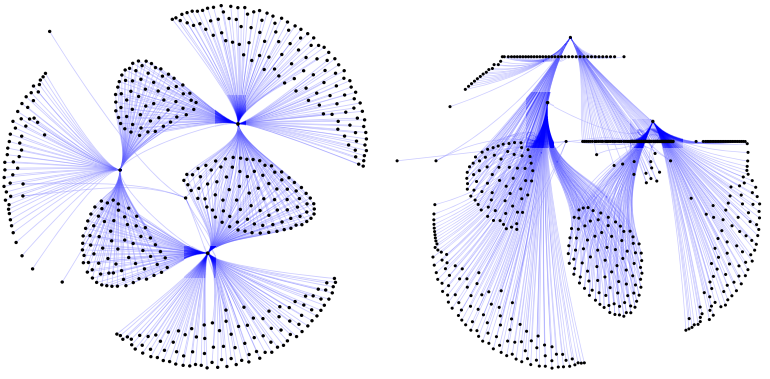
Four different graphs were chosen with a range of different node-degree distributions. The graphs were a randomly generated tree with $|V| = 1071$ and node degree ranging from 1 to 4 (Fig. 3); an Erdõs–Rényi random graph of poisson degree distribution [9] and $|V| = 1000$; a random graph with power-law degree distribution generated using the Barabási–Albert model [10] (e.g. Fig. 4); and a graph from the Matrix Market[2] that we have used before in performance testing of constrained layout methods [4].

For all methods except CG (which can not easily be extended to handle constraints) we ran both with and without a basic set of downward pointing edge constraints [4]. For the tree graph we also included ordering constraints over the x-node positions based on a simple in-order traversal of the graph. The constraints were chosen to be simple to generate, easy to visually verify, and to be similar to the types of constraints that might be useful in practical layout situations.

Numbers of stress-majorization iterations are given for each graph, with and without constraints. These are the same across all solvers since each solves the quadratic-program subproblems to optimality. For CG and GP solver methods we also give the total number of iterations required. This helps to explain the

[1] `http://www.mosek.org`

[2] `http://math.nist.gov/MatrixMarket/`



(a) Unconstrained (b) Constrained

Fig. 4. A randomly generated scale-free graph as used in our tests. It has 500 nodes with power-law distribution of degree and is drawn with and without vertical downward-pointing edge constraints.

Table 1. Results of applying stress majorization using various different techniques to solve the quadratic problems at each iteration

Graph	Constraints Hor.	Vert.	Solver	Stress Maj. Iterations	Total Iterations	Total time (secs)
Random	0	0	CG	48	646	8.82
Tree	0	0	Int. Pnt.	48	N/A	42.69
	0	0	Unscaled GP	48	1607	19.97
$\|V\| = 1071$	0	0	Scaled GP	48	833	13.88
	1070	1070	Int. Pnt.	38	N/A	341.69
	1070	1070	Unscaled GP	38	2650	33.12
	1070	1070	Scaled GP	38	1071	17.31
Poisson	0	0	CG	83	908	12.52
random	0	0	Int. Pnt.	83	N/A	62.17
	0	0	Unscaled GP	83	1907	23.51
$\|V\| = 1000$	0	0	Scaled GP	83	1244	19.34
	0	1478	Int. Pnt.	46	N/A	175.93
	0	1478	Unscaled GP	46	2336	20.88
	0	1478	Scaled GP	46	1717	15.81
Power-law	0	0	CG	91	983	13.45
random	0	0	Int. Pnt.	91	N/A	68.21
	0	0	Unscaled GP	91	2140	26.3
$\|V\| = 1000$	0	0	Scaled GP	91	1287	20.43
	0	1598	Int. Pnt.	101	N/A	390.07
	0	1598	Unscaled GP	101	1914	48.9
	0	1598	Scaled GP	101	1717	28.21
Bus 1138	0	0	CG	48	848	10.58
	0	0	Int. Pnt.	48	N/A	49.08
$\|V\| = 1138$	0	0	Unscaled GP	48	1904	25.03
	0	0	Scaled GP	48	875	16.49
	0	1458	Int. Pnt.	43	N/A	190.06
	0	1458	Unscaled GP	43	2697	36.12
	0	1458	Scaled GP	43	1148	19.97

differences in running time between the different methods. Without constraints CG was clearly fastest, solving the problem in fewer iterations and having to do slightly less work in each iteration. This is to be expected since CG is known to

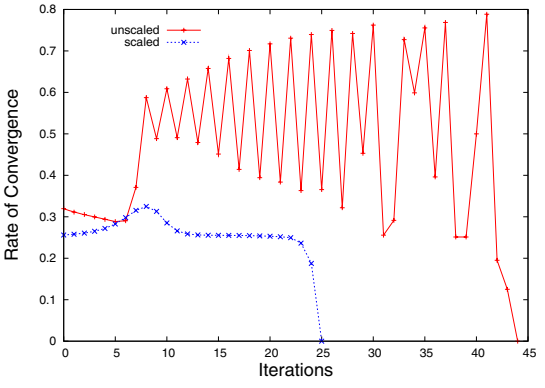


Fig. 5. Rate of convergence $\frac{|x_{k+1}-x^*|}{|x_k-x^*|}$ shown for each iteration k of the first gradient-projection iterate when applying stress majorization to the 1138bus graph. Note that x^* is simply taken as the final configuration before the threshold is reached so the final tail-off in both curves should be disregarded.

achieve super-linear convergence. Of the remaining methods, across all graphs, constrained or not scaled GP was the fastest (converging in significantly fewer iterations), followed by unscaled GP and the interior point method was slowest by several fold. In all cases scaling improved the running time by at least 20%. Interestingly, the improvement in speed in GP when scaling was applied was more marked when constraints were also solved, e.g. for the tree example it was almost twice as fast. To check scalability we also repeated the tests for random graphs between 50 and 2000 nodes and the speed improvement observed with scaling remained a fairly constant factor between 1.5 and 2.

Fig. 5 gives a graphic explanation of how scaling improves the convergence of the GP method. The figure shows rate of convergence by iteration for the first QP solved by the GP method in a stress majorization layout of the 1138bus graph. Convergence rate is, as usual, defined as the distance from an optimal solution at iteration $k+1$ divided by the distance at iteration k. As shown in Fig. 2 we stop the GP procedure when the descent vector has length smaller than some threshold τ and for this test, to ensure a reasonable number of iterations we set τ very small (10^{-15}). With scaling convergence is roughly constant and the threshold is reached after 25 iterations. Without scaling, the convergence rate oscillates and the threshold is not reached until 44 iterations.

4 Conclusion

Constrained stress majorization is a promising new technique for integrating application specific layout constraints into force-directed graph layout. The method previously suggested for solving the special kind of quadratic program arising in constrained stress majorization is a specialized gradient projection algorithm for separation constraints. We have demonstrated that by performing diagonal

scaling on the quadratic programming and generalizing the projection algorithm to handle positively scaled separation constraints, we can significantly improve the speed and convergence properties of the constrained stress-majorization technique. Importantly, this improvement comes at very little extra implementation effort. Thus, we believe that gradient projection with diagonal scaling is the method of choice for solving constrained stress majorization.

Our results have greater scope than graph layout since constrained stress majorization is immediately applicable to constrained multidimensional scaling (as the two problems are analogous). We also believe that the use of diagonal scaling may benefit other force-directed layout methods that are based on steepest descent.

References

1. Fisk, C.J., Isett, D.D.: ACCEL: automated circuit card etching layout. In: DAC 1965. Proceedings of the SHARE design automation project, pp. 9.1–9.31. ACM Press, New York (1965)
2. Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Information Processing Letters 31, 7–15 (1989)
3. Dwyer, T., Marriott, K., Wybrow, M.: Integrating edge routing into force-directed layout. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 8–19. Springer, Heidelberg (2007)
4. Dwyer, T., Koren, Y., Marriott, K.: IPSep-CoLa: An incremental procedure for separation constraint layout of graphs. IEEE Transactions on Visualization and Computer Graphics 12, 821–828 (2006)
5. Borg, I., Groenen, P.J.: Modern Multidimensional Scaling: theory and applications, 2nd edn. Springer, Heidelberg (2005)
6. Gansner, E., Koren, Y., North, S.: Graph drawing by stress majorization. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 239–250. Springer, Heidelberg (2005)
7. Bertsekas, D.P.: Nonlinear Programming. Athena Scientific (1999)
8. Dwyer, T., Marriott, K.: Constrained stress majorization using diagonally scaled gradient projection. Technical Report 217, Clayton School of IT, Monash University (2007)
9. os, P.E., Rényi, A.: On random graphs. Publ. Math. Inst. Hungar. Acad. Sci. 5, 17–61 (1960)
10. Barabási, A.L., Reka, A.: Emergence of scaling in random networks. Science 286, 509–512 (1999)