DIG-COLA: Directed Graph Layout through Constrained Energy Minimization

Tim Dwyer*
School of Computer Science and Software Engineering, Monash University

Yehuda Koren†
AT&T Labs — Research

ABSTRACT

We describe a new method for visualization of directed graphs. The method combines constraint programming techniques with a high performance force-directed placement (FDP) algorithm so that the directed nature of the graph is highlighted while useful properties of FDP — such as emphasis of symmetries and preservation of proximity relations — are retained. Our algorithm automatically identifies those parts of the digraph that contain hierarchical information and draws them accordingly. Additionally, those parts that do not contain hierarchy are drawn at the same quality expected from a non-hierarchical, undirected layout algorithm. An interesting application of our algorithm is *directional multidimensional scaling (DMDS)*. DMDS deals with low-dimensional embedding of multivariate data where we want to emphasize the overall flow in the data (e.g. chronological progress) along one of the axes.

CR Categories: G.1.6 [Numerical analysis]: Optimization—Constrained optimization; H.5.0 [Information interfaces and presentation]: General

1 INTRODUCTION

Graph drawing algorithms convert the relational structure of a graph (or network) to a diagram. Many approaches to graph drawing have been developed for different types of graphs and different application domains [2, 12]. In this paper, we consider the problem of drawing directed graphs (digraphs).

Drawing digraphs is a challenging task, requiring algorithms that faithfully represent the relative connectivity of the nodes as well as giving some sense of the overall directionality of the connections (edges). The latter requirement renders algorithms designed for undirected graph drawing inappropriate for digraphs. The dominant digraph-drawing strategy, rooted in the work of Sugiyama *et al.* [21] involves assigning x and y coordinates in separate stages with different objectives. Thus, the y-axis represents the directional information, or hierarchy, and the x-axis placement is adjusted for additional aesthetic considerations such as minimizing edge crossings. The special attributes of digraph layouts are demonstrated in Fig. 1 with two different layouts of the structure of a small gene-expression network. Each node in the network represents a gene and a directed edge indicates that the source gene affects the expression of the target gene (i.e. by binding the product of one gene to the promoter region of another gene). Undirected (or bidirected) edges (i.e. edges for which no preferred directionality should be shown) are colored blue. Otherwise, edge direction is shown with an arrow. Hierarchical layout places the genes from top to bottom in activation order. The undirected layout could be considered clearer (with fewer edge crossings, more consistent edge length and arguably better representation of cycles, clusters and symmetry) but it gives no indication of the chronological order of interactions.

*e-mail: Tim.Dwyer@infotech.monash.edu.au
†e-mail: yehuda@research.att.com

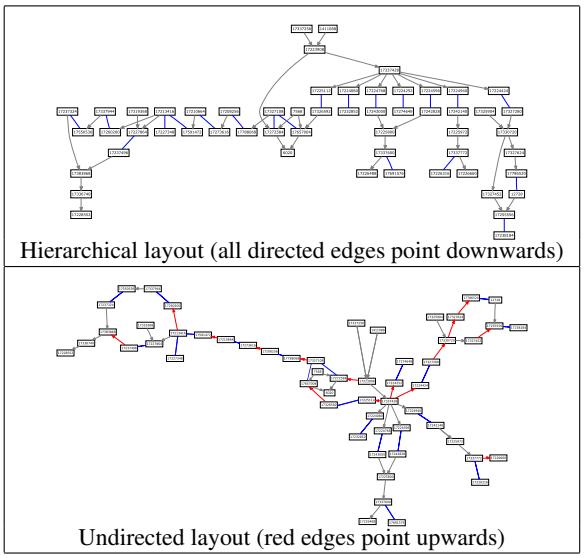


Figure 1: Two layouts of a gene network using standard layout algorithms from the GRAPHVIZ toolkit [15], blue edges are undirected

In this work we present a new approach to drawing digraphs. Following the common convention, we also dedicate one of the axes to conveying hierarchical information. However, we will not pursue the usual axis separation approach. Separate computation of different axes can be algorithmically appealing as it allows a convenient divide-and-conquer strategy. However, such a separation can make it difficult to control common aesthetic properties in the resulting drawing. Prime examples are uniformity of edge lengths (preventing very long edges) or balancing aspect ratio (equal spread along all axes). Hence, we propose a method for constrained layout of digraphs (DIG-COLA) where all axes of the layout are computed simultaneously as in standard undirected approaches. This process is made feasible by combining two optimization techniques: majorization [14] and quadratic programming [8].

2 PREVIOUS WORK

The predominant approach to drawing digraphs is based on Sugiyama *et al.* [21], which has evolved into many successful algorithms. Fig. 1 gives an example of a graph drawn with such an algorithm. In these Sugiyama-style algorithms, the y-coordinates are computed by dividing the y-axis into a finite number of layers and associating each node with exactly one layer. Edges between nodes on the same layer are not allowed and edges spanning multiple layers are split into chains of *dummy nodes*. Layer assignment is usually computed with the goal of minimizing edge-length (and hence the number of dummy nodes). The layer assignment methods used are generally incapable of handling cyclic-digraphs and so preprocessing is required to make the graph acyclic by reversing a minimal number of edges. Assigning the x-coordinates is normally done in two stages: first ordering nodes within layers to minimize crossings and then exact placement subject to the computed ordering to optimize aesthetic criteria such as minimizing edge bends. Each of the optimization problems outlined above

has been shown to be NP-hard so heuristic approaches have been designed that offer reasonable results in most cases. For more details see [2, 11, 12, 13].

Carmel *et al.* [5] use an alternative approach to drawing digraphs. The nodes are associated with continuous y-coordinates, in a way that can be applied to any kind of digraph, whether cyclic or acyclic, and which requires no graph modification or preprocessing. These y-coordinates are the unique minimizer of the *hierarchy energy*, which strongly reflects the directional information of the digraph. The x-coordinates are the minimizer of another energy function that disregards all directional information. The main characteristics of this method are its ability to deal with cyclic graphs, the fact that nodes are not partitioned into horizontal layers and the very fast execution. Unfortunately however, although solutions are defined for graphs with cycles, so-called symmetric nodes within cycles are by definition considered to be at the same hierarchy level and are thus assigned the same y-coordinate. The resulting drawings make such cycles difficult to see; see, e.g., Fig. 5(b).

The new approach introduced in this paper is distinguished from those above by the fact that it does not separate the computation of the various axes, but computes all axes simultaneously using constrained energy minimization. In this sense, we are following the force-directed placement approach (FDP) to graph drawing [2, 12]. These methods define a cost function (or a force model) whose minimization generally produces an acceptable node placement. In particular, we are aware of two other occasions where FDP was suggested for digraph drawing: Sugiyama and Misue [20] and Kamps *et al.* [18]. In both cases the force model was extended by 'angle forces' that encourage all directed edges to point in the same direction. However, this adds additional complexity to the force model and we are under the impression that the inferred energy function is complicated, prone to numerical stiffness and rich in local minima.

3 Preliminary Notions

A digraph $G = (V,E)$ comprises a set $V = \{1,\ldots,n\}$ of n nodes, and a set E of directed edges where $(i,j) \in E$ is an edge from node i to node j. We further associate with each edge (i,j) a number δ_{ij} that expresses the *relative hierarchy* of the nodes. Usually, $\delta_{ij} = 1$ for a directed edge $i \to j$, meaning that i precedes j by one unit. Similarly, $\delta_{ij} = 0$ for any undirected edge, meaning that there is no hierarchical precedence between i and j.

We denote a d-dimensional layout by an $n \times d$ matrix X. Thus, node i is located at $X_i \in \mathbb{R}^d$ and the axes of the layout are $X^{(1)},\ldots,X^{(d)} \in \mathbb{R}^n$.

3.1 The stress function

The stress function is a traditional measure of drawing quality, based on the heuristic that a nice drawing relates to good isometry: i.e. it calls for placing the nodes so that the resulting pairwise Euclidean distances will approach the corresponding target distances (e.g., graph-theoretic distances). Drawing undirected graphs by minimizing a stress function was made popular by Kamada and Kawai [17]. Specifically, we have an ideal distance d_{ij} for every pair of nodes i and j, modeled as a spring. Given a d-D layout, where node i is placed at point X_i, the energy of the system is

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

We desire a layout that minimizes this function, thereby best approximating the target distances. Here, the distance d_{ij} is typically the graph-theoretical distance between nodes i and j (i.e. the length of the shortest path connecting i and j). The normalization constant w_{ij} equals $d_{ij}^{-\alpha}$. Kamada and Kawai [17] chose $\alpha = 2$, whereas Cohen [6] also considered $\alpha = 0$ and $\alpha = 1$. Moreover, Cohen suggested setting d_{ij} to the linear-network distance to better convey any clustered structure in the graph. All results reported in this paper are based on graph-theoretical target distances and $\alpha = 2$.

3.2 Stress majorization

A recent report by Gansner *et al.* [14] suggests computing a graph layout by minimizing the stress function through *majorization*, following works in the field of multidimensional scaling [4]. Majorization is a rather global optimization process offering some distinct advantages over localized processes like gradient descent — especially guaranteed monotonic decrease of stress, improved robustness against local minima and shorter running times.

Majorization minimizes the 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 L^w, defined by

$$L^w_{i,j} = \begin{cases} -w_{ij} & i \neq j \\ \sum_{k \neq i} w_{ik} & i = j \end{cases} . \tag{2}$$

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

$$L^Z_{i,j} = \begin{cases} -w_{ij} \cdot d_{ij} \cdot \mathrm{inv}(\|Z_i - Z_j\|) & i \neq j \\ -\sum_{k \neq i} L^Z_{i,k} & i = j \end{cases} , \tag{3}$$

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

It can be shown (see [14]) that the stress function is bounded from above by the quadratic form $F^Z(X)$ defined as

$$F^Z(X) = \sum_{i<j} w_{ij} d_{ij}^2 + \sum_{a=1}^{d} \left(\left(X^{(a)}\right)^T L^w X^{(a)} - 2 \left(X^{(a)}\right)^T L^Z Z^{(a)} \right). \tag{4}$$

Thus, we have

$$\mathrm{stress}(X) \leqslant F^Z(X) \tag{5}$$

with equality when $Z = X$.

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

$$L^w X^{(a)} = L^Z Z^{(a)}, \quad a = 1,\ldots,d . \tag{6}$$

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 $F^{X(t)}(X(t)) = \mathrm{stress}(X(t))$.

We take $X(t+1)$ as the minimizer of $F^{X(t)}(X)$ by solving

$$L^w X(t+1)^{(a)} = L^{X(t)} X(t)^{(a)}, \quad a = 1,\ldots,d . \tag{7}$$

At this point, we terminate the process if

$$\Delta\mathrm{stress} = \frac{\mathrm{stress}(X(t)) - \mathrm{stress}(X(t+1))}{\mathrm{stress}(X(t))} < \varepsilon, \tag{8}$$

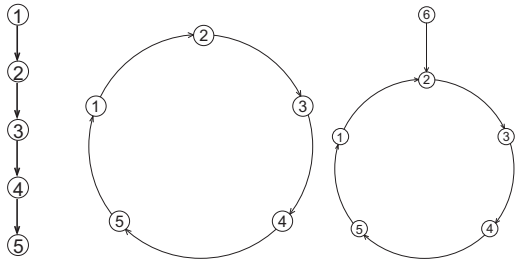
Otherwise, we set $t \leftarrow t+1$ and continue lowering the stress.

3.3 Hierarchy energy

The *hierarchy energy*, introduced by Carmel *et al.* [5], strongly reflects the directional information of the digraph. Let $G(V,E)$ be a digraph, and let $y = (y_1,\ldots,y_n)^T$ be any vector of coordinates. The *hierarchy energy* is

$$E_H(y) = \sum_{(i,j) \in E} (y_i - y_j - \delta_{ij})^2 . \tag{9}$$

The *optimal arrangement* of a digraph, $y^\star$, is defined as a minimizer of the hierarchy energy. It will try to place the nodes such that the height difference $y_i - y_j$ for any adjacent pair (i,j) will be close to



(a) five levels (b) no hierarchy (c) two levels

Figure 2: Digraphs with various degrees of hierarchy

δ_{ij}, the relative hierarchy (i,j). This optimal arrangement is given by solving the linear equations

$$\deg_i \cdot y_i = \sum_{j:(i,j)\in E} (y_j + \delta_{ij}) + \sum_{j:(j,i)\in E} (y_j - \delta_{ji}), \qquad i = 1,\ldots,n,$$

where $\deg_i = |j:(i,j)\in E| + |j:(j,i)\in E|$.

The optimal arrangement shows the height of each node in the hierarchy as induced by the digraph structure.

4 The DIG-CoLA Algorithm

Our algorithm follows a long tradition of drawing undirected graphs by minimizing a stress (or energy) function. However, since we want to show the overall directionality of the graph, we impose *constraints* on the layout.

4.1 Hierarchy as constraints

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 one of the axes to showing this hierarchy. Henceforth, the y-axis will serve for this purpose. Thus, if node i precedes node j in the hierarchy, then i will be drawn above j on the y-axis. This usually leads to the majority of directed edges pointing downwards, thereby showing a clear flow from top to bottom.

Accordingly, our first step is to compute the hierarchy induced by the digraph. This hierarchy is expressed by partitioning the node set into k disjoint sets: $V = \mathscr{L}_1 \cup \mathscr{L}_2 \cup \cdots \cup \mathscr{L}_k$, so that if $i < j$ then the nodes in $\mathscr{L}_i$ precede those in $\mathscr{L}_j$ in the hierarchy. Here $1 \leqslant k \leqslant n$ is a value depending on the graph properties. Henceforth, we will call these sets "hierarchical levels" or just "levels". Consequently, the directional properties of the digraph are expressed by imposing *hierarchy constraints*: for all $1 \leqslant i < j \leqslant k$ place all nodes of $\mathscr{L}_i$ higher than those of $\mathscr{L}_j$ in the y-axis.

Note that the hierarchical levels strongly depend on the digraph structure. For example, consider Fig. 2. In (a), we would like to say that node 1 precedes node 2, which precedes node 3 and so on. Hence, a reasonable partition would be into 5 levels where $\mathscr{L}_1 = \{1\}, \mathscr{L}_2 = \{2\}, \mathscr{L}_3 = \{3\}, \mathscr{L}_4 = \{4\}, \mathscr{L}_5 = \{5\}$. If we add only a single edge, from node 5 to node 1 we get the graph in (b). For this graph, all nodes are symmetric, and none precedes any other in the hierarchy. Hence, unless we have some external information, it would be safe to assume that no hierarchy exists here and we must assign all nodes to a single level $\mathscr{L}_1 = \{1,2,3,4,5\}$. A third case with one additional node is shown in (c). In this case, an agreeable partitioning is $\mathscr{L}_1 = \{6\}$ and $\mathscr{L}_2 = \{1,2,3,4,5\}$.

How do we get such a partitioning? Sometimes we have external information regarding the hierarchical order of the nodes. For example, if the nodes are associated with chronological data, we can partition them by years. If we know that one special node is the "root", then distance from it might dictate the hierarchical levels. Another possibility is to use the layering phase of Sugiyama-based methods [13, 21], for partitioning the nodes into different levels.

Our default choice is to base the partition on the optimal arrangement $y^\star$, which was defined in Section 3.3. This allows us to deal with all digraphs, including those that are cyclic or those that also include undirected edges. An advantageous property of the optimal arrangement is that it will not introduce hierarchy that is not induced by the graph structure[1]; see [5]. This is very important, as we don't want to impose any unjustified constraint on the layout. For example, for the digraph in Fig. 2(b), all nodes will be assigned the same position in $y^\star$ meaning that there are no hierarchical constraints; whereas for the digraph in Fig. 2(c) all nodes except node 6 will be assigned the same $y^\star$ position, yielding a constraint stating that node 6 should be placed above all other nodes.

Function PartitionToLevels $\{G(V = \{1,\ldots,n\}, E)\}$
% Partition the nodes into levels reflecting hierarchy

% Constants controlling number of levels:
$\alpha \leftarrow 0.1$, $\beta \leftarrow 0.01$ % lower values encourage more levels
Compute the optimal arrangement $y^\star \in \mathbb{R}^n$
% Sort according to $y^\star$:
Compute a permutation $1 \leqslant v_1,\ldots,v_n \leqslant n$, so that $y^\star_{v_i} \geqslant y^\star_{v_{i+1}}$
$\varepsilon \leftarrow \text{Max}\left(\alpha \frac{1}{n-1} \sum_{i=1}^{n-1} (y^\star_{v_i} - y^\star_{v_{i+1}}), \beta\right)$
$k \leftarrow 1$
for $i = 1$ to $n-1$ **do**
  $\mathscr{L}_k \leftarrow \mathscr{L}_k \cup \{v_i\}$
  if $y^\star_{v_i} - y^\star_{v_{i+1}} > \varepsilon$ **then**
    % A new level based on a significant gap in $y^\star$:
    $k \leftarrow k+1$
  end if
end for
$\mathscr{L}_k \leftarrow \mathscr{L}_k \cup \{v_n\}$
return $\mathscr{L}_1, \mathscr{L}_2, \ldots, \mathscr{L}_k$

Figure 3: Pseudocode for the level partitioning algorithm

In principle, we could sort nodes by their positions in $y^\star$ and place each node in a separate level. However, this might produce an excessive number of levels and therefore excessive constraints. Hence, we compute the hierarchical levels by clustering nodes according to their positions in $y^\star$ so that all nodes within the same cluster are at the same level of hierarchy. This is a one-dimensional clustering problem that we solve using the Single Link approach. We sort $y^\star$ and split it where we observe significant gaps, i.e. if $y^\star_i$ is significantly larger than $y^\star_{i+1}$, then nodes i and j are assigned to different levels. Detailed pseudocode for this is given in Fig. 3.

4.2 Iterative quadratic programming

We want to find a layout minimizing the stress function subject to the hierarchy constraints. He and Marriott [16] considered the addition of general linear constraints to FDP based on the localized Kamada-Kawai approach. However, we have found that constraints can be better integrated into the more global majorization optimization process described in Section 3.2. This smooth integration is made possible by using *quadratic programming*.

Recall that stress majorization involves minimizing a series of quadratic forms in a way that guarantees a monotonic decrease of stress. To address the directional information, we must also consider the hierarchy constraints, which are certainly linear. Hence, in each iteration we are minimizing a quadratic function subject to linear constraints. It can be shown that the quadratic functions that we minimize are positive definite (see next subsection). Such a constrained optimization is called *convex quadratic programming*; see, e.g., [8]. It is well known that the global minimizer of a convex quadratic program can be computed efficiently.

To summarize, when integrating stress majorization with level constraints, for each iteration we are solving a convex quadratic

[1] Unlike the layering methods used in Sugiyama methods, which, combined with cycle removal, would force any pair of connected nodes onto different levels even if they are symmetric in the original graph structure.

program, instead of minimizing a quadratic function. While solving convex quadratic programs is more involved than just minimizing quadratic functions, we can still efficiently obtain optimal solutions using a variety of available solvers. Notably, the majorization process, which must obey all constraints, still monotonically decreases the stress function and hence is convergent.

In more detail, suppose we are given some initial layout $X(0) \in \mathbb{R}^{n\times d}$. Then in the t-th iteration we compute layout $X(t)$ as the solution of the quadratic program:

$$\begin{aligned} &\min_X \sum_{a=1}^{d} \left(\left(X^{(a)}\right)^T L^w X^{(a)} - 2\left(X^{(a)}\right)^T L^{X(t-1)} X(t-1)^{(a)} \right) \\ &\text{subject to :} \quad \forall j \in \mathscr{L}_i : X_j^{(1)} \geqslant l_i, \quad i = 1,\ldots,k-1 \\ &\qquad\qquad \forall j \in \mathscr{L}_{i+1} : X_j^{(1)} + G \leqslant l_i, \quad i = 1,\ldots,k-1 \end{aligned} \tag{10}$$

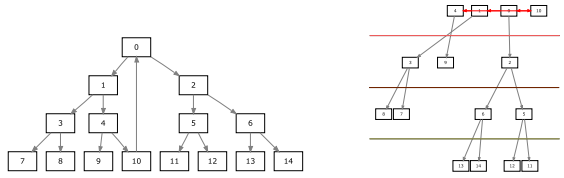
Note that the target function is the same as in the usual majorization process defined in (4). The sets $\mathscr{L}_1,\ldots,\mathscr{L}_k$ are the hierarchical levels that were defined in the previous subsection. The column $X^{(1)} \in \mathbb{R}^n$, is the axis that represents the hierarchical information (the y-coordinates in our examples). We use $k-1$ auxiliary variables $l_1, l_2, \ldots, l_{k-1}$ for expressing the constraints succinctly: l_i must be below the nodes in $\mathscr{L}_i$ but above the nodes of $\mathscr{L}_{i+1}$. The matrices L^w and $L^{X(t-1)}$ were defined in (2) and (3), respectively. Note that L^w and the constraints are fixed during the entire process. The constant G is the minimum gap between consecutive hierarchical levels. In our implementation the default value is $G = 0.1$. We terminate the process when the stress level stabilizes, i.e. when $\Delta\text{stress} < \varepsilon$, as defined in Eq. (8). A typical value for ε is 0.01.

4.3 Visual conventions

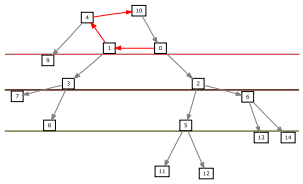
A layout produced by our algorithm can be partitioned into *horizontal bands*, each band contains all nodes belonging to a single hierarchy level. These bands decompose the $X^{(1)}$-axis (usually the y-axis) into the intervals $[-\infty, l_1], [l_1, l_2], [l_2, l_3], \ldots, [l_{k-1}, \infty]$. The sizes of the bands are non-uniform and depend on the computed layout, which reflects the graph structure.

Why do these bands interest us? The reason is that all nodes in the same band are considered to have the same level in the hierarchy. Thus, some layouts are comprised of many narrow bands and hence contain much hierarchy, whereas some other graphs do not contain much hierarchy so their layouts are composed of a very few bands. In satisfying the quadratic program, the algorithm will usually create broader bands for levels containing larger portions of the graph. In order to show this information, we let the user see the underlying bands in the form of a narrow vertical bar that is placed to the left of the graph drawing. This bar contains a sequence of colored blocks corresponding to the bands. The colors gradually change from red to green from top to bottom. To allow users to easily differentiate between consecutive bands, we alternate between saturated and unsaturated colors. Note that we do not use node colors so they can be exploited to display other attributes.

An example showing bands for a few small graphs is given in Fig. 4. In 4(a) we show a directed circle. Such a digraph contains no hierarchy and hence all layout is within a single band. In fact, when drawing such a graph DIG-COLA generates no constraints and so it becomes equivalent to undirected stress minimization algorithms. In 4(b) we show a directed tree. Each level of the tree is placed within a separate band yielding six bands. Another "tree" with the root replaced by a directed cycle is shown in 4(c). The cycle is contained within the top band, while the remaining nodes are partitioned into bands according to their distance from the root cycle. The last example 4(d) is a directed cycle with an extra node pointing to it. Naturally, the whole cycle is contained within one band, while the extra node is shown in a separate, higher band.



(a) Sugiyama-style layout by DOT (b) Layout of Carmel *et al.* [5] [15]



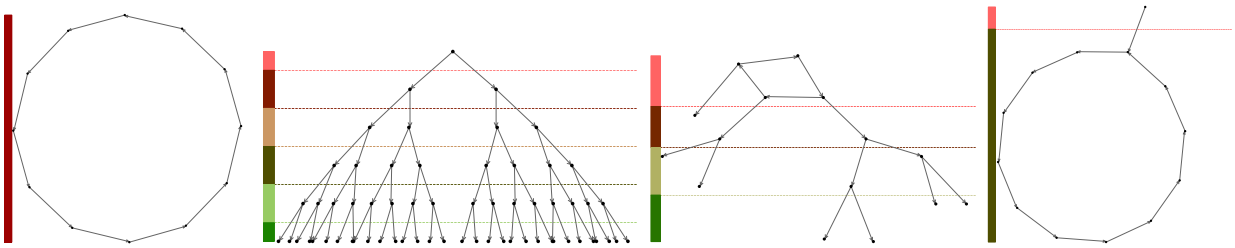
(c) DIG-COLA layout

Figure 5: Three drawings of the same directed graph containing a cycle. DIG-COLA puts all vertices of the cycle in the same level of hierarchy, while not squeezing this cycle.

In conclusion, explicit drawing of bands may provide useful information regarding the hierarchical relations between nodes and the amount of hierarchy contained in the graph.

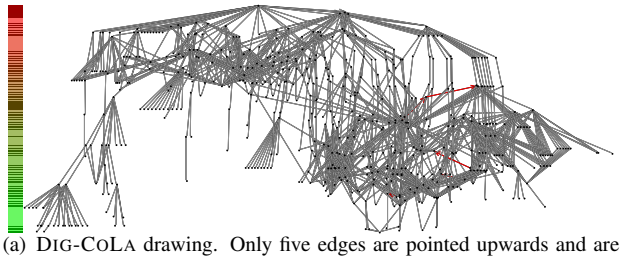
We would like to further emphasize the ability of our method to show hierarchy only where it exists, and to produce quality drawings also for non-hierarchical portions of the graph. This is nicely demonstrated in Fig. 5, where we show three drawings of a digraph containing a directed cycle. The Sugiyama-style layout (Fig. 5(a)) partitions the digraph into four hierarchical levels with the nodes involved in the cycle spread across all levels. However, based on the connectivity, there is no reason to put any one of the cycle nodes above any other. Therefore, the result of Carmel *et al.* [5] (Fig. 5(b)), which assigns y-coordinates directly from hierarchy energy, gives all nodes in the cycle the same y-coordinate. While this is true from the hierarchical standpoint, it is clearly undesirable in terms of readability. Hence, DIG-COLA (Fig. 5(c)) also places the cycle in a single top band, but adjusts this band so the cycle becomes clear.

Another visual issue concerns indicating the direction of an edge. The usual convention is by drawing an arrowhead at the end of the edge. In some cases this approach won't be efficient. When the drawing contains dense regions the arrowheads become very obscure. Moreover, these arrowheads may occupy precious real estate in the drawing area, thus impairing readability. Luckily, by the nature of the drawing algorithm the vast majority of edges follow a single trend, namely, pointing downwards. Hence, we can draw these edges without arrowheads, only using arrowheads for those edges that point upwards. An even clearer solution is to use edge coloring when colors are available. Therefore, all edges pointing downwards are colored gray/black, whereas those pointing upward are colored red. If the graph also contains undirected edges, which are permitted by our algorithm, they will be colored blue. For example, consider the graph DG_3692 from the AT&T digraph collection [23]. The layout of the digraph ($|V| = 559, |E| = 1035$) is quite dense; see Fig. 6(a). Removing arrowheads for all but the five edges that point upwards (and are colored red) improves readability while presenting the same information. In addition, since by definition the optimization task seeks to preserve edge lengths, we can assume with reasonable confidence that the neighbors of any particular node are within a relatively small radius of that node. Hence, in larger graphs, drawing edges is probably less useful than simply drawing the nodes and allowing viewers to assume that proximity implies connectedness. Sugiyama-based algorithms cannot make this guarantee, as shown in Fig. 6(b). We discuss the importance of proximity between connected nodes further in Section 6.

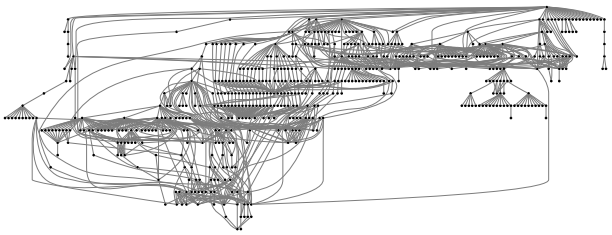


(a) 1 band (b) 6 bands (c) 4 bands (d) 2 bands

Figure 4: Hierarchy reflected as bands: each band is a horizontal slice containing nodes at equal level in the hierarchy.



(a) DIG-COLA drawing. Only five edges are pointed upwards and are colored red. Layout complexity is lessened by hiding arrow heads.



(b) Sugiyama-style drawing (produced with DOT). The occurrence of some very long edges (sometimes spanning most of the width or height of the drawing area) makes it difficult to study connectedness.

Figure 6: Two renderings of the dg_3692 digraph from the AT&T graphs repository.

4.4 Extensions and additional details

Quadratic programming solvers Many quadratic solvers are available employing a number of different optimization methods. All of them can serve for solving problem (10). They include, MOSEK [22], CPLEX [24], and OOQP [9]. The results reported in this paper were carried out using the MOSEK package which is based on the general purpose interior-point non-linear optimization method. However, several properties of the DIG-COLA method mean that there are many possibilities for designing a faster solver. Specifically, the quadratic program described above has very simple constraints and the set of constraints does not change between iterations. Also, the solution to one iteration is a good approximation to the solution of the next. Full details of a solver which takes advantage of these attributes are to appear in [1].

Fixing a single node The convexity of the quadratic program is thanks to the fact that the matrix L^w is positive semi-definite. This means that every $x \in \mathbb{R}^n$ satisfies $x^T L^w x \geqslant 0$. This inequality can be validated by using the fact that for all $i \neq j, w_{ij} > 0$ and noting that $x^T L^w x = \sum_{i<j} w_{ij}(x_i - x_j)^2 \geqslant 0$. Consequently, the quadratic form has only global minima. However, many solvers prefer dealing with strict positive definite matrices, ensuring the uniqueness of the global minimum. We can easily transform L^w into a positive definite matrix without affecting the majorization process. Observe that problem (10) has a translation degree of freedom, which is the source of multiple minima. Hence, we can fix node 1 at the origin without loss of generality, i.e. we set $X_1 = (0,0,\ldots,0)$. In the constraints we should replace all occurrences of X_1 with zeros. Moreover, we can safely remove the first row and column of L^w without affecting the value of the function. It can be shown that the resulting matrix is positive definite, and now we always have a unique global minimum, thus eliminating all degrees of freedom.

Reducing the quadratic program Note that the constraints involve only the y-axis, $X^{(1)}$ and that the target function is a sum of terms, each of which includes only a single axis. Thus, we can safely decompose problem (10) into two problems, one constrained and one unconstrained. The first problem yields $X(t+1)^{(1)}$

$$\begin{aligned} &\min_{X^{(1)}} \left(X^{(1)}\right)^T L^w X^{(1)} - 2\left(X^{(1)}\right)^T L^{X(t)} X(t)^{(1)} \\ &\text{subject to}: \quad \forall j \in \mathscr{L}_i: \; X_j^{(1)} \geqslant l_i, \quad i = 1,\ldots,k-1 \\ &\qquad\qquad \forall j \in \mathscr{L}_{i+1}: X_j^{(1)} + G \leqslant l_i, \quad i = 1,\ldots,k-1\,. \end{aligned} \tag{11}$$

And the second problem yields the rest axes $X^{(2)},\ldots,X^{(d)}$

$$\min_X \sum_{a=2}^{d} \left(\left(X^{(a)}\right)^T L^w X^{(a)} - 2\left(X^{(a)}\right)^T L^{X(t)} X(t)^{(a)} \right). \tag{12}$$

Solving problems (11), (12) will reproduce the solution of the original problem (10). However, solving two separate problems is much more efficient. The constrained problem (11) involves much fewer variables than the original (10) and hence is faster to solve. Regarding the unconstrained problem (12), its solution takes a relatively negligible time and is given by solving the equation systems $L^w X^{(a)} = L^{X(t)} X(t)^{(a)}, \quad a = 2,\ldots,d$.

When negative is a plus Interestingly, one can set a negative value for G (the minimum gap between levels). This permits slightly breaking the constraints by a quantity limited by $|G|$. Put differently, a negative G allows a limited "local" deviation of nodes from their designated bands. To make sure that these deviations are local and cannot change the global nature of the layout, we add the constraint: $l_1 \leqslant l_2 \leqslant \cdots \leqslant l_{k-1}$.

We have found that allowing small deviation sometimes gives us the necessary freedom for overcoming local inefficiencies in the layout, without affecting visualization of the overall directionality. This is a unique feature of our method that becomes possible by the clear separation between constraints and cost function.

We provide here two examples where negative gap was advantageous. Both examples are taken from the Matrix Market collection [3]. The first graph, Nos4, in Fig. 7, is based on a finite element approximation to a beam structure. Here, setting $G = -1$ improves layout quality while still being very similar to the layout that was achieved with $G = 0$. The second graph, Plat362, in Fig. 8, is based on a finite-difference model for the shallow wave equations for the Atlantic ocean. Here, we found $G = -2$ to provide better quality, while overall hierarchy depiction is not significantly affected.

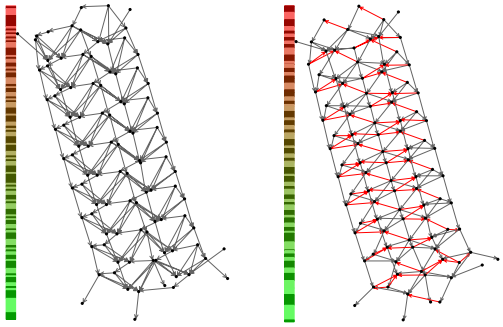


Figure 7: Two similar layouts of the Nos4 graph. In the drawing on the left all edges point downwards, whereas in the version on the right crossings are reduced by allowing some edges to point up (the red edges) without affecting overall directionality.

5 Experimental Study

In Section 4 we hinted that the Dig-CoLa method seems more suitable for larger digraphs than the popular Sugiyama-style layout since there should be less variance in edge length. On the other hand, Sugiyama style layout includes an edge crossing minimization heuristic not found in Dig-CoLa. So it would seem that there is some trade off between edge crossing reduction and consistent edge length. To see how these algorithms perform in practice we have completed an experimental study on a large collection of graphs from various application domains, comparing the performance of both the Dig-CoLa and Sugiyama methods. In this section we briefly present the results of this study.

The three datasets used in the following quantitative analysis were compiled from different repositories with the aim of capturing a variety of graph types and application domains. Our first class of graphs is a set of 258 gene-activation networks. The graphs are of varying sizes (up to $|V| = 95, |E| = 110$), generally quite sparse ($|E| \leq 2|N|$) and frequently contain directed cycles. The second set comes from the *AT&T Graphs* [23]. This latter repository contains over 5000 directed graphs. Of these, we extracted the largest connected component from each graph and filtered out very large and very small graphs (leaving 2464 graphs of $10 \leq |V| \leq 200$). Using random selection we further reduced this number to a more manageable sample of 100 graphs. We obtained a third set of 34 graphs with $200 \leq |V| \leq 1919$ from the larger graphs of the AT&T collection and the Matrix Market.

Our test systems were our own implementation of the Dig-CoLa algorithm and the freely available program Dot: an implementation of a Sugiyama-style layout distributed as part of Graphviz [15]. Both of these programs were run against each of the graphs in our test data sets and running time, edge lengths and count of crossings were recorded. We ran Dig-CoLa with $G = 0.1$ and $G = -1$.

The following results for edge length are based on normalized edge lengths l for each graph layout. That is, $l = L/\bar{L}$ where L is raw edge lengths for each layout and $\bar{L}$ is the average value of L. Thus, for each graph layout $\bar{l} = 1$. The results concerning l are summarized graphically in Fig. 9. With both layout styles the average sample standard deviation $\bar{\sigma}$ was smaller for the sparser gene-activation networks than the AT&T graphs, however for all data sets the Dot layout $\bar{\sigma}$ was at least double that of the Dig-CoLa layouts, as shown in Fig. 9(a). From the boxplots in Fig. 9(b) we can see that, while the edge-length distribution is narrow and even for Dig-CoLa layouts, DOT layouts give a large difference between the upper quartile and maximum edge lengths. That is, DOT layouts tend to have a small number of very long edges. Note that, as per speculation in Section 4, setting a negative value for G gave a further decrease in edge-length variance at the expense of more upward pointing edges. Note also, that edge lengths reported for Dot output are a conservative estimate based on Euclidean distance between start and end node positions (i.e. bends are not considered).

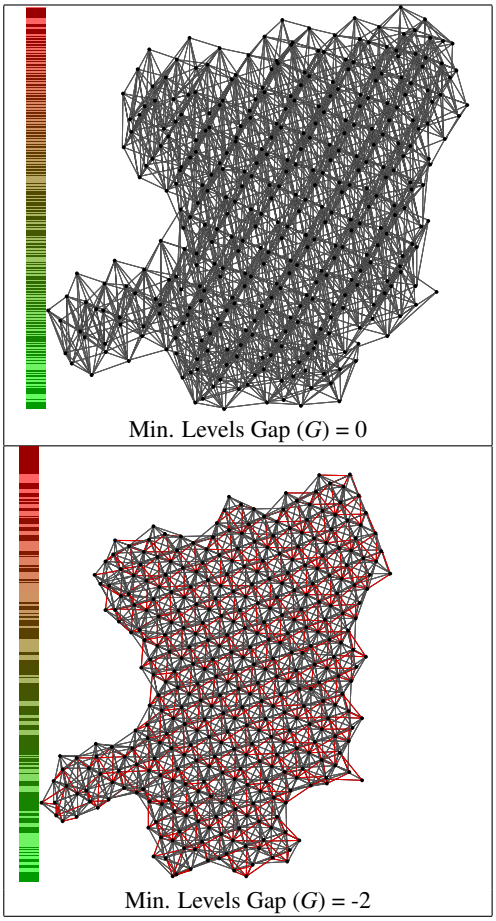
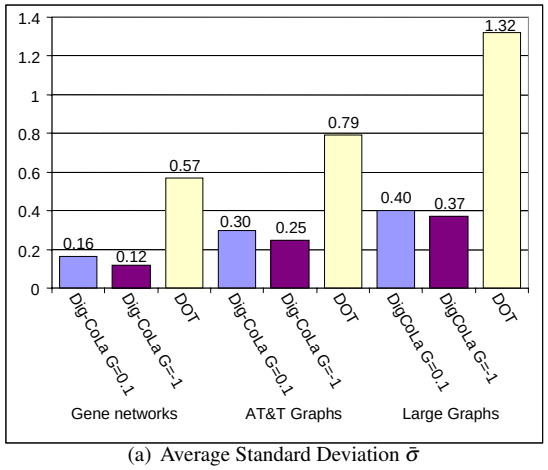


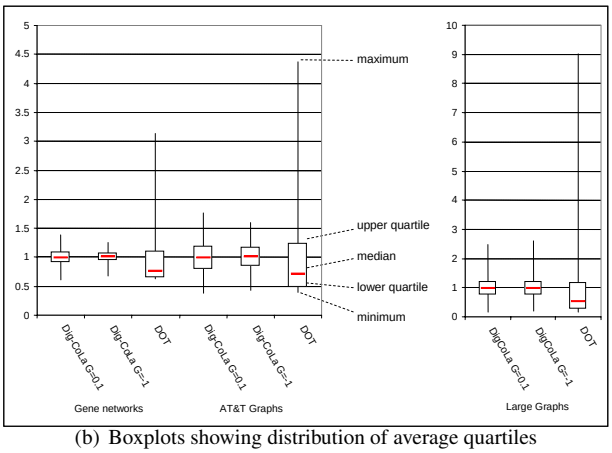
Figure 8: Two layouts the Plat362 graph. The two layouts agree on the overall structure of the graph. However, in the upper image all edges point downwards, while in the lower we allow some edges to point up (the red edges) thus achieving a clearer picture.

Regarding edge crossings, the expected trade-off described above was observed for the smaller graphs. That is, Dig-CoLa tended to produce more crossings, particularly in the denser AT&T Graphs (see Fig. 5). Again, however, we would argue that in dense graphs, drawing the edges is less important than maintaining relative proximity of nodes. Note that in the sparser gene networks Dig-CoLa with $G = 0.1$ gave an increase in crossings over Dot, while setting $G = -1$ gave fewer crossings than Dot. For the set of the larger graphs Dot gave more crossings than Dig-CoLa even with positive G.

Finally, a note on running time of Dig-CoLa. All tests were run on a 2GHz Pentium M machine. The results are shown in Fig. 11. The preprocessing step includes an $O(n|E|)$ computation of the pairwise distances, but the overall running time is significantly dominated by the iterative majorization process. Time per iteration (Fig. 11(a)) clearly increases polynomially with number of vertices and must be at least $O(n^2)$ per iteration since we are dealing with the dense $n \times n$ matrix L^w. However, the number of iterations required to satisfy $\Delta\text{stress} < \varepsilon = 0.01$ (Fig. 11(b)) is less easy to predict. In our tests graphs with $|V| < 100$ were arranged in under a second; those with $|V| < 600$ took less than 10 seconds and a matrix market graph (plat1919) with $|V| = 1919$ took 7 minutes. On average 98% of the running time is spent inside the Mosek quadratic program solver. Since completing this experiment



(a) Average Standard Deviation $\bar{\sigma}$



(b) Boxplots showing distribution of average quartiles

Figure 9: Edge length statistics by layout type and dataset. Lengths normalized such that their average is 1

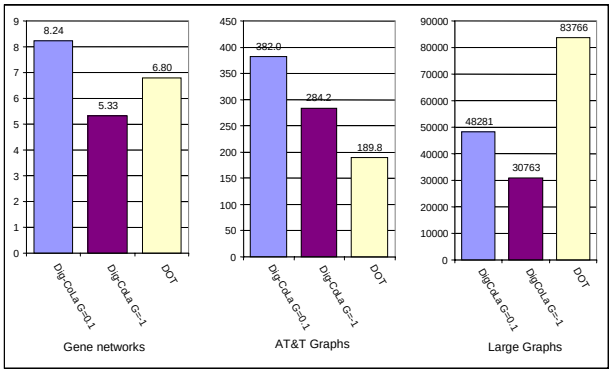
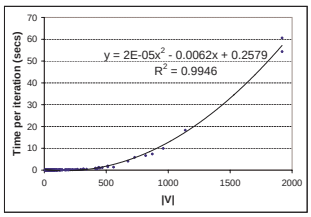
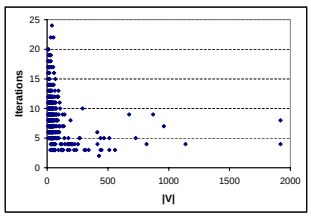


Figure 10: Average crossing counts for the different data sets and layout styles.



(a) Avg. seconds per iteration



(b) Iterations before $\Delta stress < \varepsilon$

Figure 11: Results of performance testing on graphs of various sizes.

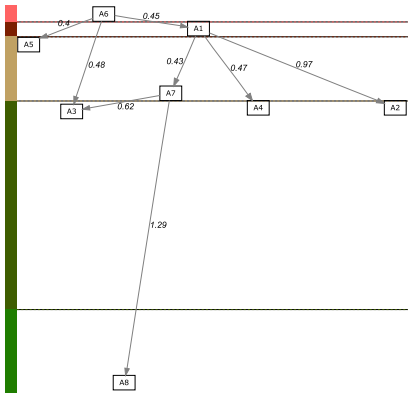


Figure 12: A small Bayesian network drawn such that edge lengths are inversely proportional to weights (probabilities).

we have been exploring alternative solvers that take advantage of the properties of the DIG-COLA quadratic program as discussed in Section 4.4. Initial results, as discussed in [1], indicate that the running time is greatly improved by employing such a solver — to the point where DIG-COLA is not significantly slower than unconstrained stress minimization and in most cases out-performs DOT.

6 VARIABLE EDGE LENGTH AND DIRECTIONAL MDS

The capability for conservation of default edge length that we have demonstrated in DIG-COLA, can be very useful in applications where relative edge weights need to be studied. For example, Bayesian networks are directed acyclic graphs where nodes represent variables and edges represent causality relationships between pairs of variables. The edges can be weighted to indicate the strength of these relationships. In Fig. 12 we give a small example of a Bayesian network drawn with DIG-COLA such that edge lengths correspond to the inverse of these weights. That is, the shorter the edge between a pair of nodes, the stronger the causality relationship between the corresponding variables. As usual, all edges point down. Based on the results of our quantitative analysis we would argue that such a close correlation between edge weight and length would be very difficult to achieve with Sugiyama layout.

We have discussed generating hierarchy constraints based on the hierarchical structure of a digraph, but the constraints could just as easily be based on another, domain specific, parameter. An example is Directional MDS to which we turn now.

The fact that the general stress majorization method described in Section 3 has its roots in the field of multidimensional scaling (MDS) leads us to wonder whether some MDS applications could also utilize the type of hierarchy constraints used in DIG-COLA. In general, graph visualization and MDS are closely related. MDS can be thought of as the visualization of the weighted, complete graph defined by the dissimilarity matrix of a set of high-dimensional data. The DIG-COLA algorithm can be applied to such a complete graph to perform MDS and we can map an additional data dimension to hierarchy constraints. We call this *Directional MDS*, or *DMDS*. For example, given high-dimensional time-series data we can produce an MDS plot separating the points corresponding to various time periods into different bands.

A practical example of DMDS is shown in Fig. 13. Here, a multivariate data set of nutritional information for popular breakfast cereals from [25] has been used to construct a dissimilarity matrix. Eight variables (fiber, starch, sodium etc.) were used. Hence, a traditional MDS algorithm would produce a 2D map of the cereals so that their relative positions correspond to their nutritional similarity. Using DMDS we can clearly show a 9$^{\text{th}}$ variable of the dataset, a dietitian's "health rating" of each cereal. By partitioning the cereals based on this health rating we introduce hierarchy to the layout such that the healthiest cereals appear at the top and the

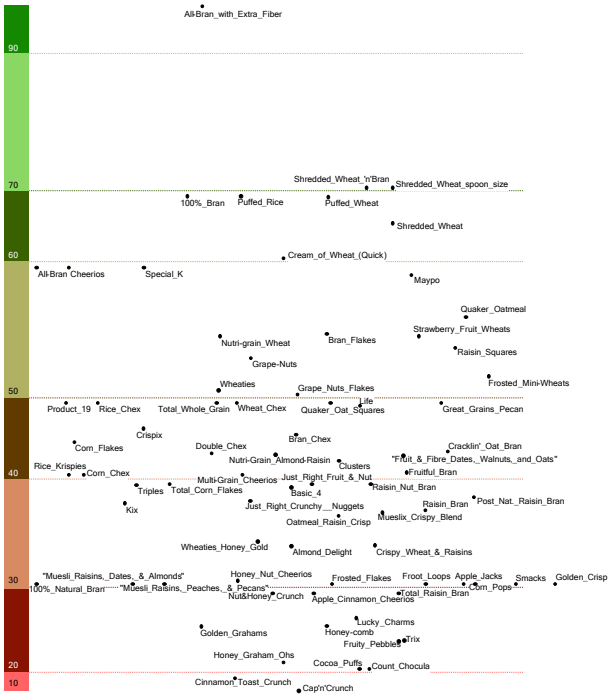


Figure 13: Directional MDS for breakfast cereal nutritional data. Cereals are arranged into bands reflecting their "health rating" and those with similar nutritional values are placed in proximity.

least healthy are constrained to the bottom. The result achieves two goals. Not only do we show similarities between the cereals based on their nutritional value, but we also allow easy recognition of the relative health rating of the cereals. Note that high-fiber cereals are generally to the top left of the figure and a cluster of starchier rice- and wheat-based cereals is visible on the upper right.

7 Discussion

Force-directed placement (FDP) is the most popular approach to drawing undirected graphs. However, despite a few attempts, FDP is not commonly employed for drawing digraphs. We believe that an important contribution of this work is in reducing this distinction between undirected and directed graph drawing methods. For the first time, we show that the same stress function can also serve for hierarchical drawing of a digraph. Moreover, the same majorization optimization method can still be used, without affecting its convergence properties.

As a consequence, the virtues of FDP that made it very popular for drawing undirected graphs, are now available in a well-behaved force-directed method for drawing directed graphs. These virtues include an ability to preserve proximity relations (as reflected by conservation of default edge length), clear decomposition to clusters when they naturally exist (especially when using linear-network distance [6]), reliable display of graph symmetries (see [7]) and a natural capability to handle multidimensional layouts.

However, FDP does possess a few shortcomings, and our method is not an exception. A notable shortcoming is handling of edge crossings. While shortening edge lengths tends to significantly decrease crossings (especially, as we have shown, for larger graphs), FDP does not consider them directly. Moreover, FDP produces "organic-looking" layouts, while sometimes more restricted, or rigid layouts are preferred, e.g. layouts where vertices are assigned to grid points or to horizontal lines. Therefore, although we have compared DIG-COLA to Sugiyama-style layout we do not argue that this type of method should replace Sugiyama algorithms, except possibly for large or dense graphs or graphs where edge weights must be represented by relative lengths. Countless papers have been written about fine tuning every aspect of Sugiyama layout. As a result implementations such as DOT offer very stable and mature solutions to layout of relatively small digraphs. However, we hope that we have demonstrated enough potential for DIG-COLA that it will become a viable alternative for other classes of directed graphs.

Finally, we would like to mention that DIG-COLA is available in the Neato program in the Graphviz open source package [15].

Acknowledgements

We would like to thank Emden Gansner for integrating DIG-COLA with the Graphviz package.

References

[1] T. Dwyer, Y. Koren and K. Marriott, "Stress Majorization with Orthogonal Ordering Constraints", *Proc. Graph Drawing (GD'05)*, Springer-Verlag, to appear.

[2] G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, 1999.

[3] 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, pp. 125–137.

[4] I. Borg and P. Groenen, *Modern Multidimensional Scaling: Theory and Applications*, Springer-Verlag, 1997.

[5] 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.

[6] J. D. Cohen, "Drawing Graphs to Convey Proximity: an Incremental Arrangement Method", *ACM Trans. Computer-Human Interaction* **4** (1997), 197–229.

[7] Peter Eades, Xuemin Lin, "Spring algorithms and symmetry", *Theoretical Computer Science* **240** (2000), 379–405.

[8] R. Fletcher, *Practical Methods of Optimization*, Wiley, 2000.

[9] E. Gertz and S. Wright, "Object-Oriented Software for Quadratic Programming", *ACM Trans. Mathematical Software* **29** (2003), 58–81.

[10] P. E. Gill, W. Murray and M. H. Wright, *Practical Optimization*, Academic Press, 1981.

[11] M. Jünger and P. Mutzel, "Exact and Heuristic Algorithms for 2-Layer Straight line Crossing Minimization", *Proc. Graph Drawing (GD'95)*, LNCS 1027, pp. 337–348, Springer-Verlag, 1995.

[12] M. Kaufmann and D. Wagner (Eds.), *Drawing Graphs: Methods and Models*, LNCS 2025, Springer Verlag, 2001.

[13] E. Gansner, E. Koutsofios, S. North and K.-P. Vo, "A Technique for Drawing Directed Graphs", *IEEE Trans. Software Engineering* **19** (1993), 214–230.

[14] E. Gansner, Y. Koren and S. North, "Graph Drawing by Stress Majorization", *Proc. 12th Int. Symp. Graph Drawing (GD'04)*, LNCS 3383, Springer Verlag, pp. 239–250, 2004.

[15] E. Gansner and S. North, "An Open Graph Visualization system and its Applications to Software Engineering", *Software Practice & Experience* **30** (2000), 1203–1233. Also, `www.graphviz.org`.

[16] W. He and K. Marriott, "Constrained Graph Layout", *Constraints* **3** (1998), 289–314, Kluwer.

[17] T. Kamada and S. Kawai, "An Algorithm for Drawing General Undirected Graphs", *Information Processing Letters* **31** (1989), 7-15.

[18] T. Kamps, J. Kleinz and J. Read, "Constraint-Based Spring-Model Algorithm for Graph Layout", *Proc. 3rd Int. Symp. Graph Drawing (GD'95)*, LNCS 1027, pp. 349–360, Springer-Verlag, 1995.

[19] K. Marriott, P. Moulder, L. Hope and C. Twardy., "Layout of Bayesian networks", *Proc. 28th Australasian Computer Science Conference (ACSC2005)*, CRPIT 38, pp. 97-106, ACS, 2005

[20] K. Sugiyama and K. Misue, "A Simple and Unified Method for Drawing Graphs: Magnetic-Spring Algorithm", *Proc. 2nd Int. Symp. Graph Drawing (GD'94)*, LNCS 894, pp. 364–375, Springer-Verlag, 1995.

[21] K. Sugiyama, S. Tagawa and M. Toda, "Methods for Visual Understanding of Hierarchical Systems", *IEEE Trans. Systems, Man, and Cybernetics* **11** (1981), 109–125.

[22] MOSEK `www.mosek.com`.

[23] AT&T Graphs `www.graphdrawing.org/data/index.html`.

[24] ILOG CPLEX `www.ilog.com/products/cplex`.

[25] Cereals data `lib.stat.cmu.edu/DASL/Datafiles/Cereals.html`.