Eurographics Conference on Visualization 2012
S. Bruckner, S. Miksch, and H. Pfister
(Guest Editors)

Volume 31 (2012), Number 3

Drawing Large Graphs by Low-Rank Stress Majorization

Marc Khoury[1] Yifan Hu[2] Shankar Krishnan[2] Carlos Scheidegger[2]

[1]The Ohio State University and AT&T Research, Florham Park, NJ
[2]AT&T Research, Florham Park, NJ

Abstract

Optimizing a stress model is a natural technique for drawing graphs: one seeks an embedding into R^d which best preserves the induced graph metric. Current approaches to solving the stress model for a graph with $|\mathcal{V}|$ nodes and $|\mathcal{E}|$ edges require the full all-pairs shortest paths (APSP) matrix, which takes $O(|\mathcal{V}|^2 \log|\mathcal{E}| + |\mathcal{V}||\mathcal{E}|)$ time and $O(|\mathcal{V}|^2)$ space. We propose a novel algorithm based on a low-rank *approximation to the required matrices. The crux of our technique is an observation that it is possible to approximate the* full *APSP matrix, even when only a small subset of its entries are known. Our algorithm takes time $O(k|\mathcal{V}| + |\mathcal{V}|\log|\mathcal{V}| + |\mathcal{E}|)$ per iteration with a preprocessing time of $O(k^3 + k(|\mathcal{E}| + |\mathcal{V}|\log|\mathcal{V}|) + k^2|\mathcal{V}|)$ and memory usage of $O(k|\mathcal{V}|)$, where a user-defined parameter k trades off quality of approximation with running time and space. We give experimental results which show, to the best of our knowledge, the largest (albeit approximate) full stress model based layouts to date.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Graphs, denoted as $G(\mathcal{V},\mathcal{E})$, are ubiquitous structures in computing. They are used to model many real-life data sources, and a visualization of the structure of G quickly gives an analyst a starting context to guide further exploration. Although there are many ways to define and draw them, in this paper we focus on connected undirected graphs with positive edge lengths, and using the shortest-path distance as the metric structure.

As computing power increases, so does the size of the graphs in which we are interested. As the size of these graphs approach or surpass the amount of memory available in typical workstations, we would like our algorithms to use no more than space proportional to the size of the input. Ideally, running times should also scale linearly with the size of the data. While solving the stress model is an effective and natural means of drawing graphs and reducing the dimensionality of data [GKN04], its formulation requires access to the entire metric structure of the graph, which means computing the all-pairs shortest paths matrix and then factorizing a fully-dense matrix. These steps respectively take $O(|\mathcal{V}|^2 \log|\mathcal{E}| + |\mathcal{V}||\mathcal{E}|)$ and $O(|\mathcal{V}|^3)$ time, and both require $O(|\mathcal{V}|^2)$ space. In this paper we propose a stress majorization scheme which leverages linear-algebraic properties of the matrices involved to bypass both the dense matrix factorization and all-pairs shortest paths (APSP) computation.

Stress majorization requires iteratively solving a(n undertermined) linear system $Ax = b$, where the matrix A is computed from the graph metric structure. As we describe in Section 3, we use recent work which has shown, remarkably, that for a large class of matrices, it is possible to get a good approximation of *every element* of the full matrix A, while only having access to the exact values of a small block of A. This approximation of A will be *low-rank*: the column space of the approximation has smaller dimension than A. We will show how this lets us solve the linear system efficiently. We summarize the basic steps of our algorithm and the paper in Figure 1. Our contributions are:

- an experimental analysis of linear-algebraic properties of matrices involved in the stress majorization, showing they are represented well by *low-rank* matrices (Section 3.1),
- a novel algorithm that can solve the full stress majorization algorithm in time roughly $O(k|\mathcal{V}|)$ per iteration, preprocessing time $O(k^2|\mathcal{V}|)$, and space $O(k|\mathcal{V}|)$, if the associated matrices are represented by a truncated SVD (Section 3.2),
- an adaptation of an algorithm by Drineas et al. [DFK*04]

Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

which approximates the SVD of the necessary matrices *without computing most of its elements* (Section 3.3),
- and an experimental analysis of the effectiveness of this algorithm for computing layouts for large graphs.

Overall, this improves the previous best full stress majorization algorithms by a factor of $O(|\mathcal{V}|/k)$ in both time and space, and allows, to the best of our knowledge, the largest stress-majorization layouts to date.

Notation

We define $\text{inv}(x) = 1/x$ if x is different than 0, and $\text{inv}(x) = 0$ otherwise. We will use $\mathbf{1}$ to be a vector of all ones, and $\hat{\mathbf{1}}$ to denote the unit-length constant vector $\mathbf{1}/\sqrt{n}$, where n will be clear from the context. We use $\text{diag}(v)$ to denote a diagonal matrix whose diagonal entries are given by v.

2. Related Work

One approach for drawing large graphs with unit edge length involves embedding the graph in R^d using its first few eigenvectors of the Laplacian [Hal70]. In a sense, the first non-degenerate eigenvector (the Fiedler vector) of the graph Laplacian is a good one-dimensional layout of the graph which minimizes the edge length. Generalizing this intuition, the first d eigenvectors should be good embeddings in R^d. Although with a multilevel implementation [KCH02] the extreme eigenvectors can be calculated very quickly, the resulting layouts tend to under-use the available space and produce a lot of local clutters. Koren later suggests to use the eigenvectors as a *basis* in which to do optimization [Kor04]. This elegantly reduces the dimensionality of the search space for stress, but requires good stress solutions to lie inside that span. Gansner et al. show this is not the case, since they use subspace optimization only as an initialization method [GKN04].

The current state-of-the-art for large-scale graph drawing utilizes a multilevel representation of the graph and a fast approximation of the electrical term in the spring-electrical force model via a Barnes-Hut scheme [HJ04, Hu05, Qui01, Tun99, Wal03]. These techniques can handle graphs with millions of vertices and edges, but assume that the graphs have unit-length edges. While it is possible to incorporate edge length into the force model, such treatment is heuristic at best and not as principled as the stress model, and its interaction with the multilevel hierarchy is not well understood. Since stress model based layouts use the graph only indirectly through the metric structure, they work on graphs with or without specified edge lengths. The algorithm described in this paper, then, improves the state-of-the-art for large-scale graph drawing for graph with arbitrary positive edge lengths.

There have been many attempts for drawing large graphs with non-unit edge lengths. For some classes of mesh-like graphs, high-dimensional embedding (HDE [HK02]) works very well, and is extremely simple and fast to implement. HDE embeds each node of the graph in a high-dimensional space by using as its coordinates a vector of graph-theoretical distances to k-centers. This high-dimensional embedding is then projected to the desired dimension by principal component analysis. The main problem with HDE is that in many graphs, different vertices tend to have exactly the same high-dimensional coordinates, and end up being projected to the same point in space. A related algorithm is PivotMDS [BP07], which finds a fast approximation to the classical scaling problem [Tor52]. This is achieved by taking a $|\mathcal{V}| \times k$ submatrix C of the APSP matrix, normalizing it, and finding the two or three top eigenvectors of the $k \times k$ matrix $C^T C$. The projection of these k-dimensional eigenvectors back to the $|\mathcal{V}|$-dimensional space by multiplication with C then gives the coordinates of the vertices. This layout is used as a starting point to solve a sparse stress model.

Another attempt at a scalable, distance-sensitive embedding is GRIP [GGK00]. This is a multilevel algorithm, with coarsening carried out through maximal independent vertex set based filtration. On coarse levels, a Kamada-Kawai algorithm [KK89] is applied to each node within a local neighborhood of the original graph, but on the finest level, a localized Fruchterman-Reingold algorithm is used [FR91]. Because of this last step, the algorithm does not strictly solve a stress model.

The problem of multidimensional scaling (MDS) is closely related to graph drawing by the stress model. Chalmers [Cha96] proposed the first linear-time iteration algorithm for dimensionality reduction in the context of visualization via stochastic sampling, and Ingram et al. [IMO09] use a multiscale variant adapted to run efficiently on graphics cards. In these papers, a single entry of the distance matrix is usually assumed to be available in constant time. This is a valid assumption for multidimensional data, but is not the case for graph data, where graph-theoretical distances have to be calculated. In contrast, we use algebraic properties of this matrix to create an approximation to all the entries in the matrix while only computing a small block. For a thorough review of the myriad techniques related to MDS, we refer the reader to the recent review of France and Carrol [FC11].

At the core of our approach is the idea of replacing a distance matrix, which requires $O(|\mathcal{V}|^2 \log |\mathcal{V}| + |\mathcal{V}||\mathcal{E}|)$ time to compute and $O(|\mathcal{V}|^2)$ space to store, with a *low-rank* version of this matrix. In other words, instead of storing an $|\mathcal{V}| \times |\mathcal{V}|$ matrix M, we store a pair of matrices A and B with dimensions $|\mathcal{V}| \times k$ and $k \times |\mathcal{V}|$, respectively. A judicious choice of matrices can be made such that $AB \approx M$. Remarkably, for a large class of matrices M it is possible to find good matrices A and B without even accessing a large portion of M, even if M is dense. This idea was first proposed by Drineas et al. [DFK*04], where it was proved that the largest singular values and vectors of matrix M can be approximated well by the singular values and left singular vectors of a $|\mathcal{V}| \times k$ matrix whose columns are sampled from the original matrix with a probablity proportional to the squared norm of the columns. This sampling technique works well in our setting,

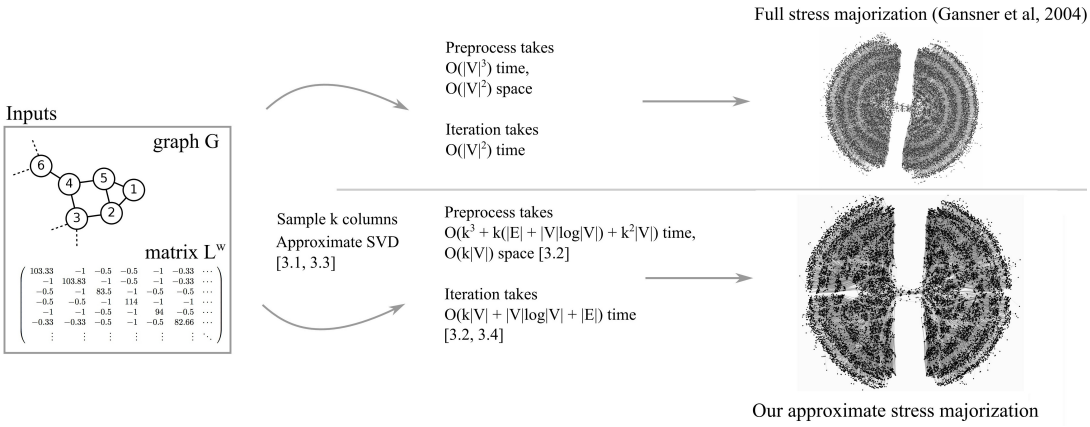

Figure 1: *An overview of our algorithm. The numbers between brackets indicate the paper sections which describe the corresponding portion. While the full majorization algorithm for the 16000-node graph (`ncvxqp9`) shown above took around forty minutes, the approximate layout we show below took about four minutes and a half. The two results are visually very close.*

since getting k columns of the distance matrix requires only k applications of Dijkstra's algorithm, with a complexity of $O(k(|\mathcal{E}|+|\mathcal{V}|\log|\mathcal{V}|))$.

There are other algorithms that approximate the SVD, such as Frieze et al.'s technique [FKV04]. Instead of sampling matrix columns, these algorithms sample *matrix entries*. This approach is not suitable for our setting, since calculating the distance between a random pair of vertices is almost as costly as one instance of single-source shortest paths. For the same reason, recent work on near linear time algorithms for solving Laplacian system, such as the work of Teng [Ten10], does not appear directly applicable to our problem.

3. Problem setup and derivation

In this section, we describe how to arrive at the approximate formulation for the stress majorization. If the reader is only interested in the algorithm for approximate layout, Section 4 shows the full algorithm.

Stress model based graph layout algorithms search for a layout X placing node i at point X_i, such that the total *stress* of the layout $S(X)$ is low, where

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

The constants w_{ij} decide the influence of pairwise interactions, and are usually taken to be $w_{ij} = d_{ij}^{-\alpha}$, where d_{ij} is the graph-theoretical distance between nodes i and j, and α is a small positive constant. The practice of taking the shortest graph distance as the ideal edge length dates back at least to 1980 in social network layout [KS80], and in graph drawing using classical MDS [BP09], but is often attributed to Kamada and Kawai [KK89]. The most popular choice is $\alpha = 2$, but in this paper we will consider only the case $\alpha = 1$, for the reasons in Section 3.4. More recently, Gansner et al. [GKN04] proposed a stress majorization procedure for solving the stress model, by showing that a resulting layout X is a local minimum of $S(X)$ if

$$L^\omega X = L^X X, \tag{1}$$

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

$$L^X_{ij} = \begin{cases} -w_{ij}d_{ij}/||X_i - X_j||, & i \neq j \\ \sum_{j\neq i} L^X_{ij}, & i = j \end{cases} \tag{3}$$

To find a minimum using stress majorization, one starts with an initial guess for X, and iteratively solves the linear system (1) using the result of the previous iteration to compute the right-hand side. We refer the reader to Gansner et al. [GKN04] for further details.

3.1. Low-rank approximations

The low-rank approximations we use in this paper are based on the singular value decomposition (SVD). Approximations of this type are only effective for some matrices. This section illustrates this phenomenon and its impact on our algorithm. Every matrix M admits a SVD decomposition of the form $M = U\Sigma V^T$, where U and V are orthogonal, and Σ is a non-negative diagonal matrix. When the diagonal elements of Σ are in non-increasing order, setting all but the first k values of Σ to zero yields a sequence of increasingly-good approximations of M as k increases. If most values of Σ are relatively close to zero, actually zeroing them does not change M too much. The *rate of decay* of these singular values determines whether M has a good low-rank approximation: the faster the values go to zero, the better low-rank approximations of M will be.

Although it is tempting to approximate L^ω directly with a low-rank version, we split L^ω into its diagonal and off-diagonal elements, $L^\omega = D^\omega + O^\omega$. The elements of O^ω_{ij} are simply $-w_{ij}$, and $D^\omega = -\text{diag}(O^\omega \mathbf{1})$. As we illustrate with Figure 2, the rate of decay of the singular values of O^ω is

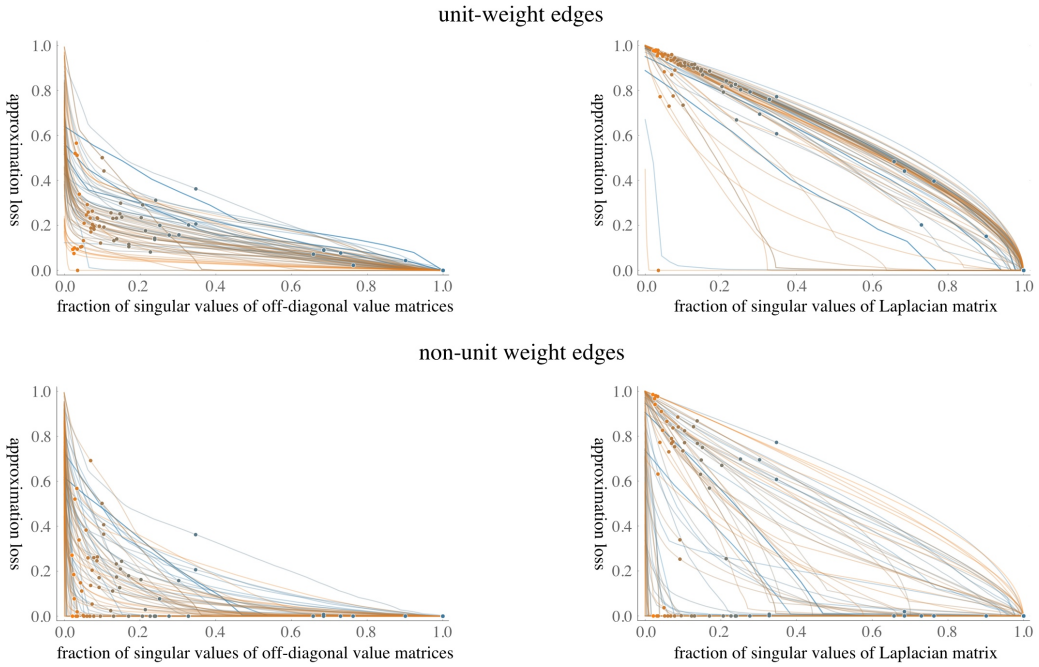

Figure 2: *Comparing the speed of decay of singular values of the Laplacian L^ω, and that of the off diagonal matrix O^ω, for the 100 smallest symmetric matrices from the University of Florida Sparse Matrix Collection [DH11]. The largest matrix in this experiment has about 4875 elements. The first row assumes a unit edge weight, while the second row takes the absolute value of the matrix entries as the edge length. Colors encode the size of the matrices – blue for small matrices and orange for larger matrices. Each dot shows the position of the 100th singular value, and the approximation error that would be incurred if that matrix was approximated by a rank-100 matrix. Note that the singular values for O^ω decay much faster; in addition, the larger the matrix, the faster the decay in general, and the better the top 100 singular values cover the whole spectrum.*

much faster than that of L^ω, and so a rank-k approximation of L^ω via O^ω is much better than a direct approximation of L^ω. Consequently, we will build a representation of L^ω of the form

$$L^\omega = D^\omega + O^\omega = -\mathrm{diag}(U\Sigma V^T\mathbf{1}) + U\Sigma V. \qquad (4)$$

In other words, we only store U, Σ, and V, and use that to build the actions of the L^ω operator. We now show how to compute U, Σ, and V, and how to solve $L^\omega x = b$ with such a representation.

3.2. SVD-based solver for $L^\omega x = b$

Our first observation is that $L^\omega x = b$ is always underdetermined, even when O^ω is full rank. This arises from the well-known property that the null space of a Laplacian matrix includes the set of all vectors with constant coordinates $c \cdot \mathbf{1}$. We assume that every distance d_{ij} is finite, and so the basis for the null space of L^ω is *exactly* $\{\mathbf{1}\}$, and hence L^ω is singular. This implies that the linear system $L^\omega x = b$ is underdetermined, and we look for its minimum-norm solution, $L^{\omega\dagger} b$, where $L^{\omega\dagger}$ is the Moore-Penrose pseudo-inverse of L^ω [GL96].

To solve $L^{\omega\dagger} b$, we use the following identities. Our basic idea is that since the Laplacian can be approximated by a low rank update of a diagonal matrix (4), we can use the matrix inversion lemma, also known as the Woodbury Matrix Identity [Hag89] (equations (9) and (10)) to find its inverse. However, since the Laplacian is singular, we work with the pseudo-inverse. This requires adding a constant matrix and projecting it out to circumvent the singularity, in Equation (5) (the identity is easily proven from the definition of the pseudo-inverse [GL96]).

$$\begin{aligned}
L^{\omega\dagger} &= (L^\omega + \hat{\mathbf{1}}\hat{\mathbf{1}}^T)^{-1} - \hat{\mathbf{1}}\hat{\mathbf{1}}^T && (5)\\
(L^\omega + \hat{\mathbf{1}}\hat{\mathbf{1}}^T)^{-1} &= (U\Sigma V^T + D^\omega + \hat{\mathbf{1}}\hat{\mathbf{1}}^T)^{-1} && (6)\\
&= (A + U\Sigma V^T)^{-1} && (7)\\
A &= (D^\omega + \hat{\mathbf{1}}\hat{\mathbf{1}}^T) && (8)\\
(A + U\Sigma V^T)^{-1} &= A^{-1} - A^{-1}UT_1{}^{-1}V^T A^{-1} && (9)\\
A^{-1} &= (D^\omega)^{-1} - t_2 t_3^{-1} t_2^T && (10)\\
T_1 &= (\Sigma^{-1} + VA^{-1}U^T) && (11)\\
t_2 &= (D^\omega)^{-1}\hat{\mathbf{1}} && (12)\\
t_3 &= 1 + \langle \hat{\mathbf{1}}, t_2 \rangle && (13)
\end{aligned}$$

In particular, Equations (9) and (10) allow us to solve the respective inverses with access only to the SVD matrices

COMPUTE-$\tilde{L}^{\omega}(G,k)$

1 $centers = \text{K-CENTERS}(G,k)$
2 $D = \text{DISTANCES}(\{i : 0 \leq i < n\}, \{j : j \in centers\})$
3 $\tilde{M}_{ij} = D_{ij}^{-1} \times column\text{-}scale$
4 $U\Sigma V = SVD(\tilde{M})$
5 $S_{ii} = sign(u_i^T \hat{M} u_i)$
6 $D^{\omega} = Diag(U\Sigma S U^T \mathbf{1})$
7 **return** $(\tilde{L}^{\omega}, centers)$ // $\tilde{L}^{\omega}$ given as $U(\Sigma S)U^T - D^{\omega}$

Figure 3: *Algorithm to compute $\tilde{L}^{\omega}$, the approximation to L^{ω}. In the final line, we do not explicitly carry out matrix operations. The matrix $\tilde{L}^{\omega}$ is returned as an abstract linear operator.*

of O^{ω}, without ever carrying out the full multiplications. Critically, if we only have the first k singular vectors and values of O^{ω}, we can still solve the approximate inverses, and in much less time than the full inverse: A^{-1} is computed via Equation (10), and T_1 is a $k \times k$ matrix, which need only be factorized once in $O(k^3)$ time. Notice that the matrix A^{-1} is never explicitly formed. Solving a system involving $L^{\omega\dagger}$ involves multiplying the right-hand side of Equations (9) with a vector, which in term involves multiplying the right-hand side of Equation (10) with vectors.

We now show that, remarkably, we can approximate the first few singular vectors of O^{ω} *without even knowing most values of* O^{ω}.

3.3. Approximate SVDs from column sampling

The derivation above is essential for our algorithm, but cannot be used directly, for the simple fact that we need the entire O^{ω} to compute its SVD, and computing the entries of O^{ω} takes $O(|\mathcal{V}|(|\mathcal{E}| + |\mathcal{V}| \log |\mathcal{V}|))$ time and $O(|\mathcal{V}|^2)$ space. The main result we use in this section is due to Drineas et al. [DFK*04]. In that paper, the authors build a matrix $\tilde{M}$ by randomly sampling k columns of a matrix M with probability p_i, where p_i is proportional to the squared length of the i-th column of M, and scaling them by $1/\sqrt{kp_i}$. They then show that the expected singular values of $\tilde{M}$ are the top singular values of M, and that the expected left singular vectors of $\tilde{M}$ are the corresponding left singular vectors of M. We cannot directly apply this technique, because computing the exact probabilities p_i requires access to the whole matrix O^{ω}. The authors also show that any probability distribution $\tilde{p}$ can be used, although this incurs an increase in the estimation variance.

In our case, we use a heuristic solution based on computing approximate k-centers which we found to work well in practice. We pick the first node randomly, and pick subsequent nodes which maximize the graph-theoretical distance to the selected nodes. These can be found in $O(k|\mathcal{V}| \log |\mathcal{V}|)$ time. We then compute the single-source shortest paths from these k vertices. Inverting the distances gives a $n \times k$ block of O^{ω}, which we scale by $1/\sqrt{kp_i}$ to obtain $\tilde{M}$. We use the left singular vectors and singular values of $\tilde{M}$ as the approximations for U and Σ.

SOLVE-$A^{-1}t(D^{\omega},t)$

1 $t_2 = (D^{\omega})^{-1}\hat{\mathbf{1}}$
2 $t_3 = 1 + \langle \hat{\mathbf{1}}, t2 \rangle$
3 **return** $(D^{\omega})^{-1}t - (t_3^{-1}\langle t2, t \rangle)t_2$

SOLVE-$\tilde{L}^{\omega\dagger}b(\tilde{L}^{\omega},b)$

1 $M = \Sigma^{-1} + UA^{-1}U^T$
2 // Use SOLVE-$A^{-1}t$ column-by-column to get $A^{-1}U^T$
3 $lu = \text{LU-DECOMPOSITION}(M)$ // Computed only once
4 $v_1 = \text{SOLVE-}A^{-1}t(D^{\omega}, b)$
5 $v_2 = U\,\text{LU-SOLVE}(lu, U^T v_1)$
6 $v_3 = \text{SOLVE-}A^{-1}t(D^{\omega}, v_2)$
7 **return** $v_1 - v_3$

Figure 4: *Algorithm to compute the pseudo-inverse of $\tilde{L}^{\omega}$ using the matrix inversion lemma.*

Drineas et al. do not address the simultaneous computation of left and right singular vectors, both of which we require. For symmetric matrices, like O^{ω}, $u_i = s_i v_i$, where s_i is either -1 or 1, and indicates whether the corresponding eigenvalue is positive or negative. We define $\hat{M}$ as a $n \times n$ *square* matrix with the columns of $\tilde{M}$ placed at the positions where they were sampled. As k increases, $\hat{M}$ approaches O^{ω}, and so $u_i^T \hat{M} u_i$ approaches the i-th eigenvalue of O^{ω}. We estimate the eigenvalues as $u_i^T \hat{M} u_i$, and flip the signs of the entries in Σ as necessary. The full algorithm for the time and space-efficient pseudo-inverse of L^{ω} is shown in Figure 3.

3.4. Computing $L^X X$ efficiently

The main stress majorization loop involves a repeated solution of the linear system $L^{\omega} X^{t+1} = L^{X^t} X^t$. While we approximate L^{ω} with the approximate SVD, the right-hand side matrix L^{X^t} is still dense, and a naive computation would still take at least $O(|\mathcal{V}|^2)$ time.

If we expand $L^{X^t} X^t$, we have

$$(L^{X^t} X^t)_i = \sum_{j \neq i} w_{ij} d_{ij} \frac{X_i - X_j}{||X_i - X_j||} \tag{14}$$

This means that each element of the right-hand-side of the linear system is a weighted sum of the unit directional vector to vertex i from the other vertices. Since $w_{ij} = 1/d_{ij}$, this further simplifies to $\sum_{j \neq i} \frac{X_i - X_j}{||X_i - X_j||}$. Following the recent practice in many large scale graph drawing algorithms [HJ04, Hu05, QE00], this summation can be approximated with a Barnes-Hut force approximation scheme [BH86], through the use of a suitable space decomposition data structure (we used a quadtree). Thus the right-hand-side of the linear system can be computed approximately in time $O(|\mathcal{V}| \log |\mathcal{V}|)$.

4. Algorithm

The complete algorithm for approximate stress majorization is described in Figure 5. With the necessary approximation

UPDATE-ANCHORS(G, *centers*, X)

1 $Y = X$
2 **for** i **in** *centers*
3 $\quad Y_i = \frac{1}{centers}(\sum_{j \leftrightarrow i} X_j)$
4 **for** i **in** *centers*
5 $\quad Y_i = \frac{\sum_{j \neq i} w_{ij}(X_j + d_{ij}(X_i - X_j) inv(||X_i - X_j||))}{\sum_{j \neq i} w_{ij}}$
6 **return** Y

APPROXIMATE-LAYOUT(G, k)

1 $\tilde{L}^{\omega}, centers = \text{COMPUTE-}\tilde{L}^{\omega}(G,k)$ // Fig 3
2 $X_{\text{new}} =$ initial guess for the layout
3 **repeat**
4 $\quad X = X_{\text{new}}$
5 $\quad rhs = L^X X$ // Through Barnes-Hut
6 $\quad X_{\text{new}} = \text{SOLVE-}\tilde{L}^{\omega\dagger} b(\tilde{L}^{\omega}, rhs)$ // Fig. 4
7 $\quad X_{\text{new}} = \text{UPDATE-ANCHORS}(G, centers, X_{\text{new}})$
8 **until** $|X_{\text{new}} - X| < \varepsilon$
9 **return** X_{new}

Figure 5: *Our algorithm for determining graph layouts using approximate stress majorization.*

algorithms for the right-hand-side and the pseudo-inverse in place, our algorithm follows essentially the steps as the stress majorization algorithm [GKN04]. The stress majorization algorithm requires an initial guess for vertex placement.

One unfortunate side effect we observed experimentally is that the k anchor nodes tend to suffer from much worse placement than the remaining $|\mathcal{V}| - k$ nodes. To eliminate this effect, we introduce an extra step per iteration, applying the localized optimization suggested by Gansner et al. [GKN04]. All the necessary w_{ij} values for these updates have already been precomputed, so this takes no extra memory and only $O(k|\mathcal{V}| + |\mathcal{E}|)$ time per iteration.

Throughout all of our experiments, we use as our initial layout the result of Graphviz's sfdp, a multiscale, spring-electric force-directed solver. This choice is not essential or critical; other fast, coarse layout algorithms such as HDE [HK02] or PivotMDS [BP07] would work as well.

Our prototype algorithm is implemented in C++, using the Armadillo matrix library for numerical linear algebra [San10], and LAPACK's SVD implementation.

4.1. Complexity

We separate the computational complexity of our algorithm into its preprocessing time and its time per iteration of the solver. There are three computationally intensive preprocessing steps. First, Dijkstra's algorithm for computing the k-columns of O^{ω} (lines 1 and 2 of COMPUTE-$\tilde{L}^{\omega}$ in Figure 3) takes time $O(k(|\mathcal{E}| + |\mathcal{V}| \log |\mathcal{V}|))$. Computing the SVD of the resulting $|\mathcal{V}| \times k$ matrix (line 3 of COMPUTE-$\tilde{L}^{\omega}$ in Figure 3) takes time $O(|\mathcal{V}|k^2)$, and the LU decomposition of M takes time k^3 (performed once on line 3 of SOLVE-$\tilde{L}^{\omega\dagger} b$ in Figure 4). All combined, these give a preprocessing time of $O(k^3 + k(|\mathcal{E}| + |\mathcal{V}| \log |\mathcal{V}|) + k^2|\mathcal{V}|)$. On every iteration, we must solve the right-hand-side matrix multiplication, which in our case can be done in $O(|\mathcal{V}| \log |\mathcal{V}|)$ time (see Section 3.4). SOLVE-$\tilde{L}^{\omega\dagger} b$ takes $O(k|\mathcal{V}|)$ time, dominated by LU-SOLVE$(lu, U^T v_1)$. Finally, UPDATE-ANCHORS takes $O(k|\mathcal{V}| + |\mathcal{E}|)$ time. The total per-iteration time is then $O(k|\mathcal{V}| + |\mathcal{V}| \log(|\mathcal{V}|) + |\mathcal{E}|)$.

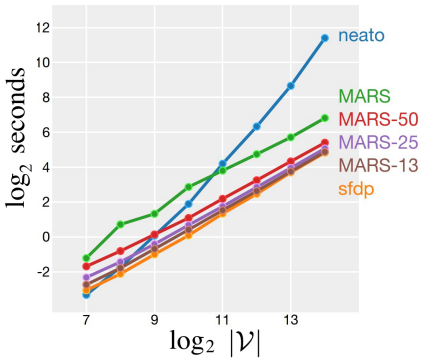

Figure 6: *Running times for neato, sfdp and our algorithm ("MARS") on complete binary trees of the specified size. Graphs larger than shown are not practical for full stress majorization: at $|\mathcal{V}| = 2^{16}$, simply storing L^w in its entirety would take 16GB of memory. "MARS-50", "MARS-25" and "MARS-13" denote running times for MARS running with $k = 50$, $k = 25$ and $k = 13$ respectively.*

5. Experiments

In this section we provide experimental evidence of the presented algorithm in Section 4. We first discuss the runtime performance of the algorithm, and follow it with a discussion of the quality of the resulting layouts. We call our low-rank stress majorization algorithm *MARS* (for "stress **M**ajorization through **A**pproximate low-**R**ank **S**VD").

5.1. Performance

We start by comparing the performance of MARS to two recent graph layout techniques, Gansner et al.'s stress-majorization solver [GKN04] ("neato") and Hu's multiscale force-directed solver [Hu05] ("sfdp"). Both implementations are open source and available in Graphviz [EGK*01]. To compare running times, we run all three algorithms on progressively larger complete binary trees, from $2^7 - 1$ to $2^{14} - 1$ nodes on a single processor of a dual quad-core Xeon running 64-bit Ubuntu 10.10 with 12GB of RAM. The results are shown in Figure 6. In general, we find that MARS takes about twice as long as sfdp to compute a layout. See Section 5.3 for the running times for larger graphs.

We ran other running-time experiments, and found that although asymptotically the number of edges in the graph is significant, in practice the running time for MARS seems to only be dependent on $|\mathcal{V}|$ and k, mostly irrespective of the graph being processed. Finally, we note that we are using the

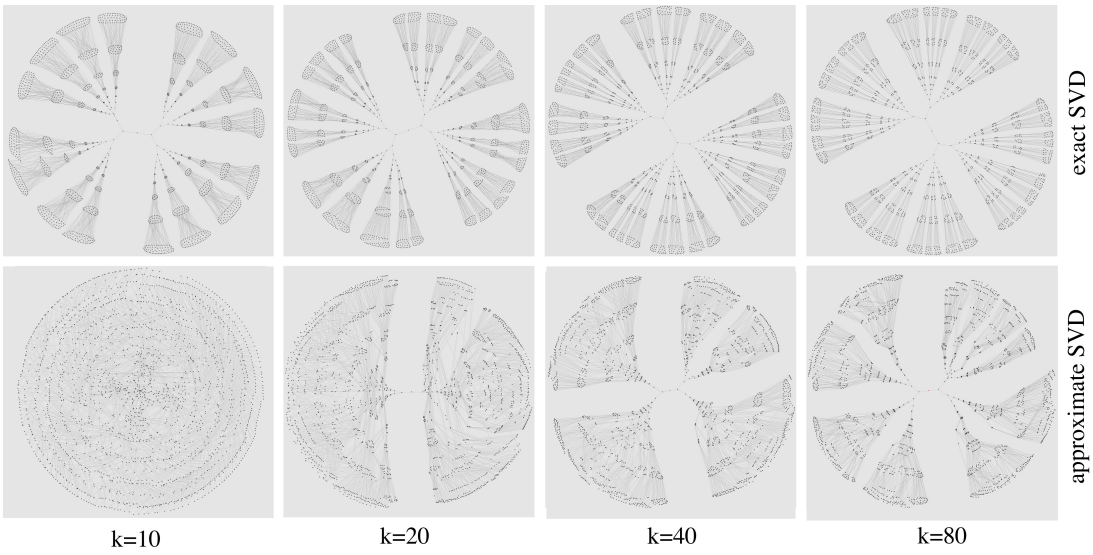

Figure 7: *A visual analysis of the approximation errors in our algorithm, computing different layouts of a complete binary tree of 2047 nodes. The top row shows the algorithm when using an exact SVD. The bottom row shows the column-sampling scheme approximate SVD we describe in the text. From left to right, we use 10, 20, 40 and 80 anchors, respectively.*

placement from sfdp as the initial guess for MARS. We have not included the running time of sfdp because it is possible to use faster techniques to provide the initial guess for MARS.

5.2. Approximation Tradeoffs and Quality

In this section, we discuss the effects of the different approximations in our algorithm. We examine these approximations using a synthetic example of a complete binary tree with $2^{11} - 1$ nodes. We choose this example since a complete binary tree has a very simple and regular structure, and so it is possible to visually evaluate the results. There are three main sources of error in our approximations to the full stress model. First, we solve a *low-rank* version of the stress model via the SVD. Second, we use an *approximation* to this low-rank version, since the exact SVD is impractical. Third, we solve an approximate version of the right-hand side of Equation (1). We have observed that computing the approximate right-hand side of Equation (1) by the Barnes-Hut algorithm (described in Section 3.4) has no noticeable loss of quality in the resulting layout, and so we only show the first two sources. These are illustrated in Figure 7. There, "exact SVD" means taking the top k SVD values of O^{ω} and setting the rest of the singular values to zero; "approximate SVD" is the procedure used in MARS where SVD is calculated on a submatrix samples from columns of O^{ω}.

Inspecting Figure 7, we note that the main source of visual artifacts for our algorithm comes not from the low-rank approximation, but from the approximate SVD. We see two consequences. The first one is that unfortunately it seems that our current algorithm depends crucially on an accurate SVD, even though the approximate SVD we use has essentially been proven optimal by the original authors [DFK*04]. Still, this points to an important direction of future work, where one would engineer alternate O^{ω} matrices, from which exact SVDs might be computed more efficiently. We speculate that there is much room for improvement in that area.

5.3. Comparison to other techniques

As a comparison to MARS, we choose HDE and PivotMDS since they are currently the fastest graph layout algorithms that also attempts to take into account edge lengths. In addition, we choose sfdp, arguably the state-of-the-art for large graph layouts, although it does not utilize edge lengths.

To compare the results of MARS to previously published techniques, we select a few graphs which highlight the relative tradeoffs of our approximation schemes. In general, we find that although the layouts produced by MARS have some obvious problems, they tend to provide an alternative "global" view of the structure of the graph. This layout is somewhat complementary from sfdp, which provides an aesthetically pleasing layout that is very well tuned locally at the expense of a global picture of the graph structure. The results are shown in Figure 8.

Specifically, we use four different graphs. The first one is `1138_bus`, a small graph with 1138 vertices, intended mostly to show that our approximation scheme appears to converge to a very pleasing layout with enough columns. sfdp took 1.4 seconds to compute this layout, while MARS took 3.9 seconds (In general, we find that for sufficiently small graphs, MARS behaves quite similarly to sfdp, neato and other high-quality graph drawing algorithms. Thus, the other examples we present are all larger graphs which challenge the current state-of-the-art techniques). Next, we use MARS to compute a layouts of two large trees. The first tree is a complete binary tree with 131071 nodes, while the second

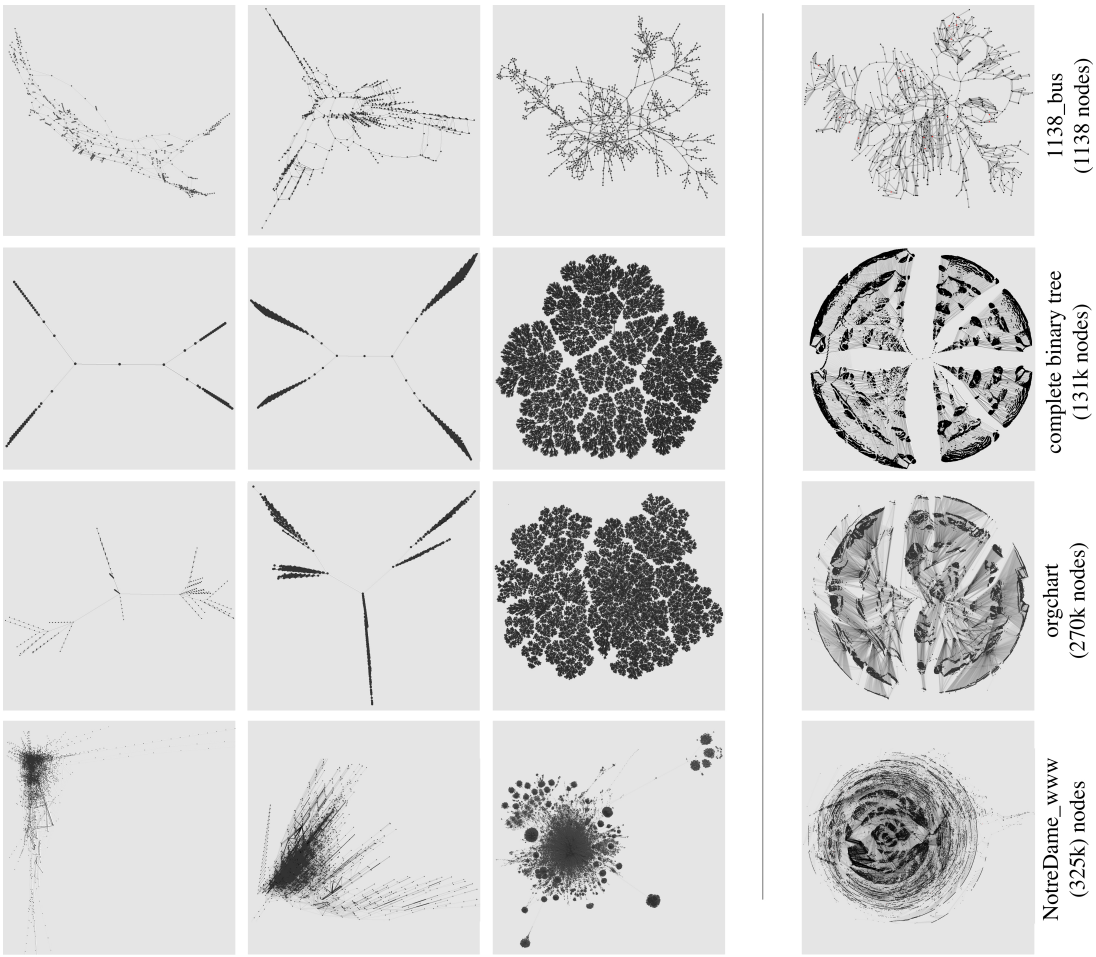

Figure 8: *The resulting layouts of MARS, compared to those of HDE, PivotMDS and sfdp. In all our examples, we set $k = 100$. HDE and PivotMDS run in negligible time for these examples. Going from top to bottom, sfdp takes 1.4, 323, 194, and 865 seconds. MARS takes 3.9, 588, 1915 and 1478 seconds.*

one is graph representing the organizational structure of a large company, with about 270000 nodes. For the binary tree, sfdp took 323 seconds, while MARS took 588 seconds. For the org chart, sfdp took 194 seconds, while MARS took 1915 seconds. The contrast between these two examples is show one advantage of stress majorization. Since it uses an underlying metric space to take into account interactions between every pair of vertices (regardless of whether an edge connects them), it can give a more global picture of the graph structure. While the layouts produced by sfdp appear more aesthetically pleasing, we call attention to the similarity between the two layouts generated by sfdp for the trees. In our opinion this is a major disadvantage of the method: *if two graphs have a significantly different structure, they should get significantly different drawings*. To the best of our knowledge, MARS satisfies this requirement better than any other published algorithm for sufficiently large graphs.

Finally, we show a graph of a crawl of the University of Notre Dame's web site, `NotreDame_www`, with 325000 nodes. Every web page is a node on this graph, and there is an edge between nodes iff there is a hyperlink that connects them. These low-diameter, highly connected graphs are known to be challenging to draw, and although sfdp succeeds in findings some well-isolated subgraphs, the main set of nodes appears hopelessly entangled. For this graph, sfdp took 865 seconds, while MARS took 1478 seconds. In the drawing produced by MARS, on the other hand, the isolated isles get pushed out to the periphery and lose their shape, but the main set of nodes appears better separated, and some coherent sets of links between clusters seem to appear. We show some more results of our algorithm in Figure 9.

6. Discussion and Future Work

Although there have been attempts at solving the stress model in a scalable way, we believe our algorithm has some unique

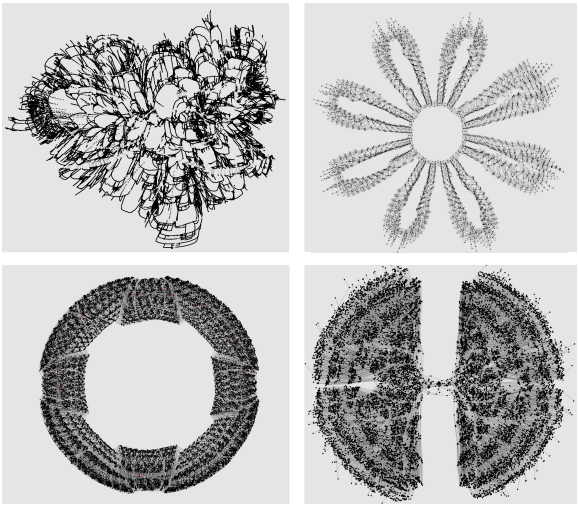

Figure 9: *Some more layouts of MARS. Top left,* `luxembourg_osm` *($|V| = 114599$, 543s); top right,* `finance256` *($|V| = 37376$, 151.9s); bottom left,* `ncvxbqp1` *(largest connected component: $|V| = 40000$, 203s); bottom right,* `ncvxqp9` *($|V| = 16554$, 66.7s). $k = 100$.*

advantages. Previous attempts have, for example, simplified the full stress matrix by assuming large blocks of the matrix are constant (via a quadtree decomposition such as the one from Chalmers [Cha96]), or by assuming large portions of the matrix are zero, such as PivotMDS [BP07] and sparse stress formulations [BP09]. MARS, on the other hand, uses a formulation for the stress matrix where every element of the matrix is equally perturbed by the low-rank approximation. Our algorithm does not explicitly store every entry of the matrix L^ω for performance reasons, but it behaves exactly as if it did. This is the crux of our insight: instead of looking for sparsifiers of the stress matrices whose approximation is all-or-nothing (either exact values or zero), we look for *low-rank* representations, whose approximation error is roughly evenly distributed along the matrix entries.

In addition, it is well-known [GKN04] that metric MDS (stress majorization) yields better results than classical MDS. Since our algorithm approximates the former, we can expect it to generate better results than those which approximate the latter, like PivotMDS.

Furthermore, the choice of $w_{ij} = d_{ij}^{-1}$ gives a fortuitous cancellation on the expression $L^X X$. In essence, this choice lets us implicitly use all d_{ij} elements without accessing any of them. Without this cancellation, all the values of the APSP matrix need to be computed, and sparsifying this matrix appears to lead to the known problem suffered by PivotMDS and HDE, where multiple vertices are projected to the same location. Our algorithm balances these two requirements. It efficiently and accurately computes the expression $L^X X$, which at present requires choosing $w_{ij} = d_{ij}^{-1}$, and it computes the pseudo-inverse of the left-hand side matrix L^ω without accessing or computing most entries of the APSP matrix.

Generally speaking, MARS appears to generate "radial" layouts at times. We speculate that this is due to the underlying metric being projected into R^2: nothing in the algorithm forces the layouts to be radial. This phenomenon might arise partly from the fact that most large graphs that are challenging to draw (and hence are the ones we picked to showcase MARS) are highly-connected and low-diameter. While we cannot rule out the possibility that the unconventional weight choice causes a radial layout (at the time of writing, no algorithm can handle other weight choices but $w_{ij} = d_{ij}^{-1}$ in large graphs), we note that the radial layout does not appear in all cases (see Figure 9), and that at least one other author has experimented with different weight assignments and this phenomenon is not visible there [Coh97].

One interesting avenue for future work is in *metric engineering*. One major advantage of stress majorization is that it is completely agnostic to the metric used in the original dataset. In most published work (and here as well), we restrict ourselves to the shortest-path graph metric. However, this metric has problems: one stray edge can "shortcut" potentially many node-to-node paths, and change the metric significantly. We intend to investigate the use of *commute-time distances* (roughly the time it takes for a random walk to go between two vertices) and stress majorization, as these metrics are more robust to such shortcuts. Along the same lines, picking an optimal value of k for the speed-accuracy tradeoff is equivalent to choosing a metric that is sufficiently accurate and easy to compute. Although we believe that our choices are reasonable, we leave a full experimental study for future work. We point out that our algorithm might be applicable to find a metric embedding for more general dimensionality reduction situations. In this case, the approximation of $L^X X$ becomes exponentially slower as the embedding dimension increases (the well-known *curse of dimensionality*). We are currently investigating the use of alternative data structures that might help circumventing some of these issues. In addition, these data structures might help solving the case where $w_{ij} = d_{ij}^{-2}$, which despite being the most popular choice in the literature, cannot be applied directly in our setting.

In this paper, we have demonstrated the feasibility of stress majorization for large graphs. Although the resulting layouts we obtain are approximate, we believe our technique opens the field for new investigations on drawing graphs that are currently believed to be too hard to draw.

Acknowledgments

The colors used in the running time plots are due to Cynthia Brewer, from `http://www.colorbrewer2.org`. We acknowledge the fruitful discussions on graph drawing with Stephen North and Emden Gansner, and Peter Lindstrom on commute-time distances.

References

[BH86] BARNES J., HUT P.: A hierarchical O(NlogN) force-calculation algorithm. *Nature 324* (1986), 446–449. 5

[BP07] BRANDES U., PICH C.: Eigensolver methods for progressive multidimensional scaling of large data. In *Proc. 14th Intl. Symp. Graph Drawing (GD '06)* (2007), vol. 4372 of *LNCS*, pp. 42–53. 2, 6, 9

[BP09] BRANDES U., PICH C.: An experimental study on distance based graph drawing. In *Proc. 16th Intl. Symp. Graph Drawing (GD '08)* (2009), vol. 5417 of *LNCS*, Springer-Verlag, pp. 218–229. 3, 9

[Cha96] CHALMERS M.: A linear iteration time layout algorithm for visualizing high dimensional data. In *Proceedings of IEEE Visualization* (1996), pp. 127–132. 2, 9

[Coh97] COHEN J. D.: Drawing graphs to convey proximity: an incremental arrangement method. *ACM Trans. Comput.-Hum. Interact. 4*, 3 (Sept. 1997), 197–229. 9

[DFK*04] DRINEAS P., FRIEZE A. M., KANNAN R., VEMPALA S., VINAY V.: Clustering large graphs via the singular value decomposition. *Machine Learning 56* (2004), 9–33. 1, 2, 5, 7

[DH11] DAVIS T. A., HU Y. F.: University of Florida Sparse Matrix Collection. *ACM Transaction on Mathematical Software 38* (2011), 1–18. http://www.cise.ufl.edu/research/sparse/matrices/. 4

[EGK*01] ELLSON J., GANSNER E. R., KOUTSOFIOS E., NORTH S. C., WOODHULL G.: Graphviz – open source graph drawing tools. In *Graph Drawing* (2001), pp. 483–484. 6

[FC11] FRANCE S., CARROL J. D.: Two-way multidimensional scaling: A review. *IEEE Transactions on System, Man and Cybernetics – Part C 41*, 5 (2011), 644–661. 2

[FKV04] FRIEZE A., KANNAN R., VEMPALA S.: Fast monte-carlo algorithms for finding low-rank approximations. *Journal of ACM 51* (2004), 1025–1041. 3

[FR91] FRUCHTERMAN T. M. J., REINGOLD E. M.: Graph drawing by force directed placement. *Software - Practice and Experience 21* (1991), 1129–1164. 2

[GGK00] GAJER P., GOODRICH M. T., KOBOUROV S. G.: A fast multi-dimensional algorithm for drawing large graphs. *LNCS 1984* (2000), 211 – 221. 2

[GKN04] GANSNER E. R., KOREN Y., NORTH S. C.: Graph drawing by stress majorization. In *Proc. 12th Intl. Symp. Graph Drawing (GD '04)* (2004), vol. 3383 of *LNCS*, Springer, pp. 239–250. 1, 2, 3, 6, 9

[GL96] GOLUB G., LOAN C.: *Matrix computations*. Johns Hopkins studies in the mathematical sciences. Johns Hopkins University Press, 1996. 4

[Hag89] HAGER W. W.: Updating the inverse of a matrix. *SIAM Review 31*, 2 (June 1989), 221–239. 4

[Hal70] HALL K. M.: An *r*-dimensional quadratic placement algorithm. *Management Science 17* (1970), 219–229. 2

[HJ04] HACHUL S., JÜNGER M.: Drawing large graphs with a potential field based multilevel algorithm. In *Proc. 12th Intl. Symp. Graph Drawing (GD '04)* (2004), vol. 3383 of *LNCS*, Springer, pp. 285–295. 2, 5

[HK02] HAREL D., KOREN Y.: Graph drawing by high-dimensional embedding. *lncs* (2002), 207–219. 2, 6

[Hu05] HU Y. F.: Efficient and high quality force-directed graph drawing. *Mathematica Journal 10* (2005), 37–71. 2, 5, 6

[IMO09] INGRAM S., MUNZNER T., OLANO M.: Glimmer: Multilevel MDS on the GPU. *IEEE Trans. Vis. Comp. Graph. 2*, 15 (2009), 249–261. 2

[KCH02] KOREN Y., CARMEL L., HAREL D.: Ace: A fast multiscale eigenvectors computation for drawing huge graphs. In *INFOVIS '02: Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02)* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 137–144. 2

[KK89] KAMADA T., KAWAI S.: An algorithm for drawing general undirected graphs. *Information Processing Letters 31* (1989), 7–15. 2, 3

[Kor04] KOREN Y.: Graph drawing by subspace optimization. In *Proc. VisSym* (2004), pp. 65–74. 2

[KS80] KRUSKAL J. B., SEERY J. B.: Designing network diagrams. In *Proceedings of the First General Conference on Social Graphics* (Washington, D.C., July 1980), U. S. Department of the Census, pp. 22–50. Bell Laboratories Technical Report No. 49. 3

[QE00] QUIGLEY A., EADES P.: Fade: Graph drawing, clustering, and visual abstraction. *LNCS 1984* (2000), 183–196. 5

[Qui01] QUIGLEY A.: *Large scale relational information visualization, clustering, and abstraction*. PhD thesis, Department of Computer Science and Software Engineering, University of Newcastle, Australia, 2001. 2

[San10] SANDERSON C.: *Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments*. Tech. rep., NICTA, 2010. 6

[Ten10] TENG S.-H.: The laplacian paradigm: Emerging algorithms for massive graphs. In *Proceedings of Theory and Applications of Models of Computation* (2010), Kratochvíl J., Li A., Fiala J., Kolman P., (Eds.), pp. 2–14. 3

[Tor52] TORGERSON W. S.: Multidimensional scaling: I. theory and method. *Psychometrika 17* (1952), 401–419. 2

[Tun99] TUNKELANG D.: *A Numerical Optimization Approach to General Graph Drawing*. PhD thesis, Carnegie Mellon University, 1999. 2

[Wal03] WALSHAW C.: A multilevel algorithm for force-directed graph drawing. *J. Graph Algorithms and Applications 7* (2003), 253–285. 2