SSDE: Fast Graph Drawing Using Sampled Spectral Distance Embedding

Ali Çivril, Malik Magdon-Ismail and Eli Bocek-Rivele

Computer Science Department, RPI, 110 8th Street, Troy, NY 12180
{civria,magdon}@cs.rpi.edu,boceke@rpi.edu

Abstract. We present a fast spectral graph drawing algorithm for drawing undirected connected graphs. Classical Multi-Dimensional Scaling yields a quadratic-time spectral algorithm, which approximates the real distances of the nodes in the final drawing with their graph theoretical distances. We build from this idea to develop the linear-time spectral graph drawing algorithm SSDE. We reduce the space and time complexity of the spectral decomposition by approximating the distance matrix with the product of three smaller matrices, which are formed by sampling rows and columns of the distance matrix. The main advantages of our algorithm are that it is very fast and it gives aesthetically pleasing results, when compared to other spectral graph drawing algorithms. The runtime for typical 10^5 node graphs is about one second and for 10^6 node graphs about ten seconds.

1 Introduction

A graph $G = (V, E)$ is a pair where V is the vertex set and E is the edge set, which is a binary relation over V. The graph drawing problem is to compute an aesthetically pleasing layout of vertices and edges so that it is easy to grasp visually the inherent structure of the graph. In this paper, we only consider straight-line edge drawings for which a variety of aesthetic criteria have been studied: number of edge crossings; uniform node densities; symmetry. Depending on the aesthetic criteria of interest, various approaches have been developed, and a general survey can be found in [13, 22].

For straight-line edge drawings, the graph drawing problem reduces to the problem of finding the coordinates of the vertices in two dimensions. A popular approach is to define an energy function or a force-directed model with respect to vertex positions, and to iteratively compute a local minimum of the energy function. The positions of the vertices at the local minimum produce the final layout. This approach is generally simple and easy to extend to new energy functions. Various energy functions and force models have been studied (see for example [6, 12]) and there exist several improvements to handle large graphs, most of them concentrating on a multi-scale paradigm. Multi-scale approaches involve laying out a coarser level of the graph first, and then taking advantage of this coarse layout to compute the vertex positions at a finer level (see for example [9, 24]).

Spectral graph drawing was first proposed by Hall in 1970 [8] and it has become popular recently. We use the term spectral graph drawing to refer to any approach that produces a final layout using the spectral decomposition of some matrix derived from the vertex and edge sets of the graph. A general introduction can be found in [14]. In this paper, we present the spectral graph drawing algorithm SSDE (Sampled Spectral Distance Embedding), using a similar formulation that was introduced in [5], which uses Classical Multi-Dimensional Scaling (CMDS) techniques for graph drawing. CMDS for graph drawing was first introduced in [17] and recently, a similar idea using CMDS technique, was proposed by Koren and Harel in [15] using a slightly different formulation. CMDS uses the spectral decomposition of the graph theoretical distance matrix to produce the final layout of the vertices. In the final layout, the pair-wise Euclidean distances of the vertices approximate the graph theoretical distances. The main disadvantage of this technique from the computational perspective is that one must perform an all-pairs shortest path computation, which takes $O(|V||E|)$ time. The space complexity of the algorithm is also quadratic since one needs to keep all the pair-wise distances. This prevents large graphs having more than $10,000$ nodes from being drawn efficiently.

SSDE uses an approximate decomposition of the distance matrix, reducing the space and time complexity considerably. Some theoretical properties of such matrix decompositions have been studied in [20]. The fact that the distance matrix is symmetric allows us to express the decomposition in a simpler way. SSDE consists of three main steps:

(i) *Sampling:* a constant number c of nodes are sampled from the graph for which the graph theoretical distances to all other nodes are computed. Let the matrices C and R denote the corresponding rows and columns of the distance matrix that have been computed, where $R = C^T$. The complexity of this step is $O(c|E|)$ for unweighted graphs, using BFS for each sampled node.

(ii) *Computing Φ^+:* Based on the information in C and R, we form Φ, which is a $c \times c$ matrix keeping the entries which are common in C and R. Since we need its pseudo-inverse Φ^+, the complexity of this step is $O(c^3)$, which involves computing a pseudo-inverse via the singular value decomposition (SVD) of Φ.

(iii) *Spectral Decomposition:* We find the optimal rank-d spectral reconstruction of the product $C\Phi^+R$, to embed in d-dimensions. The complexity of this step is $O(cd|V|)$, using the power iteration, which finds the largest eigenvalues of a matrix and its associated eigenvectors.

SSDE can be used to produce a d-dimensional embedding, the most practical being $d = 2,3$. We focus on $d = 2$ in this paper. We present the results of our algorithm through several examples, including run-times and embedding errors. Compared to similar techniques, we observe that our algorithm is fast enough to handle graphs up to 10^6 nodes in about 10 seconds. A comparison of SSDE with two popular spectral graph drawing algorithms (HDE and ACE) is given in Figure 1: SSDE produces very good drawings of almost every mesh-like graph we have tried, with comparable or better running times. One of the main exceptions is tree-like graphs or more generally graphs with low algebraic connectivity, which are problematic for all three spectral graph drawing techniques mentioned.

The problem our algorithm addresses is that of embedding a finite metric space in $\mathbb{R}^2$ under the l_2-norm [19]. Most research in this area of mathematics has focused on determining what kinds of finite metric spaces are embeddable using low-distortion

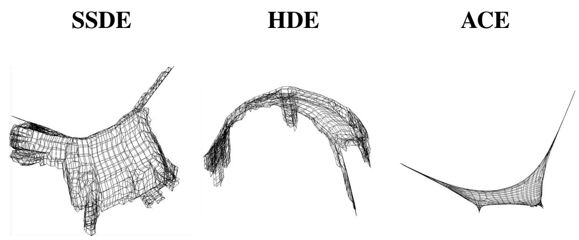

Fig. 1. Comparison of SSDE with other spectral methods (HDE and ACE) on the finite element mesh of a cow with $|V| = 1820, |E| = 7940$.

embeddings. Our work does not provide any guarantees on the distortion of the resulting embedding, which is an active area of research. We do, however, give the intuition behind why our algorithm constructs a good embedding using limited data on the distances between the points. Another paper using this kind of approach is [21], which introduces a different formulation via the Nystrom approximation.

1.1 Related Work

There are general methods to draw graphs, and detailed information about different approaches can be found in [13, 22]. Our algorithm is based on spectral decomposition which yields the problem of computing the eigenvalues and eigenvectors of certain matrices related to the structure of the graph. The formulation is mathematically clean, in that exact solutions can be found, because eigenvectors and eigenvalues can be computed exactly in $O(|V|^3)$ time. Our work falls within the category of fast spectral graph drawing algorithms, which is the related work we elaborate upon.

High-Dimensional Embedding (HDE) described in [10] by Harel and Koren embeds the graph in a high dimension (typically 50) with respect to carefully chosen pivot nodes. One then projects the coordinates into two dimensions by using a well-known multivariate analysis technique called principal component analysis (PCA), which involves computing the first few largest eigenvalues and eigenvectors of the covariance matrix of the points in the higher dimension.

ACE (Algebraic multigrid Computation of Eigenvectors) [16] minimizes Hall's Energy function $E = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2$ in each dimension, modulo some non-degeneracy and orthogonality constraints (n is the number of nodes, x_i is the one-dimensional coordinate of the i^{th} node and w_{ij} is the weight of the edge between i and j). This minimization problem can be reduced to obtaining the eigen-decomposition of the Laplacian of the graph. A multi-scaling approach is also used, creating coarser levels of the graph and relating them to the finer levels using an interpolation matrix.

Both of the methods described above are fast due to the small sizes of the matrices processed. Specifically, the running time of ACE depends on the structure of the graph while HDE provides better image quality and run-times. But, they may result in aesthetically unpleasant drawings of certain graphs and some of these problems are illustrated in Figure 1.

1.2 Notation

We use $i, j, k, \ldots$ for indices of vectors and matrices, bold uncapitalized letters $\mathbf{x}, \mathbf{y}, \mathbf{z}$ for vectors in $\mathbb{R}^d$ and bold capitalized letters for matrices. Typically, $\mathbf{M}, \mathbf{N}$ are used to represent $n \times n$ matrices and $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ for $n \times d$ matrices, which represent n vectors in $\mathbb{R}^d$. $\mathbf{A}_{(i)}$ denotes the i^{th} row of the matrix $\mathbf{A}$ and $\mathbf{A}^{(i)}$ denotes its i^{th} column. The pseudo-inverse of a matrix $\mathbf{A}$ is denoted as $\mathbf{A}^+$. The norm of a vector $\|\mathbf{x}\|$ is the standard Euclidean norm. The transpose of a vector or a matrix is denoted as $\mathbf{x}^T, \mathbf{M}^T$.

2 Spectral Decomposition of the Distance Matrix and CMDS

Given a graph $G = (V, E)$ with n nodes, let $V = \{v_1, v_2, \ldots, v_n\}$. The distance matrix $\mathbf{D}$ is the symmetric $n \times n$ matrix containing all the pair-wise distances, i.e., D_{ij} is the length of the shortest path between v_i and v_j. Suppose that the position at which vertex v_i is placed is $\mathbf{x}_i$. We are seeking a positioning that approximates the graph theoretical distances with the Euclidean distances, i.e,

$$\|\mathbf{x}_i - \mathbf{x}_j\| \approx D_{ij}, \qquad \text{for } i, j = 1, 2, \ldots, n. \tag{1}$$

A suitable algebraic manipulation as presented in [4] on (1) yields the following equation, which is the basic idea of CMDS:

$$\mathbf{Y}\mathbf{Y}^T \approx -\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma} = \mathbf{M}. \tag{2}$$

where $\mathbf{Y}$ is an $n \times d$ matrix containing the coordinates of the points, $\mathbf{L}$ is the $n \times n$ matrix keeping the squares of the distances between nodes and $\boldsymbol{\gamma} = \frac{1}{n}\mathbf{I}_n - \mathbf{1}_n\mathbf{1}_n{}^T$ is a projection matrix. We want to approximate $\mathbf{M}$ as closely as possible. The metric that CMDS chooses is the spectral norm, so we wish to find the best rank-d approximation to $\mathbf{M}$ with respect to the spectral norm. This is a well-known problem, which is equivalent to finding the largest d eigenvalues of $\mathbf{M}$. The final centralized coordinates are then given by $\mathbf{Y} = [\sqrt{\lambda_1}\mathbf{u}_1, \ldots, \sqrt{\lambda_d}\mathbf{u}_d]$, where $\lambda_1, \ldots, \lambda_d$ are the first d eigenvalues of $\mathbf{M}$ and $\mathbf{u}_1, \ldots, \mathbf{u}_d$ are the associated eigenvectors.

CMDS(G)
1: Compute $\mathbf{D}$ using an APSP algorithm on G
2: Define matrix $\mathbf{L}$ such that $L_{ij} = D_{ij}^2$.
3: **return** $\mathbf{Y}$ = PowerIteration($-\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma}, \varepsilon$)

Fig. 2. The spectral graph drawing algorithm based on CMDS.

Finding a rank-d approximation of $\mathbf{M} = -\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma}$, which corresponds to computing the largest d eigenvalues and eigenvectors, is performed by a standard procedure typically referred to as the power iteration, rather than by an exact algorithm which would have $O(|V|^3)$ time complexity.

3 Approximate Distance Matrix Reconstruction

The running time of the CMDS technique is quadratic in terms of the number of nodes even for sparse graphs since one needs to compute and store all-pairs' shortest path lengths. In this section, we will first briefly explain the intuition behind SSDE, which breaks the quadratic complexity of this technique and actually yields a fast, linear time algorithm. Then, we will present the mathematical formulation.

3.1 Intuition

SSDE tries to construct an approximation to the distance matrix without computing all the entries in it. In the previous section, we noted that the distance matrix might have rank larger than d. But, the rank of the distance matrix is expected to be small in terms of the number of nodes in the graph, even if it is larger than d. This intuitive reasoning stems from a famous result from low distortion embedding theory. In 1984, Johnsson and Lindenstrauss [11] proved that n points in high dimension can be embedded into $O(\frac{\log n}{\varepsilon^2})$ dimensions with ε distortion. This means, roughly speaking that the set of points can be reconstructed in low dimension while preserving all the pair-wise distances and hence that the effective rank of the distance matrix is much smaller than its full dimension. This suggests that one can extract much of the information about the matrix by performing computations on matrices having much smaller ranks.

Specifically, SSDE approximates the distance matrix with the product of three smaller matrices, which have linear size in terms of the number of nodes in the graph. In order to do this, reasoning from the fact that the distance matrix has low rank, the columns of the distance matrix can approximately be expressed as a linear combination of a small number of its columns. The algorithm essentially consists of choosing this small number of columns, constructing the whole matrix appropriately and computing the coordinates of the vertices via the spectral decomposition of this matrix. A variant of the particular approximation that we will use has been studied in [20]. In [20], the sampling approach used assumes that the whole matrix is known using one pass. Since, this would lead to a quadratic time algorithm, our approach must use online sampling. One can either sample the columns randomly or use a simple greedy algorithm, which seems to give a better set of columns.

3.2 Formulation

Let $i_1, i_2, \ldots, i_c$ be a set of distinct indices where c is a predefined positive integer smaller than n and $1 \leq i_k \leq n$ for $k = 1, \ldots c$. Let $\mathbf{C} = [\mathbf{L}^{(i_1)}, \mathbf{L}^{(i_2)}, \ldots, \mathbf{L}^{(i_c)}]$. If $\mathbf{C}$ is chosen carefully, under the assumptions mentioned above, any column $\mathbf{L}^{(i)}$ can approximately be written as a linear combination of the columns of $\mathbf{C}$, i.e.

$$\mathbf{L}^{(i)} \approx \mathbf{C}\boldsymbol{\alpha}^{(i)} \qquad \text{for } i = 1, 2, \ldots, n, \tag{3}$$

where $\boldsymbol{\alpha}^{(i)}$ is a $c \times 1$ vector. Denoting $\boldsymbol{\alpha} = [\boldsymbol{\alpha}^{(1)}, \boldsymbol{\alpha}^{(2)}, \ldots, \boldsymbol{\alpha}^{(n)}]$, we have

$$\mathbf{L} \approx \mathbf{C}\boldsymbol{\alpha} \tag{4}$$

Let $\boldsymbol{\Phi}$ be the $c \times c$ matrix such that $\boldsymbol{\Phi}_{jk} = \mathbf{L}_{i_j i_k}$ for $j,k = 1,\dots c$. Note that since we also have $\mathbf{C}^T = [\mathbf{L}_{(i_1)}, \mathbf{L}_{(i_2)}, \dots, \mathbf{L}_{(i_c)}]$, $\boldsymbol{\Phi}$ can be interpreted as the intersection of $\mathbf{C}$ and $\mathbf{C}^T$ on the matrix $\mathbf{L}$. Now, since the columns of $\mathbf{L}$ can approximately be expressed as a linear combination of the columns of $\mathbf{C}$, the columns of $\mathbf{C}^T$ can also be expressed as a linear combination of the columns of $\boldsymbol{\Phi}$. This gives

$$\mathbf{C}^T \approx \boldsymbol{\Phi}\boldsymbol{\alpha} \tag{5}$$

where $\boldsymbol{\alpha}$ is the same matrix as we defined above. If $\boldsymbol{\Phi}$ has full rank, (5) yields $\boldsymbol{\alpha} = \boldsymbol{\Phi}^{-1}\mathbf{C}^T$. Combining this with (4), we have $\mathbf{L} = \mathbf{C}\boldsymbol{\Phi}^{-1}\mathbf{C}^T$. More generally, we do not assume that $\boldsymbol{\Phi}$ has full rank, so we have

$$\mathbf{L} \approx \mathbf{C}\boldsymbol{\Phi}^{+}\mathbf{C}^T \tag{6}$$

where $\boldsymbol{\Phi}^{+}$ is the pseudo-inverse of $\boldsymbol{\Phi}$ (See [7] for the definition of pseudo-inverse). The last expression indicates that we can approximate the distance matrix $\mathbf{L}$ by the multiplication of three smaller matrices, which all have at most linear size in terms of n. Note that $\mathbf{C}$ is $n \times c$ and $\boldsymbol{\Phi}$ is $c \times c$.

4 The Algorithm SSDE

The algorithm SSDE, which uses the procedures that we will define shortly is summarized in Figure 5. As stated in the introduction, the algorithm consists of three main steps:

(1) *Sampling:* The first step of the algorithm is to compute the columns that define $\mathbf{C}$ and $\boldsymbol{\Phi}$. This is equivalent to choosing a particular set of nodes and computing the graph theoretical distances to all other nodes in the graph. We propose two methods to sample c nodes:

   (i) *Random Sampling:* The c nodes are sampled uniformly at random.

   (ii) *Greedy Sampling:* The first node is chosen uniformly at random. Then, at each step, we choose the furthest node to the set of nodes that have already been chosen until c nodes are chosen.

   Note that, the second method stated above is also known to be a 2-approximation algorithm to the k-center problem [23]. This method was also used in [9] and [10]in different contexts. The procedure for performing these operations is presented in Figure 3. Even though c can be treated as a parameter to the algorithm, we have experienced that setting $c = 25$ is enough for getting good results on practically all graphs we have tried. The sampling step, overall requires $O(c|E|)$ time as we initiate a BFS from c nodes in the graph.

(2) *Computing Φ^{+}:* We find the pseudo-inverse $\boldsymbol{\Phi}^{+}$ by first computing the singular value decomposition $\boldsymbol{\Phi} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$, which can be performed in $O(c^3)$ time using standard procedures (see for example [7]). The pseudo-inverse can then be computed by the expression $\boldsymbol{\Phi}^{+} = \mathbf{V}\boldsymbol{\Sigma}^{+}\mathbf{U}^T$. Here, $\boldsymbol{\Sigma}^{+}$ is the diagonal matrix keeping the reciprocals of the non-zero singular values, which are stored in $\boldsymbol{\Sigma}$. Unfortunately, in order to get numerically stable results, it is not enough to compute the

ComputeCandPhi(G, method, c)

if method = random **then**
  Select c vertices uniformly at random
  for $k = 1$ to c **do**
    $\mathbf{C}^k \leftarrow dist(i_k, V)$ // BFS
  end for
else if method = greedy **then**
  $i_1 \leftarrow unifrnd(1, |V|)$ // Choose uniformly at random
  $\mathbf{C}^1 \leftarrow dist(i_1, V)$ // BFS
  for $k = 2$ to c **do**
    $i_k \leftarrow \max\limits_{1 \le j \le n} \min\limits_{1 \le l \le k} \{\mathbf{C}_{jl}\}$ // Choose the furthest node
    $\mathbf{C}^k \leftarrow dist(i_k, V)$ // BFS
  end for
end if
Compute $\boldsymbol{\Phi}$ // $\boldsymbol{\Phi}_{kj} = \mathbf{C}_{i_k j}$
return $(\mathbf{C}, \boldsymbol{\Phi})$

Fig. 3. The procedure computing the matrices $\mathbf{C}$ and $\boldsymbol{\Phi}$.

reciprocals of the singular values, since the small singular values which are close to zero should actually be ignored, as they may be the result of numerical imprecision and will result in huge instability in $\boldsymbol{\Sigma}^+$. To prevent such instability, we use a *regularization* method that was presented in [18], which uses the expression

$$\frac{\sigma_i}{{\sigma_i}^2 + \alpha / {\sigma_i}^2} \tag{7}$$

for the reciprocals in $\boldsymbol{\Sigma}^+$, where σ_i is the i^{th} diagonal entry in $\boldsymbol{\Sigma}$. The parameter α is the regularization parameter, which must be chosen judiciously in order not to distort the reciprocals of the large singular values too much. On the other hand, it should result in values close to zero for the small singular values. Our experiments revealed that $\alpha = {\sigma_1}^3$ is good enough for practical purposes where σ_1 is the largest singular value. However, we keep it as a parameter of the procedure.

(3) *Spectral Decomposition:* Having computed the pseudo-inverse of $\boldsymbol{\Phi}$, we compute $\hat{\mathbf{L}} = \mathbf{C}\boldsymbol{\Phi}^+\mathbf{C}^T$ from which we obtain $\hat{\mathbf{M}} = -\frac{1}{2}\boldsymbol{\gamma}\hat{\mathbf{L}}\boldsymbol{\gamma}$. Then, analogous to (2), we obtain the coordinates of the points in the embedding using the spectral decomposition of $\hat{\mathbf{M}}$, which approximates $\mathbf{M}$. This requires computing the top d eigenvalues and eigenvectors, for which we use a standard procedure called the power iteration (See Figure 4). In the power iteration, the main computational task is to repetitively multiply a randomly chosen vector with the matrix whose eigenvalues and eigenvectors are sought. In our power iteration, starting from the right, the matrix-vector multiplications (line 5 and line 15) can be performed using $O(c|V|)$ scalar additions and multiplications. The total number of iterations until a predefined convergence condition holds, depends on the matrix processed. But, since the convergence is

exponential (see for example [7]), in practice, a constant number of iterations is enough. Overall, the running time of the power iteration step of the algorithm is $O(c|V|)$.

PowerIteration($\mathbf{C}$, $\boldsymbol{\Phi}^+$, ε)
$\mathit{current} \leftarrow \varepsilon$; $\mathbf{y}_1 \leftarrow \mathbf{random}/\|\mathbf{random}\|$
repeat
  $\mathit{prev} \leftarrow \mathit{current}$
  $\mathbf{u}_1 \leftarrow \mathbf{y}_1$
  $\mathbf{y}_1 \leftarrow -\frac{1}{2}\boldsymbol{\gamma}\mathbf{C}\boldsymbol{\Phi}^+\mathbf{C}^T\boldsymbol{\gamma}\mathbf{u}_1$
  $\lambda_1 \leftarrow \mathbf{u}_1 \cdot \mathbf{y}_1$ % compute the eigenvalue
  $\mathbf{y}_1 \leftarrow \mathbf{y}_1/\|\mathbf{y}_1\|$
  $\mathit{current} \leftarrow \mathbf{u}_1 \cdot \mathbf{y}_1$
until $|\mathit{current}/\mathit{prev}| \leq 1+\varepsilon$
$\mathit{current} \leftarrow \varepsilon$; $\mathbf{y}_2 \leftarrow \mathbf{random}/\|\mathbf{random}\|$
repeat
  $\mathit{prev} \leftarrow \mathit{current}$
  $\mathbf{u}_2 \leftarrow \mathbf{y}_2$
  $\mathbf{u}_2 \leftarrow \mathbf{u}_2 - \mathbf{u}_1(\mathbf{u}_1 \cdot \mathbf{u}_2)$ % orthogonalize against $\mathbf{u}_1$
  $\mathbf{y}_2 \leftarrow -\frac{1}{2}\boldsymbol{\gamma}\mathbf{C}\boldsymbol{\Phi}^+\mathbf{C}^T\boldsymbol{\gamma}\mathbf{u}_2$
  $\lambda_2 \leftarrow \mathbf{u}_2 \cdot \mathbf{y}_2$ % compute the eigenvalue
  $\mathbf{y}_2 \leftarrow \mathbf{y}_2/\|\mathbf{y}_2\|$
  $\mathit{current} \leftarrow \mathbf{u}_2 \cdot \mathbf{y}_2$
until $|\mathit{current}/\mathit{prev}| \leq 1+\varepsilon$
return ($\sqrt{\lambda_1}\mathbf{y}_1$ $\sqrt{\lambda_2}\mathbf{y}_2$)

Fig. 4. The power iteration method for finding eigenvectors and eigenvalues ($d = 2$).

The embedding is obtained directly from the eigenvectors and eigenvalues, which are returned by the power iteration.

5 Results

We have implemented our algorithm in C++, and Table 1 gives the running time results on a Pentium 4HT 3.0 GHz processor system with 1 GB of memory. We present the results of running the algorithm on several graphs of varying sizes up to about $2,000,000$ nodes. We set $c = 25$, since our experiments have revealed that this is enough to get good drawings. For the power iteration, we set the tolerance $\varepsilon = 10^{-7}$. The running times in Table 1 do not include the file I/O that is used to access and store the coordinates of the nodes. In Table 1, we present the results for CMDS and SSDE with $c = 25, 50$ and greedy sampling. Along with the running time, we also give the Frobenius norm of the relative error matrix for the embedding, $\boldsymbol{\varepsilon}$, where $\varepsilon_{ij} = 1 - D'_{ij}/D_{ij}$ and $\boldsymbol{D}, \boldsymbol{D'}$ are the

SSDE(G, method)
$(\mathbf{C}, \boldsymbol{\Phi}) \leftarrow$ ComputeCandPhi(G, method, c)
$(\mathbf{U}, \boldsymbol{\Sigma}, \mathbf{V}^T) \leftarrow$ SVD($\boldsymbol{\Phi}$)
$\boldsymbol{\Sigma}^+ \leftarrow$ Regularize($\boldsymbol{\Sigma}$, α)
$\boldsymbol{\Phi}^+ \leftarrow \mathbf{V}\boldsymbol{\Sigma}^+\mathbf{U}^T$
return $\mathbf{Y}$ = PowerIteration($\mathbf{C}, \boldsymbol{\Phi}^+, \varepsilon$)

Fig. 5. The spectral graph drawing algorithm SSDE.

true distance matrix and distance matrix implied by the embedding respectively. The normalized Frobenius errors computed in Table 1 are defined as

$$\|\boldsymbol{\varepsilon}_{F'}\| = \sqrt{\frac{1}{n^2}\sum_{i \neq j}(1 - \frac{D'_{ij}}{D_{ij}})^2}. \tag{8}$$

These errors might be interpreted as a quantification of the quality of the embedding, and can be used to compare SSDE to CMDS. As can be inferred from Table 1, SSDE is a good approximation to CMDS, which becomes more so as c increases.

SSDE is able to draw graphs up to 10^6 nodes in about ten seconds, which is comparable to the other fast spectral methods. The last three graphs in the table are road maps of states [1]. As is empirically verified from these graphs, the asymptotic running time of the algorithm is linear. Figure 6 demonstrates the quality of the drawings for some benchmark graphs. In all the graphs except finan512, we used the greedy sampling method. Random sampling seems to work better for finan512 because of its special structure. We have observed that the algorithm is able to reveal the general structure of almost all the graphs we tested, as well as the finer structure of some of the graphs successfully, where other spectral methods have difficulty. An example is the finan512 graph, where the overall structure is clearly visible, and one can also see the finer structure of the small "towers" attached to the main cycle. Figure 7 compares the results of the exact algorithm CMDS, and SSDE, which is approximate but far more efficient. We demonstrate the results of SSDE for both random and greedy sampling. The figure shows that SSDE does not sacrifice much in the way of picture quality as compared to CMDS. For all the drawings mentioned, it is important to note that exact pictures may change depending on which specific nodes are sampled, but the typical structure is consistent. The quality of the drawing for random and greedy sampling also doesn't differ much, but our experiments showed that the greedy sampling tends to give more consistent results.

Graph	|V|	|E|	CMDS		SSDE(c=25)		SSDE(c=50)	
			$\|\varepsilon_{F'}\|$	Time(sec)	$\|\varepsilon_{F'}\|$	Time(sec)	$\|\varepsilon_{F'}\|$	Time(sec)
3elt	4720	13722	0.382	8.47	0.432	0.015	0.398	0.04
sierpinski08	9843	19683	0.17	24.72	0.203	0.03	0.19	0.07
Grid 100x100	10000	19800	0.17	29.73	0.192	0.03	0.186	0.06
crack	10240	30380	0.085	45.00	0.103	0.045	0.09	0.10
4elt2	11143	32818	0.252	48.77	0.291	0.07	0.283	0.14
4elt	15606	45878	0.308	133.33	0.375	0.13	0.342	0.25
sphere	16386	49152	0.291	136.69	0.334	0.14	0.312	0.27
finan512	74752	261120	-	-	-	0.68	-	1.43
ocean	143437	409593	-	-	-	1.65	-	3.56
144	144649	1074393	-	-	-	2.85	-	6.03
wave	156317	1059331	-	-	-	2.40	-	4.78
auto	448695	3314611	-	-	-	9.96	-	21.67
Florida	1048506	1330551	-	-	-	10.04	-	23.45
California	1613325	1989149	-	-	-	17.91	-	36.13
Texas	2073870	2584159	-	-	-	21.69	-	45.89

Table 1: Running time and embedding errors of CMDS and SSDE for several graphs. (Most of these graphs can be downloaded from [1], [2] and [3]). Missing entries are graphs where it was too costly to compute the entire distance matrix.

6 Conclusion and Future Work

We have presented a fast spectral graph drawing algorithm, which significantly improves the idea of Classical Multi-Dimensional Scaling (CMDS), by using sampling techniques over nodes to reduce the time complexity of computing the distance matrix. We use a sparse approximation to the distance matrix obtained through sampling. The spectral decomposition of this sampled matrix yields the desired embedding. The running time of our algorithm is mainly governed by the shortest path computations for the sampled nodes and the power iteration procedure where we compute the coordinates of the points via the spectral decomposition, which in total is linear in the size of the graph. SSDE gives competitive running times with very good drawings for a broad range of graphs, and at the same time it does not sacrifice quality as compared to CMDS.

The typical graphs for which SSDE is not suited are graphs with low algebraic connectivity (such as trees for which special purpose algorithms exist) and dense graphs which are difficult to visualize anyway. Usually, as the graph gets denser, the sampled nodes cannot extract enough information about the spectrum of the distance matrix. We would like to mention that this shortcoming of SSDE applies to many real world graphs. However, these are issues faced by all the fast spectral methods discussed here.

The rigorous mathematical analysis of sampling methods and specifically their implications on the error of the difference between the real distance matrix and the approximation is the context of future work. The sampling step intuitively tries to pick a set of columns whose volume in $|V|$ dimensions is as large as possible, which implies a better approximation to the distance matrix. An interesting problem would be to consider the performance of greedy sampling with respect to the optimal choice of samples.

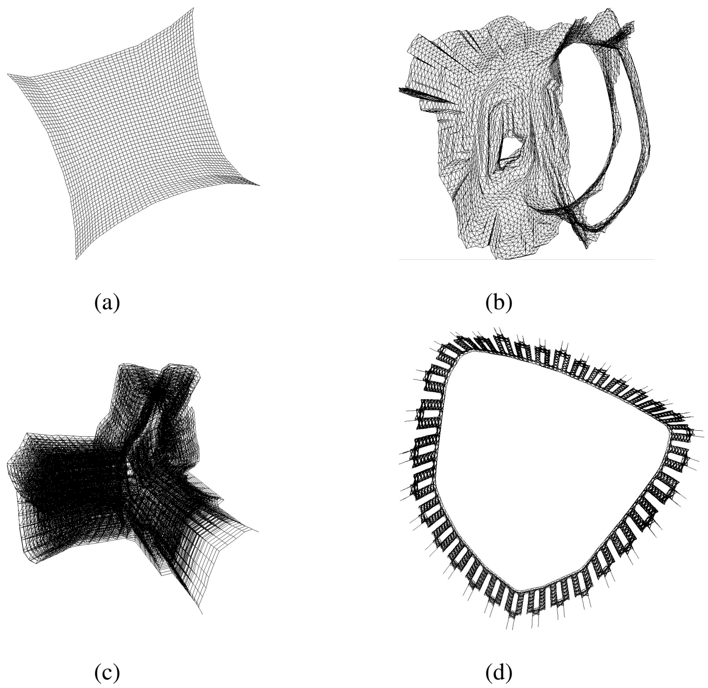

Fig. 6. Layouts of (a) 50x50 grid with $|V| = 2500, |E| = 4900$, (b) 3elt with $|V| = 4720, |E| = 13722$, (c) cti with $|V| = 16840, |E| = 48232$, (d) finan512 with $|V| = 74752, |E| = 261120$.

References

1. `http://www.dis.uniroma1.it/~challenge9/data/tiger/`.
2. `http://wwwcs.uni-paderborn.de/fachbereich/AG/monien/RESEARCH/PART/graphs.html`.
3. `http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/`.
4. I. Borg and P. Groenen. *Modern Multidimensional Scaling*. Springer-Verlag, 1997.
5. A. Çivril, M. Magdon-Ismail, and E. Bocek-Rivele. SDE: Graph drawing using spectral distance embedding. In *GD'05*, pages 512–513, 2005.
6. T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice And Experience*, 21(11):1129–1164, 1991.
7. G. H. Golub and C.H. Van Loan. *Matrix Computations*. Johns Hopkins U. Press, 1996.
8. K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17:219–229, 1970.
9. D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. In *GD'00*, volume 1984, pages 183–196, 2000.
10. D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. In *GD'02*, 2002.
11. W. Johnson and J. Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemp. Math.*, 26:189–206, 1984.
12. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

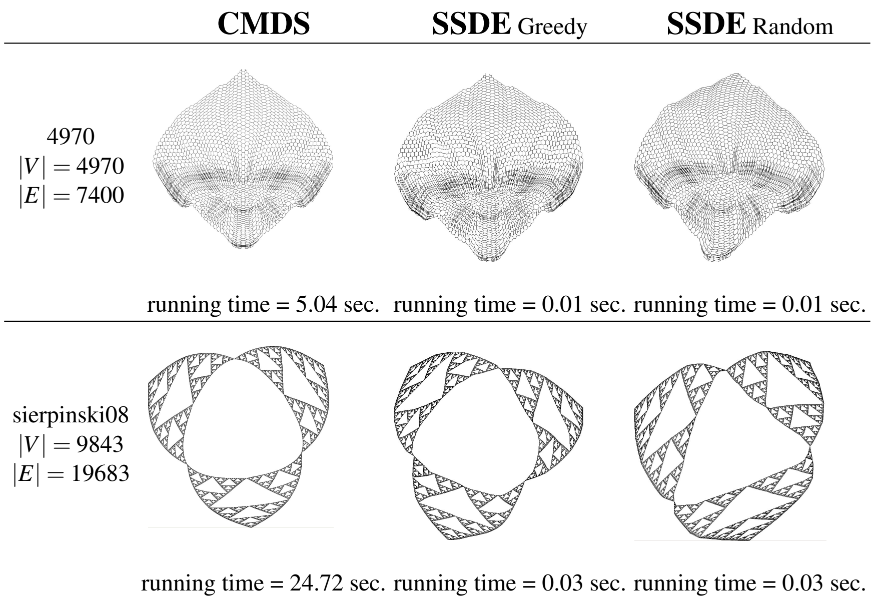

Fig. 7. Comparison of pure CMDS and SSDE

13. M. Kaufmann and D. Wagner, editors. *Drawing Graphs: Methods and Models*. Number 2025 in LNCS. Springer-Verlag, 2001.
14. Y. Koren. On spectral graph drawing. In *COCOON 03*, volume 2697, pages 496–508, 2003.
15. Y. Koren. One dimensional layout optimization, with applications to graph drawing by axis separation. *Computational Geometry: Theory and Applications*, 32:115–138, 2005.
16. Y. Koren, D. Harel, and L. Carmel. Drawing huge graphs by algebraic multigrid optimization. *Multiscale Modeling and Simulation*, 1(4):645–673, 2003. SIAM.
17. J. B. Kruskal and J. B. Seery. Designing network diagrams. In *Proc. First General Conference on Social Graphics*, 1980.
18. J. Maeda and K. Murata. Restoration of band-limited images by an iterative regularized pseudoinverse method. *Journal of Optical Society of America*, 1(1):28–34, 1984.
19. J. Matousek. Open problems on embeddings of finite metric spaces. *Discr. Comput. Geom.*, to appear.
20. P.Drineas, R. Kannan, and M. W. Mahoney. Fast Monte Carlo algorithms for matrices III: Computing a compressed approximate matrix decomposition. *SIAM Journal on Computing*, 36(1):184–206, 2006.
21. J. C. Platt. FastMap, MetricMap, and landmarkMDS are all Nystrom algorithms. In *Proc. 10th Int. Workshop on Artificial Intelligence and Statistics*, pages 261–268, 2005.
22. I. G. Tollis, G. Di Battista, P. Eades, and R. Tamassia. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.
23. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.
24. C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *GD'00*, volume 1984, 2000.