International Journal of Foundations of Computer Science

SDE: GRAPH DRAWING USING SPECTRAL DISTANCE EMBEDDING *

ALI CIVRIL†
Computer Science Department,
Rensselaer Polytechnic Institute,
110 8th Street, Troy, New York 12180, United States of America

and

MALIK MAGDON-ISMAIL
Computer Science Department,
Rensselaer Polytechnic Institute,
110 8th Street, Troy, New York 12180, United States of America

and

ELI BOCEK-RIVELE
Computer Science Department,
Rensselaer Polytechnic Institute,
110 8th Street, Troy, New York 12180, United States of America

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

We present a novel algorithm for drawing undirected connected graphs, by using a spectral decomposition of the distance matrix to approximate the graph theoretical distances. The main advantages of our algorithm are that it is "exact" (as opposed to iterative), and it gives results that preserve symmetry and uniform node density, i.e., the drawings are aesthetically pleasing. Our approach has the benefits of fast spectral techniques, but at the same time it produces drawings of a quality comparable to or better than the much slower force-directed approaches. The computational complexity of our algorithm is governed by its two main steps: distance matrix computation using an all-pairs shortest path algorithm, which is $O(|V||E|)$; and low-order spectral decomposition, which is $O(|V|^2)$. The runtime for typical $20,000$ node graphs ranges from 100 to 150 seconds.

Keywords: eigenvector, singular value decomposition, power iteration, finite metric

---

*A preliminary version of this paper will appear in the 13th International Symposium on Graph Drawing GD'05.

†To whom correspondence should be addressed.

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. The quantification of the notion "aesthetically pleasing" in an efficiently tractable way is one of the main tasks in graph drawing. 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; graph theoretic distance preservation (distortion); symmetry. Depending on the aesthetic criteria of interest, various approaches have been developed, and a general survey can be found in [14, 21].

For the 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 [4, 5, 6, 13] 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 [7, 9, 10, 18, 23]. The force-directed approach can accommodate complex aesthetic criteria, however the more complex the criteria, the less efficient the algorithm. Typical force-directed algorithms incorporate a repulsion term and an attraction term, and tend to give aesthetically pleasing drawings. The main drawbacks of force-directed algorithms are that they are inefficient (even in multi-scale approaches) and are sensitive to the ratio of repulsive to attraction forces.

Spectral graph drawing approaches have 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. In this paper, we present a spectral graph drawing algorithm SDE (Spectral Distance Embedding), in which we use 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. SDE consists of two main steps:

(i) all-pairs shortest path computation, which takes $O(|V||E|)$ time.

(ii) spectral decomposition of the distance matrix, in which we find the optimal rank-d reconstruction to embed in d-dimensions. The complexity of this step is $O(d|V|^2)$.

SDE 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

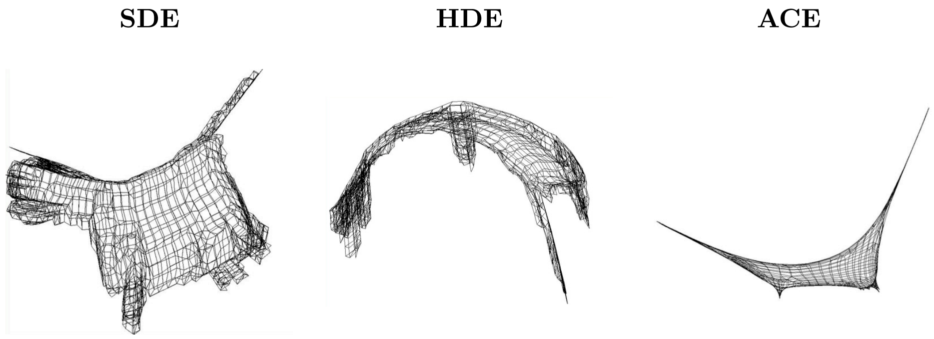


Figure 1: Comparison of SDE with other spectral methods (HDE and ACE) on the finite element mesh of a cow with $|V| = 1820, |E| = 7940$.

our algorithm through several examples, including run-times. Compared to similar techniques, we observe that our results achieve superior drawings, while at the same time not significantly sacrificing computation time. A comparison of SDE with two popular spectral graph drawing algorithms (HDE and ACE) is given in Figure 1 (Additional comparisons are given in Figures 7 and 8). ACE and HDE are very efficient spectral graph drawing techniques, which give reasonably good drawings for some graphs [11, 16], but can give drawings with very large distortion on many (even simple) graphs. On the other hand, SDE produces good drawings on almost every graph we have tried, with reasonable running times. The main exception is trees which are problematic for all three spectral graph drawing techniques; however, many good specialized algorithms exist for trees [21].

The breakdown of the paper is as follows: first, we discuss some related work on spectral graph drawing. In Section 2, we discuss the spectral decomposition of the distance matrix, followed by the algorithm and the results in Sections 3 and 4 respectively. We then give an analysis of the performance characteristics of the algorithm, followed by some concluding remarks in Section 6, where we also discuss possible improvements to the algorithm.

1.1. Related Work

There are general methods to draw graphs and detailed information about different approaches can be found in [14, 21]. Our algorithm is based on spectral decomposition which is a relatively new approach to graph drawing. Spectral graph drawing formulates graph drawing as a 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 (as opposed to iterative) solutions can be found, because eigenvectors and eigenvalues can be computed exactly in $O(|V|^3)$ time. Our work falls within the category of spectral graph drawing, which is the related work we elaborate upon.

The method described in [11] 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. The running time of the algorithm is $O(m|E| + m^2|V|)$ which is driven by the computation of the $m \times m$ covariance matrix and the power iteration to compute the largest eigenvalues and associated eigenvectors, where m is the dimension in which the graph is drawn in the first stage.

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. The multi-scaling allows the power iteration to converge faster as the initial starting vectors in the power iteration makes use of the previously calculated results at the coarser levels. An iterative approach to minimizing Hall's energy E could also be used and results in an update of the form, $x_i = \frac{\sum_{j \in N(i)} x_j}{|N(i)|}$, i.e., x_i is placed at the center of mass of its neighbors. This basic method was first introduced by Tutte [22], and is known as *the barycenter method*. To avoid the degenerate solution in which all the nodes are placed at the same location, Tutte proposed to split the nodes into two sets S_{fixed} and $S_{variable}$. The nodes in S_{fixed} are "nailed" to the corners of a polygon, and the nodes in $S_{variable}$ are updated iteratively. In [15], all of the nodes are positioned simultaneously by solving a constrained quadratic optimization. The solution once again reduces to the eigen-decomposition of a matrix associated with the graph.

Both of the methods described above are fast due to the small sizes of the matrices processed. Specifically, ACE also takes advantage of the simple form of Hall's energy function by using a multi-scaling approach to the eigen-decomposition. The drawings reflect the general structure of the graph, however there is nothing that prevents the nodes from becoming too close to one another since there is no repulsion term in the energy function. This may result in aesthetically unpleasant drawings of certain graphs and some of these problems are illustrated in Figures 1, 7 and 8.

We propose a new spectral graph drawing algorithm that explicitly approximates the graph theoretical distances between nodes. It sits between the fast spectral methods, which may sacrifice on quality, and slow force-directed approaches, which produce high quality drawings. Other related algorithms that try to embed distance matrices on manifolds and which have been used in different contexts (localization from incomplete distance matrices and dimensionality reduction using local distance information) are Multi-Dimensional Scaling [12], Semi-Definite Embedding [3], Isomap [20] and Locally Linear Embedding [19].

The problem our algorithm (SDE) addresses is that of embedding a finite metric space in $\mathbb{R}^2$ under the l_2-norm [17]. Most research in this area of mathematics has focused on determining what kinds of finite metric spaces are embeddable using low distortion embeddings. Our work does not provide any guarantees on the distortion of the resulting embedding, which is an active area of research. However, we do provide some results indicating that if a low distortion optimal embedding exists, then SDE will find an embedding with near-optimal distortion.

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 $\mathbb{R}^d$. The norm of a vector $\| \mathbf{x} \|$ is the standard Euclidean norm and for a matrix, we mostly use the Frobenius norm $\| \mathbf{M} \|_F$ or the spectral norm $\| \mathbf{M} \|_S$ [8]. The transpose of a vector or a matrix is denoted as $\mathbf{x}^T, \mathbf{M}^T$.

2. Spectral Decomposition of the Distance Matrix

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 shortest path length 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,

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

Taking squares of both sides, we have

$$\mathbf{x}_i{}^2 + \mathbf{x}_j{}^2 - 2\mathbf{x}_i \cdot \mathbf{x}_j \approx D_{ij}^2. \tag{2}$$

To write this expression in matrix notation, we will need to define some special matrices. Let $\mathbf{L}$ be an $n \times n$ symmetric matrix such that $L_{ij} = D_{ij}^2$, for $i, j = 1, 2, \ldots, n$. Let $\mathbf{X}, \mathbf{Q}$ and $\mathbf{1}_n$ be defined as follows:

$$\mathbf{X}^T = [\mathbf{x}_1, \ldots, \mathbf{x}_n], \quad \mathbf{Q}^T = [\|\mathbf{x}_1\|^2, \ldots, \|\mathbf{x}_n\|^2], \quad \mathbf{1}_n^T = [1, \ldots, 1].$$

Note that $\mathbf{X}$ is an $n \times d$ matrix containing the positions, $\mathbf{Q}$ is an $n \times 1$ matrix containing the magnitude of the positions and $\mathbf{1}_n$ is the $n \times 1$ vector of ones. We discuss general d; however, $d = 2$ is the case of practical interest. Now (2) can be written as

$$[\mathbf{Q}\mathbf{1}_n{}^T]_{ij} + [\mathbf{Q}\mathbf{1}_n{}^T]_{ji} - 2[\mathbf{X}\mathbf{X}^T]_{ij} \approx L_{ij}. \tag{3}$$

Since $[\mathbf{A}]_{ij} = [\mathbf{A}^T]_{ji}$, and $(\mathbf{Q}\mathbf{1}_n{}^T)^T = \mathbf{1}_n\mathbf{Q}^T$, the entire set of equations in matrix form is

$$\mathbf{Q}\mathbf{1}_n{}^T + \mathbf{1}_n\mathbf{Q}^T - 2\mathbf{X}\mathbf{X}^T \approx \mathbf{L}. \tag{4}$$

Note that $\mathbf{Q}$ is a function of $\mathbf{X}$. The goal is to find $\mathbf{X}$ for which the above equality approximately holds. This set of equalities may not be exactly satisfied if $\mathbf{L}$ cannot be embedded in $\mathbb{R}^d$. As it stands, (4) is hard to solve on account of the dependance of $\mathbf{Q}$ on $\mathbf{X}$. We massage it into a more convenient form by using a projection matrix

$$\boldsymbol{\gamma} = \mathbf{I}_n - \frac{1}{n}\mathbf{1}_n{\mathbf{1}_n}^T, \tag{5}$$

where $\mathbf{I}_n$ is the $n \times n$ identity matrix. Multiplying both sides of (4) by $\boldsymbol{\gamma}$ from the left and the right, we obtain

$$\boldsymbol{\gamma}\mathbf{Q}{\mathbf{1}_n}^T\boldsymbol{\gamma} + \boldsymbol{\gamma}\mathbf{1}_n\mathbf{Q}^T\boldsymbol{\gamma} - 2\boldsymbol{\gamma}\mathbf{X}\mathbf{X}^T\boldsymbol{\gamma} \approx \boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma}. \tag{6}$$

Since $\boldsymbol{\gamma}$ is a projection operator, (4) becomes

$$(\boldsymbol{\gamma}\mathbf{X})(\boldsymbol{\gamma}\mathbf{X})^T \approx -\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma}, \tag{7}$$

where we have used the fact that $\boldsymbol{\gamma} = \boldsymbol{\gamma}^T$. We may interpret this equation more easily by setting

$$\mathbf{Y} = \boldsymbol{\gamma}\mathbf{X} = (\mathbf{X} - \frac{1}{n}\mathbf{1}_n{\mathbf{1}_n}^T\mathbf{X}). \tag{8}$$

$\mathbf{Y}$ is an $n \times d$ matrix containing the coordinates in $\mathbf{X}$, each translated by the same vector $\frac{1}{n}{\mathbf{1}_n}^T\mathbf{X}$, i.e., each translated by the mean of the $\mathbf{X}$ coordinates. Thus, $\mathbf{Y}$ is the same set of coordinates as $\mathbf{X}$ in a different coordinate system; one in which $mean(\mathbf{Y}) = 0$. Since the distance matrix is invariant to rotations and translations, a solution for $\mathbf{Y}$ is just as acceptable as a solution for $\mathbf{X}$. Letting $\mathbf{M} = -\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma}$, we get

$$\mathbf{Y}\mathbf{Y}^T \approx \mathbf{M}. \tag{9}$$

Note that $\mathbf{Y}$ has rank d. If $\mathbf{D}$ were a true Euclidean distance matrix, then $\mathbf{M}$ would have rank at most d and we could exactly recover $\mathbf{Y}$, solving our problem. Since $\mathbf{D}$ may not be a true distance matrix , i.e., $\mathbf{D}$ may not be embeddable in $\mathbb{R}^d$, $\mathbf{M}$ will generally have rank greater than d. Naturally, we want to approximate $\mathbf{M}$ as closely as possible. The metric we choose 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 $\mathbf{M}$. Specifically, order the eigenvalues of $\mathbf{M}$ such that $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$ and let $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ be the associated eigenvectors. Then, the spectral decomposition of $\mathbf{M}$ yields $\mathbf{M} = \sum_{k=1}^{n} \lambda_k \mathbf{u}_k {\mathbf{u}_k}^T$, and the rank-$d$ approximation of $\mathbf{M}$ is $\mathbf{M}_d = \sum_{k=1}^{d} \lambda_k \mathbf{u}_k {\mathbf{u}_k}^T$.

Theorem 1 (see for example[8]) $\mathbf{M}_d$ *is the best rank-d approximation to* $\mathbf{M}$ *w.r.t. the spectral norm.*

The final centralized coordinates are then given by $\mathbf{Y} = [\sqrt{\lambda_1}\mathbf{u}_1, \ldots, \sqrt{\lambda_d}\mathbf{u}_d]$.

SDE(G)
1: Compute the distance matrix $\mathbf{D}$ using an APSP algorithm on G
2: Define matrix $\mathbf{L}$ such that $L_{ij} = D_{ij}^2$.
3: **return** $\mathbf{Y}$ = PowerIteration($-\frac{1}{2}\gamma\mathbf{L}\gamma, \epsilon$) % epsilon is a tolerance

Figure 2: The spectral graph drawing algorithm SDE.

3. The Graph Drawing Algorithm

The algorithm can now be succintly stated as an implementation of Theorem 1. Specifically, there are two stages:

(i) computing all-pairs shortest path lengths to obtain $\mathbf{L}$.

(ii) 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.

In order to implement (i), we run a BFS for each node. The complexity of this step is $O(|V||E|)$. For (ii), rather than using an exact algorithm which will be $O(|V|^3)$, we use a standard procedure typically referred to as the power iteration to compute the eigenvalues and eigenvectors of $\mathbf{M}$. The power iteration typically produces results as good as the exact algorithm, but much more efficiently. It starts with some random initial vectors and iteratively multiplies them with the relevant matrix modulo orthonormality constraints. The procedure stops when some termination condition is met, for example, when the change in direction of the eigenvector is negligible, i.e., the cosine of the dot product of the previous estimate and the newly computed estimate is above $1 - \epsilon$ for some small ϵ. We impose one additional condition for termination, which ensures that the ratio of the direction change between two consecutive iterations is above some value, $1 + \epsilon$ in our case. The convergence of the power iteration depends on the eigenvalues of the matrix; in practice it takes some constant number of iterations to compute the eigenvalues to some precision, since convergence is exponentially fast. The matrix multiplications we perform in the power iteration take $O(|V|^2)$ time, which makes the overall complexity of the power iteration $O(d|V|^2)$. Thus, the complexity of our algorithm is $O(|V||E| + d|V|^2)$, which is equal to $O(|V||E|)$ for $d = 2$. The space complexity is $O(|V|^2)$ since we need to store all the pair-wise distances. The algorithm is summarized in Figure 2 with a detailed implementation of the power iteration given in Figure 3.

4. Results

We have implemented our algorithm in C++, and Table 1 gives the running time results on a Pentium IV 3.2 GHz processor system. We present the results of running the algorithm on several graphs of varying size up to about 20,000 nodes. We show results for some small graphs in order to illustrate explicitly how the symmetries are preserved, in addition to several benchmark graphs. Finally in Figure 7 and 8, PowerIteration($\mathbf{M}$,ϵ)

$\mathit{current} \leftarrow \epsilon$; $\mathbf{y}_1 \leftarrow \mathbf{random}/\|\mathbf{random}\|$
repeat
  $\mathit{prev} \leftarrow \mathit{current}$
  $\mathbf{u}_1 \leftarrow \mathbf{y}_1$
  $\mathbf{y}_1 \leftarrow \mathbf{M}\mathbf{u}_1$
  $\lambda_1 \leftarrow \mathbf{u}_1 \cdot \mathbf{y}_1$ % compute the eigenvalue
  $\mathbf{y}_1 \leftarrow \mathbf{y}_1/\|\mathbf{y}_1\|$
  $\mathit{current} \leftarrow \mathbf{u}_1 \cdot \mathbf{y}_1$
until $\mathit{current} \geq 1-\epsilon$ or $|\mathit{current}/\mathit{prev}| \leq 1+\epsilon$
$\mathit{current} \leftarrow \epsilon$; $\mathbf{y}_2 \leftarrow \mathbf{random}/\|\mathbf{random}\|$
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 $\mathbf{u}_1$
  $\mathbf{y}_2 \leftarrow \mathbf{M}\mathbf{u}_2$
  $\lambda_2 \leftarrow \mathbf{u}_2 \cdot \mathbf{y}_2$ % compute the eigenvalue
  $\mathbf{y}_2 \leftarrow \mathbf{y}_2/\|\mathbf{y}_2\|$
  $\mathit{current} \leftarrow \mathbf{u}_2 \cdot \mathbf{y}_2$
until $\mathit{current} \geq 1-\epsilon$ or $|\mathit{current}/\mathit{prev}| \leq 1+\epsilon$
return $(\sqrt{\lambda_1}\mathbf{y}_1 \quad \sqrt{\lambda_2}\mathbf{y}_2)$

Figure 3: The power iteration method for finding eigenvectors and eigenvalues ($d = 2$).

we compare some drawings generated by our algorithm, together with the results of other spectral graph drawing algorithms HDE and ACE [11, 16]. Note that the results for HDE are produced by the standard application of the algorithm, which uses the first and the second principal components [11]. For the power iteration, we set the tolerance $\epsilon = 10^{-7}$.

Table 1 shows that SDE is reasonably fast for graphs up to 20,000 nodes. As can be seen from Figures 4, 5 and 6, it also produces aesthetically pleasing drawings of a wide range of graphs varying in size, node density and degree of symmetry. Figure 7 and 8 highlight the main advantages of SDE over other spectral graph drawing methods. Specifically, it preserves the symmetries in the graph and maintains uniform node density to a large extent, and hence produces a better representation of the overall graph structure. To be fair, we note that on some graphs (eg. 3elt), ACE and HDE give comparable drawings to SDE, much more quickly. However, they suffer from many problems illustrated in the figures. Typically, node densities are not preserved resulting in distortions of the general structure.

In the last column of Table 1, we also give the Frobenius norm of the embedding error $\boldsymbol{\epsilon} = \mathbf{D} - \mathbf{D}'$ where $D_{ij} = \sqrt{L'_{ij}}$ and $\mathbf{L}' = \mathbf{Q}\mathbf{1}_n{}^T + \mathbf{1}_n\mathbf{Q}^T - 2\mathbf{Y}\mathbf{Y}^T$, as a measure of how accurate the drawing is with respect to the graph theoretical distances. The Frobenius errors computed in Table 1 are defined as

Graph	$\lvert\mathbf{V}\rvert$	$\lvert\mathbf{E}\rvert$	APSP	PI	Total	$\lVert \boldsymbol{\epsilon} \rVert_F$	$\lVert \boldsymbol{\epsilon} \rVert_{F'}$
Jagmesh1	936	2664	0.1	0.1	0.2	2.26	0.154
Can1072	1072	5686	0.2	0.3	0.5	1.52	0.314
Grid 50x50	2500	4900	0.7	0.5	1.2	4.49	0.171
Torus 50x50	2500	5000	0.7	0.6	1.3	10.1	0.357
Nasa1824	1824	18692	1.0	0.8	1.8	1.45	0.250
Blckhole	2132	6370	0.6	1.5	2.1	5.90	0.334
Nasa2146	2146	35052	2.1	0.7	2.8	1.49	0.179
Lshp3466	3466	10215	1.8	2.0	3.8	2.48	0.111
4970	4970	7400	2.8	2.3	5.1	4.77	0.162
Grid 70x70	4900	9660	2.8	2.4	5.2	6.28	0.170
Airfoil1	4253	12289	3.4	3.5	6.9	5.81	0.265
3elt	4720	13722	4.7	3.8	8.5	10.2	0.382
Sierpinski08	9843	19683	16	9	25	14.4	0.170
Grid 100x100	10000	19800	18	12	30	8.96	0.170
Whitaker3	9800	28989	25	8	33	1.51	0.054
Crack	10240	30380	28	17	45	2.04	0.085
4elt2	11143	32818	34	15	49	9.16	0.252
Bcsstk33	8738	291583	77	15	92	1.67	0.252
4elt	15606	45878	86	47	133	11.5	0.308
Sphere	16386	49152	107	30	137	17.6	0.291
Vibrobox	12328	165250	125	40	165	2.99	0.562
Cti	16840	48232	91	78	169	7.40	0.328

Table 1: Running time of SDE for several graphs. (Most of these graphs can be downloaded from [1, 2].)

$$\lVert \boldsymbol{\epsilon} \rVert_F = \sqrt{\frac{1}{n^2}\sum_{i,j}(D_{ij} - D'_{ij})^2}, \qquad \lVert \boldsymbol{\epsilon} \rVert_{F'} = \sqrt{\frac{1}{n^2}\sum_{i\neq j}\left(1 - \frac{D'_{ij}}{D_{ij}}\right)^2} \tag{10}$$

The Frobenius errors may not be considered as an exact quantification of what embodies an aesthetic drawing. Furthermore, for certain graphs, no matter how good a graph drawing algorithm is, these errors may not be under some certain threshold as they depend on the embedability of the graph with respect to the graph theoretical distances, but one expects these errors to be as small as possible for any intuitively pleasing drawing.

5. Performance Analysis

The formal problem we are attempting to solve is a problem in minimum distortion finite metric embedding [17]. We approach this problem using a spectral decomposition of the matrix of squared distances. We now give some results that explain the intuition behind why and when our algorithm will work well in practice. The distance matrix $\mathbf{D}$ represents a finite metric space. Our approach is to use a spectral technique to estimate $\mathbf{L}$, where $L_{ij} = D_{ij}^2$. Suppose that the optimal embedding (which we define below) is given by the coordinates $\mathbf{z}_1, \ldots, \mathbf{z}_n$, which we collect in the matrix $\mathbf{Z}$ (analogous to $\mathbf{X}, \mathbf{Y}$). Let $\mathbf{D_Z}$ and $\mathbf{L_Z}$ be the distance matrix and the matrix of squared distances implied by $\mathbf{Z}$. We can then write

$$\mathbf{L} = \mathbf{L_Z} + \boldsymbol{\epsilon}. \tag{11}$$

We refer $\boldsymbol{\epsilon}$ as the metric embedding error. $\mathbf{Z}$ is optimal in that $\| \boldsymbol{\epsilon} \|_S$ is infimum over all possible $\mathbf{Z}$. $\mathbf{L}$ is *embeddable* if $\boldsymbol{\epsilon} = \mathbf{0}$.

Theorem 2 *If $\mathbf{L}$ is embeddable, then, up to an orthogonal transformation, our algorithm returns $\mathbf{Z} - \frac{1}{n}\mathbf{1}_n{\mathbf{1}_n}^T\mathbf{Z}$.*

Proof. Multiplying both sides of (11) by $\boldsymbol{\gamma}$ from the left and right, we obtain

$$\begin{aligned} \boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma} &= \boldsymbol{\gamma}\mathbf{L_Z}\boldsymbol{\gamma} \\ &= -2(\mathbf{Z} - \frac{1}{n}\mathbf{1}_n{\mathbf{1}_n}^T\mathbf{Z})(\mathbf{Z} - \frac{1}{n}\mathbf{1}_n{\mathbf{1}_n}^T\mathbf{Z})^T \\ &= -2\mathbf{A}\mathbf{A}^T, \end{aligned} \tag{12}$$

where $\mathbf{A} = \mathbf{Z} - \frac{1}{n}\mathbf{1}_n{\mathbf{1}_n}^T\mathbf{Z}$ is rank-d. Since our algorithm computes a rank-d approximation $\mathbf{A}_d$ of the matrix $-\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma}$, and the right hand side of (12) is rank-d, we exactly recover $-\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma}$, i.e., $\mathbf{A}_d{\mathbf{A}_d}^T = -\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma} = \mathbf{A}\mathbf{A}^T$. Since the singular values of a matrix $\mathbf{A}$ are the nonnegative square roots of the eigenvalues of $\mathbf{A}\mathbf{A}^T$ and the left singular vectors of that matrix are the eigenvectors of $\mathbf{A}\mathbf{A}^T$, we can write the singular value decompositions of $\mathbf{A_d}$ and $\mathbf{A}$ as $\mathbf{A_d} = \mathbf{U}\boldsymbol{\Sigma}{\mathbf{V}_1}^T$ and $\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}{\mathbf{V}_2}^T$. Multiplying the equation for $\mathbf{A}_d$ from left by $\mathbf{V}_1$, we have $\mathbf{A}_d\mathbf{V}_1 = \mathbf{U}\boldsymbol{\Sigma}$ since $\mathbf{V}_1$ is orthogonal. Substituting this expression in the equation for $\mathbf{A}$, we obtain $\mathbf{A} = \mathbf{A}_d\mathbf{V}_1{\mathbf{V}_2}^T$. Note that $\mathbf{V} = \mathbf{V}_1{\mathbf{V}_2}^T$ is also orthogonal since $\mathbf{V}\mathbf{V}^T = \mathbf{I}_d$ and therefore $\mathbf{A}_d$ differs from $\mathbf{A}$ by at most an orthogonal transformation. □

Since the distance matrix is invariant to orthogonal transformations, we obtain the following corollary, which is the basic intuition behind SDE:

Corollary 1 *When the distance matrix is embeddable, the coordinates recovered by SDE exactly reproduce the distance matrix.*

When the distance matrix is not exactly embeddable, but the embedding error $\boldsymbol{\epsilon}$ is small, SDE should approximately reproduce the distance matrix. Suppose that $\boldsymbol{\epsilon} \neq \mathbf{0}$, and let

$$\begin{aligned} \mathbf{M} &= -\frac{1}{2}\boldsymbol{\gamma}\mathbf{L_Z}\boldsymbol{\gamma} \\ &= -2(\mathbf{Z} - \frac{1}{n}\mathbf{1}_n{\mathbf{1}_n}^T\mathbf{Z})(\mathbf{Z} - \frac{1}{n}\mathbf{1}_n{\mathbf{1}_n}^T\mathbf{Z})^T. \end{aligned} \tag{13}$$

Let $\boldsymbol{\epsilon}_1 = -\frac{1}{2}\boldsymbol{\gamma}\boldsymbol{\epsilon}\boldsymbol{\gamma}$. Then, using (11), $-\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma} = \mathbf{M} + \boldsymbol{\epsilon}_1$. The next theorem states that the optimal rank-d approximation for $-\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma}$ is a also a good approximation to $\mathbf{M}$ (up to the embedding error $\boldsymbol{\epsilon}$).

Theorem 3 *Let $\mathbf{M}_d$ be the optimal rank-d approximation to $-\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma} = \mathbf{M} + \boldsymbol{\epsilon}_1$. Then, $\| \mathbf{M}_d - \mathbf{M} \|_S \leq \| \boldsymbol{\epsilon} \|_S$.*

Proof. By the triangle inequality, we have

$$\begin{aligned}\| \mathbf{M}_d - \mathbf{M} \|_S &= \| \mathbf{M}_d - \mathbf{M} - \boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_1 \|_S \\ &\le \| \mathbf{M}_d - \mathbf{M} - \boldsymbol{\epsilon}_1 \|_S + \| \boldsymbol{\epsilon}_1 \|_S.\end{aligned} \tag{14}$$

Since $\mathbf{M}_d$ is the best rank-d approximation to $\mathbf{M} + \boldsymbol{\epsilon}_1$ and $\mathbf{M}$ is itself rank-d,

$$\begin{aligned}\| \mathbf{M}_d - \mathbf{M} - \boldsymbol{\epsilon}_1 \|_S &\le \| \mathbf{M} - \mathbf{M} - \boldsymbol{\epsilon}_1 \|_S \\ &= \| \boldsymbol{\epsilon}_1 \|_S.\end{aligned} \tag{15}$$

Thus, $\| \mathbf{M}_d - \mathbf{M} \|_S \le 2\| \boldsymbol{\epsilon}_1 \|_S$. To conclude, note that since $\boldsymbol{\gamma}$ is a projection matrix, $\| \boldsymbol{\gamma} \|_S \le 1$, so by sub-multiplicativity, $\| \boldsymbol{\epsilon}_1 \|_S = \| -\frac{1}{2}\boldsymbol{\gamma}\boldsymbol{\epsilon}\boldsymbol{\gamma} \|_S \le \frac{1}{2}\| \boldsymbol{\gamma} \|_S^2 \| \boldsymbol{\epsilon} \|_S \le \frac{1}{2}\| \boldsymbol{\epsilon} \|_S$. □

Thus, if SDE returns the coordinates $\mathbf{Y}$, then $\| \boldsymbol{\gamma}\mathbf{Y}\mathbf{Y}^T\boldsymbol{\gamma} - \boldsymbol{\gamma}\mathbf{Z}\mathbf{Z}^T\boldsymbol{\gamma} \|_S$ is small provided that $\mathbf{L}$ is nearly embeddable (which depends on the perturbation $\boldsymbol{\epsilon}$). Unfortunately this alone does not guarantee that $\boldsymbol{\gamma}\mathbf{Y} \approx \boldsymbol{\gamma}\mathbf{Z}\mathbf{O}$ for some orthogonal $d \times d$ matrix $\mathbf{O}$, i.e., in general, the eigenvectors are not stable to perturbations. The next theorem relates the change in eigenvectors to the spectrum.

Theorem 4 (Corollary 7.2.6 in [8]) *Let* $\hat{\mathbf{e}}_1$ *and* $\hat{\mathbf{e}}_2$ *be the top two eigenvectors of* $\mathbf{M}+\boldsymbol{\epsilon}_1$*, and* $\mathbf{e}_1$ *and* $\mathbf{e}_2$ *be the top two eigenvectors of* $\mathbf{M}$*. Then there exists functions* f_1 *and* f_2 *of* $\mathbf{M}$ *and* $\boldsymbol{\epsilon}_1$ *such that* $\| \mathbf{e}_1 - \hat{\mathbf{e}}_1 \| \le f_1(\mathbf{M}, \boldsymbol{\epsilon}_1)$ *and* $\| \mathbf{e}_2 - \hat{\mathbf{e}}_2 \| \le f_1(\mathbf{M}, \boldsymbol{\epsilon}_1)$*. Furthermore, these functions depend on how well separated the eigenvalues of* $\mathbf{M}$ *are, the size of the error* $\boldsymbol{\epsilon}_1$*, and how well aligned the error is to* $\mathbf{M}$*.*

The functions f_1, f_2 in Theorem 4 have the expected dependence on $\mathbf{M}, \boldsymbol{\epsilon}_1$. In particular, the better separated the spectrum of $\mathbf{M}$ is and the smaller $\boldsymbol{\epsilon}_1$ is, the smaller the values of f_1, f_2 will be. An additional concern is how well aligned $\boldsymbol{\epsilon}_1$ is to $\mathbf{M}$. The precise formulation of this can be found in [8], but in a nutshell, the perturbations in the small singular value subspaces of $\mathbf{M}$ should be small (i.e. the relative perturbations should be small). According to Theorem 4, when the eigenvalues of $\mathbf{M}$ are well separated and $\boldsymbol{\epsilon}$ is small, then not only will $\mathbf{M}_d$ be close to $\mathbf{M}$, but so will the top two eigenvectors match closely. Thus, SDE will recover a close approximation to $\mathbf{ZO}$ for some orthogonal matrix $\mathbf{O}$, i.e., the distance matrix is nearly recovered. Note that the separation of the eigenvalues is a necessary condition for the power iteration to converge quickly. Thus, when the power iteration fails to converge quickly, it is already a sign that SDE may produce undesirable results. However, it also means that any rank-2 decomposition, including the optimal embedding must have a high relative embedding error. To see this, note that $\| \boldsymbol{\epsilon}_1 \|_F^2 \ge \sum_{i=3}^{n} \sigma_j^2(\mathbf{L})$, and since $\sigma_3^2(\mathbf{L}) \approx \sigma_{1,2}^2(\mathbf{L})$ (because the singular values are not well separated), the embedding error will be a constant fraction of $\| \mathbf{L} \|_F$.

6. Conclusion

We have presented a novel graph drawing algorithm SDE which is based on computing the spectral decomposition of the matrix of squared graph theoretical distances. Our algorithm has the advantages of spectral graph drawing techniques, for example exact (as opposed to iterative) computation of the coordinates, and efficiency, while at the same time producing drawings that are comparable in quality to slower force directed-methods.

The running time of our algorithm is dominated by an APSP computation, which is $O(|V||E|)$ using $O(|V|^2)$ space for storing pair-wise distances. SDE can readily be extended to weighted graphs with run time $O(|V||E|\log|V|)$, again dominated by the APSP algorithm. Our algorithm is not as efficient as other methods such as ACE, which runs in linear time. However, it is able to draw quite large graphs in a reasonable amount of time, and therefore can be extremely useful as an auxiliary drawing algorithm to obtain precise pictures of smaller regions contained within a huge graph. The general structure of our algorithm also allows for zooming into a graph, which corresponds to running SDE on a principle sub-matrix of the matrix of squared distances.

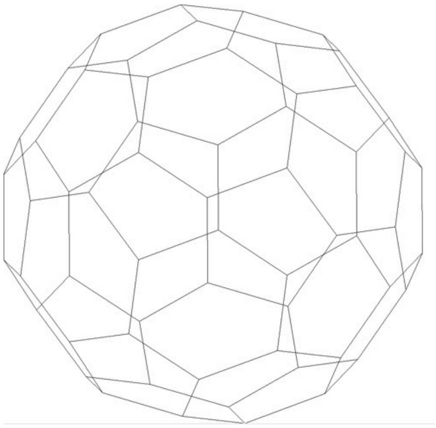
The main bottleneck of SDE is caused by the fact that it needs to compute the shortest path lengths between every pair of nodes. This also affects the power iteration, where large $(|V| \times |V|)$ matrices need to be processed. We will briefly discuss approaches to improve the efficiency of our algorithm without significant sacrifice in quality (which will be the topic of forthcoming work). Since any valid matrix of squared distances has rank 4, i.e. all the rows of the matrix can be expressed as a linear combination of 4 vectors, the BFS algorithm need not be run on all vertices, but rather on a small number $(O(1))$ of carefully chosen vertices. This results in a sparse matrix representation of $\mathbf{L}$, which will be accurate provided that the perturbation $\boldsymbol{\epsilon}$ is small, i.e., if $\mathbf{L}$ is embeddable. All the algorithms can then be run on the sparse approximation, which will result in a runtime of $O(mE)$, where $m = O(1)$ is the dimensionality of the sparse representation. Further increases in efficiency can be obtained by using a multi-scaling approach in the power iteration, which was introduced in [16]. This method relates the actual graph to a coarser level of the graph using an interpolation matrix. Finding the eigenvalues and eigenvectors of coarser level graphs makes the convergence faster at the finer levels.

References

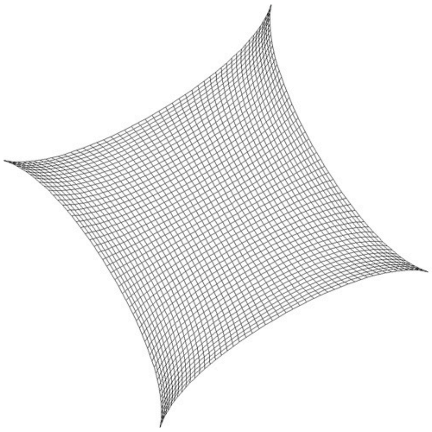
1. http://wwwcs.uni-paderborn.de/fachbereich/ag/monien/research/part/graphs.html.
2. http://www.gre.ac.uk/ c.walshaw/partition/.
3. P. Biswas and Y. Ye. Semidefinite programming for ad-hoc wireless localization. IPSN, Berkeley, CA, 2004.
4. R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, 1996.
5. P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
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. P. Gajer, M. T. Goodrich, and S. G. Kobourov. A fast multi-dimensional algorithm for drawing large graphs. In *Proc. 8th Int. Symp. Graph Drawing(GD'00)*, volume

1984 of *LNCS*, pages 211–221. Springer-Verlag, 2000.

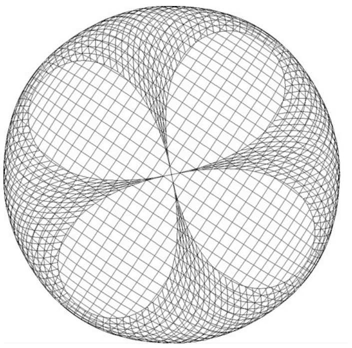
8. G. H. Golub and C. V. Loan. *Matrix Computations*. Johns Hopkins U. Press, 1996.
9. S. Hachul and M. Jünger. Drawing large graphs with a potential-field-based multi-level algorithm. In *Proc. 12th Int. Symp. Graph Drawing(GD'04)*, LNCS, 2004.
10. D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. In *Proc. 8th Int. Symp. Graph Drawing(GD'00)*, volume 1984 of *LNCS*, pages 183–196. Springer-Verlag, 2000.
11. D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. In *GD02*, LNCS. Springer-Verlag, 2002.
12. X. Ji and H. Zha. Sensor positioning in wireless ad-hoc sensor networks using multidimensional scaling. IEEE Infocom, March 7-11, 2004.
13. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.
14. M. Kaufmann and D. Wagner, editors. *Drawing Graphs: Methods and Models*. Number 2025 in LNCS. Springer-Verlag, 2001.
15. Y. Koren. On spectral graph drawing. In *COCOON 03*, volume 2697 of *LNCS*, pages 496–508. Springer-Verlag, 2003.
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. Matousek. Open problems on embeddings of finite metric spaces. *Discr. Comput. Geom.*, to appear.
18. A. Quigley and P. Eades. FADE: Graph drawing, clustering and visual abstraction. In *GD00*, volume 1984 of *LNCS*, pages 197–210. Springer-Verlag, 2000.
19. S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.
20. J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.
21. I. G. Tollis, G. D. Battista, P. Eades, and R. Tamassia. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.
22. W. T. Tutte. How to draw a graph. *Proc. London Mathematical Society*, 13:743–768, 1963.
23. C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *GD00*, volume 1984. Springer-Verlag, 2000.



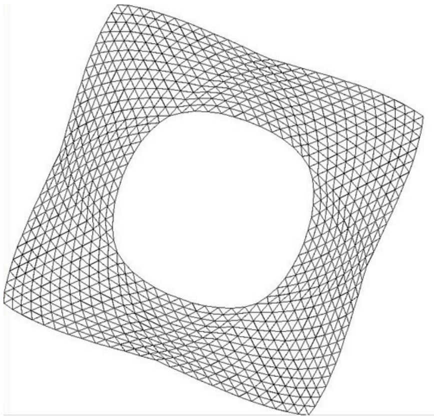
(a) Buckyball; $|V| = 60, |E| = 90$.



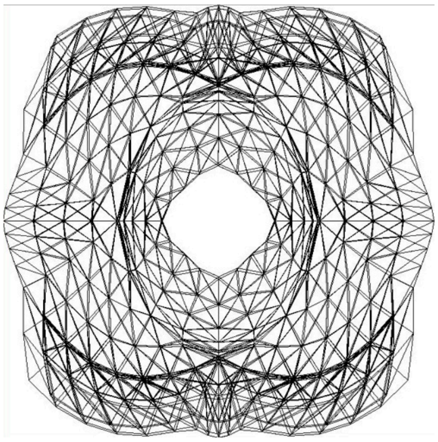
(b) 50×50 Grid; $|V| = 2500, |E| = 4900$.



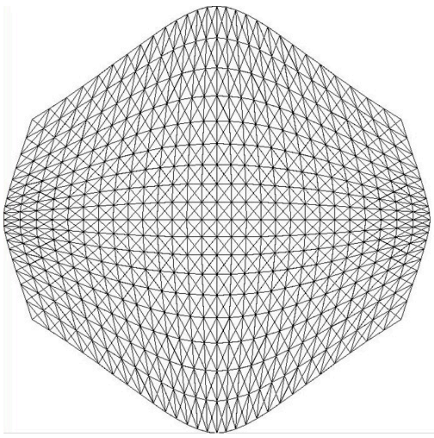
(c) 50×50 Bag; $|V| = 2497, |E| = 4900$.



(d) Jagmesh1; $|V| = 936, |E| = 2664$.

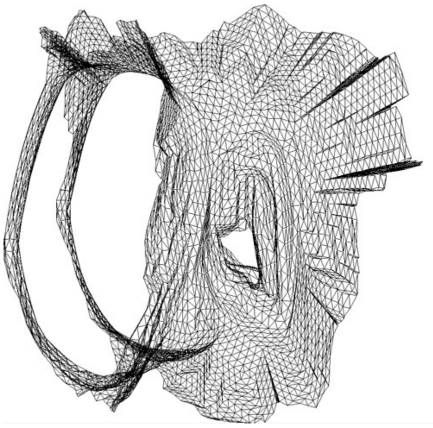


(e) Nasa1824; $|V| = 1824, |E| = 18692$.

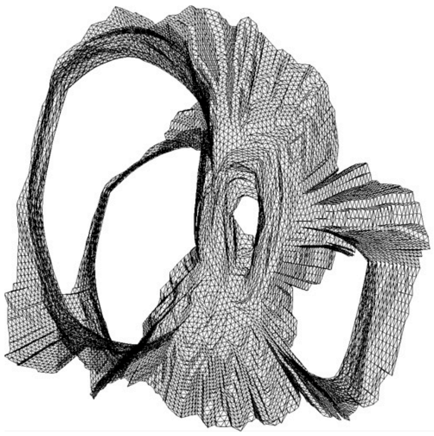


(f) Nasa2146; $|V| = 2146, |E| = 35052$.

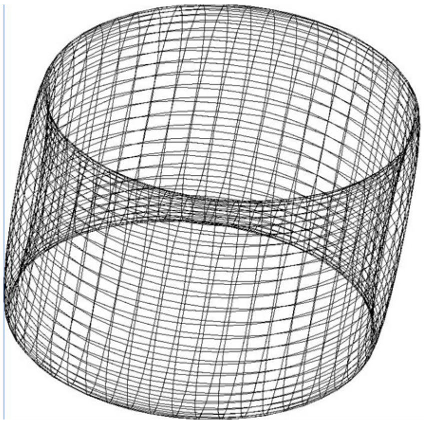
Figure 4: Layouts of some of the graphs tested **I**.



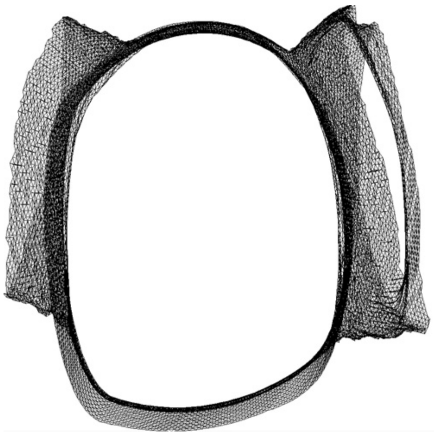
(a) 3elt; $|V| = 4720, |E| = 13722$.



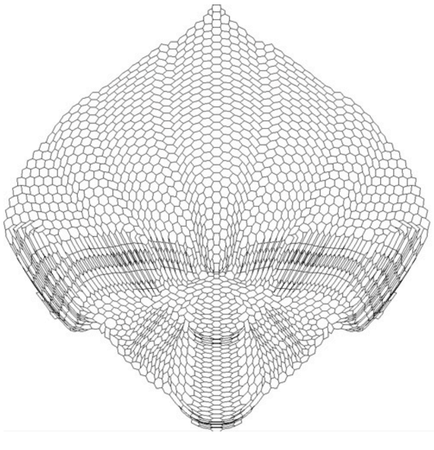
(b) 4elt; $|V| = 15606, |E| = 45878$.



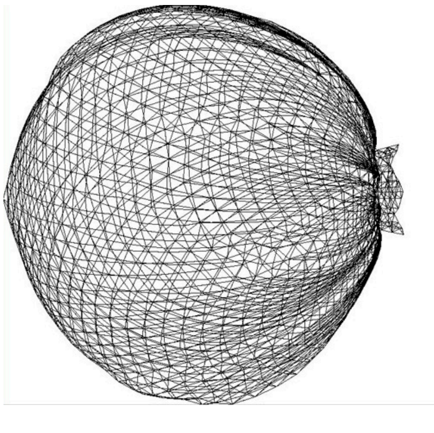
(c) 50×50 Torus; $|V| = 2500, |E| = 5000$.



(d) 4elt2; $|V| = 11143, |E| = 32818$.

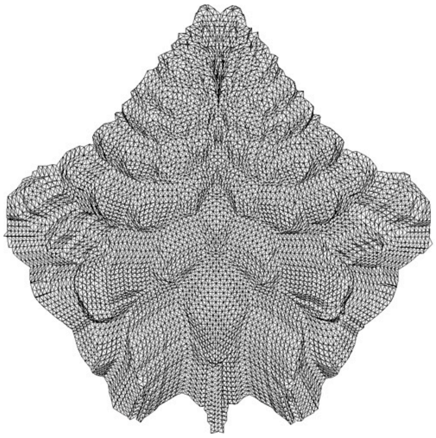


(e) Graph 4970; $|V| = 4970, |E| = 7400$.

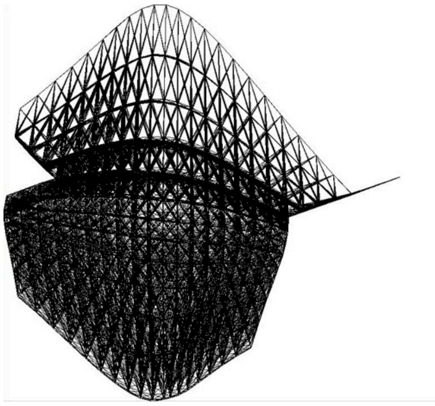


(f) Blckhole; $|V| = 2132, |E| = 6370$.

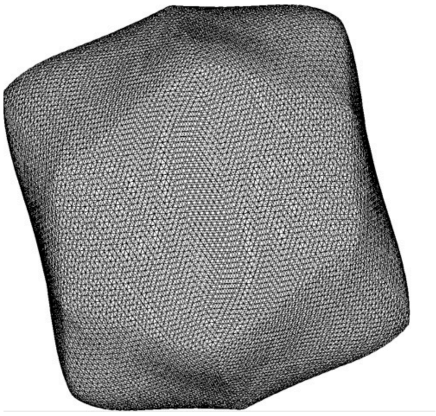
Figure 5: Layouts of some of the graphs tested **II**.



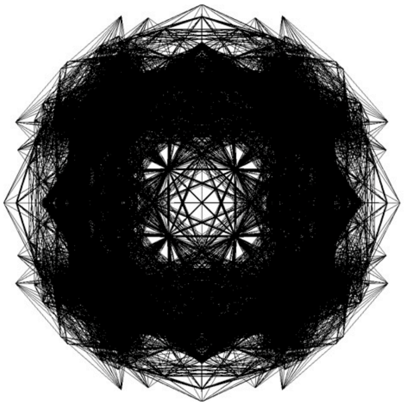
(a) Crack; $|V| = 10240, |E| = 30380$.



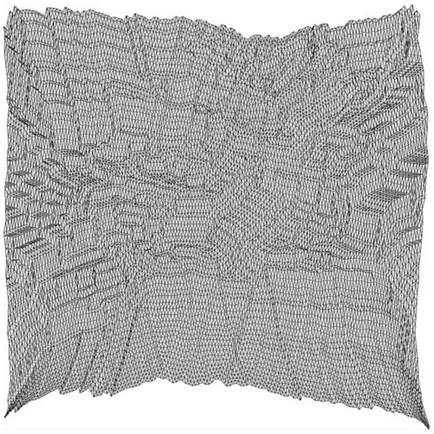
(b) Bcsstk33;$|V| = 8738|E| = 291583$.



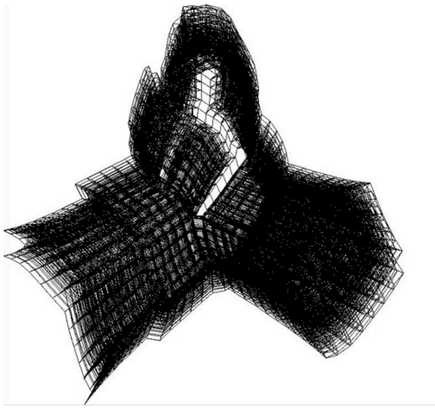
(c) Sphere; $|V| = 16386, |E| = 49152$.



(d) Vibrobox; $|V| = 12328, |E| = 165250$.



(e) Whitaker3; $|V| = 9800, |E| = 28989$.

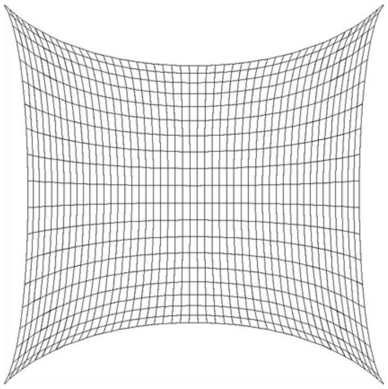


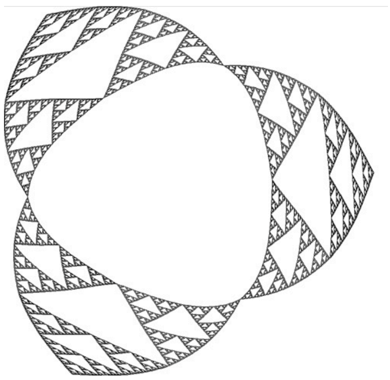
(f) Cti; $|V| = 16840, |E| = 48232$.

Figure 6: Layouts of some of the graphs tested **III**.

20 × 50 Grid; $|V| = 1000, |E| = 1930$. Sierpinski 8; $|V| = 9843, |E| = 19683$.

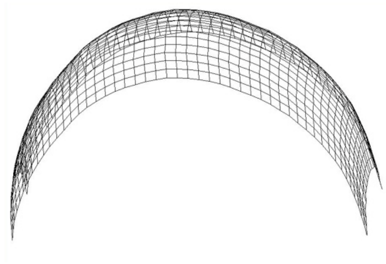
SDE

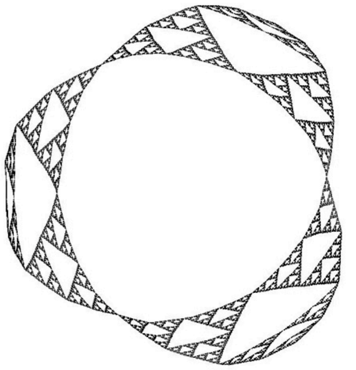




running time = 0.21 sec. running time = 24.72 sec.

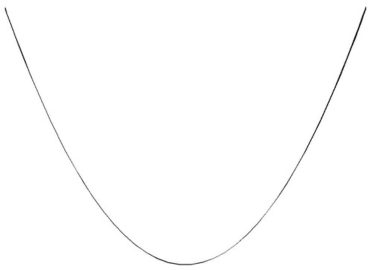
HDE

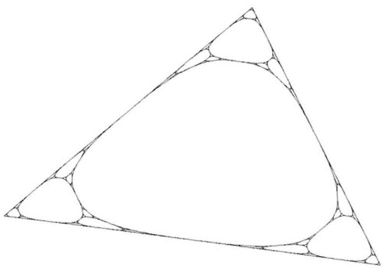




running time = 0.01 sec. running time = 0.11 sec.

ACE





running time = 0.03 sec. running time = 0.78 sec.

Figure 7: Comparison of SDE with HDE and ACE **I**.

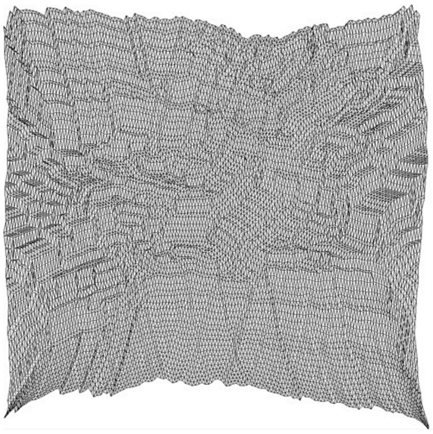
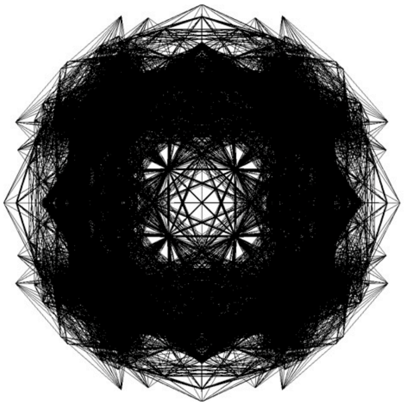
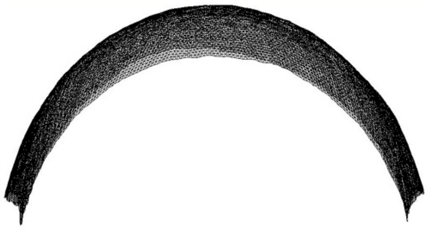
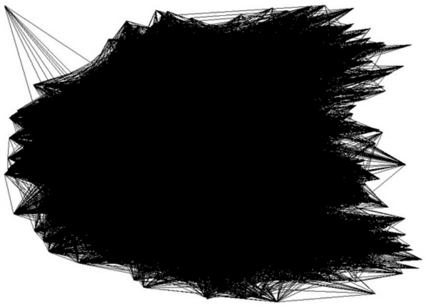
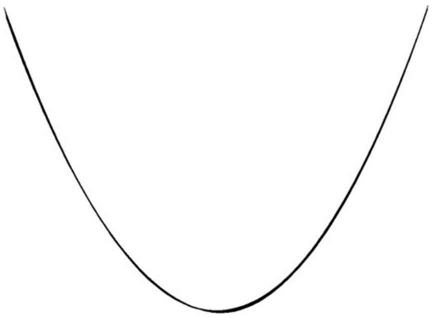
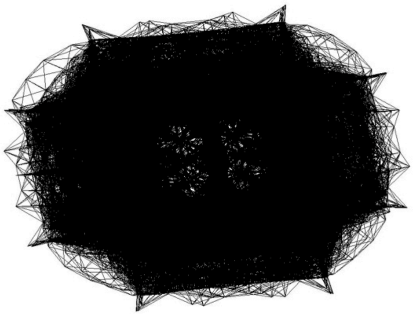
	Whitaker3; $|V| = 9800, |E| = 28989$.	Vibrobox; $|V| = 12328, |E| = 165250$.
SDE	 running time = 33.42 sec.	 running time = 164.81 sec.
HDE	 running time = 0.11 sec.	 running time = 0.24 sec.
ACE	 running time = 0.25 sec.	 running time = 0.92 sec.

Figure 8: Comparison of SDE with HDE and ACE **II**.