Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2004)
O. Deussen, C. Hansen, D.A. Keim, D. Saupe (Editors)

Graph Drawing by Subspace Optimization

Yehuda Koren

AT&T Labs – Research
Florham Park, NJ, USA
yehuda@reserach.att.com

Abstract
We show a novel approach for accelerating the computation of graph drawing algorithms. The method is based on the notion that we can find a subspace with a relatively low dimensionality that captures the "nice" layouts of the graph. This way each axis of the drawing is a linear combination of a few basis vectors, instead of being an arbitrary vector in $\mathbb{R}^n$ (n is the number of nodes). We describe ways of constructing these basis vectors and also algorithms for optimizing the graph drawing in the resulting subspace.

1. Introduction

A graph $G(V = \{1,\ldots,n\},E)$ is an abstract structure that is used to model a relation E over a set V of nodes. Visualizing graphs is a challenging task, requiring algorithms that faithfully represent the graph's structure and the relative similarities of the nodes. Consequently, many approaches to graph drawing have been developed [dBETT99, KW01]. We have focused on the problem of drawing undirected graphs with straight-line edges. The most popular approaches to this appear to be those that define a cost function, whose minimization determines the optimal drawing. The resulting algorithms are known as *force-directed* methods; see, e.g., [Ead84, FR91, KK89].

A particularly challenging problem is drawing large graphs containing 10^3–10^6 nodes, which has gained much interest recently because of the rapid growth of data collections. Most existing graph drawing algorithms do not scale up well and face substantial difficulties when applied to large graphs. Consequently, we suggest here a new approach to drawing large graphs. As in force-directed methods, we utilize cost functions to assess the quality of the drawing. However, instead of optimizing these functions in the full $\mathbb{R}^n$, we optimize them in a carefully designed, much smaller subspace that we expect to contain nice drawings of the graph. Our experiments with two appropriate energy models show an appreciable reduction of running time without a considerable loss of drawing quality.

2. Related Work

Multi-scale graph drawing In recent years the multi-scale strategy has been recognized as an effective way for improving the performance of graph-related optimization problems by overcoming the localized nature of various optimization heuristics. Probably, the most well-known application is for the graph partitioning problem; see, e.g., [HL95]. Multi-scale graph drawing [GGK00, HH01, HK00, KCH02, Wal00] is a rather rapid method suitable for drawing large graphs. The main idea is to approximate the final drawing using a drawing of much smaller related graph (e.g., of the half size), called *a coarse graph*. After obtaining the approximated layout, it is refined locally to become a truly nice layout.

The method that we introduce in this paper shares some conceptual relations with the multi-scale approach. Multi-scale algorithms accelerate the computation by considering layouts of smaller, simplified graphs; our approach also works with simplified layouts but in a very different way – we use special vector spaces in which the layout must lie. Unlike the multi-scale algorithms that have to refine the simplified layout in order to get a nice one, the subspace-restricted layouts are already quite nice and can be used without further refinement.

Eigen-projection — drawing graphs with eigenvectors The spectral approach to graph visualization, which is rooted in the 1970's work of Hall [Hal70], computes the layout of a graph using certain (generalized) eigenvectors of the related Laplacian matrix. Consequently, we call this method the *eigen-projection*. Two distinct advantages of eigen-projection make it very attractive. First, it provides us with an exact solution to the layout problem, whereas almost all other formulations result in an NP-hard problem, which can only be approximated. The second advantage is computation speed: spectral drawings can be computed extremely fast [KCH02].

In this work eigen-projection can be used both for constructing the subspace in which we want to draw the graph (Subsection 3.2) and for finding the graph layout within a given subspace (Subsection 4.1). In fact, in the context of visualizing multidimensional data, a few recent works already deal with optimizing eigen-projection-related energies within a subspace; see [Bra03, KC03].

PHDE — graph drawing by high-dimensional embedding An approach to drawing large graphs, which is especially relevant to this paper, is described in [HK02]. This method involves a two-steps process: it first constructs a high-dimensional layout (HDE) of the graph (e.g., in 50 dimensions) and then projects the layout into a low dimension using *principal components analysis* (PCA). Henceforth, we refer to this PCA-based approach as PHDE (short for PCA of High-dimensional Embedding). In the following sections we show how to replace the PCA projections with more sophisticated methods that take into account the structure of the graph and typically yield better drawings.

Visualization challenges One of the main difficulties when drawing large graphs is the limited display area. Obviously, graphs with many thousands of nodes cannot be conveniently displayed or printed, and new display tools would be required. A promising direction is to display only a portion of a graph at any given time, using various smooth navigation tools. Another interesting approach is to display abstractions of the original graph in order to ease the visual burden of grasping the full complex graph. A comprehensive survey of various visualization approaches for the display of large graphs can be found in, e.g., [HMM00]. In this paper, we tackle the problem from an algorithmic point of view, which is complementary to the display point of view.

3. Adequate Subspaces for Graph Drawing

Given a graph $G(V=\{1,\dots,n\},E)$, we can define its 2-D layout using two vectors $x,y \in \mathbb{R}^n$, such that the coordinates of node i are (x_i,y_i).[†] Consequently, a traditional force directed algorithm can be formulated as:

> "Find vectors $x,y \in \mathbb{R}^n$ that minimize a certain cost-function".

We want to alter this formulation and suggest the following:

> "Find vectors $x,y \in \mathbb{S}$ that minimize a certain cost-function".

Here, $\mathbb{S} \subseteq \mathbb{R}^n$ represents some subspace (vector space) in which we want to optimize the layout. Our usual way of defining such a subspace is by defining its basis vectors. This way, the subspace spanned by the basis vectors $\{v^1,\dots,v^m\}$ is

$$\mathrm{span}(v^1,\dots,v^m) \overset{def}{=} \{\alpha_1 v^1 + \cdots + \alpha_m v^m \mid \alpha_1,\dots,\alpha_m \in \mathbb{R}\}.$$

Clearly, constraining the drawing to lie within a subspace might result in arbitrarily bad layouts. However, as we will show, we can find some subspaces that contain reasonably nice layouts, so we are not loosing much by working only within such subspaces. In this section we describe two such subspaces that meet our demands. In the next section we describe how to optimize certain cost functions within these subspaces.

Technical notes We assume, without loss of generality, that the basis vectors are orthonormal, what will be proved useful later. For any set of vectors $v^1,v^2,\dots,v^m$, this characteristic can be achieved without altering $\mathrm{span}(v^1,v^2,\dots,v^m)$. Also, note that adding a multiple of $1_n \overset{def}{=} (1,1,\dots,1) \in \mathbb{R}^n$ to the coordinates is equivalent to translation that has no visual effect. Hence, it will be convenient for us that all vectors are orthogonal to 1_n (that is *centered*) eliminating the redundant translation degree-of-freedom. We achieve all these requirements by a variant of the Gram-Schmidt orthonormalization procedure shown in Fig. 1. Note that vectors that are (almost) linearly dependent will get the value 0 and therefore should be removed. The entire orthonormalization process takes $O(m^2n)$ time.

```
Orthonormalize ($\{u^1,\dots,u^m\}$)
% This function orthonormalizes a set
% of vectors. Also, it orthogonalizes
% the vectors against $1_n$

  const $u^0 \leftarrow \frac{1_n}{\|1_n\|}$, $\varepsilon \leftarrow 0.001$
  for $i = 1$ to $m$ do
    for $j = 0$ to $i-1$ do
      $u^i \leftarrow u^i - \left(\left(u^i\right)^T u^j\right) u^j$
    end for
    if $\|u^i\| < \varepsilon$ then
      % a linearly dependent vector
      $u^i \leftarrow 0$
    else
      $u^i \leftarrow \frac{u^i}{\|u^i\|}$
    end if
  end for
```

Figure 1: *Gram-Schmidt orthonormalization*

[†] The methods discussed here can be easily extended to 3-D, but for presentation simplicity we deal with 2-D layouts that are much more common.

3.1. High-dimensional embedding

An appropriate subspace is HDE (high-dimensional embedding) that has already been used by [HK02]. In order to construct an m-dimensional subspace spanned by $\{v^1, v^2, \ldots, v^m\}$, we choose m *pivot* nodes $\{p_1, p_2, \ldots, p_m\}$ that are uniformly distributed on the graph and link each of the m basis vectors with a unique node. The vector v^i, which is associated with pivot node p_i, represents the graph from the "viewpoint" of p_i. This is done by assigning the j-th component of v^i to the graph-theoretic distance between nodes p_i and j. Henceforth, we denote this graph-theoretical distance by $d_{p_i j}$, so in symbols $v^i_j = d_{p_i j}$.

The resulting algorithm for constructing the high-dimensional embedding is given in Fig. 2. The graph-theoretical distances are computed using breadth-first-search (BFS). The pivots $p_1, p_2, \ldots, p_m$ are chosen as follows. The first member, p_1, is chosen at random. For $j = 2, \ldots, m$, node p_j is a node that maximizes the shortest distance from $\{p_1, p_2, \ldots, p_{j-1}\}$. The time complexity of this algorithm is $O(m \cdot |E|)$, since we perform BFS in each of the m iterations.

```
HDE $(G(V = \{1, \ldots, n\}, E), m)$
% This function finds an $m$-dimensional
% high-dimensional embedding of $G$

  Choose node $p_1$ randomly from $V$
  $d[1, \ldots, n] \leftarrow \infty$
  for $i = 1$ to $m$ do
      % Compute $v^i$ using BFS
      $d_{p_i *} \leftarrow \mathrm{BFS}(G(V,E), p_i)$
      for every $j \in V$
          $v^i_j \leftarrow d_{p_i j}$
          $d[j] \leftarrow \min\{d[j], v^i_j\}$
      end for
      % Choose next pivot
      $p_{i+1} \leftarrow \arg\max_{\{j \in V\}} \{d[j]\}$
  end for
  return $v^1, \ldots, v^m$
```

Figure 2: *Constructing an m-dimensional HDE*

The PHDE method [HK02] uses PCA projections of the high-dimensional embedding to yield nice layouts. Projections are just a type of linear combinations, so restricting the layout to lie inside span$(u_1, u_2, \ldots, u_m)$ is plausible. In practice, we have found that choosing $m \sim 50$ serves very well for producing a nice layout. It is important to note two fundamental differences between the current approach and PHDE:

1. In PHDE we looked for *projections* of the high-dimensional embedding, whereas here we are looking for the more general case of all *linear combinations* of the high-dimensional embedding vectors.
2. In PHDE all the knowledge about the graph is encapsulated in the part of the algorithm that generates the high-dimensional embedding, whereas in the projection part of the algorithm (i.e., PCA) nothing is known about the graph's structure. However, here, as will be shown later, we utilize the graph's structure also in an algorithm that computes a nice layout within the subspace.

3.2. Low eigenspace of the Laplacian

The Laplacian, L, is an $n \times n$ symmetric positive-semidefinite matrix associated with the graph, defined as:

$$L_{ij} = \begin{cases} -1 & \langle i,j \rangle \in E \\ \deg_i & i = j \\ 0 & \text{otherwise} \end{cases} \qquad i,j = 1, \ldots, n,$$

where $\deg_i = |\{j \mid \langle i,j \rangle \in E\}|$. Throughout this paper we have assumed that G is connected and contains no self loops or parallel edges.

It is well-known that the low eigenvectors of the Laplacian are useful for obtaining nice graph layouts [Hal70]. In the graph drawing field only the 2–4 low eigenvectors of the Laplacian were used so far [BW02, KCH02, Kor02]; however, in [KG00] many more low eigenvectors of the Laplacian were used in compressing mesh geometry (in the context of computer graphics). Consequently, [KG00] implied that combinations of low Laplacian eigenvectors can yield very good layouts.

Let us denote the $m+1$ lowest eigenvectors of L by $u^1, u^2, \ldots, u^{m+1}$. It is known that u^1 is proportional to 1_n and that all other eigenvectors are orthogonal to 1_n. Recall that we remove the superfluous translation degree-of-freedom by excluding u^1 from our subspace. Therefore, an adequate m-dimensional subspace that possesses important features of the graph is span$(u^2, \ldots, u^{m+1})$.

Technical notes Apparently, taking a subspace spanned by $u^2, \ldots, u^{m+1}$ involves a very expensive computation of m eigenvectors. Fortunately, we have found a way to get around this computational problem. Observe that it is not necessary to accurately distinguish between the eigenvectors $u^2, \ldots, u^{m+1}$, since we are not interested in each individual eigenvector, but rather in the space spanned by all of them together (that is, an *eigenspace*). It is necessary to distinguish between vectors that lie inside span$(u^2 \ldots, u^{m+1})$ and the rest of the vectors. Algorithms that compute eigenvectors have a hard time in distilling a particular eigenvector from adjacent eigenvectors (those with similar eigenvalues). However, here, for most vectors (especially the more important, low ones), it is not necessary to distinguish between adjacent eigenvectors, since we need all of them. Consequently, we use an algorithm for computing a basis for the eigenspace span$(u^2 \ldots, u^{m+1})$, where each vector in the basis is not necessarily an eigenvector. The algorithm is given in Fig. 3. Note that since we are interested in the low eigenspace, we invert the order of the eigenvectors by using the matrix $g \cdot I - \mathcal{X}^T L \mathcal{X}$. The scalar g is the Gershgorin bound [GvL96], which is a theoretical upper bound for (the absolute value of) the largest eigenvalue of a matrix.

We also utilize the fact that the lower eigenvectors (those around u^2) are more important for us than the higher ones (those around u^{m+1}), since the lower eigenvectors convey the graph better in terms of energy minimization [Hal70, Kor02]. In this manner, it is not crucial if, for example, we replace u^{m+1} with u^{m+2} in the basis. Hence, we halt the algorithm prematurely, after about 100 iterations. At this point, the low eigenvectors are already contained within the vector space, whereas the expression of higher eigenvectors might be worse.

In terms of running time, each iteration takes time $O(m^2 n + m|E|)$ for performing orthonormalization and m matrix-vector multiplications. Since the number of iterations is constant, this is also the overall time complexity. In practice, construction of the high-dimensional embedding subspace is faster.

```
SubspaceIteration ($L$, $\{u^2, \dots, u^{m+1}\}$)
% This function computes a basis for
% the low eigenspace of the Laplacian $L$

  const #iterations $\leftarrow$ 100
  % Compute Gershgorin bound:
  $g \leftarrow \max_i \left(L_{ii} + \sum_{j \neq i} |L_{ij}|\right)$

  $\{u^2, \dots, u^{m+1}\} \leftarrow$ random
  Orthonormalize($\{u^2, \dots, u^{m+1}\}$)
  for $i = 1$ to #iterations do
    for $j = 2$ to $m+1$ do
      $u^j \leftarrow (g \cdot I - L) u^j$
    end for
    if $i$ mod 3 = 0 then
      % orthogonalize each 3 iterations
      Orthonormalize($\{u^2, \dots, u^{m+1}\}$)
    end if
  end for
```

Figure 3: *The subspace iteration algorithm*

4. Optimization within Subspaces

In this section we show how to optimize two cost functions within a subspace. For convenience, let us assume that the subspace is spanned by the columns of an $n \times m$ matrix $\mathcal{X}$. Hence, the subspace is just the range of $\mathcal{X}$ denoted by $\mathcal{R}(\mathcal{X})$. Since the basis vectors were assumed to be orthonormal we obtain $\mathcal{X}^T \mathcal{X} = I$. Such a matrix representation is very convenient, since it allows us to describe the vectors in the subspace as the matrix-vector product $\mathcal{X} v$, where $v \in \mathbb{R}^m$.

4.1. Eigen-projection in a subspace

Here, we follow the eigen-projection and define the nice layout as the minimizer of:

$$\min_{x \in \mathbb{R}^n} \frac{\sum_{\langle i,j \rangle \in E} (x_i - x_j)^2}{\sum_{i<j} (x_i - x_j)^2} . \qquad (1)$$

The energy to be minimized strives to make edge lengths short (to minimize the numerator) while scattering the nodes in the drawing area preventing an overcrowding of the nodes (to maximize the denominator). This follows a common strategy to graph drawing stating that adjacent nodes should be drawn closely, while, generally, nodes should not be drawn too close to each other; see, e.g., [Ead84, FR91].

Eliminating the translation degree-of-freedom, we are interested only with centered coordinates, i.e., $x^T 1_n = 0$. In this case it can be shown that $\sum_{i<j} (x_i - x_j)^2$ is proportional to $x^T x$, making problem (1) equivalent to:

$$\min_{1_n \perp x \in \mathbb{R}^n} \frac{x^T L x}{x^T x} , \qquad (2)$$

where L is the Laplacian matrix defined in Subsection. 3.2. The optimizer is just the eigenvector of L with the lowest positive eigenvalue.

However, in our case, we want to optimize x within a subspace so problem (1) becomes:

$$\min_{x \in \mathcal{R}(\mathcal{X})} \frac{\sum_{\langle i,j \rangle \in E} (x_i - x_j)^2}{\sum_{i<j} (x_i - x_j)^2} . \qquad (3)$$

Or, equivalently:

$$\min_{x \in \mathcal{R}(\mathcal{X})} \frac{x^T L x}{x^T x} . \qquad (4)$$

In this case, we can replace x with $\mathcal{X} v$. Hence, (4) becomes $\min_{v \in \mathbb{R}^m} \frac{(\mathcal{X} v)^T L (\mathcal{X} v)}{(\mathcal{X} v)^T (\mathcal{X} v)}$, or equivalently:

$$\min_{v \in \mathbb{R}^m} \frac{v^T \mathcal{X}^T L \mathcal{X} v}{v^T v} . \qquad (5)$$

The denominator could be simplified because $\mathcal{X}^T \mathcal{X} = I$. Note, that here we do not impose orthogonality to 1_n, as it is already achieved by the fact that $1_n \notin \mathcal{R}(\mathcal{X})$.

Since the columns of $\mathcal{X}$ are linearly independent and orthogonal to 1_n, the matrix $\mathcal{X}^T L \mathcal{X}$ is positive-definite. Consequently, it is known that the minimizer of (5), which is the *Rayleigh quotient*, is the eigenvector of $\mathcal{X}^T L \mathcal{X}$ with the lowest eigenvalue. Another uncorrelated axis can be obtained by the second lowest eigenvector, and so on.

To summarize, let us be restricted to a subspace spanned by the columns of the orthogonal matrix $\mathcal{X}$. The drawing can be obtained by first computing the two lowest eigenvectors of $\mathcal{X}^T L \mathcal{X}$, denoted by v and u, and then taking the coordinates to be $\mathcal{X} v$ and $\mathcal{X} u$.

It is interesting to compare this approach to the PCA projection used in PHDE [HK02]. We can reshape minimization problem (1) as the *maximization* of the ratio

$$\frac{\sum_{i<j}(x_i - x_j)^2}{\sum_{\langle i,j\rangle \in E}(x_i - x_j)^2} . \qquad (6)$$

However, we have shown in [KC03] that PCA maximizes $\sum_{i<j}(x_i - x_j)^2$, which was just the numerator in (6). Therefore, unlike PCA, which strives to maximize the scatter of the nodes, without considering the structure of the graph, here we take into account the graph, and strive to keep edge lengths short.

Optimizing (1) in the subspace spanned by the low eigenspace of the Laplacian will yield the lowest positive eigenvectors of the Laplacian that lie within this subspace. This is not an interesting result, since this is exactly the eigen-projection layout obtained by optimizing (1) in the full $\mathbb{R}^n$.

Nonetheless, we obtain very interesting outcomes by optimizing (1) in the high-dimensional embedding subspace. This is a very quick way to approximate the eigen-projection, which replaces solving an $n \times n$ eigen-equation with an $m \times m$ eigen-equation. Regarding drawing quality, the fact that all coordinates in the high-dimensional embedding are integral (up to translation), helps in avoiding very dense regions in the drawing, which are common in eigen-projection results. In a sense, optimizing (1) in the high-dimensional embedding subspace results in an integration of PHDE and eigen-projection. It merges the ability of the eigen-projection to find the global layout of the graph, with the ability of PHDE to show delicate details that are often hidden in eigen-projection layouts.

Figure 4 demonstrates optimization of (1) in the high-dimensional embedding subspace, by comparing side-by-side three methods: (1) eigen-projection, (2) PHDE and (3) eigen-projection in the HDE subspace. We provide there three layouts of the Bfw782a graph [MM]. Clearly, the new method agrees with the eigen-projection regarding the global structure of the layout, but provides finer details like PHDE. We also show the results of the three methods for the 4elt graph [Wal]. Here, PHDE fails to show the global structure of the graph optimally, since its top boundary is folded; this is solved by using our new method. Our last example here is the larger graph Finan512 [Wal], (|V|=74,752, |E|=261,120). As usual, the eigen-projection shows its overall circular structure well, but fails to exhibit the delicate details. PHDE provides some hints concerning the micro structure, but a better and more symmetric layout is obtained by the method we have described here.

Running time Computation of the product $\mathcal{X}^T L\mathcal{X}$ is done in two steps: first, we compute $L\mathcal{X}$ in time $O(m|E|)$ utilizing the sparsity of L, and then we compute $\mathcal{X}^T(L\mathcal{X})$ in time $O(m^2 n)$. Note that $\mathcal{X}^T L\mathcal{X}$ is an $m \times m$ matrix, where typically $m \sim 50$, so the eigenvectors' calculation takes negligible time (about a millisecond). We recommend that a very accurate calculation be performed; this improves the layout quality with an insignificant affect on running time. In practice, we invert the order of the eigenvectors by using the matrix $B = \mu \cdot I - \mathcal{X}^T L\mathcal{X}$, and compute the highest eigenvectors of B using the power-iteration [GvL96]. The scalar μ is the highest eigenvalue of $\mathcal{X}^T L\mathcal{X}$ that can be computed directly by the power-iteration, or alternatively, one can set μ to the Gershgorin bound [GvL96].

A distinct advantage of optimization within the HDE subspace is the substantial reduction of running time thanks to replacing the $n \times n$ eigen-equation with an $m \times m$ eigen-equation. We cannot provide a recipe for the value of m, but as hinted before, in all our experiments $m = 50$ served us well, regardless of the graph's size. Note that unlike all iterative eigen-solvers (including the rapid algebraic-multigrid implementation in [KCH02]) for which the number of iterations depends on the structure of the matrix, the running time of our algorithm depends only on the graph's size and is $O(m^2 n + m|E|)$. Moreover, the dimensionality of the drawing has virtually no effect on the running time, whereas for (unconstrained) eigen-projection running time is linear in the dimensionality of the layout (e.g., time for drawing a graph in 3-D will grow by $\sim$50% relative to drawing it in 2-D).

Table 1 provides the actual running time of the various components of the subspace-constrained algorithm, as measured on a Pentium IV 2GHz PC. In addition to the total running time, we also provide the time needed for computing and orthogonalizing the HDE subspace (in the HDE-titled column), and the time needed for calculating the matrix $\mathcal{X}^T L\mathcal{X}$ (in the last column).

4.2. Stress minimization in a subspace

Graph drawing algorithms based on minimizing the so-called stress energy strive to place nodes in accordance with target distances. Such algorithms were first introduced to the graph drawing field by Kamada and Kawai [KK89]. Given a layout x, the concrete form of the energy is:

$$E(x) \stackrel{\text{def}}{=} \sum_{\{i,j\} \in \mathcal{S}} k_{ij} \left(|x_i - x_j| - d_{ij}\right)^2 . \qquad (7)$$

Here, the target distance, d_{ij}, is typically the graph-theoretical distance between nodes i and j. The normalization constant k_{ij} equals $d_{ij}^{-\alpha}$, where $0 \leqslant \alpha \leqslant 2$; we worked with $\alpha = 2$. The set $\mathcal{S} \subseteq \{\{i,j\} \mid i,j \in V\}$ contains those node pairs whose respective pairwise distances should be preserved. The most obvious choice is to take $\mathcal{S}$ as the set of *all node pairs*. The resulting layouts are usually nice; however, the space complexity would be quadratic. Shortly, we will show a much more efficient choice for $\mathcal{S}$.

Standard stress minimization algorithms are based on node-by-node local optimization methods, where in each step a single node is relocated in a way that decreases the

graph	|V|	|E|	running time (sec.)		
			total	HDE	$\mathcal{X}^T L\mathcal{X}$
516 [Wal]	516	729	0.02	0.00	0.00
Bfw782a [MM]	782	3,394	0.06	0.02	0.00
Fidap006 [MM]	1651	23,914	0.06	0.03	0.02
4970 [Wal]	4970	7400	0.77	0.09	0.64
3elt [Wal]	4720	13,722	0.77	0.09	0.64
Crack [Wal]	10,240	30,380	1.80	0.25	1.45
4elt2 [Wal]	11,143	32,818	1.84	0.28	1.52
4elt [Wal]	15,606	45,878	2.59	0.44	2.13
Sphere [Wal]	16,386	49,152	2.91	0.55	2.33
Fidap011 [MM]	16,614	537,374	3.28	0.73	2.52
Finan512 [Wal]	74,752	261,120	8.17	2.83	5.30
Sierpinski (depth 10)	88,575	177,147	13.89	3.19	10.56
grid 317 × 317	100,489	200,344	7.59	3.28	4.24
Ocean [Wal]	143,437	409,593	25.73	8.00	17.50

Table 1: *Running time (in seconds) of the various components of eigen-projection within HDE subspace*

stress energy. However, since the basic entities of these methods are nodes and not axes, they seem to be inappropriate for subspace-restricted optimization. An alternative approach that suits subspace-restricted optimization is the novel algorithm for axis-by-axis minimization of the stress energy [KH03]. The algorithm iteratively solves problems of the form:

$$\min_{x\in\mathbb{R}^n} x^T \mathcal{L}x - 2x^T b^{\tilde{x}}. \qquad (8)$$

Here, the $n\times n$ matrix $\mathcal{L}$ is an appropriate Laplacian, defined as:

$$\mathcal{L}_{ij} = \begin{cases} -k_{ij} & \{i,j\}\in\mathcal{S} \\ \sum_{j:\{i,j\}\in\mathcal{S}} k_{ij} & i=j \\ 0 & \text{otherwise} \end{cases} \qquad i,j=1,\dots,n.$$

Let the vector $\tilde{x}\in\mathbb{R}^n$ be another layout, usually the one obtained by a previous iteration. The vector $b^{\tilde{x}}\in\mathbb{R}^n$ is defined as:

$$b_i^{\tilde{x}} = \sum_{\substack{j:\\ \{i,j\}\in\mathcal{S},\tilde{x}_j\leqslant\tilde{x}_i}} k_{ij}d_{ij} - \sum_{\substack{j:\\ \{i,j\}\in\mathcal{S},\tilde{x}_j>\tilde{x}_i}} k_{ij}d_{ij} \qquad i=1,\dots,n. \qquad (9)$$

The solution of (8) is obtained by solving the $n\times n$ system of equations, $\mathcal{L}x = b^{\tilde{x}}$. It is shown in [KH03] that $E(x) < E(\tilde{x})$, unless $x=\tilde{x}$.

By Limiting x to lie within the subspace $\mathcal{R}(\mathcal{X})$, we can replace it with $\mathcal{X}v$, and now we can rewrite (8) as:

$$\min_{v\in\mathbb{R}^m} v^T\mathcal{X}^T\mathcal{L}\mathcal{X}x - 2x^T\mathcal{X}^T b^{\tilde{x}}. \qquad (10)$$

The unique minimizer is obtained by solving the tiny $m\times m$ system: $\left(\mathcal{X}^T\mathcal{L}\mathcal{X}\right)v = \mathcal{X}^T b^{\tilde{x}}$, and taking $x=\mathcal{X}v$ as the layout. Consequently, we modify the iterative 1-D layout algorithm of [KH03] as follows:

```
**1-D_stress_subspace** ($G(V,E)$, $\mathcal{X}\in\mathbb{R}^{n\times m}$, $x\in\mathbb{R}^n$)
  Compute the Laplacian $\mathcal{L}$
  $A \leftarrow \mathcal{X}^T\mathcal{L}\mathcal{X}$
  **do**
    $\tilde{x} \leftarrow x$
    Compute $b^{\tilde{x}}$ (by using (9))
    Compute $v$ for which $Av = \mathcal{X}^T b^{\tilde{x}}$
    $x \leftarrow \mathcal{X}v$
  **while** ($x \neq \tilde{x}$)
```

As was proved in [KH03], the stress energy of x strictly decreases in each iteration, and the process must converge.

We gain two benefits by minimizing the stress in a subspace. First, while using the original method, we solved in each iteration an $n\times n$ system of equations; here we solve an $m\times m$ system, which takes negligible time.

The second advantage is even more important and is related to the choice of $\mathcal{S}$. Since we optimize in subspaces that contain only “quality" drawings, we find that taking a very small set $\mathcal{S}$ is enough. Our way of choosing $\mathcal{S}$ is as follows. First, we construct a set $\mathcal{P}$ that contains k *pivot nodes* uniformly scattered over the graph. The construction of $\mathcal{P}$ is performed exactly like the way we have chosen the pivots for constructing the high-dimensional embedding in Subsection 3.1. Then, the set of pairs will be $\mathcal{S} = \{\{i,j\} \mid i\in\mathcal{P}, j\in V, i\neq j\}$. The distances between these pairs are enough to distinguish the nice layout from other layouts in the carefully crafted subspace. This is unlike the case when we optimize in the full $\mathbb{R}^n$, when we also have to add to $\mathcal{S}$ all pairs of closely located nodes. We have found that taking $k=40$ is often enough. Thus, $|\mathcal{S}|$ is linear in n, and it is independent of E. This is very important since $|\mathcal{S}|$ is a dominant magnitude, governing the time and space complexity of the algorithm. Also note that the pivot-smoothing mentioned in [KH03] is not needed here, since the layouts in the subspace are already “smooth".

As explained in [KH03], when we want to compute a 2-D layout by our algorithm, we have to compute $b^{\tilde{x}}$ (defined in equation (9)) by taking the target distances to be the *residual distances*. This way we account for the fact that some fraction of the target distances is already achieved in some other axis. Specifically, given some layout $z \in \mathbb{R}^n$, these residual distances are defined as:

$$d^z_{ij} = \begin{cases} \sqrt{d_{ij}^2 - (z_i - z_j)^2} & d_{ij} > |z_i - z_j| \\ 0 & \text{otherwise} \end{cases} \quad i, j = 1, \ldots, n. \tag{11}$$

Now, we compute a 2-D layout within a subspace by alternating between the computation of the x- and y-coordinates, as follows:

```
2-D_stress_subspace ($G(V,E)$, $\mathcal{X} \in \mathbb{R}^{n \times m}$, $x, y \in \mathbb{R}^n$)
  Compute the Laplacian $\mathcal{L}$
  $A \leftarrow \mathcal{X}^T \mathcal{L} \mathcal{X}$
  do
    % Improve the $x$-axis:
    $\tilde{x} \leftarrow x$
    Compute $b^{\tilde{x}}$ using the residual distances $d^y_{ij}$
    Compute $v$ for which $Av = \mathcal{X}^T b^{\tilde{x}}$
    $x \leftarrow \mathcal{X} v$
    % Improve the $y$-axis:
    $\tilde{y} \leftarrow y$
    Compute $b^{\tilde{y}}$ using the residual distances $d^x_{ij}$
    Compute $v$ for which $Av = \mathcal{X}^T b^{\tilde{y}}$
    $y \leftarrow \mathcal{X} v$
  while ($x \neq \tilde{x}$ or $y \neq \tilde{y}$)
```

The most expensive part of the algorithm is the computation of the relevant residual distances (those corresponding to pairs in $\mathcal{S}$) and the calculation of $b^{\tilde{x}}$ and $b^{\tilde{y}}$. Performance is improved if instead of switching between the axes in each iteration, we iterate with only one axis for a few iterations and then switch to the other axis. This saves the necessity of recomputing the residual distances in each iteration. We are still looking for ways for alleviating the cost of computing $b^{\tilde{x}}$ and $b^{\tilde{y}}$.

In our experiments we have found that optimization of the stress energy within a subspace yields very nice results, usually much better than optimizing the eigen-projection within the subspace. As expected, we have to pay in running time, which is significantly increased compared with eigen-projection in a subspace. However, subspace-restricted stress minimization is the fastest way that we know for optimization of the stress function, and for some graphs it is probably the fastest way of achieving a reasonable drawing.

As for the choice of the subspace, usually we have found that working with the high-dimensional embedding is superior to working with the low eigenspace of the Laplacian, since it contains high quality layouts with much fewer basis vectors. However, for a few graphs the low eigenspace of the Laplacian is advantageous, so our default setting is to form the subspace by joining the 10-D low eigenspace of the Laplacian with a 40-D HDE, obtaining a 50-D subspace. This choice usually works well, but it is certainly not optimal.

Figure 5 shows the results of the 2-D algorithm; we quote the running times measured on a Pentium IV 2GHz PC with 256MB RAM. The set of node pairs, $\mathcal{S}$, is constructed using 40 pivots. We initialized the process with the optimizer of eigen-projection within the subspace, as described in Subsection. 4.1 (this is included in the reported running times). If the iterative process exceeds 200 iterations, we halt it before convergence.

Especially interesting for us is the result for the Finan512 graph (|V|=74,752, |E|=261,120) given in Fig. 6. This graph was also drawn in Fig. 4, but only the stress-minimization strategy can show its impressive micro-structure. Interestingly, the Fruchterman-Reingold cost function used by Walshaw (see Fig. 6 in [Wal00]) also hides the micro-structure, producing a layout that resembles the result of the eigen-projection. Therefore, it is very important to be able to draw such a graph by minimizing the stress. Because of its large size, optimization within a subspace is the only way we could use for minimizing the stress in reasonable time and space. However, the results for this graph when building $\mathcal{S}$ using 40 pivots were not satisfactory and consequently we have worked with 70 pivots, resulting in a running time of 225 seconds.

Last, somewhat funny example is the horse polygonal mesh. The original mesh, shown in Fig. 7(a) consist of both connectivity and 3-D geometry (that is, we are given a graph and its layout). We used only the connectivity, and computed a 2-D layout using our method. The result, which shares some resemblance with the original mesh, is shown in Fig. 7(b). For a detailed study on drawing polygonal meshes based on their connectivity refer to [IGG01].

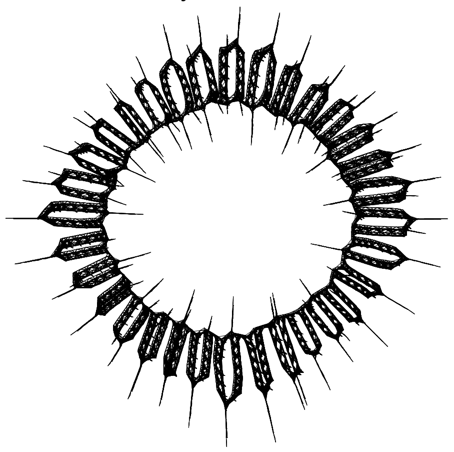

Figure 6: *Stress minimization within a 50-D subspace of the graph Finan512 [Wal]. $|V|$=74,752, $|E|$=261,120, 225 sec*

5. Conclusions and Future Work

We have shown that optimizing graph-drawing in carefully designed subspaces (vector spaces) is a powerful approach for drawing large graphs quickly. Two constructions of subspaces were suggested, one is based on the high-dimensional embedding of [HK02] and one is based on the low eigenspace of the Laplacian matrix associated with the graph. We have shown how to optimize within a subspace two cost-functions that measure layout quality. We are still looking for ways to optimize additional interesting cost functions within a subspace and also seeking new ways of constructing adequate subspaces.

References

[BW02] U. Brandes and T. Willhalm, “Visualizing Bibliographic Networks with a Reshaped Landscape Metaphor", *Proc. 4th Joint Eurographics - IEEE TCVG Symp. Visualization (VisSym'02)*, pp. 159-164, ACM Press, 2002.

[dBETT99] G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, 1999.

[Bra03] M. Brand, “Nonlinear Dimensionality Reduction by Kernel Eigenmaps", *Proc. 18th Inter. Joint Conference on Artificial Intelligence (IJCAI'2003)*, 2003.

[Ead84] P. Eades, “A Heuristic for Graph Drawing”, *Congressus Numerantium* **42** (1984), 149–160.

[FR91] T.M.G. Fruchterman and E. Reingold, “Graph Drawing by Force-Directed Placement”, *Software-Practice and Experience* **21** (1991), 1129–1164.

[GGK00] P. Gajer, M. T. Goodrich and S. G. Kobourov, “A Multi-dimensional Approach to Force-Directed Layouts of Large Graphs", *Proc. 8th Graph Drawing (GD'00)*, LNCS 1984, pp. 211–221, Springer-Verlag, 2000.

[GvL96] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, 1996.

[HH01] R. Hadany and D. Harel, “A Multi-Scale Method for Drawing Graphs Nicely", *Discrete Applied Mathematics* **113** (2001), 3-21.

[Hal70] K. M. Hall, “An r-dimensional Quadratic Placement Algorithm", *Management Science* **17** (1970), 219–229.

[HK00] D. Harel and Y. Koren, “A Fast Multi-Scale Method for Drawing Large Graphs", *Proc. 8th Graph Drawing (GD'00)*, LNCS 1984, pp. 183–196, Springer-Verlag, 2000.

[HK02] D. Harel and Y. Koren, “Graph Drawing by High-Dimensional Embedding”, *Proc. 10th Graph Drawing (GD'02)*, LNCS 2528, pp. 207–219, Springer-Verlag, 2002.

[HL95] B. Hendrickson and R. Leland, “A Multilevel Algorithm for Partitioning Graphs", *Proc. Supercomputing 1995*, ACM press, 1995.

[HMM00] I. Herman, G. Melancon and M. S. Marshall, “Graph Visualisation and Navigation in Information Visualisation", *IEEE Trans. on Visualization and Computer Graphics* **6** (2000), 24–43.

[IGG01] M. Isenburg, S. Gumhold and C. Gotsman C., “Connectivity Shapes", *Proc. Visualization (Vis'01)*, pp. 135–142, IEEE, 2001.

[KK89] T. Kamada and S. Kawai, “An Algorithm for Drawing General Undirected Graphs", *Information Processing Letters* **31** (1989), 7–15.

[KG00] Z. Karni Z. and C. Gotsman, “Spectral Compression of Mesh Geometry" *Computer Graphics (Proc. SIGGRAPH'00)*, pp. 279–286, 2000

[KW01] M. Kaufmann and D. Wagner (Eds.), *Drawing Graphs: Methods and Models*, LNCS 2025, Springer-Verlag, 2001.

[KCH02] Y. Koren, L. Carmel and D. Harel, “ACE: A Fast Multiscale Eigenvectors Computation for Drawing Huge Graphs", *Proc. IEEE Information Visualization (InfoVis'02)*, IEEE, pp. 137–144, 2002.

[Kor02] Y. Koren, “On Spectral Graph Drawing", *Proc. 9th Inter. Computing and Combinatorics Conference (COCOON'03)*, LNCS 2697, Springer-Verlag, pp. 496–508, 2003.

[KC03] Y. Koren and L. Carmel, “Visualization of Labeled Data Using Linear Transformations", *Proc. IEEE Information Visualization (InfoVis'03)*, IEEE, pp. 121–128, 2003.

[KH03] Y. Koren and D. Harel, “Axis-by-Axis Stress Minimization", *Proc. 11th Graph Drawing (GD'03)*, Springer-Verlag, 2003, in press.

[Wal00] C. Walshaw, “A Multilevel Algorithm for Force-Directed Graph Drawing", *Proc. 8th Graph Drawing (GD'00)*, LNCS 1984, pp. 171–182, Springer-Verlag, 2000.

[Pet] Petit's collection: `www.lsi.upc.es/˜jpetit/MinLA/Experiments`

[Wal] Walshaw's collection: `www.gre.ac.uk/˜c.walshaw/partition`

[MM] The Matrix Market collection: `math.nist.gov/MatrixMarket`

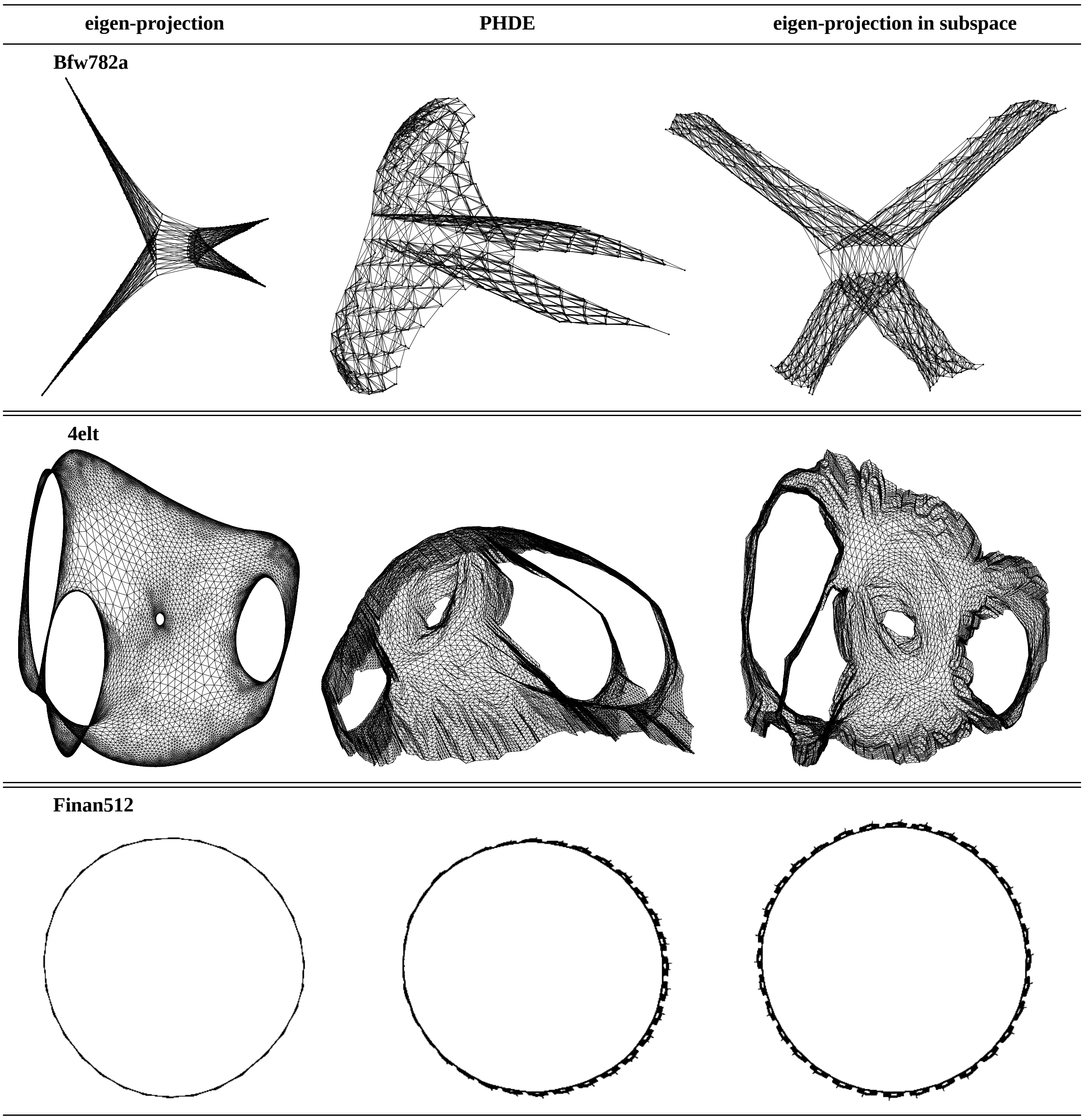

Figure 4: *Comparison of three drawing algorithms:* ***left:*** *eigen-projection,* ***middle:*** *PHDE (PCA of high-dimensional embedding), and* ***right:*** *eigen-projection optimized in high-dimensional embedding subspace. The results are given for the three graphs:* ***top:*** *the Bfw782a graph [MM] ($|V|$=782, $|E|$=3,394),* ***center:*** *the 4elt graph [Wal] ($|V|$=15,606, $|E|$=45,878), and* ***bottom:*** *the Finan512 graph [Wal] ($|V|$=74,752, $|E|$=261,120).*

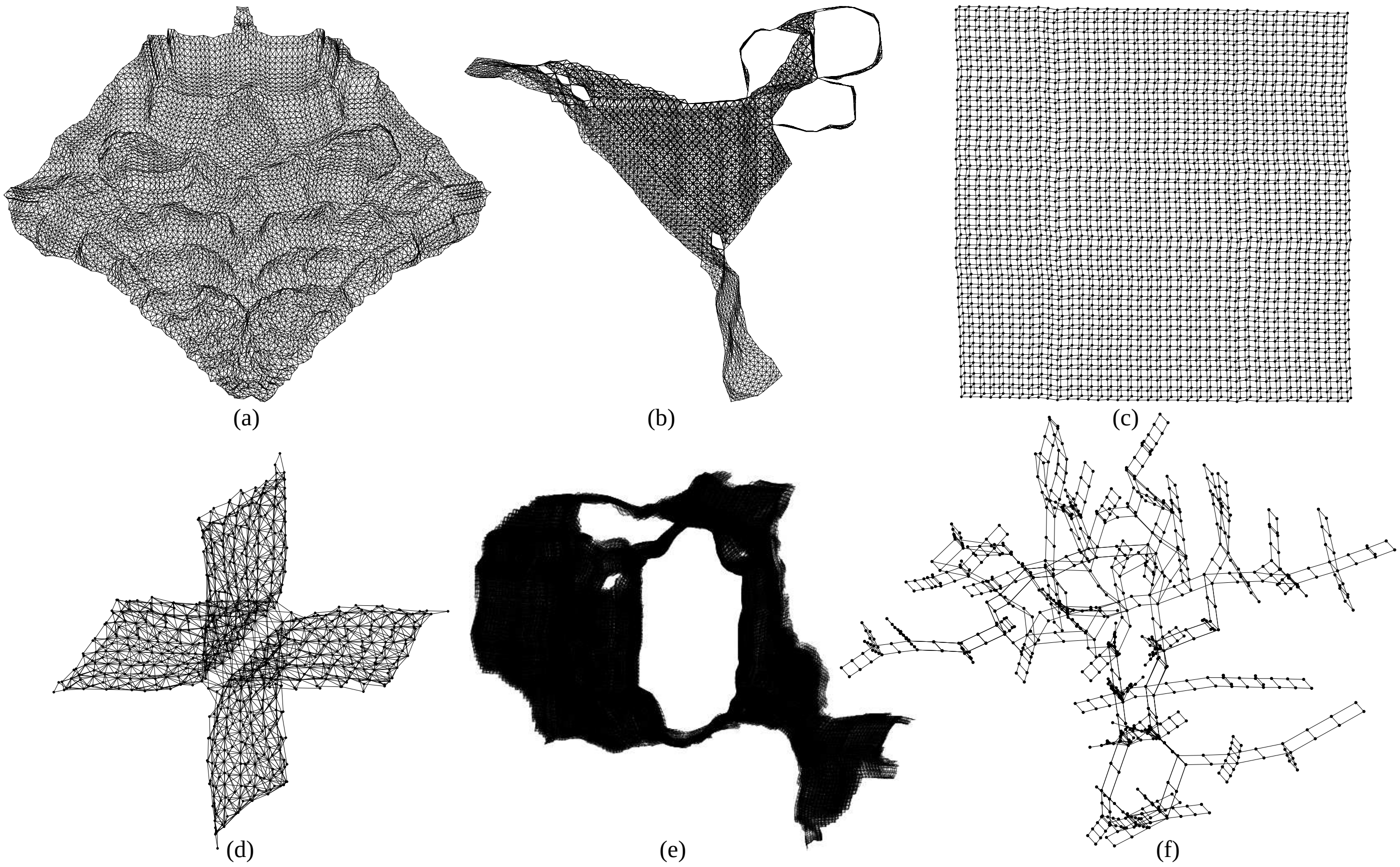

Figure 5: *Stress minimization within a 50-D subspace of the graphs* ***(a)*** *Crack [Pet]: |V|=10,240, |E|=30,380, running time is 9 sec;* ***(b)*** *Shuttle [Wal]: |V|=3200, |E|=7840, 2 sec;* ***(c)*** *Rdb3200l [MM]: |V|=3200, |E|=7840, 1 sec;* ***(d)*** *Bfw782a [MM]: |V|=782, |E|=3394, 0.3 sec;* ***(e)*** *Ocean [Wal]:|V|=143,437, |E|=409,593, 4 minutes;* ***(f)*** *Qh882 [MM]: |V|=882, |E|=1533, 0.5sec*

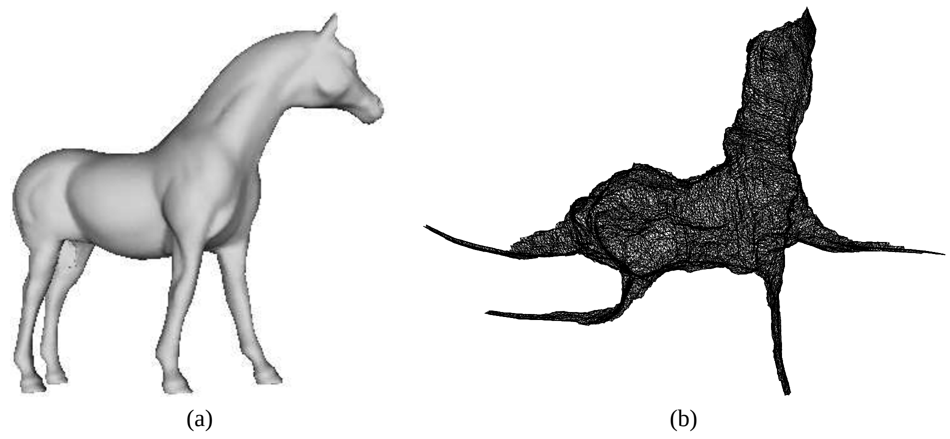

Figure 7: *The horse model:* ***(a)*** *Original polygonal mesh;* ***(b)*** *Drawing the connectivity of the mesh using stress minimization within a 50-D subspace. |V|=19,851, |E|=59,547, 29sec*