ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computational Geometry 32 (2005) 115–138

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

One-dimensional layout optimization, with applications to graph drawing by axis separation

Yehuda Koren [a,*], David Harel [b]

[a] *AT&T Labs—Research, Florham Park, NJ, USA*

[b] *Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel*

Received 5 February 2004; received in revised form 28 October 2004; accepted 1 March 2005

Available online 17 May 2005

Communicated by D. Wagner

Abstract

In this paper we discuss a useful family of graph drawing algorithms, characterized by their ability to draw graphs in one dimension. We define the special requirements from such algorithms and show how several graph drawing techniques can be extended to handle this task. In particular, we suggest a novel optimization algorithm that facilitates using the Kamada and Kawai model [Inform. Process. Lett. 31 (1989) 7–15] for producing one-dimensional layouts. The most important application of the algorithms seems to be in achieving graph drawing by *axis separation*, where each axis of the drawing addresses different aspects of aesthetics.

Keywords: Graph drawing; Force-directed algorithms; Stress energy; Multidimensional scaling; Principal component analysis; Eigenprojection; Digraph drawing; Newton–Raphson method

1. Introduction

A graph $G(V, E)$ is an abstract structure that is used to model a relation E over a set V of entities. Graph drawing is a standard means for the visualization of relational information, and its ultimate usefulness depends on the readability of the resulting layout; that is, the drawing algorithm's capability of

* Corresponding author.
E-mail addresses: yehuda@research.att.com (Y. Koren), dharel@weizmann.ac.il (D. Harel).

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2005.03.003

conveying the meaning of the diagram quickly and clearly. Consequently, many approaches to graph drawing have been developed [8,18]. We concentrate on the problem of drawing graphs so as to convey pictorially the proximity relations between the nodes. The most popular approaches to this appear to be *force-directed algorithms*. These define a cost function (or a force model), whose minimization determines the optimal drawing. Graph drawing research traditionally deals with drawing graphs in two or three dimensions. In this paper, we identify and discuss a new family of graph drawing algorithms whose goal is to draw the graph in one dimension. In fact, the methods that we utilize are not limited to traditional graph drawing and are also intended for general low-dimensional visualization of a set of objects according to their pairwise similarities/distances and even their multidimensional coordinates; see, e.g., Fig. 1.

An obvious question that the reader might have is: why could we be interested in one-dimensional graph drawing? Well, of course by using two dimensions we can convey much more information about the graph, but as we are going to show, in certain cases, working in two dimensions is impossible. The most common case is graph drawing by *axis separation*; a technique that was employed by, e.g., [2,28]. Here, we would like to build a multidimensional layout axis-by-axis, so that each axis can be computed using a different algorithm, perhaps accounting for different aesthetical considerations. This facilitates an appealing "divide-and-conquer" approach to graph drawing. In particular, we have implemented two axis-separation-based applications from which we drew our inspiration and that served us for exploring and evaluating the various algorithms. One application deals with drawing directed graphs (digraphs), whereas the other one is concerned with the visualization of clustered data. We now turn to describe these applications.

Drawing digraphs. A well-known example of using axis separation is the problem of drawing directed graphs (digraphs), where we want to give some sense of the overall directionality as well as to faithfully represent the relative similarities of the nodes. The dominant strategy, rooted in the work of Sugiyama et al. [28], is based on separating the axes, where the y-axis represents the directional information, or hierarchy, and the x-axis allows for additional aesthetic considerations, such as shortening edge lengths or minimizing the number of edge crossings. Our implementation is based on the graph drawing algorithm of [4], where the y-axis conveys the hierarchy structure of the graph by minimizing the hierarchy energy. The x-axis shows additional properties of the graph, which are unrelated to edge directions, using the various 1-D graph drawing algorithms that are developed here.

Visualization of clustered data. As another example, we have recently worked on visualizing clustered data using axis separation [21]. There, the x-coordinates guarantee the visual separation of clusters, whereas the y-coordinates address additional aesthetics while ignoring the clustering structure. The data are represented by a weighted graph that reflects pairwise similarities/distances. We are also given a hierarchical clustering of the data represented as a *dendrogram*—a full binary tree in which each subtree is a cluster and the leaves are individual elements

Our goal is to convey the data visually by associating each data element with a point in the plane, while also guaranteeing the visual separation of every two disjoint clusters in the dendrogram. Algorithmically, this is done by utilizing the axis-separation paradigm. Specifically, the x-coordinates are responsible for preserving the clustering structure of the data. This is based on a dynamic programming algorithm for calculating the "best" ordering of the dendrogram. We convey additional information about the data using the y-coordinates, disregarding the clustering information that is exhaustively taken care of by

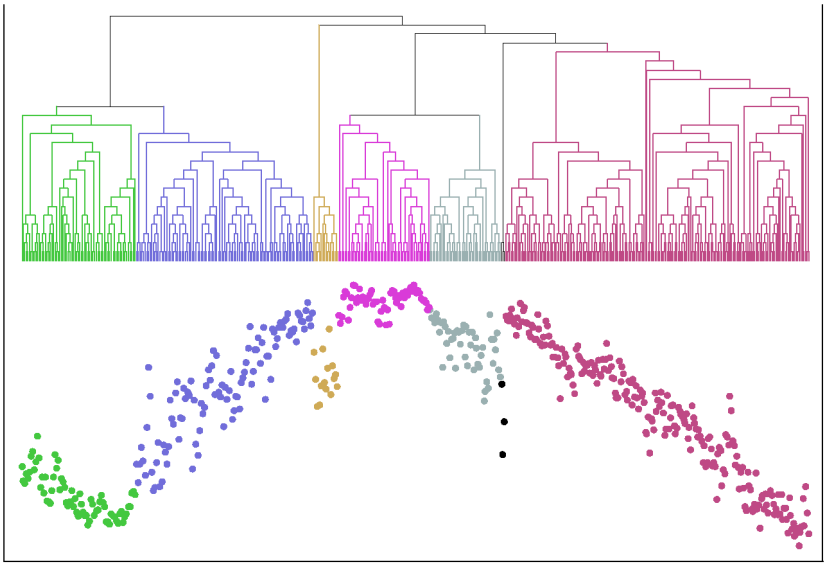

Fig. 1. (Taken from [21].) Using axis separation to draw hierarchically clustered fibroblast gene expression data. We convey both the similarities between the nodes and their clustering decomposition, using an ordered dendrogram coupled with a 2-D layout that adheres to its structure. We have colored six salient clusters that are clearly visible. (For interpretation of the references in color in this figure legend, the reader is referred to the web version of this article.)

the x-axis. Suitable methods for computing the y-axis are appropriately tuned one-dimensional graph drawing techniques.

Fig. 1 shows a sample result of this, containing a hierarchically-clustered biological dataset (modeled by a weighted graph). Consequently, the x-axis was computed so as to adhere to the dendrogram structure, while the y-axis was computed by a 1-D graph drawing algorithm (using the classical-MDS method, as described in Section 3.3).

Further applications. Sometimes a single dataset can be modeled by different graphs. Consequently, it might be instructive to draw the data by assigning each of the axes to a different graph, and then simultaneously examine and compare the characteristics of the two models. For example, proximity relationships between web pages can be modeled either by connecting pages that have a similar content, or by relying on their link structure. We can draw the web pages as points in the plane according to these two models by using axis separation, thus making it possible to see at a glance which elements are related by each of them.

Another tightly related case is when we already have one coordinate for each node. Such a coordinate might be a numeric attribute of the nodes that we want to convey spatially. In order to reflect proximity relationships, we would like to add another coordinate computed by a 1-D graph drawing algorithm. A nice example of this appears in [2]. There, a link structure (like the WWW) is visualized by associating one axis with a ranking of the nodes (some measure for node-prominency) and the other axis is computed by 1-D graph drawing (using the eigenprojection method described in Section 3.1). See Fig. 2.

Linear arrangement. So far, we have described situations were the 1-D graph drawing is used to construct a multidimensional drawing. However, in some cases, additional axes are not necessary, and we

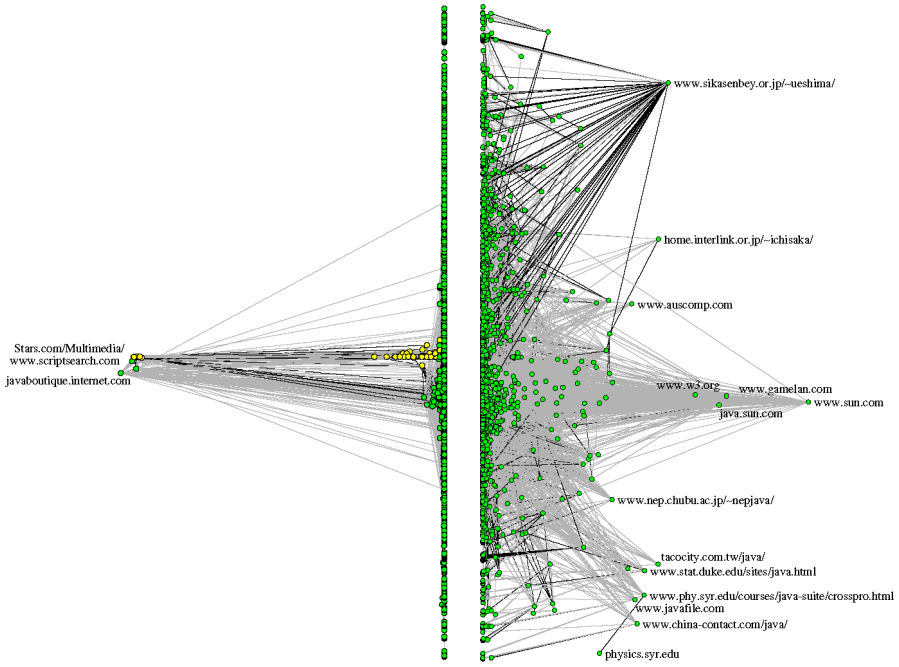

Fig. 2. Authority and PageRank visualization of the "java" query result, taken (with permission) from [2]. Each web-page is given two numerical values that measure its importance (Authority and PageRank). These values determine the x-coordinates of the drawing. The y-coordinates, which reflect the similarity between the web-pages, are computed by a graph drawing algorithm.

simply need an algorithm for ordering the vertices of a graph. Such an ordering usually optimizes one of several related cost functions; see, e.g., [7]. In particular, the problem is called *linear arrangement*, in the case that the ordering minimizes the sum of edge lengths. In the graph drawing context, such a problem arises in code and data layout applications [1], and in laying out software diagrams [26].

Fig. 3 shows how such a linear arrangement can be used to visualize a (weighted) adjacency matrix. The figure shows the relations between odor patterns measured by an electronic nose using a complete weighted graph; see [5]. As seen in part (a) of the figure, the "raw" adjacency matrix does not show any structure. However, the same matrix, shown in part (b) after permuting its rows and columns according to a linear arrangement of the graph, reveals much of the structure of the data. Ordering problems are naturally formulated as discrete optimization problems, where the coordinates are permutation of $\{1, \ldots, n\}$. However, such formulations lead to NP-hard problems that are difficult to solve. One way to eliminate part of this difficulty is to allow the nodes to take on non-integer coordinates. The resulting continuous problems can be efficiently solved, and their solution is used as an approximation of the optimal discrete ordering, by taking a sorted ordering of the nodes' coordinates; see [16,20]. In this way, the continuous formulations given in this paper can be used for discrete linear arrangement problems too.

Organization of the paper. A nice drawing of a graph is, in general, an ill defined notion and different aesthetical considerations are relevant under different situations. We provide four different methods for 1-D graph drawing. These methods address different aesthetic criteria and also differ in their computational complexity. Before describing the methods, we prepare the background by defining some essential

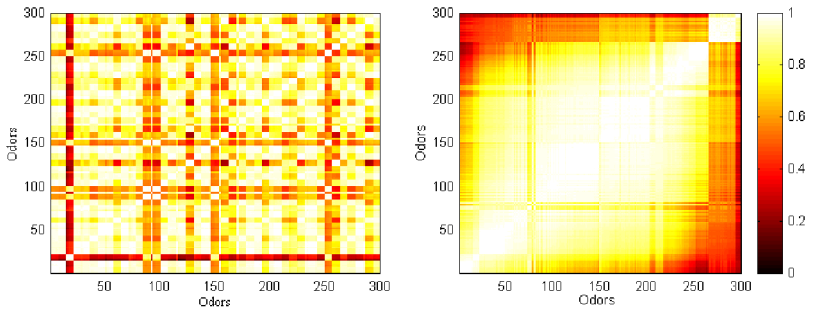

Fig. 3. Using linear arrangement for matrix visualization. (a) A similarity matrix of odor patterns as measured by an electronic nose; more similar patterns get higher (= brighter) similarity values. (b) The same similarity matrix after re-ordering rows and columns by a linear arrangement algorithm; the emerged homogeneous regions correspond to clusters of similar patterns.

mathematical notions in Section 2. Then, we discuss three algorithms that are based on exact optimization of a convenient mathematical nature. All these algorithms can be smoothly generalized for dealing with 1-D layouts using the "uncorrelation principle" as described in Section 3. While these algorithms are usually very fast, they are not the traditional way for drawing graphs; to wit, they are not mentioned in the two main surveys of graph drawing [8,18]. More heuristic approaches will often produce better results. Consequently, in Section 4 we explain how to harness the well-known method of Kamada and Kawai [17] to 1-D graph drawing. Here the necessary algorithmic adaptations are more involved, and hence we provide a longer description of the drawing process.[1] Whereas this last approach cannot use exact optimization like the first three methods, its heuristic optimization will produce nicer results for a wide family of graphs. Results and comparisons between the methods are described towards the end of Sections 3 and 4.

2. Basic notions

We begin with some basic notions from linear algebra. Let A be an $n \times n$ real matrix. It is called *symmetric* if for all $1 \leqslant i, j \leqslant n$: $A_{ij} = A_{ji}$. A pair $\lambda \in \mathbb{C}$ and $v \in \mathbb{C}^n$ that satisfy the equation

$$Av = \lambda v$$

is called an *eigenpair*, where the scalar λ is an *eigenvalue* and v is an *eigenvector*.

If all eigenvalues associated with a matrix are real positive, the matrix is called *positive definite*. When all eigenvalues are real non-negative, the matrix is called *positive semi-definite*. A positive definite matrix A is very convenient when solving the equation $Ax = b$, since we can use fast and robust direct solvers like Cholesky factorization or known iterative solvers like Conjugate Gradient and Gauss Seidel. These algorithms are implemented in various linear algebra packages and described in [13].

The *null space* of a matrix A is the vector space containing all vectors $x \in \mathbb{R}^n$ for which $Ax = 0$. Consequently, solutions of the equation $Ax = b$ are invariant under addition of vectors belonging to the null space.

[1] Section 4 is mostly taken from [22].

We denote the normalized value of x by $\hat{x} = x/\|x\|$. Henceforth, we will extensively use the *elementary orthogonal projector* $I - \hat{x}\hat{x}^{\mathrm{T}}$, which is a symmetric $n \times n$ matrix. This matrix possesses the important property of being an orthogonalization operator: for any vector $y \in \mathbb{R}^n$, the result of orthogonalizing y against x is $(I - \hat{x}\hat{x}^{\mathrm{T}})y$, so: $(I - \hat{x}\hat{x}^{\mathrm{T}})y \perp x$.

Throughout the paper, we assume that we are given an n-node connected weighted graph $G(V, E)$, with $V = \{1, \ldots, n\}$ and the weight of the edge $\{i, j\}$ is $w_{ij} \geqslant 0$. A key entity that describes relations between nodes is the *Laplacian*, which is an $n \times n$ symmetric positive semi-definite matrix denoted by L, where

$$L_{ij} = \begin{cases} -w_{ij} & \{i, j\} \in E \\ 0 & \{i, j\} \notin E,\ i \neq j \\ \deg(i) & i = j \end{cases} \qquad i, j = 1, \ldots, n.$$

Here, $\deg(i) \stackrel{\text{def}}{=} \sum_{j:\ \{i,j\}\in E} w_{ij}$. It is easy to check that $1_n \stackrel{\text{def}}{=} (1, \ldots, 1)^{\mathrm{T}} \in \mathbb{R}^n$ is an eigenvector of L with associated eigenvalue 0. When the graph is connected, all other eigenvalues are strictly positive.

The Laplacian plays a key role in spectral graph theory, and some of its interesting properties are mentioned in [27]. For the purposes of this work the usefulness of the Laplacian stems from the fact that the quadratic form associated with it is just the sum of weighted squared edge lengths. We formulate this for a 1-D layout:

Lemma 1. *Let L be an $n \times n$ Laplacian, and let $x \in \mathbb{R}^n$. Then*

$$x^{\mathrm{T}}Lx = \sum_{\{i,j\}\in E} w_{ij}(x_i - x_j)^2.$$

The proof of the lemma is straightforward, and it can be extended to multidimensional layouts too.

We now recall some basic statistical notions. The *mean* of a vector $x \in \mathbb{R}^n$, denoted by $\bar{x}$, is defined as $\frac{1}{n}\sum_{i=1}^n x_i$. The *variance* of x, denoted by $\mathrm{Var}(x)$, is defined as $\frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})^2$. The *covariance* between two vectors $x, y \in \mathbb{R}^n$ is defined as $\mathrm{Cov}(x, y) = \frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$. The *correlation coefficient* between x and y is defined as

$$\frac{\mathrm{Cov}(x, y)}{\sqrt{\mathrm{Var}(x)\mathrm{Var}(y)}}.$$

This measures the colinearity between the two vectors. If the correlation coefficient is 0, x and y are *uncorrelated*. If x and y are independent, then they are uncorrelated (but the inverse does not necessarily hold).

As explained earlier, 1-D drawing algorithms are often used in the context of multidimensional drawings. Henceforth, for simplicity, assume we have to compute the y-coordinates, while (possibly) being given precomputed x-coordinates. Thus, the layout is characterized by two vectors $x, y \in \mathbb{R}^n$, with the x-coordinates being $x_1, \ldots, x_n$, and the y-coordinates $y_1, \ldots, y_n$. Other cases, where we have more than one precomputed axis or where we want to produce several dimensions, can be addressed by small changes in our techniques. Moreover, we assume, without loss of generality, that the x- and y-coordinates are *centered*, so their means are 0. In symbols, $\sum_{i=1}^n x_i = \sum_{i=1}^n y_i = 0$. This can be achieved by a simple translation, and it will simplify notation.

3. Algorithms based on the uncorrelation principle

In principle, we could have used a classical force-directed algorithm for computing the 1-D layout. However, when trying to modify the customary two-dimensional optimization algorithm for use in our one-dimensional case, convergence was rarely achieved. Traditionally, node-by-node optimization is performed, by moving each node to a point that decreases the cost function. Common methods for this are Gradient-Descent and localized Newton–Raphson. However, these methods tend to get stuck in bad local minima when used for 1-D drawing [4,30]. Interestingly, 2-D drawing is much easier for such methods. Probably, the reason is that there is less space for maneuver in one dimension when seeking a nice layout, which prevents convergence to an optimum. Furthermore, in several works even higher-dimensional layout (e.g., 3-D) is used to avoid local minima, see, e.g., [3,12,30].

Another possible approach could be to use algorithms for computing (approximated) minimum linear arrangements (MinLA). These set the coordinates to be a permutation of $\{1, \ldots, n\}$ in a way that minimizes the sum of edge lengths. However, a major disadvantage of MinLA is that it cannot consider precomputed coordinates. Note that a careless computation that ignores such precomputed coordinates can be very problematic. Such a computation might yield y-coordinates that are very similar to the x-coordinates, resulting in a drawing whose intrinsic dimensionality would really be 1, meaning that one axis would be wasted; for example consider the layouts of Nos3 and Plat362 in Fig. 6 that will be discussed later.

In the rest of this section, we describe three different methods that fit the 1-D layout task: eigenprojection, PCA and CMDS. A common characteristic of these methods, which makes them suitable for 1-D optimization, is that they compute the layout *axis-by-axis*, instead of the *node-by-node* optimization mechanism of force-directed methods. Furthermore, when these methods are used to produce a multidimensional layout, the different axes are uncorrelated. This suggests an effective way to generalize the methods so that they can deal with the precomputed coordinates: we simply require *no correlation* between the precomputed x-coordinates and the y-coordinates, so that the latter ones will provide us with as much new information as possible. Technically, since we have assumed x and y to be centered, the no-correlation requirement can be formulated simply as $y^{\mathrm{T}} \cdot x = 0$, which states that x and y are orthogonal.

We now describe the three methods. For each method we survey its mathematical foundations and then explain how to generalize it to handle the predefined x-coordinates. We conclude this section with experiments and comparisons showing the merits of the various methods.

3.1. Eigenprojection

The eigenprojection, which is rooted in the 1970's work of Hall [14], computes the layout of a graph using low eigenvectors of its Laplacian. Some important advantages of this approach are its ability to compute optimal layouts (according to specific requirements) and a very short computation time [19]. As we will see, this method is a natural choice for 1-D layouts, and has already been used for such tasks in [1,2,4,16]. However, previous work did not consider the relation with the given coordinates, which we handle by the uncorrelation requirement. We begin with a brief derivation of the method in a way that shows its tight relationship with force-directed graph drawing.

The optimal 1-D layout $y \in \mathbb{R}^n$ is defined as the solution of:

$$\min_y \frac{\sum_{\{i,j\}\in E} w_{ij}(y_i - y_j)^2}{\sum_{i<j}(y_i - y_j)^2}. \tag{1}$$

The energy to be minimized strives to make edge lengths short (to minimize the numerator) while scattering the nodes in the drawing area, thus preventing their overcrowding (to maximize the denominator). This way, we adopt a classical strategy to graph drawing by which adjacent nodes should be drawn closely, while, generally, nodes should not be drawn too close to each other; see, e.g., [9,11]. Since the sum is weighted by edge-weights, "heavy" edges have a stronger impact and hence will be typically shorter. Consequently, when using eigenprojection edge weights should reflect pairwise similarities. The most common case dealt with in the literature is drawing unweighted graphs where all edge weights are set to 1.

It is easy to see that the energy to be minimized is invariant under translation of the data. Thus, for convenience, we eliminate this degree of freedom by requiring that y be centered; that is, $y^{\mathrm{T}}1_n = 0$. Consequently, we can simplify (1) by replacing $\sum_{i<j}(y_i - y_j)^2$ with the proportional $y^{\mathrm{T}}y$. Moreover, using Lemma 1, we can write $\sum_{\{i,j\}\in E} w_{ij}(y_i - y_j)^2$ as the quadratic form $y^{\mathrm{T}}Ly$. We can now reformulate our minimization problem in the equivalent form:

$$\begin{aligned} &\min_y \frac{y^{\mathrm{T}}Ly}{y^{\mathrm{T}}y} \quad \text{in the subspace:} \\ &y^{\mathrm{T}}1_n = 0. \end{aligned} \tag{2}$$

By substituting $\hat{x} = 0$ in Proposition 3.1 below, we find that the optimal 1-D layout is the eigenvector of L with the smallest positive eigenvalue.

This way, the eigenprojection method provides us with an efficient way to calculate optimal 1-D layouts. We still have to show how the eigenprojection can be extended to deal with the uncorrelation requirement: that is a case where we already have a coordinate vector x, and we require that y is orthogonal to x. Now, the optimal layout will be the solution of:

$$\begin{aligned} &\min_y \frac{y^{\mathrm{T}}Ly}{y^{\mathrm{T}}y} \quad \text{in the subspace:} \\ &y^{\mathrm{T}}1_n = 0, \quad y^{\mathrm{T}}x = 0. \end{aligned} \tag{3}$$

Fortunately, the optimal layout is still a solution of a related eigen-equation:

Proposition 3.1. *The solution of* (3) *is the eigenvector of* $(I - \hat{x}\hat{x}^{\mathrm{T}})L(I - \hat{x}\hat{x}^{\mathrm{T}})$ *with the smallest positive eigenvalue.*

Proof. Without loss of generality, we can assume that $y^{\mathrm{T}}y = 1$, because changing the scale still gives an optimal solution: it can be easily checked that if for y_0 we get $y_0^{\mathrm{T}}Ly_0/y_0^{\mathrm{T}}y_0 = \lambda$, then we will also get $y^{\mathrm{T}}Ly/y^{\mathrm{T}}y = \lambda$ for each $y = c \cdot y_0$ ($c \neq 0$). Thus, the new form of the optimization problem will be:

$$\begin{aligned} &\min_y y^{\mathrm{T}}Ly \quad \text{in the subspace:} \\ &y^{\mathrm{T}}1_n = 0, \quad y^{\mathrm{T}}x = 0 \\ &\text{given:} \quad y^{\mathrm{T}}y = 1. \end{aligned} \tag{4}$$

The matrix $(I - \hat{x}\hat{x}^{\mathrm{T}})L(I - \hat{x}\hat{x}^{\mathrm{T}})$ is symmetric, so it has n orthogonal eigenvectors spanning $\mathbb{R}^n$. We will use the convention $\lambda_1 \leqslant \lambda_2 \leqslant \cdots \leqslant \lambda_n$ for the eigenvalues of $(I - \hat{x}\hat{x}^{\mathrm{T}})L(I - \hat{x}\hat{x}^{\mathrm{T}})$, and denote the corresponding real orthonormal eigenvectors by $u_1, u_2, \ldots, u_n$. Clearly, $(I - \hat{x}\hat{x}^{\mathrm{T}})L(I - \hat{x}\hat{x}^{\mathrm{T}}) \cdot x = 0$. Utilizing the fact that $x^{\mathrm{T}}1_n = 0$ and that 1_n is the only zero eigenvector of L, we obtain $\lambda_1 = \lambda_2 = 0$, $u_1 = \hat{x}$, $u_2 = (1/\|1_n\|) \cdot 1_n$, and $\lambda_3 > 0$.

We can now decompose every $y \in \mathbb{R}^n$ as a linear combination, where $y = \sum_{i=1}^{n} \alpha_i u_i$. Moreover, since the solution is constrained to be orthogonal to u_1 and u_2, we can restrict ourselves to linear combinations of the form $y = \sum_{i=3}^{n} \alpha_i u_i$.

Use the constraint $y^{\mathrm{T}}y = 1$ to obtain $\sum_{i=3}^{n} \alpha_i^2 = 1$ (a generalization of the Pythagorean law). Similarly, $y^{\mathrm{T}}(I - \hat{x}\hat{x}^{\mathrm{T}})L(I - \hat{x}\hat{x}^{\mathrm{T}})y = \sum_{i=3}^{n} \alpha_i^2 \lambda_i$. Note that since $y^{\mathrm{T}}\hat{x} = \hat{x}^{\mathrm{T}}y = 0$, we get:

$$y^{\mathrm{T}}(I - \hat{x}\hat{x}^{\mathrm{T}})L(I - \hat{x}\hat{x}^{\mathrm{T}})y = y^{\mathrm{T}}Ly - y^{\mathrm{T}}\hat{x}\hat{x}^{\mathrm{T}}L(I - \hat{x}\hat{x}^{\mathrm{T}})y - y^{\mathrm{T}}L\hat{x}\hat{x}^{\mathrm{T}}y = y^{\mathrm{T}}Ly.$$

So the target value is

$$y^{\mathrm{T}}Ly = y^{\mathrm{T}}(I - \hat{x}\hat{x}^{\mathrm{T}})L(I - \hat{x}\hat{x}^{\mathrm{T}})y = \sum_{i=3}^{n} \alpha_i^2 \lambda_i \geqslant \sum_{i=3}^{n} \alpha_i^2 \lambda_3 = \lambda_3.$$

Thus, for any y that satisfies the constraints, we get $y^{\mathrm{T}}Ly \geqslant \lambda_3$. Since $u_3^{\mathrm{T}}Lu_3 = u_3^{\mathrm{T}}(I - \hat{x}\hat{x}^{\mathrm{T}})L(I - \hat{x}\hat{x}^{\mathrm{T}})u_3 = \lambda_3$, we can deduce that the minimizer is u_3, the lowest positive eigenvector. □

Interestingly, posing the problem as in (3) and solving it as in Proposition 3.1, constitutes a smooth generalization of the eigenprojection method: when x is the lowest positive eigenvector of L, then the solution y will be the second lowest positive eigenvector of L. This coincides with the way the eigenprojection computes 2-D layouts; see [14]. However, we allow the more general case of arbitrary x-coordinates.

As to computational complexity, the space usage of the algorithm is $O(|E|)$ when using a sparse representation of the Laplacian. The computation can be carried out using iterative algorithms, such as the Power-Iteration that can be easily implemented or the considered faster Lanczos method that is implemented in public libraries such ARPACK [25]; see, e.g., [13]. The time complexity of a single iteration is $O(|E|)$. The number of iterations (and hence the actual running time) depends on the structure of the graph, or more precisely on the separation between relevant eigenvalues. For many graphs it is possible to initialize the process with a smart placement that results in a significant reduction of the number of iterations and makes it possible to deal with millions of nodes in a reasonable amount of time; see [19,23] where concrete running times are also given.

When using sparse solvers caution is needed, since an explicit calculation of $(I - \hat{x}\hat{x}^{\mathrm{T}})L(I - \hat{x}\hat{x}^{\mathrm{T}})$ would destroy the sparsity of L rendering all iterative algorithms impractical. To get around this, we utilize the fact that the iterative algorithms for computing eigenvectors use the matrix as an operator, i.e., they access it only via multiplication with a vector. This settles the issue, since carrying out the product $(I - \hat{x}\hat{x}^{\mathrm{T}})L(I - \hat{x}\hat{x}^{\mathrm{T}}) \cdot v$ is equivalent to orthogonalizing v against x, multiplying the result with the sparse matrix L, and then again orthogonalizing the result against x.

3.2. Principal component analysis

In the previous section we described a method that addresses data modeled by pairwise similarities. Now we turn to the visualization of multivariate data, where each node has multidimensional coordinates

(or, attributes). The most popular technique for visualizing such data is principal component analysis (PCA). PCA computes a projection of multidimensional data that optimally preserves their variance; see [10]. The fact that PCA uses the data coordinates apparently renders it useless for most graph drawing applications. However, in [15] we show that it is possible to generate artificial k-dimensional coordinates of the nodes that preserve some of the graph structure, thus making it possible to use PCA.

Let us denote the nodes' coordinates (either produced artificially or given by an external source) by an $n \times k$ coordinate matrix called $\mathcal{X}$, so the k coordinates of node i constitute the ith row of $\mathcal{X}$. We assume each of the columns of $\mathcal{X}$ is centered, something that can be achieved by translating the data. In order to compute a 1-D projection, PCA computes a unit vector $d \in \mathbb{R}^k$, which is the direction of the projection. The vector d is the top eigenvector of the *covariance matrix* $\frac{1}{n}\mathcal{X}^{\mathrm{T}}\mathcal{X}$. The projection itself is $\mathcal{X}d$, and, as mentioned, it is the best 1-D projection in terms of variance preservation.

When given x-coordinates, we will be interested only in the component of the projection that is orthogonal to the x-coordinates. This component is exactly $(I - \hat{x}\hat{x}^{\mathrm{T}}) \cdot (\mathcal{X}d)$, and we want to maximize its variance. However,

$$(I - \hat{x}\hat{x}^{\mathrm{T}}) \cdot (\mathcal{X}d) = \big((I - \hat{x}\hat{x}^{\mathrm{T}}) \cdot \mathcal{X}\big)d,$$

so our problem is reduced to finding the most variance-preserving projection of the coordinates $(I - \hat{x}\hat{x}^{\mathrm{T}}) \cdot \mathcal{X}$. The optimal solution is obtained by performing PCA on $(I - \hat{x}\hat{x}^{\mathrm{T}}) \cdot \mathcal{X}$, which is equivalent to orthogonalizing each of $\mathcal{X}$ columns against x and then performing PCA on the resulting matrix.

Again, this is a smooth generalization of PCA that enables it to deal with predefined x-coordinates. The reason is that if x was also computed by PCA, then one would obtain the regular 2-D PCA projection. One of the advantages of the PCA approach is its excellent time and space complexity: space complexity is linear and time complexity is $O(k^2 n)$ for the PCA itself together with an additional $O(k|E|)$ time that is needed to construct the "artificial" coordinates; see [15].

3.3. Classical multidimensional scaling

Multidimensional scaling (MDS) is a general term for techniques that generate coordinates of points from information about pairwise distances. Here we are interested in a technique called classical-MDS (CMDS) [10], which produces (multidimensional) coordinates that preserve the given pairwise distances perfectly; i.e., the pairwise Euclidean distances in the generated space are equal to the given distances. The graph drawing application of CMDS was suggested long ago, in [24]. The distance between nodes i and j is defined as d_{ij}, the graph-theoretical distance between the nodes. Therefore, CMDS can be used to find a Euclidean embedding of the graph that preserves the graph-theoretical distance. Anyway, our derivation is not limited to the graph-theoretical distance, and can be applied to other distances like those extracted from attributes associated with the nodes.

We now provide a short technical description of the method. Given points in Euclidean space, it is possible to construct a matrix $\mathcal{X}$ of centered coordinates if we know the pairwise distances among the points. The way to do this is to construct the $n \times n$ inner product matrix $B = \mathcal{X}\mathcal{X}^{\mathrm{T}}$, which can be computed using the cosine law, as follows:

$$B_{ij} = -\frac{1}{2}\left(d_{ij}^2 - \frac{1}{n}\sum_{k=1}^{n} d_{ik}^2 - \frac{1}{n}\sum_{k=1}^{n} d_{kj}^2 + \frac{1}{n^2}\sum_{k=1,\ l=1}^{n} d_{lk}^2\right). \tag{5}$$

Note that B is invariant under orthogonal transformations of $\mathcal{X}$. That is, given some orthogonal matrix Q (i.e., $QQ^{\mathrm{T}} = I$), we can replace $\mathcal{X}$ with $\mathcal{X}Q$, without changing the inner-product matrix:

$$\mathcal{X}Q(\mathcal{X}Q)^{\mathrm{T}} = \mathcal{X}QQ^{\mathrm{T}}\mathcal{X} = \mathcal{X}\mathcal{X}^{\mathrm{T}} = B.$$

Therefore, B determines the coordinates up to orthogonal transformation. This is reasonable, since such a transformation does not alter pairwise distances. There is always an orthogonal transformation that makes the axes orthogonal (i.e., the singular value decomposition), which allows us to restrict ourselves to a coordinate matrix with orthogonal columns. Such a matrix can be obtained by factoring B using the eigenvalue decomposition $B = U\Delta U^{\mathrm{T}}$ (U is orthogonal and Δ is diagonal), which makes it possible to define the coordinates of the points as $\mathcal{X} = U\Delta^{1/2}$. This way, the columns of $\mathcal{X}$ are centered and are mutually orthogonal. In practice, we do not need all the coordinates but only a low-dimensional projection of the points, and here only a 1-D embedding is needed. Thus, as in PCA, we seek the 1-D projection of $\mathcal{X}$ having the maximal variance. Since the columns of $\mathcal{X}$ are uncorrelated, we simply take the column with the maximal variance, which is equivalent to the column of U with the highest corresponding eigenvalue. Therefore, we are interested in the top eigenvector u_1 and the corresponding eigenvalue, λ_1, of B. After computing this eigenpair, we can define the embedding of the data as $\sqrt{\lambda_1}u_1$. Additional coordinates can be obtained using the subsequent eigenpairs.

It appears that CMDS is closely related to PCA. In fact, CMDS is a way of performing PCA without explicitly defining the coordinate matrix. Thus, if the pairwise distances are Euclidean distances based on the coordinate matrix, the results of CMDS are identical to PCA. Consequently, in our case, when we want the embedding to be orthogonal to x, we can use the same technique we used in PCA.

Once again, we would like to perform PCA on $(I - \hat{x}\hat{x}^{\mathrm{T}})\mathcal{X}$, and of course we do not have this matrix explicitly. However, it is possible to compute the inner-product matrix $(I-\hat{x}\hat{x}^{\mathrm{T}})\mathcal{X}\mathcal{X}^{\mathrm{T}}(I-\hat{x}\hat{x}^{\mathrm{T}})$, since this matrix is simply $(I-\hat{x}\hat{x}^{\mathrm{T}})B(I-\hat{x}\hat{x}^{\mathrm{T}})$. Using the same reasoning as above, the first principal component of $(I - \hat{x}\hat{x}^{\mathrm{T}})\mathcal{X}$ can be found by computing the top eigenpair of $(I - \hat{x}\hat{x}^{\mathrm{T}})B(I - \hat{x}\hat{x}^{\mathrm{T}})$.

However, there is one theoretical flaw in applying CMDS to graph-drawing. Computing a coordinate matrix $\mathcal{X}$ that preserves pairwise distances is not always possible, and will fail when the graph-theoretical metric is not *Euclidean*. Technically, there might be some negative eigenvalues to the matrix B, preventing the square-root operation from being carried out. However, in practice this is not a serious problem, since we are not interested in recovering the full multidimensional coordinates, but only the few leading ones.

When the given x-coordinates are also the result of CMDS, our method produces the same y-coordinates as CMDS. Therefore, we have a smooth generalization of CMDS that allows it to deal with predefined coordinates.

One note on complexity. When performing this CMDS, we have to store the matrix B, which requires $O(n^2)$ space complexity, much worse than in the eigenprojection or PCA cases.

3.4. Experiments and comparison

So far, we have generalized three graph drawing algorithms using the unified paradigm of computing the layout axis-by-axis, while maintaining non-correlation with the precomputed coordinates. The three algorithms represent two different attitudes to the definition of a nice layout. The eigenprojection defines the nice layout as the minimizer of an energy that calls for placing similar (adjacent) nodes closely. On the other hand, the CMDS/PCA approach is based on preserving multidimensional coordinates. This

Table 1
Comparison of the three methods for 1-D graph drawing

Method	Representation	Principle	Space complexity	Time complexity
Eigenprojection	pairwise similarities	energy minimization	linear	iterative, linear time per iteration
CMDS	pairwise distances	variance preservation of multidimensional coords.	quadratic	iterative, quadratic time per iteration
PCA	coordinates	variance preservation of (artificial) multidimensional coords.	linear	non-iterative, linear

distinction induces different outputs for the different approaches, being sometimes in favor of the eigenprojection and other times in favor of CMDS. Whereas, eigenprojection and CMDS produce globally optimal layouts (according to different criteria), the PCA-based graph drawing relies on artificially generated coordinates. Since PCA and CMDS are both based on the same variance-preservation principle, the naturalness of CMDS makes it a better choice than PCA in terms of output quality.

Regarding computational complexity, the situation is different. The PCA-based approach does not involve an optimization process and hence is the only one that guarantees linear time and space complexity. On the other hand, CMDS is less efficient and impractical for large graphs containing more than a few thousand nodes. We summarize the characteristics of the methods in Table 1.

We have used the three methods for computing the x-axis in the digraph drawing algorithm of [4]. To get a flavor of the results we provide here the layouts of three digraphs using the three algorithms. (Details regarding all graphs given throughout this paper are provided in Table 2.) The Nos3 graph is drawn differently by the three algorithms, with the PCA result failing to show the symmetries of the graph. The Nos6 graph is arguably drawn best by CMDS, which shows the most rigid layout with close to 90° angles for most of the meeting points of edges. However, eigenprojection and PCA produce smoother results that have a certain spherical appeal. In addition, the eigenprojection result shows very dense areas at the boundaries, which is a general characteristic of this method. In the layout of the Plat362 graph, we also prefer the CMDS result, which scatters the nodes more uniformly.

Clearly, CMDS is not always superior. In fact when we worked with visualization of clustered data the eigenprojection frequently yielded better results. For example, consider the odor dataset, which contains 262 elements corresponding to measurements over 30 volatile odorous chemicals. We are given a hierarchical clustering of the dataset, represented by a dendrogram containing 261 (nested) clusters. We know that 30 of these clusters are really meaningful; see more explanations in [21]. We computed a 2-D layout of the database. The x-coordinates were computed so as to adhere to the dendrogram structure. The y-coordinates were computed in order to convey additional properties of the data and to allow picking the interesting clusters out of the dendrogram. We have computed the y-axis twice, once using eigenprojection and once using CMDS. The result is given in Fig. 5, where each of the given 30 clusters gets a different (hand-prepared) color. Note that the embedding algorithm worked with a dendrogram containing 261 clusters, and was not informed that 30 of these clusters are the "natural" ones. Nonetheless, in the eigenprojection result we can see that many of these 30 clusters were clearly distinguished in the embedding, being well separated from the other clusters. Still, the CMDS result is of lower quality: it shows many interesting properties of the data, but separation between clusters is less clear.

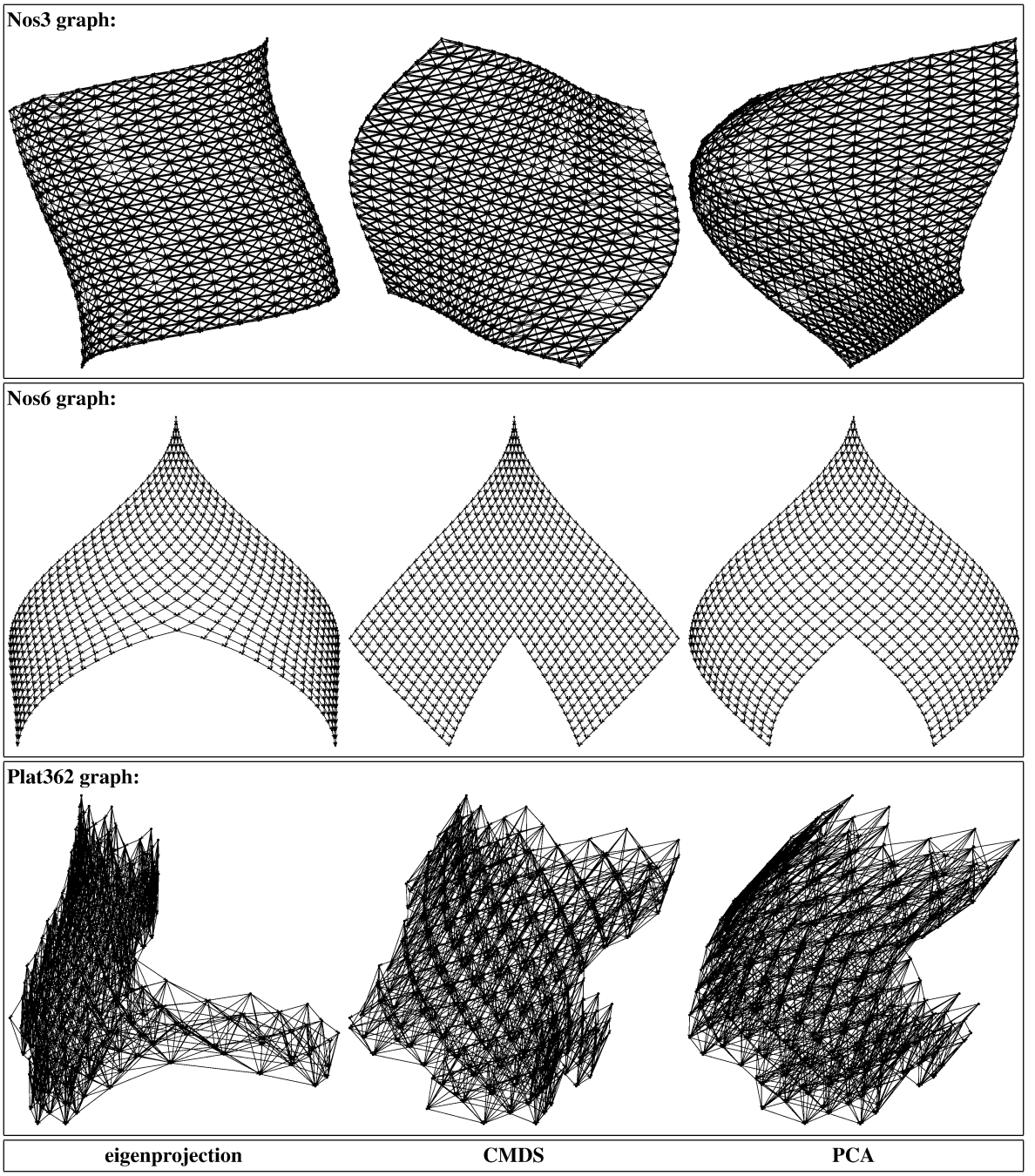

Fig. 4. Demonstration of drawings produced by our algorithm. The different columns are distinguished by the method used to produce the x-coordinates. The eigenprojection was used in the left column, CMDS was used in the middle column, and PCA was used in the right column. Results are shown for the digraphs Nos3 ($|V| = 960$, $|E| = 7442$), Nos6 ($|V| = 675$, $|E| = 1290$) and Plat362 ($|V| = 362$, $|E| = 2712$), all based on matrices from [32]. The y-coordinates are identical for the three versions of each digraph.

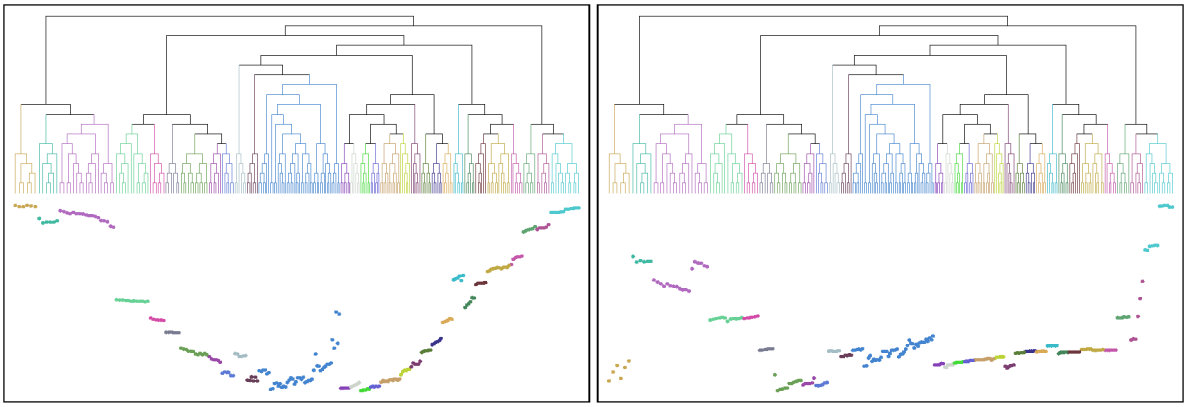

Fig. 5. Odors dataset containing 262 measurements by electronic nose over 30 different chemicals, suggesting a partitioning of the data into 30 clusters. Results are color-coded according to the partitioning. (For interpretation of the references in color in this figure legend, the reader is referred to the web version of this article.)

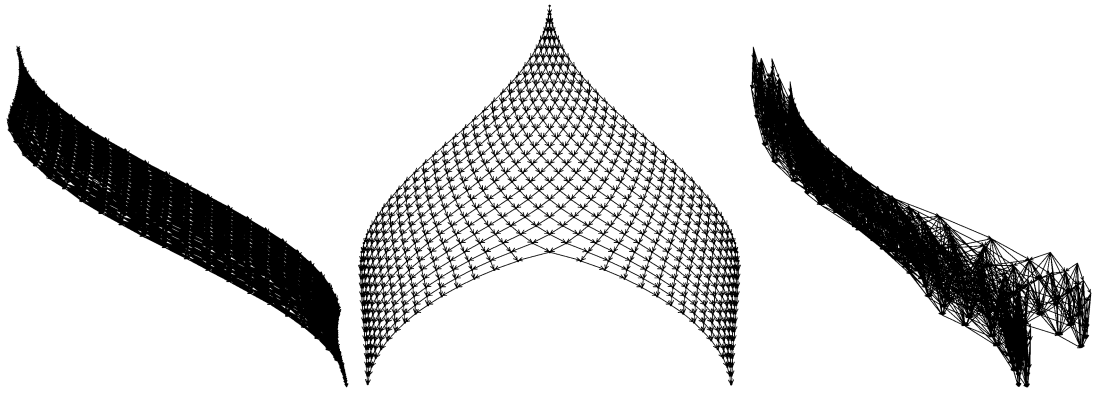

Fig. 6. Drawing the graphs of Fig. 4 using eigenprojection for the x-axis, but without imposing uncorrelation of the axes. Now in the Nos3 and Plat362 graphs the axes are significantly correlated, wasting much of the drawing area.

We conclude this section by illustrating the usefulness of the no-correlation requirement. In Fig. 6 we draw again the three digraphs of Fig. 4, but now we do not impose uncorrelation between the axes, hence when computing the x-coordinates we ignored the y-coordinates. We give the results only of the eigenprojection algorithm, but CMDS and PCA will show very similar behavior. It can be seen clearly that now the drawings of Nos3 and Plat362 are far less informative, as the two axes are strongly correlated wasting much of the drawing area. Regarding the Nos6, the result is still good, very similar to the one in Fig. 4. In this case, the x- and y-coordinates end up uncorrelated even without imposing it explicitly. Of course, in general we cannot expect such "lucky" cases and should enforce the uncorrelation.

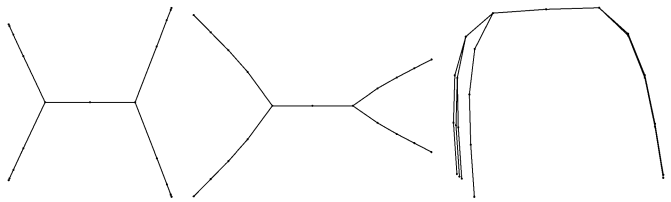

Fig. 7. Drawings of a depth 5 full binary tree by: (a) eigenprojection, (b) CMDS, (c) PCA.

4. One-dimensional stress minimization[2]

The three methods discussed in the previous section are characterized by relying on a global optimization that guarantees convergence to optimal solution. However, there is a price to pay for this convenience. In general, these methods are inferior to traditional force-directed methods in terms of drawing quality. For example, in many of the experiments we conducted, we noticed that the drawings obtained by these methods are often over-smooth, and hence they cannot faithfully describe abrupt changes in the graph structure. More specifically, the methods cannot deal with trees or with graphs containing loosely connected components. We provide two layouts demonstrating the shortcomings of the methods. For illustration we show 2-D layouts (both axes are computed using the same method), but we note that 1-D layouts share similar deficiencies. Fig. 7 shows drawings of a full binary tree of depth 5 on which the methods completely fail. Another graph, shown in Fig. 8, is Plsk1919, which (as we shall see in Fig. 9) comprises 3 natural components, for which all methods are only able to show the central larger component.

In this section, we show how the familiar method of Kamada and Kawai [17], which minimizes the function known as *the stress energy*,[3] can be used for 1-D graph drawing. In optimizing the stress energy we rely on its well studied aesthetical properties. Consequently, optimization of this energy produces more aesthetic results than the methods described in Section 3. To appreciate this, we show in Fig. 9 the drawings of the two graphs that were shown in Figs. 7–8, but which now utilize our algorithm for stress minimization. Clearly, the results are aesthetically superior. However, note that in terms of running time, the new algorithm cannot compete with the ultra-fast eigenprojection or PCA. Moreover, the stress energy is non-convex, and therefore finding its global minimizer is not guaranteed.

4.1. One-dimensional optimization process

Stress energy is a traditional measure of drawing quality, based on the heuristic that a nice drawing relates to good isometry: it calls for placing the nodes so that the resulting pairwise Euclidean distances will approach the corresponding target (graph-theoretical) distances. The concrete form of the energy is:

$$E(y) \stackrel{\text{def}}{=} \sum_{i<j} k_{ij}\big(|y_i - y_j| - d_{ij}\big)^2. \tag{6}$$

[2] This section is mostly taken from [22].

[3] Indeed, the stress energy was originally used for multidimensional scaling, where its name originated; see, e.g., [6,29].

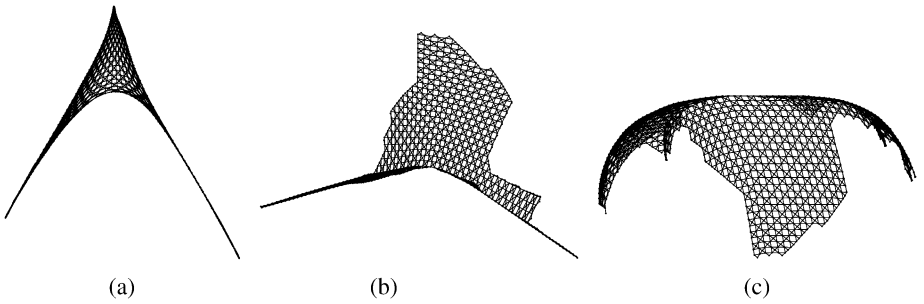

Fig. 8. Drawings of the Plsk1919 graph by: (a) eigenprojection, (b) CMDS, (c) PCA.

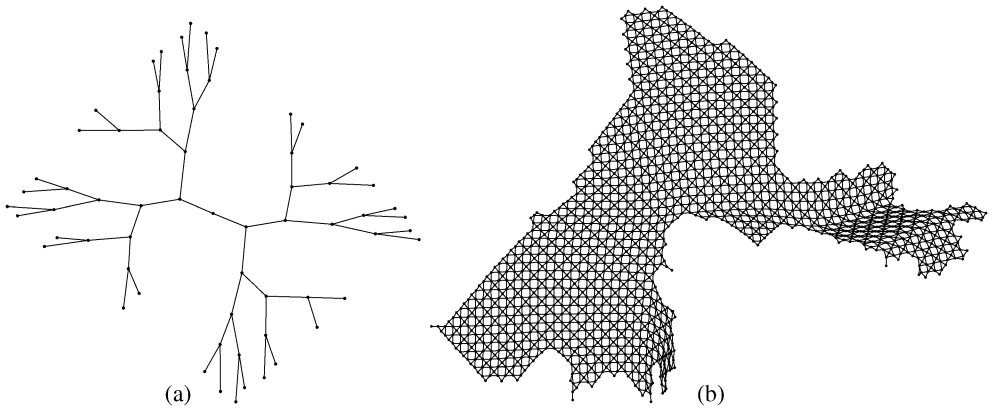

Fig. 9. Drawing by stress minimization of (a) depth 5 full binary tree, (b) the Plsk1919 graph.

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 $0 \leqslant \alpha \leqslant 2$. Kamada and Kawai [17] picked $\alpha = 2$, whereas Cohen [6] also considered $\alpha = 0$ and $\alpha = 1$. Moreover, Cohen suggested alternative target distances by setting d_{ij} to the linear-network distance. Throughout this paper, we worked only with the original choices of Kamada and Kawai; however, other choices can be directly incorporated into our formulations.

Given two 1-D layouts, $y, \tilde{y} \in \mathbb{R}^n$, we define the following family of auxiliary functions:

$$\delta_{ij}^{\tilde{y}}(y) = \begin{cases} y_i - y_j & \tilde{y}_i \geqslant \tilde{y}_j \\ y_j - y_i & \tilde{y}_i < \tilde{y}_j \end{cases} \qquad 1 \leqslant i < j \leqslant n. \tag{7}$$

Next, we define the following energy function of the layout y:

$$E^{\tilde{y}}(y) \stackrel{\text{def}}{=} \sum_{i<j} k_{ij} \big(\delta_{ij}^{\tilde{y}}(y) - d_{ij}\big)^2. \tag{8}$$

It is important to understand the relations between $E^{\tilde{y}}(y)$ and the stress energy, $E(y)$.

Lemma 2. *For every* $y, \tilde{y} \in \mathbb{R}^n$, $E(y) \leqslant E^{\tilde{y}}(y)$.

Proof. Let us pick some pair $i < j$, and analyze the corresponding terms of $E(y)$ and $E^{\tilde{y}}(y)$. Observe that $|\delta_{ij}^{\tilde{y}}(y)| = |y_i - y_j|$. In addition, it is always the case that $k_{ij}, d_{ij} \geqslant 0$. Therefore, $k_{ij}(\delta_{ij}^{\tilde{y}}(y) - d_{ij})^2 \geqslant k_{ij}(|y_i - y_j| - d_{ij})^2$. The lemma follows. □

Lemma 3. *For every* $y \in \mathbb{R}^n$, $E(y) = E^y(y)$.

Proof. Simply observe that $\delta_{ij}^{y}(y) = |y_i - y_j|$. □

By the last two lemmas we conclude:

Corollary 1. *For every* $y, \tilde{y} \in \mathbb{R}^n$, $E^y(y) \leqslant E^{\tilde{y}}(y)$.

The usefulness of the energy $E^{\tilde{y}}(y)$ stems from the fact that it can be minimized optimally. To realize this we need some additional notations. First, we define a related $n \times n$ Laplacian matrix $\mathcal{L}$, where

$$\mathcal{L}_{ij} = \begin{cases} -k_{ij} & i \neq j \\ \sum_{j \neq i} k_{ij} & i = j \end{cases} \quad i, j = 1, \ldots, n. \tag{9}$$

Note that the Laplacian depends only on the graph, regardless of its layout $\tilde{y}$. We also use the vector $b^{\tilde{y}} \in \mathbb{R}^n$, where:

$$b_i^{\tilde{y}} = \sum_{j \neq i:\ \tilde{y}_j \leqslant \tilde{y}_i} k_{ij} d_{ij} - \sum_{j \neq i:\ \tilde{y}_j > \tilde{y}_i} k_{ij} d_{ij}, \quad i = 1, \ldots, n. \tag{10}$$

Now, using some elementary algebra, it can be shown that:

$$E^{\tilde{y}}(y) = y^{\mathrm{T}} \mathcal{L} y - 2 y^{\mathrm{T}} b^{\tilde{y}} + C,$$

where C is a constant that is independent of y. Since the Laplacian is known to be positive semi-definite, we conclude by differentiation:

Lemma 4. *The minimizer of* $E^{\tilde{y}}(y)$ *is the solution of the system of equations*: $\mathcal{L} y = b^{\tilde{y}}$.

As mentioned in Section 2, for a connected underlying graph, the matrix $\mathcal{L}$ has a single zero eigenvalue that is associated with the eigenvector 1_n. Hence, the null-space of $\mathcal{L}$ is spanned by 1_n. Since addition of 1_n is equivalent to translation, the minimizer y is unique up to translation. To reduce this degree-of-freedom, we can assume that the first node is located at the origin (so $y_1 = 0$).

All these observations suggest the following process for minimizing the stress energy:

Function 1-D_stress_minimization $(G(V, E), y \in \mathbb{R}^n)$
  Compute the Laplacian $\mathcal{L}$
  do
    $\tilde{y} \leftarrow y$
    Compute $b^{\tilde{y}}$
    Compute y for which $\mathcal{L} y = b^{\tilde{y}}$
  while $(y \neq \tilde{y})$

We show that each iteration of this process (except for the last one) must decrease the stress energy: $E(y) < E(\tilde{y})$. This stems from the above lemmas: by Lemma 3, $E(\tilde{y}) = E^{\tilde{y}}(\tilde{y})$, and by Lemma 4, $E^{\tilde{y}}(y) < E^{\tilde{y}}(\tilde{y})$ (since $\tilde{y} \neq y$, the energy must strictly decrease). Now, by Corollary 1, $E^{y}(y) \leqslant E^{\tilde{y}}(y)$. Taken together, we obtain:

$$E(y) = E^{y}(y) \leqslant E^{\tilde{y}}(y) < E^{\tilde{y}}(\tilde{y}) = E(\tilde{y}).$$

Of course, the energy is bounded below by zero, so the process must converge. Unlike node-by-node local optimization methods, the new optimization process does not suffer from working in a single dimension. Later, we introduce some encouraging experimental results. Refer also to Appendix A, where we explain a rather significant advantage of our method over the node-by-node approach of Kamada and Kawai. However, note that like other optimization methods, we can only guarantee convergence to a local minimum.

Technical notes. As stated above, to make the minimizer unique, we remove the translation degree-of-freedom by assuming that $y_1 = 0$. Therefore, we can remove the first row and column of $\mathcal{L}$, as well as the first component of $b^{\tilde{y}}$, and solve an $(n-1) \times (n-1)$ system of equations. The resulting $(n-1) \times (n-1)$ matrix is positive definite. This is very convenient, since methods like Conjugate Gradient, Gauss Seidel, and Cholesky factorization are guaranteed to work [13].

4.2. Working with a sparse Laplacian

A significant drawback of stress optimization is that its space complexity is $\Theta(n^2)$, since we have to store all pairwise distances. This, of course, slows down running time, and even more importantly, it prevents us from dealing with large graphs containing more than around 10,000 nodes.

However, it appears that many graphs that can be drawn nicely possess much redundancy in the pairwise distances. We utilize this redundancy by neglecting most pairwise distances. Specifically, we define the set $\mathcal{S}$ of the "interesting" node pairs, so the stress energy is defined as:

$$\sum_{\{i,j\} \in \mathcal{S}} k_{ij} \big(|y_i - y_j| - d_{ij}\big)^2.$$

Accordingly, we can use a sparse Laplacian, whose nonzero entries correspond to the pairs in $\mathcal{S}$:

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

Similarly, the vector $b^{\tilde{y}}$ is defined as:

$$b_i^{\tilde{y}} = \sum_{\substack{j: \\ \{i,j\} \in \mathcal{S}, \tilde{y}_j \leqslant \tilde{y}_i}} k_{ij} d_{ij} - \sum_{\substack{j: \\ \{i,j\} \in \mathcal{S}, \tilde{y}_j > \tilde{y}_i}} k_{ij} d_{ij}, \quad i = 1, \ldots, n.$$

Strategies regarding the choice of pairs in $\mathcal{S}$ and more details on the sparse method are given in [22].

4.3. Working in the presence of predefined coordinates

The main application of 1-D graph drawing is where one axis of the drawing is given in advance. In Section 3 we suggested treating this issue by requiring the new axis to be uncorrelated with the given

one. Using our method to optimize the stress, we can directly handle such a requirement. It can be shown that stress minimization of y, accompanied by the constraint of uncorrelation with x, can be achieved by solving, in each iteration, the system:

$$(I - \hat{x}\hat{x}^{\mathrm{T}})\mathcal{L}(I - \hat{x}\hat{x}^{\mathrm{T}})y = (I - \hat{x}\hat{x}^{\mathrm{T}})b^{\tilde{y}},$$

where $\hat{x} = x/\|x\|$.

However, for stress minimization, we recommend another very effective way of taking care of the predefined axis. The stress minimization strategy strives to achieve the target distances in the drawing. Some fraction of the target distances is already achieved in the predefined axis, x. Hence, when computing a new axis we would like the achieve the *residual target distances* instead of the original ones. The new residual target distances are defined as:

$$d_{ij}^{x} = \begin{cases} \sqrt{d_{ij}^2 - (x_i - x_j)^2} & d_{ij} > |x_i - x_j| \\ 0 & \text{otherwise} \end{cases} \qquad i, j = 1, \ldots, n. \tag{11}$$

Note that there is no reason to alter the normalization weights k_{ij}.

However, there is still one problem remaining: the target distances set the scale of the layout. Therefore, it is possible that the predefined axis, x, has an entirely different scale (e.g., it may be very small), so our computation of residual target distances makes no sense. To overcome this problem, we re-scale x in order to bring it to the scale of the target distances (alternatively, we could re-scale the target distances in order to bring them to the scale of x). More specifically, we want to compute a constant $c > 0$ that minimizes the stress energy of the scaled $c \cdot x$. To find the exact value of c, we differentiate:

$$\frac{\partial}{\partial c} E(c \cdot x) = 0.$$

The solution is:

$$c = \frac{\sum_{\{i,j\}\in\mathcal{S}} k_{ij} d_{ij} |x_i - x_j|}{\sum_{\{i,j\}\in\mathcal{S}} k_{ij} (x_i - x_j)^2}. \tag{12}$$

To summarize, when an axis x is given in advance, we replace it with $c \cdot x$ to bring it into the scale of the target distances. The constant c is computed by (12). Then, we take into account the distances already captured by x, by computing the residual target distances, d_{ij}^{x}, as in (11). The additional axis y is computed in order to minimize the stress function $\sum_{\{i,j\}\in\mathcal{S}} k_{ij}(|y_i - y_j| - d_{ij}^{x})^2$.

4.4. Experimental results

Application to drawing digraphs. We have tested the capabilities of our algorithm by using it to compute the x-axis in the digraph drawing algorithm of [4]. As expected, our experiments show that in most cases using stress minimization improves the quality of the layouts compared to the results of the three methods described in Section 3. In Fig. 10 we show the stress-minimization-based layouts of the digraphs that appear in Fig. 4 using other methods. In all the layouts, the superiority of stress minimization is evident, especially in showing sharp changes in the graph structure. One of the most important advantages of stress-minimization is its ability to deal with tree or tree-like graphs. This is unlike the three methods of Section 3, which completely fail with trees. We demonstrate the advantage of stress minimization for drawing a rooted tree in Fig. 11.

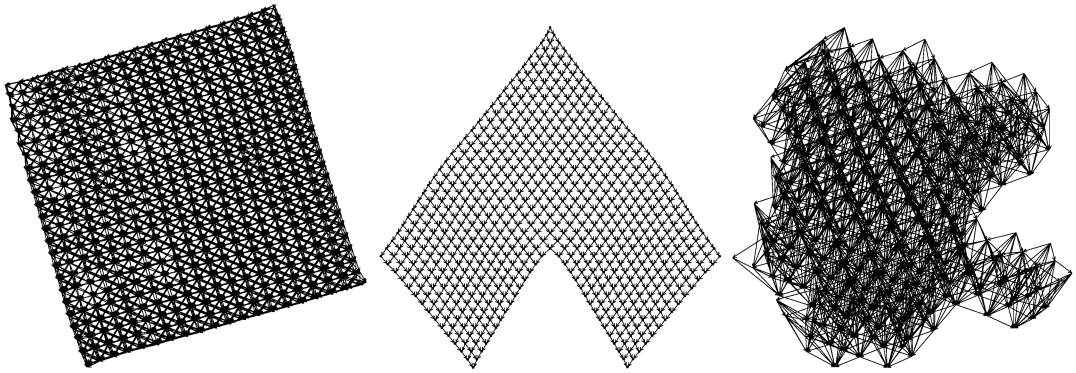

Fig. 10. Layouts of the digraphs: Nos3, Nos6 and Plat362. The x coordinates are computed by stress minimization and the y coordinates are the minimizer of the hierarchy energy.

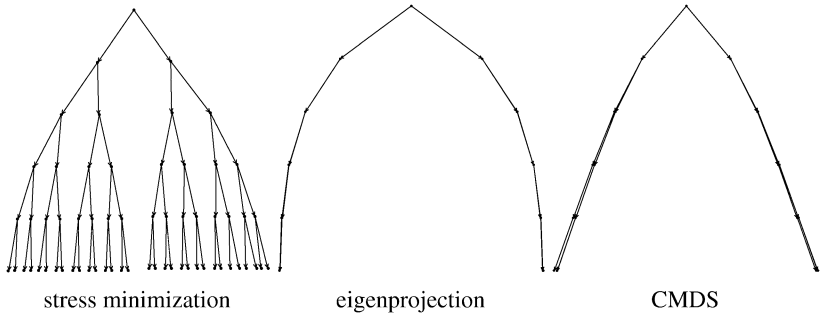

Fig. 11. Layout of a depth 5, full, rooted binary tree. In all layouts the y coordinates are the same minimizer of the hierarchy energy. However, the x coordinates were computed by three different methods: stress minimization, eigenprojection and classical-MDS.

4.4.1. Speed of computation for selected graphs

Smart initialization. We recommend initializing our algorithm with a smart (rather than random) placement of the nodes. Such an initialization could result from faster methods like eigenprojection or PCA. This way the probability of avoiding poor local minima is improved. Also, running time is significantly decreased as the number of iterations is reduced.

We initialized the algorithm by first running a few iterations of the method described in [23], which accelerates the stress minimization by constraining the layout to lie in a unique small subspace. This significantly reduced the number of iterations, and allowed us to run the sparse version of the algorithm with good results for all the given graphs.

To give an impression of actual performance, Table 2 shows the number of iterations and running time for several graphs. For each graph we have computed a 1-D layout, and stopped the iterative process when the layout stabilized. Our experience shows that the number of required iterations grows with graph size but also strongly depends on the graph's structure and the initial position. The overall running time, which includes the smart initialization, was measured on a Pentium IV 2GHz PC with 256MB RAM.

Table 2
Number of iterations and running time (in seconds) for various graphs

Graph name	$\|V\|$	$\|E\|$	Number of iterations	Time (sec.)
Torus 64×16	1024	2048	11	0.19
4970 [31]	4970	7400	19	1.7
Shuttle (Data) [33]	2851	15093	64	2.71
Crack [33]	10240	30380	41	9.64
Fidap006 [32]	1651	23914	49	1.65
Nos3 [32]	960	7442	20	0.36
Nos5 [32]	468	2352	18	0.13
Nos6 [32]	675	1290	10	0.1
Nos7 [32]	729	1944	15	0.13
Plat362 [32]	362	2712	9	0.08
Plat1919 [32]	1919	15240	17	0.62
Plsk1919 [32]	1919	4831	8	0.34

5. Discussion

We have explored one-dimensional graph drawing algorithms and have studied their special features and applications. One important application of these is in graph drawing by axis separation, where each axis is computed separately, so as to address specific aesthetical considerations. Since node-by-node local optimization in one dimension is a poor strategy for force-directed models, traditional graph drawing algorithms are not suitable for the 1-D drawing task, while less traditional algorithms are. We generalized three such algorithms using the unified paradigm of computing the layout axis-by-axis, while maintaining non-correlation with the precomputed coordinates.

As an alternative, we then introduced a novel process based on minimizing the so-called stress energy that strives to place nodes in accordance with target distances. Such a strategy was first introduced into graph drawing by Kamada and Kawai [17]. However, they use a localized 2-dimensional Newton–Raphson process, that tends to get stuck in poor local minima when used for computing 1-D layouts. We devised a rather global optimization process that replaces this usual local node-by-node optimization. Significantly, the algorithm is suitable for computing a one-dimensional layout. We prove in Appendix A that our process is equivalent to a full n-dimensional Newton–Raphson process, which clearly explains its advantage over [17].

None of the four methods developed in this paper possesses a definite advantage over the other methods. It turns out that some methods excel when dealing with large graphs, where computational efficiency becomes a critical issue. On the other hand some other methods are preferable for certain kinds of graphs, as they produce more readable layouts. In fact, the stress-based approach that usually produces most aesthetical results is also the less stable one as it involves a heuristic optimization process. Therefore, as usually is the case with graph drawing algorithms, we believe that all four methods should be considered as candidates when a 1-D graph layout is needed. The actual choice of a method might be based on comparing concrete layouts, on previous domain-specific experience and on computational limitations.

Appendix A. An alternative viewpoint to the stress minimization process

In this appendix we provide a new explanation of the stress minimization algorithm described in Section 4.1, which allows a direct comparison with the process suggested by Kamada and Kawai [17].

Given a 2-D stress energy $E(x, y)$ $(x, y \in \mathbb{R}^n)$ we could optimize it using the Newton–Raphson process as follows. Define the *gradient* $\nabla E(x, y) \in \mathbb{R}^{2n}$ as the vector of partial derivatives, i.e.,

$$\nabla E(\tilde{x}, \tilde{y}) = \begin{pmatrix} \frac{\partial E}{\partial x_1}(\tilde{x}, \tilde{y}) \\ \vdots \\ \frac{\partial E}{\partial x_n}(\tilde{x}, \tilde{y}) \\ \frac{\partial E}{\partial y_1}(\tilde{x}, \tilde{y}) \\ \vdots \\ \frac{\partial E}{\partial y_n}(\tilde{x}, \tilde{y}) \end{pmatrix}.$$

Also, define the $2n \times 2n$ *Hessian matrix* $H(x, y)$, which is the matrix of partial second derivatives, i.e.

$$H(x, y) = \begin{pmatrix} \frac{\partial^2 E}{\partial x_1^2}(\tilde{x}, \tilde{y}) & \dots & \frac{\partial^2 E}{\partial x_1 x_n}(\tilde{x}, \tilde{y}) & \frac{\partial^2 E}{\partial x_1 y_1}(\tilde{x}, \tilde{y}) & \dots & \frac{\partial^2 E}{\partial x_1 y_n}(\tilde{x}, \tilde{y}) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial y_1 x_1}(\tilde{x}, \tilde{y}) & \dots & \frac{\partial^2 E}{\partial y_1 x_n}(\tilde{x}, \tilde{y}) & \frac{\partial^2 E}{\partial y_1^2}(\tilde{x}, \tilde{y}) & \dots & \frac{\partial^2 E}{\partial y_1 y_n}(\tilde{x}, \tilde{y}) \end{pmatrix}.$$

The exact formulas for all these partial derivatives can be found in [17]. A single iteration of the Newton–Raphson process changes the current layout $\tilde{x}$, $\tilde{y}$ into a "better" layout x, y by solving the $2n \times 2n$ system of equations

$$H(\tilde{x}, \tilde{y}) \cdot \left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} \right) = -\nabla E(\tilde{x}, \tilde{y}).$$

However, working with such a full Newton–Raphson process is impractical in our case, since it requires a costly recomputation of the Hessian matrix in each iteration. Moreover, the Hessian might be singular or ill-conditioned which may cause additional numerical difficulties. Consequently, Kamada and Kawai have opted for a localized 2-D Newton–Raphson process. In this process all nodes except some node i are fixed, so only two variables remain: x_i and y_i. The new coordinates of node i are computed by solving a tiny system of equations involving the corresponding 2×2 Hessian.

Now, let us consider the 1-D case in which the layout is defined by $x \in \mathbb{R}^n$. We use the definition of $E(x)$ in (6), and by a simple derivation we conclude that

$$\left(\nabla E(\tilde{x})\right)_i = 2 \sum_{j \neq i} k_{ij}(x_i - x_j) - 2b_i^{\tilde{x}},$$

where $b^{\tilde{x}}$ is defined in (10).

Moreover, regardless of the value of $\tilde{x}$, the Hessian is nothing but $2\mathcal{L}$, where $\mathcal{L}$ is the Laplacian defined in (9). Consequently, a single iteration of Newton–Raphson optimization is based on solving the linear system

$$\mathcal{L}x = \mathcal{L}\tilde{x} - \frac{1}{2}\nabla E(\tilde{x}).$$

Let us consider the ith row of this system

$$(\mathcal{L}x)_i = (\mathcal{L}\tilde{x})_i - \sum_{j \neq i} k_{ij}(x_i - x_j) + b_i^{\tilde{x}}.$$

We can simplify this equation by observing that $(\mathcal{L}\tilde{x})_i = \sum_{j \neq i} k_{ij}(x_i - x_j)$. Consequently we get that the ith equation is

$$(\mathcal{L}x)_i = b_i^{\tilde{x}}.$$

Thus, in each iteration we are solving the system $\mathcal{L}x = b^{\tilde{x}}$. This is exactly what happens in our function 1-D_ stress_minimization. Therefore our process is equivalent to performing full n-dimensional Newton–Raphson optimization, while Kamada and Kawai perform only a 2-D localized Newton–Raphson optimization. Naturally, this explains why our process converges better than the one of Kamada and Kawai when working in 1-D.

Note that the two shortcomings of the Newton–Raphson method that prevent its use for 2-D layouts, do not exist in the 1-D case. First, we do not have to recompute the Hessian in each iteration since it is constant and independent of the current layout. Second, the Hessian matrix is positive definite (and certainly not singular) after removing the translation degree of freedom as explained in the technical notes in Section 4.1. Consequently, solving the system of equations is very convenient from a numerical viewpoint.

References

[1] B. Beckman, Theory of spectral graph layout, Technical Report MSR-TR-94-04, Microsoft Research, 1994.
[2] U. Brandes, S. Cornelsen, Visual ranking of link structures, J. Graph Algorithms Appl. 7 (2003) 181–201.
[3] I. Bruss, A. Frick, Fast interactive 3-D graph visualization, in: Proc. 3rd Inter. Symposium on Graph Drawing (GD'95), in: Lecture Notes Comput. Sci., vol. 1027, Springer, Berlin, 1996, pp. 99–110.
[4] L. Carmel, D. Harel, Y. Koren, Combining hierarchy and energy for drawing directed graphs, IEEE Trans. Vis. Comput. Graph. 10 (2004) 46–57.
[5] L. Carmel, Y. Koren, D. Harel, Visualizing and classifying odors using a similarity matrix, in: Proc. 9th International Symposium on Olfaction and Electronic Nose (ISOEN'02), Aracne, 2003, pp. 141–146.
[6] J.D. Cohen, Drawing graphs to convey proximity: an incremental arrangement method, ACM Trans. Computer-Human Interact. 4 (1997) 197–229.
[7] J. Diaz, J. Petit, M. Serna, A survey on graph layout problems, ACM Comput. Surv. 34 (2002) 313–356.
[8] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, Graph Drawing: Algorithms for the Visualization of Graphs, Prentice-Hall, Englewood Cliffs, NJ, 1999.
[9] P. Eades, A heuristic for graph drawing, Congressus Numerantium 42 (1984) 149–160.
[10] B.S. Everitt, G. Dunn, Applied Multivariate Data Analysis, Arnold, 1991.
[11] T.M.G. Fruchterman, E. Reingold, Graph drawing by force-directed placement, Software-Practice Exp. 21 (1991) 1129–1164.
[12] P. Gajer, M.T. Goodrich, S.G. Kobourov, A multi-dimensional approach to force-directed layouts of large graphs, in: Proc. 8th Inter. Symposium on Graph Drawing (GD'00), in: Lecture Notes Comput. Sci., vol. 1984, Springer, Berlin, 2000, pp. 211–221.
[13] G.H. Golub, C.F. Van Loan, Matrix Computations, Johns Hopkins University Press, 1996.
[14] K.M. Hall, An r-dimensional quadratic placement algorithm, Management Sci. 17 (1970) 219–229.
[15] D. Harel, Y. Koren, Graph drawing by high-dimensional embedding, in: Proc. 10th Inter. Symposium on Graph Drawing (GD'02), in: Lecture Notes Comput. Sci., vol. 2528, Springer, Berlin, 2002, pp. 207–219.
[16] M. Juvan, B. Mohar, Optimal linear labelings and eigenvalues of graphs, Discrete Appl. Math. 36 (1992) 153–168.

[17] T. Kamada, S. Kawai, An algorithm for drawing general undirected graphs, Inform. Process. Lett. 31 (1989) 7–15.
[18] M. Kaufmann, D. Wagner (Eds.), Drawing Graphs: Methods and Models, Lecture Notes Comput. Sci., vol. 2025, Springer, Berlin, 2001.
[19] Y. Koren, L. Carmel, D. Harel, Drawing huge graphs by algebraic multigrid optimization, Multiscale Modeling and Simulation 1 (2003) 645–673.
[20] Y. Koren, D. Harel, A multi-scale algorithm for the linear arrangement problem, in: Proc. 28th Inter. Workshop on Graph-Theoretic Concepts in Computer Science (WG'02), in: Lecture Notes Comput. Sci., vol. 2573, Springer, Berlin, 2002, pp. 293–306.
[21] Y. Koren, D. Harel, A two-way visualization method for clustered data, in: Proc. 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03), ACM Press, New York, 2003, pp. 589–594.
[22] Y. Koren, D. Harel, Axis-by-axis stress minimization, in: Proc. 11th Inter. Symposium on Graph Drawing (GD'03), Springer, Berlin, 2003, pp. 450–459.
[23] Y. Koren, Graph drawing by subspace optimization, in: Proc. 6th Eurographics—IEEE TCVG Symposia on Visualization (VisSym'04), Eurographics, 2004, pp. 65–74.
[24] J. Kruskal, J. Seery, Designing network diagrams, in: Proc. First General Conference on Social Graphics, US Department of the Census, 1980, pp. 22–50.
[25] B. Lehoucq, D.C. Sorensen, C. Yang, ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods, SIAM, 1998.
[26] A.J. McAllister, A new heuristic algorithm for the linear arrangement problem, Technical Report 99_126a, Faculty of Computer Science, University of New Brunswick, 1999.
[27] B. Mohar, The Laplacian spectrum of graphs, Graph Theory Combin. Appl. 2 (1991) 871–898.
[28] K. Sugiyama, S. Tagawa, M. Toda, Methods for visual understanding of hierarchical systems, IEEE Trans. Systems Man Cybernet. 11 (1981) 109–125.
[29] J.W. Sammon, A nonlinear mapping for data structure analysis, IEEE Trans. Comput. 18 (1969) 401–409.
[30] D. Tunkelang, A numerical optimization approach to general graph drawing, Ph.D. Thesis, Carnegie Mellon University, 1999.
[31] C. Walshaw, A multilevel algorithm for force-directed graph drawing, J. Graph Algorithms Appl. 7 (2003) 253–285.
[32] The Matrix Market collection, http://math.nist.gov/MatrixMarket.
[33] The University of Greenwich Graph Partitioning Archive, http://www.gre.ac.uk/~c.walshaw/partition.