Graph Drawing by Stress Majorization

Conference Paper *in* Lecture Notes in Computer Science · September 2004

DOI: 10.1007/978-3-540-31843-9_25

3 authors, including:

Emden Gansner
self
96 PUBLICATIONS **6,892** CITATIONS
SEE PROFILE

Stephen C. North
Infovisible
100 PUBLICATIONS **6,799** CITATIONS
SEE PROFILE

Graph Drawing by Stress Majorization

Emden R. Gansner, Yehuda Koren and Stephen North

AT&T Labs — Research
Florham Park, NJ 07932
{erg,yehuda,north}@research.att.com

Abstract. One of the most popular graph drawing methods is based of achieving graph-theoretic target ditsances. This method was used by Kamada and Kawai [15], who formulated it as an energy optimization problem. Their energy is known in the multidimensional scaling (MDS) community as *the stress function*. In this work, we show how to draw graphs by stress majorization, adapting a technique known in the MDS community for more than two decades. It appears that majorization has advantages over the technique of Kamada and Kawai in running time and stability. We also present a few extensions to the basic energy model which can improve layout quality and computation speed in practice. Majorization-based optimization is essential to these extensions.

1 Introduction

A graph is a structure $G(V{=}\{1,\ldots,n\},E)$ representing a binary relation E over a set of nodes V. Visualizing graphs is a challenging problem, requiring algorithms that faithfully represent the graph's structure and the relative similarities of the nodes [4, 16]. Here we will focus on drawing undirected graphs with straight-line edges.

The most popular approach defines an energy, or a cost function, sometimes implicitly, based on some virtual physical model of the graph. Minimizing this function determines an optimal drawing. In the approach considered here, originally proposed by Kamada and Kawai[15], a nice drawing relates to good isometry. We have an ideal distance d_{ij} given for every pair of nodes i and j,modeled as a springs. Given a 2-D layout, where node i is placed at point X_i, the energy of the system is

$$\sum_{i<j} w_{ij} \left(\|X_i - X_j\| - d_{ij}\right)^2 . \tag{1}$$

We desire a layout that will minimize this function, thereby best approximating the target distances. Here, the distance d_{ij} is typically the graph-theoretical distance between nodes i and j. The normalization constant w_{ij} equals $d_{ij}^{-\alpha}$. Kamada and Kawai [15] chose $\alpha = 2$, whereas Cohen [6] also considered $\alpha = 0$ and $\alpha = 1$. Moreover, Cohen suggested setting d_{ij} to the linear-network distance to convey the clustering structure of the graph.

The function (1), with $\alpha = 0$, appeared earlier, as the stress function in multidimensional scaling (MDS) [5, 6, 17]. Whereas Kamada and Kawai proposed a localized 2-D Newton-Raphson process for minimizing the stress function, several researchers in the MDS field have proposed a different, more global approach called *majorization*. Majorization seems to offer some distinct advantages over localized processes like Newton-Raphson or gradient descent. These include guaranteed monotonic decrease of the energy value, improved robustness against local minima and shorter running times. The main contribution of this work is the introduction of this technique in the framework of graph layout.

We also describe three useful extensions to stress optimization which require the power and flexibility of majorization optimization. The first extension, described in Section 3, deals with weighting edge lengths in a way that better utilizes the drawing area, and is especially useful for drawing real-life graphs whose degree distribution follows a power law. We have found empirically that traditional stress optimization is unstable under such a weighting,

while majorization works very well. The second extension, described in Section 4, deals with sparse stress functions, where only a small fraction of all pairwise distances are considered. This is essential for reducing the time and space complexity of stress optimization, and allows in-core layout of much larger graphs. We have found that sparse stress optimization is practically impossible when using the Kamada-Kawai technique (unless one has a very good initialization). Again, with majorization, it is easy to work with sparse models.

The last extension, described in Section 5, deals with obtaining an approximate drawing of the graph by constraining the layout axes to lie within a carefully selected small vector space. Such a technique was recently introduced by Koren[14] and can be integrated into layout algorithms based on matrix algebra. Fortunately, the algebraic nature of the majorization process allows us to perform rapid subspace-restricted stress minimization.

2 Stress Majorization

In this section we briefly summarize stress majorization as described in the MDS literature [3, 5].

We denote a d-dimensional layout by an $n \times d$ matrix X. Thus, node i is located at $X_i \in \mathbb{R}^d$ and the axes of the layout are $X^{(1)}, \ldots, X^{(d)} \in \mathbb{R}^n$. The associated stress function is

$$\mathrm{stress}(X) = \sum_{i<j} w_{ij} \left(\|X_i - X_j\| - d_{ij} \right)^2 . \tag{2}$$

We always take $w_{ij} = d_{ij}^{-2}$, which seems to produce the best drawings in most cases. Decompose (2) to obtain

$$\mathrm{stress}(X) = \sum_{i<j} w_{ij} d_{ij}^2 + \sum_{i<j} w_{ij} \|X_i - X_j\|^2 - 2 \sum_{i<j} \delta_{ij} \|X_i - X_j\| , \tag{3}$$

where $\delta_{ij} \stackrel{\mathrm{def}}{=} w_{ij} d_{ij} \quad i, j = 1, \ldots, n$.

The first term of (3), $\sum_{i<j} w_{ij} d_{ij}^2$, is a constant independent of the current layout. The second term, $\sum_{i<j} w_{ij} \|X_i - X_j\|^2$, is a quadratic sum, and can be written using the quadratic form of the weighted Laplacian L^w

$$\sum_{i<j} w_{ij} \|X_i - X_j\|^2 = \mathrm{Tr}(X^T L^w X) , \tag{4}$$

where the $n \times n$ weighted Laplacian is defined as

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

for $i, j = 1, \ldots, n$.

The third term, $\sum_{i<j} \delta_{ij} \|X_i - X_j\|$, is more involved and we will bound it from below. We will make use of the Cauchy-Schwartz inequality

$$\|x\| \|y\| \geqslant x^T y$$

with equality when $x = y$. Consequently, given any $n \times d$ matrix Z,

$$\|X_i - X_j\| \|Z_i - Z_j\| \geqslant (X_i - X_j)^T (X_i - Z_j)$$

with equality when $X = Z$. We can now bound the third term as follows

$$\sum_{i<j} \delta_{ij} \|X_i - X_j\| \geqslant \sum_{i<j} \delta_{ij} \mathrm{inv}(\|Z_i - Z_j\|) (X_i - X_j)^T (Z_i - Z_j) \tag{5}$$

where $\mathrm{inv}(x) = \begin{cases} 1/x & x \neq 0 \\ 0 & x = 0 \,. \end{cases}$

Inequality (5) can be written in a more convenient matrix form

$$\sum_{i<j} \delta_{ij} \|X_i - X_j\| \geqslant \mathrm{Tr}(X^T L^Z Z) \,,$$

where the $n \times n$ matrix L^Z is defined as

$$L^Z_{i,j} = \begin{cases} -\delta_{ij} \mathrm{inv}(\|Z_i - Z_j\|) & i \neq j \\ -\sum_{j \neq i} L^Z_{i,j} & i = j \end{cases}$$

for $i, j = 1, \ldots, n$,

Combining all the above, we can bound the stress function using $F^Z(X)$ defined as

$$F^Z(X) = \sum_{i<j} w_{ij} d_{ij}^2 + \mathrm{Tr}(X^T L^w X) - 2\mathrm{Tr}(X^T L^Z Z) \tag{6}$$

So

$$\mathrm{stress}(X) \leqslant F^Z(X) \tag{7}$$

with equality when $Z = X$.

Note that Z is a constant $n \times d$ matrix. This way we have bounded the stress with a quadratic form $F^Z(X)$. We differentiate by X and find that the minima of $F^Z(X)$ are given by solving

$$L^w X = L^Z Z \,.$$

Or, equivalently, for each axis we have to solve

$$L^w X^{(a)} = L^Z Z^{(a)}, \quad a = 1, \ldots, d \,. \tag{8}$$

The characteristic of the minima is determined by the nature of the weighted Laplacian L^w, which is known to be positive semi-definite with a one-dimensional null space spanned by $1_n = (1, \ldots, 1) \in \mathbb{R}^n$. Hence, $F^Z(X)$ has only global minima, which are invariant under translation (addition of $\alpha \cdot 1_n$ is equivalent to translation). This makes sense, since the stress function is also invariant under translation.

Numerically, it is better to make the minimizer unique. Hence we recommend removing the translation degree-of-freedom by taking $X_1 = 0$. Therefore, we can remove the first row and column of L^w, as well as the first row of $L^Z Z$. The resulting $(n-1) \times (n-1)$ matrix is strictly diagonal dominant and hence positive-definite. This is very convenient, since methods like Conjugate Gradient, Gauss-Seidel, and Cholesky factorization are guaranteed to work [9].

The optimization process

The above formulation leads to the following iterative optimization process. Given some layout $X(t)$, we want to compute a layout $X(t+1)$ so that $\mathrm{stress}(X(t+1)) < \mathrm{stress}(X(t))$. We use the function $F^{X(t)}(X)$ that satisfies $F^{X(t)}(X(t)) = \mathrm{stress}(X(t))$.

We take $X(t+1)$ as the minimizer of $F^{X(t)}(X)$ by solving

$$L^w X(t+1)^{(a)} = L^{X(t)} X(t)^{(a)}, \quad a = 1, \ldots, d \,. \tag{9}$$

At this point, if $X(t+1) = X(t)$, we terminate the process. Otherwise, we get

$$\mathrm{stress}(X(t+1)) \leqslant F^{X(t)}(X(t+1)) < F^{X(t)}(X(t)) = \mathrm{stress}(X(t)) \,.$$

The first inequality is by (7) and the second inequality is by the uniqueness of the minimum.

In practice we terminate the process when

$$\frac{\text{stress}(X(t)) - \text{stress}(X(t+1))}{\text{stress}(X(t))} < \epsilon\,, \tag{10}$$

where ϵ is the tolerance of the process. Typically, $\epsilon \sim 10^{-4}$.

To summarize, the majorization process involves iteratively solving (9). The matrix L^w is constant throughout the entire process, whereas the matrix $L^{X(t)}$ would be re-computed at each iteration.

2.1 Equation solvers

In practice we recommend using either Cholesky factorization or Conjugate Gradient (CG) [9] to solve (9) (by first fixing $X_1 = 0$ as discussed above). Using Cholesky factorization implies that at a preprocessing stage we find the LL^T factorization of L^w using $n^3/3$ flops (floating point operations). Then in each iteration we solve the linear system using back substitution in time $O(n^2)$. Hence, the significant cost in Cholesky factorization is independent of the number of iterations, making it is suitable for graphs requiring many iterations of process (9).

On the other hand, CG optimization involves no preprocessing and its running time is evenly distributed among the iterations. Almost the entire solving time is devoted to performing matrix-vector multiplication. Each such multiplication takes n^2 flops. Thus, if the total number of matrix multiplications is less than about $n/3$, the CG process is expected to be faster than Cholesky factorization. Otherwise, Cholesky factorization is recommended. In practice, for most graphs we have experimented with, CG outperformed Cholesky since the total number of matrix-vector multiplications is typically less than $n/3$. Note that CG benefits by the fact that we have an initial approximate solution from the previous iteration. However, we observed that the overall number of iterations increases very moderately with the size of the graph. Therefore, for large graphs (over 10,000 nodes), we encountered cases where the total number of matrix-vector multiplications exceeded even n, so Cholesky factorization should do much better. In any case, all the results reported here employ CG.

2.2 Intuitive interpretation

Let us concentrate on axis a, and denote the current coordinates by $\hat{x} = X(t)^{(a)}$. The majorization process determines the new coordinates $x = X(t+1)^{(a)}$ by solving the system of equations (9). Eliminating x_i in equation i, we rewrite the system in an equivalent form

$$x_i = \frac{\sum_{j \neq i} w_{ij}\,(x_j + d_{ij}(\hat{x}_i - \hat{x}_j)\text{inv}(\|X(t)_i - X(t)_j\|))}{\sum_{j \neq i} w_{ij}}\,. \tag{11}$$

The intuitive interpretation of this process is simple. A node j located at x_j strives to place node i (on current axis a) at $x_j + d_{ij}\frac{\hat{x}_i - \hat{x}_j}{\|X(t)_i - X(t)_j\|}$.

Based on the current placement, this is node j's best strategy to assure that node i will be at distance d_{ij} from j in the full multidimensional layout. To see this, notice that the distance between the nodes depends on all the axes. Therefore, node j's estimate of the contribution of axis a for the distance between i and j is the fraction $\alpha = |\frac{\hat{x}_i - \hat{x}_j}{\|X(t)_i - X(t)_j\|}|$. So the magnitude of displacement should be d_{ij} scaled-down by α. Now, after deciding the magnitude of the 1-D displacement, the direction must be decided: should we place x_i at $x_j + \alpha d_{ij}$ or at $x_j - \alpha d_{ij}$? Again, the decision is based on the current placement, whether currently $\hat{x}_i < \hat{x}_j$ or vice versa.

This way, each node j votes for its desired placement of x_i. The final position is determined by taking the weighted average of the suggested positions. This intuition also suggests a localized optimization process, which we next describe.

2.3 Localized optimization

Following the idea of Kamada and Kawai [15], we can fix the positions of all nodes, except some node i. Then, by the same argument given above for the full majorization process, it can be shown that the stress function is decreased by setting the position of i as follows

$$X_i^{(a)} \leftarrow \frac{\sum_{j \neq i} w_{ij} \left(X_j^{(a)} + d_{ij}(X_i^{(a)} - X_j^{(a)}) \mathrm{inv}(\|X_i - X_j\|) \right)}{\sum_{j \neq i} w_{ij}}, \quad a = 1, \ldots, d. \quad (12)$$

This way we can iterate through all nodes, and in each iteration relocate all the d coordinates of node i according to (12). Each iteration is guaranteed to strictly decrease the stress until convergence. Hence, oscillations and non-convergence are impossible.

In practice, we have only used the more involved global process (9) and have no experience yet with the local version. We provide this local version here mainly because it is simple and easy to implement, requiring no equation solver.[1]

2.4 Comparisons

A natural question is whether we should replace the traditional Kamada-Kawai based optimization with majorization. Based on several months of experimenting with both approaches, our definite answer is yes. We base this recommendation on several considerations.

We experimented with various example graphs. On each graph, we ran each of the two algorithms 25 times with different random initializations. At certain times during each execution, we measured the elapsed running time and the current value of the stress function, and averaged over all 25 executions. From this we obtained stress-vs.-time charts for the graphs. While it is impossible to present here all of the charts, we show a few representative ones in Figures 1-3. We can make some important observations.

Layout quality We observed that most of the time, the two methods eventually achieved about the same stress level. In certain cases, the Kamada-Kawai approach would yield a slightly better layout in terms of the stress value, but the difference was always small; see Figure 2. However, in other cases, the majorization approach yielded significantly better layouts as can be seen in Figure 3. Hence, probably due to its more global nature, majorization can be considered better in terms of layout quality.

Monotonicity of convergence A significant advantage of majorization is that iterations monotonically decrease the stress until convergence. This way, termination of the process is determined naturally by a condition like (10). However, our experience with the Kamada-Kawai approach, as implemented in Neato [7], shows that in some cases the latter process may cycle without converging, while the energy is oscillating. This requires an artificial or more convoluted termination condition.

Our experiments show that, as expected, the majorization approach was always monotonic in decreasing the stress value. The non-monotonicity problem of the Kamada-Kawai method was extremely rare (remember that we averaged over 25 executions, lessening the impact of a single bad non-monotonic execution). However, we observed this non-monotonic behavior, when experimenting with the Qh882 graph [1]. The result is provided in Fig. 1 that compares the average behavior of both approaches on this graph. We should note that here we weighted edges as explained in Section 3. The reader can see that after 2 seconds of running, the stress value in the Kamada-Kawai approach increases for some period. Here, this did not prevent it from converging at about the same stress level as the majorization process.

Running time The running time of the majorization process is consistently less than that of the Kamada-Kawai process. In all runs, it can be observed that majorization reaches the low stress level much before Kamada-Kawai.

[1] Process (12) should not be confused with the similar Gauss-Seidel process that can be used to solve (9).

A partial explanation is that majorization's running time is dominated by matrix operations (matrix-vector multiplication or Cholesky factorization). These operations are implemented in libraries like BLAS and LAPACK that are highly optimized on the machine instruction level for common platforms. We are using the Intel Math Kernel Library [21]; another well-known implementation is Atlas [22].

For implementations not relying on special matrix software, we found the situation to be similar to that of the stress function. Sometimes the Kamada-Kawai approach would be marginally faster; on the other hand, when the majorization process was faster, it was significantly faster. And as the size of the graphs increased, the advantage swung completely to majorization.

Extensions In the following sections, we will show some useful extensions to the basic stress model. When using these extensions, the advantage of majorization is even more decisive.

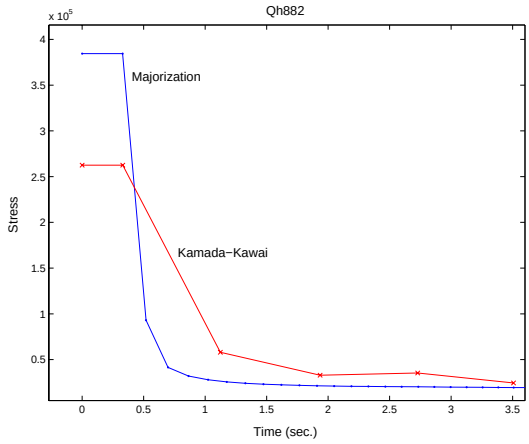


Fig. 1. Stress function vs. running time for the graph Qh882 [1] (|V|=882, |E|=1533). Here both methods reached about the same stress. Interestingly, Kamada-Kawai is not monotonic.

Before leaving this topic, we must point out that our implementation of the Kamada and Kawai process on which we based our comparisons differs slightly from the implementation originally suggested [15]. We are using the more common implementation which replaces the two nested loops with a single loop; see [2, 11]. As noted in Brandenburg, Himsolt, and Rohrer [2], this leads to a significant speed-up over the original implementation. This more efficient implementation is also the one used in Neato [7] and GraphLet [20].

3 Weighting Edge Lengths

In many real life graphs, the degree distribution decays at a much lower rate than in random graphs. Usually this distribution follows a power law and is proportional to $d^{-\lambda}$. However, setting desired edge lengths to a uniform length (typically 1), inevitably makes the neighborhood of high degree nodes too dense in the layout. Consequently, we suggest weighting edges by their neighborhood size.

Specifically, we set the length of each edge $\langle i, j\rangle \in E$ as

$$l_{ij} = |N_i \cup N_j| - |N_i \cap N_j|\,, \tag{13}$$

where $N_i = \{j | \langle i, j\rangle \in E\}$. Then, each target distance d_{ij} is the length of the shortest weighted path between i and j.

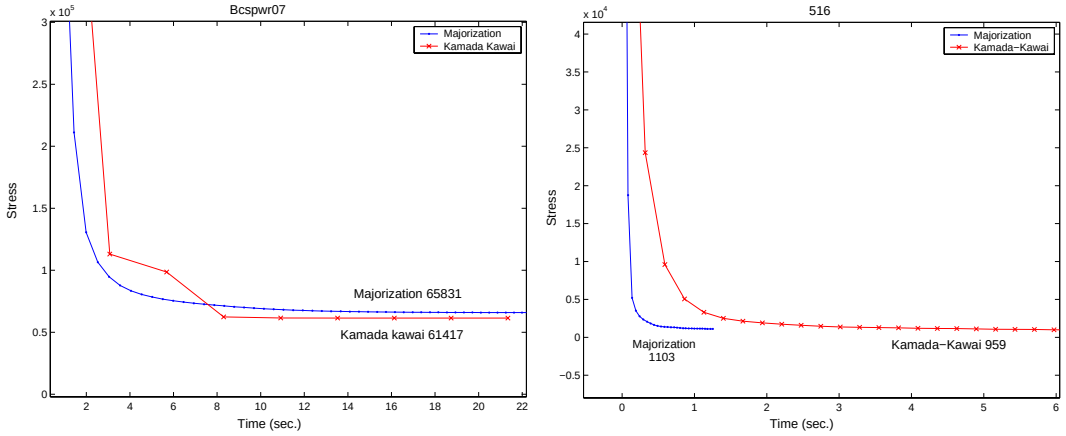


Fig. 2. Stress function vs. running time for the graphs Bcspwr07 [1] (|V|=882, |E|=1533) and 516 [18] (|V|=516, |E|=729).

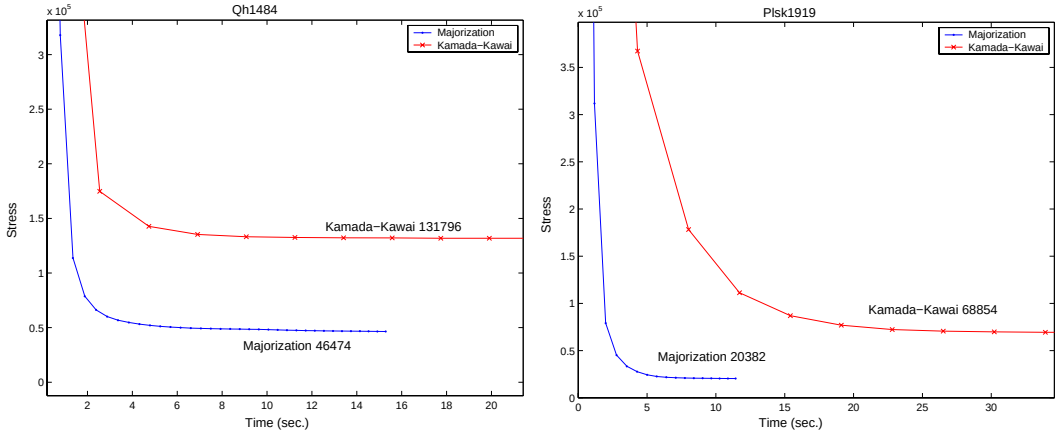


Fig. 3. Stress function vs. running time for the graphs Qh1484 [1] (|V|=1470, |E|=6420) and Plsk1919 [1] (|V|=1919, |E|=4831).

This simple change is surprisingly effective in many real life irregular graphs that have highly non-uniform degree distributions. We present here two examples. The first example is the 1138Bus graph (|V|=1138, |E|=1458) from the Matrix Market repository [1]. This graph models a network of high-voltage power distribution lines. Figure 4 shows two layouts of this graph. In one layout, edges were weighted according to (13). The other layout was made with unweighted edges. Nodes are much better dispersed in the weighted-edge-based layout. By weighting edges, more space is allocated to the dense areas avoiding many of the edge crossings.

Another interesting example is a BGP connectivity graph representing communications between autonomous systems (|V|=3847, |E|=11539). This graph has a few nodes of high degree (e.g., one node is degree 695 and a few others are around 100), as well as 3257 nodes of degree 1. We show two layouts of this graph in Figure 5. Again, it is clear that when weighting edges, the resulting layout is much more informative. For example, in both layouts the central node is the one of degree 695. In the weighted version, its neighborhood is placed

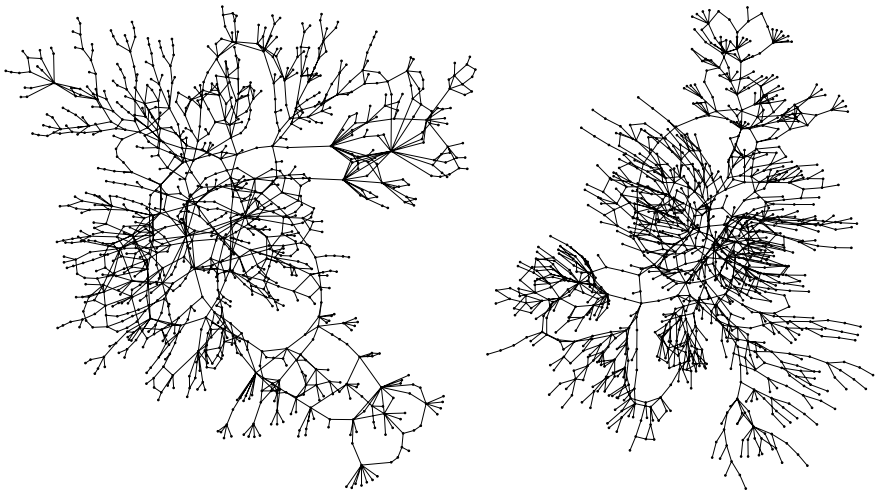


Fig. 4. Two layouts of the 1138Bus graph [1]

far enough from it to make it fairly visible. However, in the unweighted version, all of its neighbors are positioned densely around it, hiding its structure completely.

We have frequently found that when there are large deviations in edge lengths, as in the BGP graph, classic Kamada-Kawai optimization fails to find a nice layout. The result of Kamada-Kawai optimization on the edge-weighted BGP graph is shown in Figure 6(a). It is clearly inferior to the majorization result shown in Figure 5. We also compare the average stress-vs.-time behavior of the two methods in Figure 6(b), where it is clear the Kamada-Kawai-type optimization is pretty helpless here. Although do not fully understand this limitation of Kamada-Kawai optimization, it seems that its local nature somehow limits its ability to deal with significantly unbalanced edge lengths.

4 Sparse Stress Functions

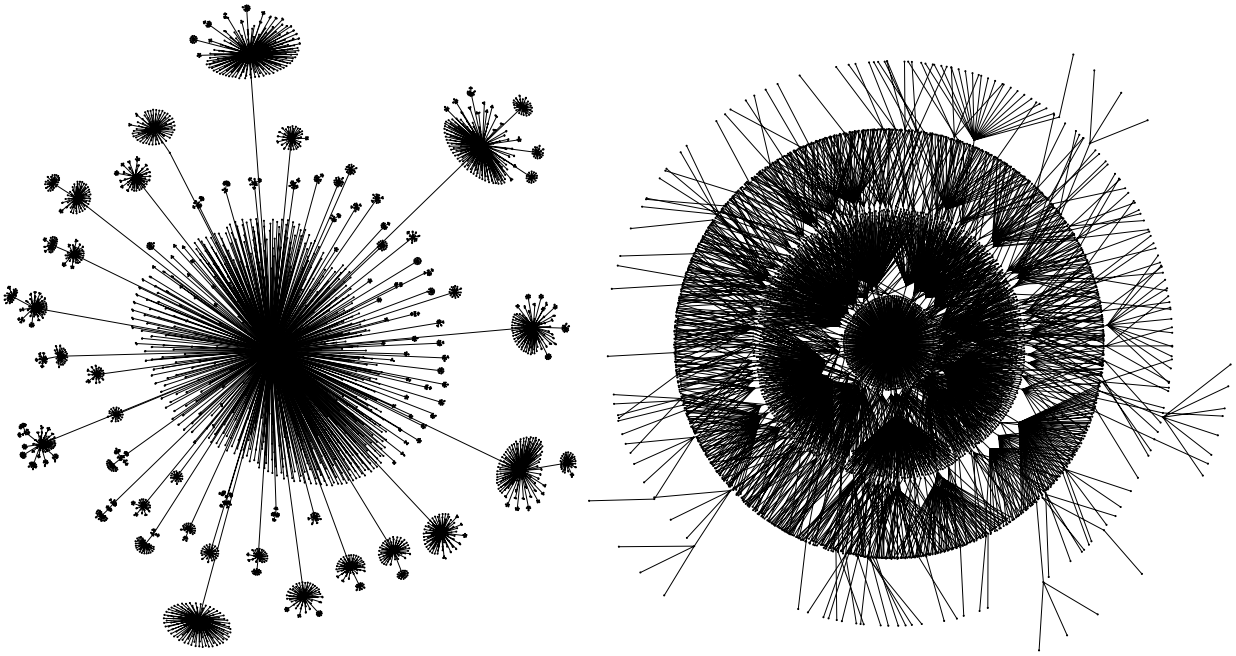
The full stress function (2) addresses all $n(n-1)/2$ pairwise distances. Its inherent quadratic complexity limits its practicality on large graphs. In fact, our implementation requires $2n^2 + O(n)$ bytes of memory, meaning that in a conventional 32-bit address space, we cannot even represent models of graphs with more than about 25,000 nodes.

A way to overcome this complexity is to consider sparser versions of the energy model which address far fewer relations than the full model. Specifically, we can define the sparse stress function as

$$\text{stress}^{\mathcal{S}}(X) = \sum_{\{i,j\}\in\mathcal{S}} w_{ij} \left(\|x_i - x_j\| - d_{ij}\right)^2 . \tag{14}$$

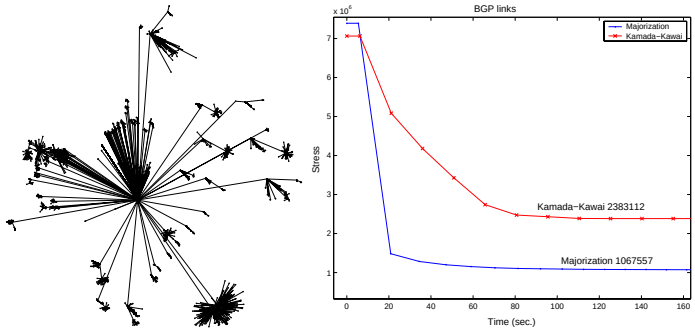
The set $\mathcal{S}$ defines the pairs that explicitly contribute to the total energy. For many graphs there is enough redundancy in the pairwise distances that by addressing a portion of the pairs, all pairwise distances will be preserved.

We follow the construction of Koren and Harel [13]. The pairs-set, $\mathcal{S}$, consists of two kinds of node-pairs: (1) "pivot-based" pairs and (2) pairs of close nodes. This construction is made in the following way. First, we construct a set $\mathcal{P}$ that contains m pivot nodes uniformly scattered over the graph. Although such pivots can be chosen randomly, a better method, which we use in practice, is described in an earlier paper [12] by the same authors. We then initialize $\mathcal{S}$ with all of the pairs $\{\{i,j\} \mid i \in \mathcal{P}, j \in V, i \neq j\}$. The distances between these pairs preserve the overall structure of the graph. Then, we add to $\mathcal{S}$ all pairs that are close in



weighted edges uniform edges

Fig. 5. Two majorization-based layouts of BGP connectivity, with a skewed degree distribution.



(a) (b)

Fig. 6. (a) Layout of the edge weighted BGP connectivity graph using Kamada-Kawai optimization. **(b)** Stress-vs.-time behavior of majorization and Kamada-Kawai on weighted BGP connectivity example graph.

the graph: $\{\{i,j\} \mid d_{ij} \leqslant B\}$. Typical values for B are between 3 and 6. This way we account for the local aesthetics of the drawing, which are not captured well by the pivots.

Koren and Harel have noted [13] that this sparse scheme completely fails under traditional Kamada-Kawai optimization. The Kamada-Kawai process cannot deal successfully with the two kinds of nodes, “pivots” and “non-pivots”, and yields unacceptable results. They thus propose a 1-D optimization process to replace the Kamada-Kawai process and show that heuristically this process can also make 2-D layouts. Using such an optimization process, they obtained nice layouts using very sparse energy.

As discussed below, the 1-D process of Koren and Harel is equivalent to 1-D majorization. However, majorization is more powerful as it can deal with higher dimensions. Therefore, as

expected, majorization works well with sparse stress functions, surpassing the results originally reported [13].

Technically, this means that the matrices L^w and L^Z defined in Section 2 are sparse matrices, allowing cheaper storage and faster matrix-vector multiplication. As mentioned in [13], pivot nodes may appear to "break out" of the drawing. This can be alleviated by smoothing each individual pivot locally: we iteratively execute process (12) on each pivot considering only its local neighborhood; see [13] for more details.

Working with sparse stress is effective for reducing the time and space complexity of the optimization process, and is possible only if we replace the Kamada-Kawai process with majorization.[2]

5 Subspace Restricted Stress Optimization

A recent approach [14] to accelerating graph layout algorithms is based on restricting the layout axes to lie within a specially crafted subspace. Here we briefly show how the majorization process fits nicely into this framework.

A k-D subspace is spanned by the columns of an $n \times k$ matrix A, so each layout axis is constrained to be of the form Av, for $v \in \mathbb{R}^k$. It is crucial to find a smart construction of A so we can find axes $Av_1, \ldots, Av_k$ which correspond to a nice layout of the graph. Ways of constructing A were described previously [14]. We can easily integrate the subspace restriction into graph drawing algorithms based on matrix algebra by substituting every appearance of the axes $X^{(1)}, \ldots, X^{(k)}$ with $Av_1, \ldots, Av_k$. This usually allows a significant reduction of the dimensionality of the problem.

Kamada-Kawai optimization does not fit in this framework. Koren has suggested [14] that one optimize the stress within a subspace using the 1-D stress optimization algorithm of [13]. However, this algorithm is intended only for the 1-D case and the convergence properties of its heuristic 2-D extension are unclear.

Using the majorization technique, which generalizes the algorithm in [13], we can naturally deal with higher dimensions and guarantee its convergence. Since each iteration of the stress process is defined by a pure quadratic form (6), we can easily integrate it with subspace restriction.

Technically, when the layout axes should be spanned with the columns of A, we can replace the unknown $n \times d$ coordinate matrix X (see Section 2) with an unknown $k \times d$ matrix V, so that the coordinates are AV. Substituting this in (6), we get the following expression for the majorizing function

$$G^{Z,A}(V) = \sum_{i<j} w_{ij} d_{ij}^2 + \mathrm{Tr}(V^T A^T L^w A V) - 2\mathrm{Tr}(V^T A^T L^Z Z)\,. \tag{15}$$

We should iteratively minimize this function. In analogy with (8), we do this by solving

$$A^T L^w A V(t+1)^{(a)} = A^T L^{X(t)} X(t)^{(a)}, \quad a = 1, \ldots, d\,. \tag{16}$$

Thus, we have replaced solving an $n \times n$ system (9) with solving a $k \times k$ system. The value of k, which is the dimensionality of the subspace, is usually around 50, so we can achieve a significant decrease of complexity. The main complexity of this process lies in the need to recompute in each iteration the right-hand side of (16). Results of optimizing the stress within the subspace are provided in [14]. Working with majorization can further improve the quality of the results and guarantee the monotonic convergence of the process.

[2] Due to space limitations, full experimental results are omitted from this abstract.

Smart initialization Another key use of optimizing within a subspace is to provide a smart initialization for the more accurate full process. In the default setting of our implementation, we precede the full stress minimization with a sparse optimization restricted to a 30-D subspace for at most 50 iterations. This is helpful for two reasons. First, it improves the robustness of our process, avoiding undesired minima. Second, it usually cuts overall running time by about a factor of two.

Figure 7 compares the energy-vs.-time performance for two graphs: 516 [18] (|V|=516, |E|=729) which was already analyzed in Figure 2 and the Plat1919 graph [1] (|V|=1919, |E|=15240). As usual, we give results averaged over 25 runs. The results given for the Plat1919 are typical, showing how the smart initialization significantly reduces running time and slightly improves the average stress level. The results for the 516 graph are far more dramatic, as the stress level obtained with smart initialization is almost 3 times better. In fact, the 516 graph is an elongated mesh (see [18] for its layout) and should be easily drawn without smart initialization. However, we observed that, once every few iterations, we obtained a twisted layout with a significantly increased stress. Since the plot averages 25 runs, this raised the average stress level when using random initialization. This is a nice demonstration of how smart initialization by subspace restriction can make the layout process more robust.

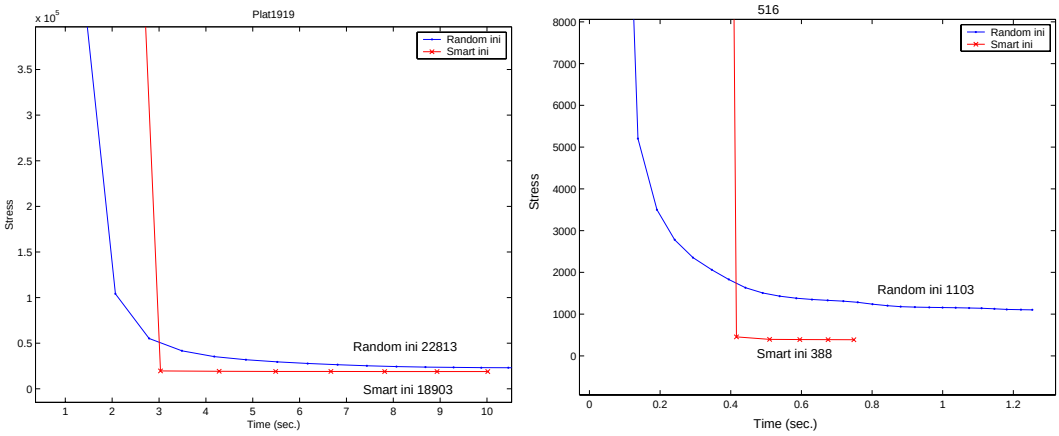


Fig. 7. The affect of smart initialization: stress-vs.-time for the graphs Plat1919 [1] and 516 [18].

6 Related Work

Substantial work in statistical MDS deals with the properties of the majorization process, including proofs of its convergence rate [3]. The MDS literature suggests solving equation (9) by computing $(L^w)^+$, the Moore-Penrose inverse of the singular matrix L^w. Our suggestion to set $X_1 = 0$ allows much faster solution by Cholesky factorization.

Several studies in the graph drawing field suggest improving stress computation by multi-scale extensions [8, 10, 11] which approximate the graph by a smaller one, to quickly obtain an initial layout. We see these approaches as complementary to our proposal, as one can apply majorization to optimizing the stress at each scale. In general, our recommendation is to get an initial placement either by multi-scale techniques or by subspace-restricted computation.

Recent work by Koren and Harel [13] describes an algorithm for monotonically decreasing the stress function in 1-D, and a heuristic extension to higher dimensions whose convergence properties are unknown. It is easy to prove that this 1-D algorithm is equivalent to 1-D majorization, although derived differently. However, majorization is more powerful as

it can be generalized to higher dimensions. Interestingly, the optimization process of [13] is equivalent to full, n-D Newton-Raphson process. Accordingly, we conclude that in 1-D, the majorization process is equivalent to full n-D Newton-Raphson process. This is unlike the Kamada-Kawai process that is based on a localized 2-D Newton-Raphson process.

7 Conclusions

Majorization, a technique developed in studies of statistical MDS, is relevant to practical graph drawing. The MDS community has studied it extensively from the standpoint of optimizing stress function and escaping local minima. Further ideas along these lines may also prove useful in graph drawing.

The main algorithms discussed here are available in the Neato program in the GraphViz open source package [19].

References

1. R. F. Boisvert et al., "The Matrix Market: A web resource for test matrix collections", in *Quality of Numerical Software, Assessment and Enhancement*, R. F. Boisvert, ed., Chapman Hall, 1997, pp. 125–137 `math.nist.gov/MatrixMarket`.
2. F.J. Brandenburg, M. Himsolt and C. Rohrer, "An Experimental Comparison of Force-Directed and Randomized Graph Drawing Algorithms", *Proceedings of Graph Drawing '95*, LNCS 1027, pp. 76–87, Springer Verlag, 1995.
3. J. De Leeuw, "Convergence of the Majorization Method for Multidimensional Scaling", *Journal of Classification* **5** (1988), pp. 163–180.
4. G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, 1999.
5. I. Borg and P. Groenen, *Modern Multidimensional Scaling: Theory and Applications*, Springer-Verlag, 1997.
6. J. D. Cohen, "Drawing Graphs to Convey Proximity: an Incremental Arrangement Method", *ACM Transactions on Computer-Human Interaction* **4** (1997), pp. 197–229 .
7. E. R. Gansner and S. C. North, "Improved force-directed layouts", *Proceedings of Graph Drawing '98*, LNCS 1547, pp. 364–373, Springer-Verlag, 1998.
8. P. Gajer, M. T. Goodrich and S. G. Kobourov, "A Multi-dimensional Approach to Force-Directed Layouts of Large Graphs", *Proceedings of Graph Drawing 2000*, LNCS 1984, pp. 211–221, Springer-Verlag, 2000.
9. G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, 1996.
10. R. Hadany and D. Harel, "A Multi-Scale Method for Drawing Graphs Nicely", *Discrete Applied Mathematics* **113** (2001), pp. 3-21.
11. D. Harel and Y. Koren, "A Fast Multi-Scale Method for Drawing Large Graphs", *Journal of Graph Algorithms and Applications* **6** (2002), pp. 179–202.
12. D. Harel and Y. Koren, "Graph Drawing by High-Dimensional Embedding", *Proceedings of Graph Drawing 2002*, LNCS 2528, pp. 207–219, Springer-Verlag, 2002.
13. Y. Koren and D. Harel, "Axis-by-Axis Stress Minimization", *Proceedings of Graph Drawing 2003*, Springer-Verlag, pp. 450–459, 2003.
14. Y. Koren, "Graph Drawing by Subspace Optimization", *Proceedings 6th Joint Eurographics - IEEE TCVG Symposium Visualization (VisSym '04)*, pp. 65–74, Eurographics, 2004.
15. T. Kamada and S. Kawai, "An Algorithm for Drawing General Undirected Graphs", *Information Processing Letters* **31** (1989), pp. 7–15.
16. M. Kaufmann and D. Wagner (Eds.), *Drawing Graphs: Methods and Models*, LNCS 2025, Springer-Verlag, 2001.
17. J. W. Sammon, "A Nonlinear Mapping for Data Structure Analysis", *IEEE Trans. on Computers* **18** (1969), pp. 401–409.
18. C. Walshaw, "A Multilevel Algorithm for Force-Directed Graph Drawing", *Proceedings 8th Graph Drawing (GD'00)*, LNCS 1984, pp. 171–182, Springer-Verlag, 2000.
19. GraphViz `www.research.att.com/sw/tools/graphviz/`
20. Graphlet `www.infosun.fmi.uni-passau.de/Graphlet/`
21. Intel Math Kernel Library `www.intel.com/software/products/mkl/`
22. Automatically Tuned Linear Algebra Software (ATLAS) `math-atlas.sourceforge.net/`