The Binary Stress Model for Graph Drawing

Yehuda Koren[1] and Ali Çivril[2]

[1] AT&T Labs — Research
[2] Rensselaer Polytechnic Institute

Abstract. We introduce a new force-directed model for computing graph layout. The model bridges the two more popular force directed approaches – the stress and the electrical-spring models – through the *binary stress* cost function, which is a carefully defined energy function with low descriptive complexity allowing fast computation via a Barnes-Hut scheme. This allows us to overcome optimization pitfalls from which previous methods suffer. In addition, the binary stress model often offers a unique viewpoint to the graph, which can occasionally add useful insight to its topology. The model uniformly spreads the nodes within a circle. This helps in achieving an efficient utilization of the drawing area. Moreover, the ability to uniformly spread nodes regardless of topology, becomes particularly helpful for graphs with low connectivity, or even with multiple connected components, where there is not enough structure for defining a readable layout.

1 Introduction

A popular approach to drawing graphs is based on measuring the quality of the layout through a formal cost function. The layout of the graph is formed by an optimization algorithm that finds a local minimum of the cost function. This family of algorithms is known in the graph drawing literature as force-directed algorithms; see, e.g., [3,14].

Broadly speaking, force-directed cost functions (also known as *energies*) define a desired layout based on either the electric-spring metaphor or on a stress function. Electric spring functions liken the graph to a physical system where nodes correspond to electrically charged particles, and edges correspond to springs with zero rest length. Repulsive electric forces ensure that nodes are well separated, while attractive spring forces tend to shorten edges and pack closely connected components. Two well known early versions of this scheme are by Eades [4] and by Fruchterman and Reingold [6].

The stress function relates a nice drawing to good isometry. We have an ideal target distance d_{ij} for every pair of nodes i and j. Given a 2-D layout, where node i is placed at point p_i, the stress function is:

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

We desire a layout that minimizes this function, thereby best realizing the target distances. Here, the distance d_{ij} is typically the graph-theoretical distance between nodes

I.G. Tollis and M. Patrignani (Eds.): GD 2008, LNCS 5417, pp. 193–205, 2009.
© Springer-Verlag Berlin Heidelberg 2009

i and j. The normalization constant w_{ij} equals $d_{ij}^{-\alpha}$. The function (1) appeared earlier as the stress function in multidimensional scaling [2], where it was applied to graph drawing [16]. It became a popular graph drawing tool by Kamada and Kawai [13].

Both electric-spring and stress approaches enjoy successful implementations and offer pleasing layouts to many graphs. In terms of layout appearance, there are distinct differences between the models, though they are hard to define. As for computational aspects, the two approaches induce different optimization processes, and each has a unique advantage. Electric-spring models have the advantage of a lower descriptive complexity compared to the stress model. This is because all repulsive forces are uniform, whereas attractive forces involve only the $|E|$ pairs of adjacent nodes. On the other hand, the stress function requires encoding a different target distance for each node pair. This fundamental difference bounds stress models to quadratic space complexity, while efficient implementations of electric-spring models scale to larger graphs.

On the other hand, the stress function has a mild landscape, which allows utilizing powerful optimization techniques such as majorization [7]. This way, good minima are usually achieved regardless of the initial positions. This is untrue for the electric-spring models, which induce an intricate landscape as repulsive forces make the energy go to infinity when nodes overlap. This causes serious convergence problems even for moderately sized graphs. Past works [9,11,19] used sophisticated initialization techniques through multilevel approximation to overcome these problems.

In this work we introduce the binary-stress model (*bStress*) for drawing graphs. Computationally, it is able to merge the advantages of both the electric-spring model and the stress model. Namely, it offers a low descriptive complexity, thus being scalable to very large graphs. At the same time, it is similar in its form to the known stress function, thus enabling the use of the majorization optimization scheme.

As for the quality of the layout, bStress frequently offers a unique perspective to the graph structure. More than other models, bStress emphasizes uniform spread of the nodes within a circular drawing area. This may lead to distinctive layouts, which can serve as useful addition to those produced by other algorithms. Moreover, the emphasis on uniform spread is advantageous for graphs with low connectivity, whose structure alone is not capable of defining a good layout. For example, bStress will naturally handle graphs with multiple connected components by packing all connected components together without requiring any post-processing or special treatment that alternative methods require. In addition, bStress is suitable for drawing large graphs, not only because of its improved scalability, but also because it achieves good area utilization that is important for placing a large number of nodes.

2 Basic Notions

We are seeking a layout for a graph $G(V = \{1, \ldots, n\}, E)$, where the position of node i is $p_i = (x_i, y_i)$. Sometimes, we will refer to the vectors $x, y \in \mathbb{R}^n$, which represent all x- or y-coordinates, respectively. Notice that while this work addresses the more common case of 2-D layouts, as usual with force-directed algorithms, extensions to 3-D are naturally possible.

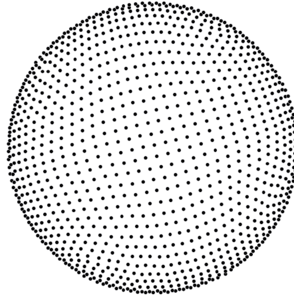


Fig. 1. A Layout of 1024 points that minimizes $G(p)$, by scattering the points within a circle

3 The Binary Stress Model

One of the earliest cost functions involved in defining a nice layout strives to shorten the squared edge lengths:

$$H(p) = \sum_{\langle i,j \rangle \in E} \|p_i - p_j\|^2 \tag{2}$$

However, minimizing $H(p)$ on its own is not sufficient for defining a useful layout, as nothing prevents all nodes from collapsing at a single point. Thus, Tutte [18] and Hall [10] augmented $H(p)$ with simple constraints that prevented the formation of trivial layouts. Nonetheless, both solutions tend to generate layouts with very uneven sparsity, where many nodes are overcrowded together. Moreover, Tutte's and Hall's methods fail to produce adequate layouts for graphs of low connectivity such as tree-like graphs.

A hypothetical possible way to make $H(p)$ working for general graphs, is to lay out the graph over a grid and then minimize $H(p)$ while requiring that each node is positioned at a unique grid cell. This will ensure a uniform spread of the nodes and prevent nodes from getting too close to each other. However, practical implementation of such a strategy would be quite complicated. The primary issue is that constraining positions to grid cells transforms the problem into integer optimization, which would be much harder to solve and less scalable.

We avoid integer optimization by adopting a continuous relaxation of the grid layout strategy. The relaxation is based on the following cost function:

$$G(p) = \sum_{i \neq j \in V} (\|p_i - p_j\| - 1)^2 \tag{3}$$

This function strives to place all nodes such that their pairwise distances are uniform. Notice that $G(p)$ is independent of the graph structure. The minimum of $G(p)$, as we have found experimentally, will position the nodes almost uniformly within a circle. For example, consider Fig. 1, where 1024 nodes are positioned so as to minimize $G(p)$.

The function $G(p)$ gives us the necessary tool to combat the over dense areas which are typical to minimization of $H(p)$. Thus, the binary stress function for computing a layout of a graph is defined as a linear combination of the two functions:

$$B(p) = \sum_{\langle i,j \rangle \in E} \|p_i - p_j\|^2 + \alpha \sum_{i \neq j \in V} (\|p_i - p_j\| - 1)^2 \quad (4)$$

The first term relates the layout to the graph structure by ensuring that edges are short, whereas the second term makes the nodes spread uniformly within a circle. The constant α (discussed later) controls the balance between the two terms.

Our experience shows that bStress results in useful layouts for wide families of graphs. However, before we dwell into the quality of layouts generated by the bStress model, we would like to discuss computational aspects.

4 Minimizing the Binary Stress Function

The bStress function (4) is structured as a sum of two stress functions (Eq. (1)), one with target distances equal to 0, and the other with target distances equal to 1. This is the reason for choosing the "binary stress" name. Though, the particular value of 1 has no influence on the resulting layout and any other positive value could be used as well.

As sum of stress functions, the majorization optimization technique can be exploited to optimizing bStress. Derivation of the stress majorization was given by Gansner et al. [7]. The process used here is as follows:

Let us define two $n \times n$ matrices, L and M. The matrix L is the *Laplacian* of graph G, whose associated quadratic form is the sum of squared edge lengths $H(p)$. The other matrix, M, is associated with a quadratic form that bounds $G(p)$:

$$L_{i,j} = \begin{cases} -1 & \langle i,j \rangle \in E \\ \sum_{k \neq i} L_{ik} & i = j \\ 0 & \text{otherwise} \end{cases}, \qquad M_{i,j} = \begin{cases} -1 & i \neq j \\ n-1 & i = j \end{cases}$$

We also define two vectors, $b^x, b^y \in \mathbb{R}^n$, which sum all cosines and sines associated with each node:

$$b_i^x = \sum_{j \neq i} \frac{x_i - x_j}{\|(x_i, y_i) - (x_j, y_j)\|}, \quad b_i^y = \sum_{j \neq i} \frac{y_i - y_j}{\|(x_i, y_i) - (x_j, y_j)\|} \quad (5)$$

Given a current placement $p(t) = (x(t), y(t))$, an improved placement $p(t+1) = (x(t+1), y(t+1))$, which lowers $B(p)$, is computed by solving the system of equations:

$$(M + \alpha L)x(t+1) = b^{x(t)}, \quad (M + \alpha L)y(t+1) = b^{y(t)} \quad (6)$$

Now, let us consider computational complexity. The number of entries in matrix L is $n+|E|$. The other matrix – M – is, strictly speaking, dense. However its highly uniform structure makes it sparse for practical purposes. Typical to the stress majorization process is solving (6) by using the conjugate gradient method, which accesses $(M+\alpha L)$ as a linear operator. Thus, all we need to ensure is that the product $(M+\alpha L)x$, can be computed efficiently. This is indeed the case, as L is sparse, and $(Mx)_i = nx_i - \sum_j x_j$, which is computed in a constant time after precomputing $\sum_j x_j$. Thus, the product $(M + \alpha L)x$, is computed in time $O(n + |E|)$.

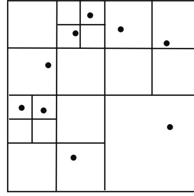


Fig. 2. A quad-tree hierarchical space decomposition

The more challenging operation is the computation of the b^x and b^y vectors of Eq. (5). This essentially involves computing the angles formed by all node pairs. Here we follow several recent graph drawing works [9,11,17] and use the Barnes-Hut scheme [1] for approximating the $O(n^2)$ interactions in practically $O(n \log n)$ time. Thus, we use a hierarchical geometric decomposition of the drawing area through a quad-tree data structure. The whole area is assigned to a square (or, a rectangle). Then, each square is subsequently partitioned into four identical squares, till each node is lying within a unique leaf square. See Fig. 2 for an illustration.

Computation of b_i^x and b_i^y is based on a top-bottom traversal of the quad-tree. Let v be a quad-tree vertex corresponding to square s with side length l. We compare l to d - the distance between node i and the center of square s. If $l/d > \theta$, then we continue the traversal recursively with the four children of v. Otherwise, we halt the traversal while taking the approximation that all graph nodes lying within square s are at the same location, and thus can be processed at once. Our default value for θ is 0.5.

In order to give a flavor of actual running times, we report our experience with graphs of varying sizes in Table 1. Times were measured on a Pentium 4 PC. We let the majorization process run for 200 iterations, while it was terminated earlier once $\|p(t+1) - p(t)\| / \|p(t)\| < 0.001$. Overall running time is divided among the two components of the algorithm: (1) solving Eq. (6) through the conjugate gradients iterative process. (2) Computing b^x and b^y (Eq. (5)) using a Barnes-Hut approximation. The table shows that the Barnes-Hut approximation is indeed closely following an $O(n \log n)$ running time. The conjugate gradient component takes $(n + |E|)$ time per internal iteration, but the number of those iterations is less consistent. Since the Barnes-Hut calculation is independent of the number edges, as graphs become denser the conjugate gradient component becomes more significant (see graphs 'plustk10' and 'gearbox'). Wall-clock measured running times are not directly comparable across different papers, due to differences in platforms and code optimization. However, we believe that the ability of bStress to lay out of 100,000 nodes in a few minutes, places it among the more efficient graph drawing techniques.

5 Results and Implementation Details

The binary stress model is based on unique principles, which in many cases lead to layouts quite different than those produced by other algorithms. Hence, a key to assessing the utility of the new model is a qualitative analysis of typical results. In the following subsections we discuss various aspects of bStress through concrete layout examples.

Table 1. Running time characteristics for graphs of varying sizes. We measure times for the two components of the algorithm: a conjugate gradient solver, and Barnes-Hut approximation of vectors b^x and b^y. The last two columns show the dependency of running time with graph size. Graphs are taken from [12].

name	nodes	edges	iterations	conjugate gradient time/it (sec.)	Barnes-Hut time/it (sec.)	$10^6 \times \frac{\text{C.G. time}}{\lvert E\rvert+n}$	$10^6 \times \frac{\text{B.H. time}}{n \cdot \log n}$
nopoly	10774	30034	133	0.019	0.182	0.477	4.181
skirt	12598	91961	109	0.082	0.272	0.784	5.264
tuma2	12992	20925	13	0.015	0.238	0.454	4.462
poli_large	15575	17468	200	0.106	0.305	3.199	4.666
powersim	15838	36430	200	0.045	0.357	0.869	5.366
ncvxqp9	16554	22493	200	0.023	0.405	0.598	5.797
lpl1	32460	147788	200	0.408	0.763	2.261	5.212
finance256	37376	130560	200	0.192	0.749	1.145	4.385
bcircuit	68902	153328	200	0.328	1.874	1.476	5.621
plustk10	80676	2114154	159	5.169	2.125	2.355	5.367
Ford2	100196	222246	33	0.582	2.230	1.806	4.450
gearbox	107624	3250488	200	5.874	3.317	1.749	6.124
lung2	109460	273646	137	0.272	3.477	0.710	6.304

5.1 Balancing the System

Recall that bStress is parametrized by α, which controls the balance between uniform spread and structure preservation. As α grows, the model will prefer shortening edges over uniformly spreading the nodes. This can significantly influence the appearance of the layout. For example, in Fig. 3 we show two layouts of the same graph, one computed with $\alpha = 1$ and the other with $\alpha = 1000$. When α is low (=1), the model emphasizes uniform spread, thus nodes are well separated and visible. On the other hand, when α is high (=1000), the model cares mostly about exposing the graph's structure through shortening edges. Thus, the different hubs that form the graph are clearly shown.

Notice that $G(p) = \sum_{i \neq j \in V} (\|p_i - p_j\| - 1)^2$ contains about $n^2/2$ terms, whereas the other part of bStress, $H(p) = \sum_{\langle i,j \rangle \in E} \|p_i - p_j\|^2$, contains only $|E|$ terms. Thus, $G(p)$ becomes more and more dominant as $n^2/|E|$ grows. This is undesirable, as it makes the determination of parameter α less stable across varying graphs. To offset some of this phenomenon, our experience shows that as $|E|/n$ grows, it is beneficial to overweight $H(p)$ over $G(p)$. In other words, for sparse graphs, there is no much structure in the graph and it is reasonable to pay much attention to uniform spread. However, for denser graphs, there is much structure to be captured from the connectivity information. Combining these considerations, we learned that a sensible choice to α is $c \cdot n$, for some positive constant c. Hence, the bStress model becomes:

$$B(p) = \sum_{\langle i,j \rangle \in E} \|p_i - p_j\|^2 + c \cdot n \sum_{i \neq j \in V} (\|p_i - p_j\| - 1)^2 \qquad (7)$$

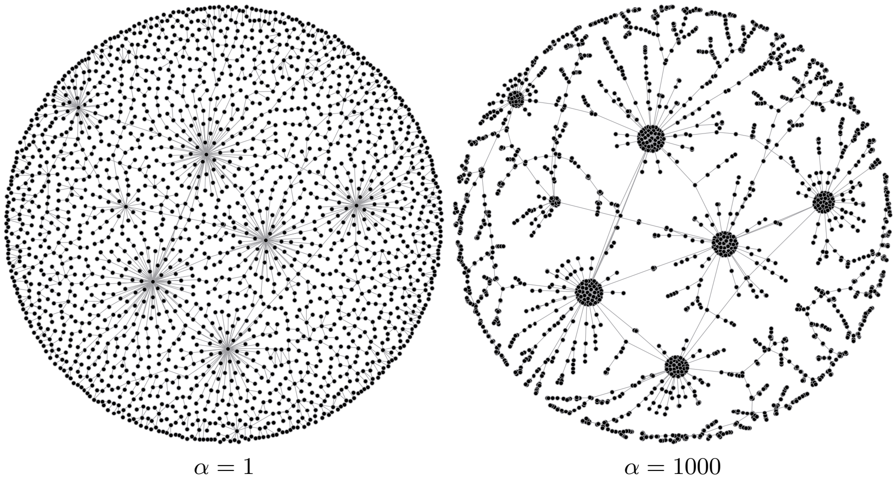


Fig. 3. Two bStress layouts of a graph with 1933 nodes and 2043 edges. Setting $\alpha = 1$ achieves better separation of nodes and improved area utilization. However, some may prefer $\alpha = 1000$, for the better abstraction of the graph's structure.

Focusing on values of c is easier than focusing on values of α. In fact, our experiments show that $c = 1$ is a universally reasonable choice, being our default value. In some cases, better results are obtained with lower values of c.

There is another implication to the value of c, beyond layout appearance. We have found that the majorization optimization process may encounter bad local minima when c is too low. To avoid this, we first run the algorithm with higher values of c, and then use the resulting layout for seeding a process with a lower c value. That is, a typical run would start with c=100, and then restart with c=1. Usually, the number of majorization iterations after restarting the run is relatively low thanks to the improved initialization.

5.2 Drawing Trees

Prior adaptation of the $H(p)$ function to drawing graphs [10,18] could not handle trees and tree-like graphs adequately. The major issue was the inability to prevent many nodes from collapsing at the same location, thus resulting in a highly imbalanced layout with much unused area and a few overcrowded locations. Such an issue does not exist with bStress, as could be evident from the drawing of a tree-like graph given in Fig. 3. In fact, as graphs become sparser, results of bStress look increasingly different than those computed by alternative models such as the aforementioned stress and electric-spring models. This is because, the lack of sufficient connectivity information let the uniform spread component, $G(p)$, be more dominant in shaping the layout.

As an example, in Fig. 4–5 we present the drawings of two trees, which are derived from an Internet map and a BGP connectivity map. Results of bStress are compared to the results of the stress function. The known stress model seems to be better at exposing the decomposition of the tree, whereas bStress achieves more uniform node distribution.

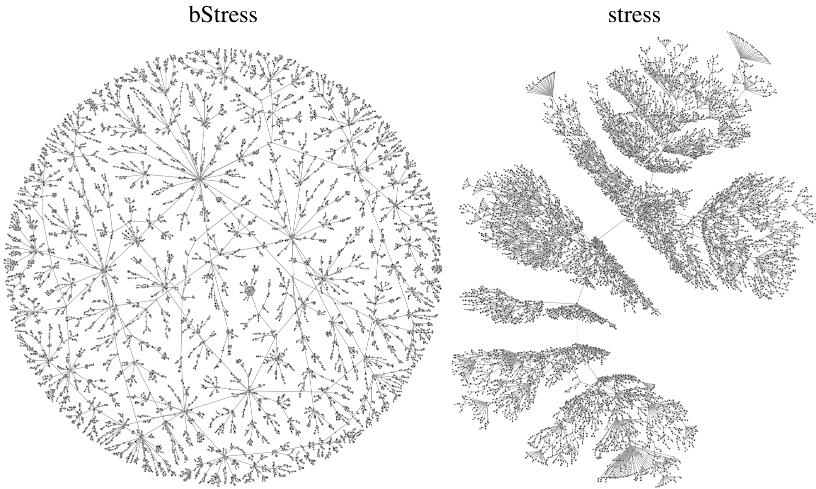


Fig. 4. Comparing stress to bStress in drawing an Internet map tree ($|V|$=9227, $|E|$= 9226)

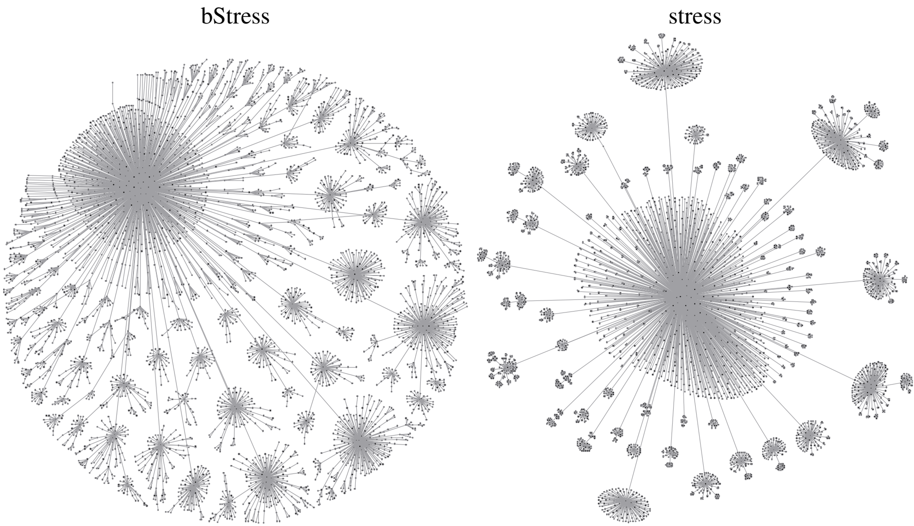


Fig. 5. Comparing stress to bStress in drawing a BGP connectivity tree ($|V|$=3487, $|E|$= 3486)

The uniform spread achieved by bStress becomes particularly useful when the number of nodes is large making area utilization a high priority.

5.3 Disconnected Graphs

Most force-directed methods cannot directly handle disconnected graphs. For example, the stress model requires defining the distance between each two nodes, which is

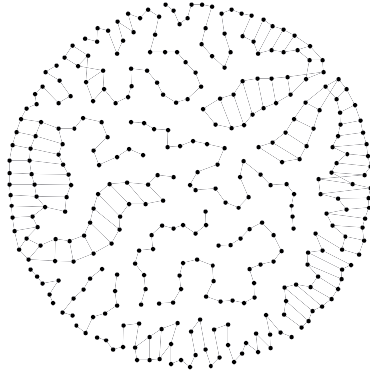


Fig. 6. A graph with 11 connected components ($|V|$=333, $|E|$=397)

not naturally defined for disconnected nodes. Likewise, the electric spring model assumes only repulsive forces among connected components, ultimately pushing them away from each other till infinity. Certainly, various modifications to those models can enable working with disconnected graphs. Most notably, each connected component can be drawn separately, and later a smart packing algorithm squeezes all components within the drawing area [5].

Interestingly, bStress handles disconnected graphs exactly the same way it handles connected graphs. Thus, unlike other methods, it does not require any modification or postprocessing when addressing disconnectivity. This is thanks to the uniform spread model ($G(p)$), which strives for a fairly uniform node distribution, regardless of connectivity. A small artificial example is brought in Fig. 6, where we draw a graph with 11 connected components. As can be seen, bStress could pack all components efficiently together within a circle, while no two components overlap, and each component is dawn reasonably. A larger, more realistic example is given in Fig. 7, where we show a graph consisting of many Internet traces. The graph contains 3743 connected components, which are all packed pretty well within the layout.

5.4 Filling a Circle

A notable feature of bStress is packing the graph within a circle. Admittedly, the circular shape of the layout is not a design goal but rather an outcome of the chosen cost function. However, filling the interior of the circle is indeed a design goal of the bStress model. In some cases this can lead to surprisingly looking layouts. For example, some layouts would be expected to lie on the periphery of a circle. However, bStress will "insist" on filling the circle with some of the nodes, due to the strict uniform spread requirement. This might look odd at first, but we argue that it has an advantage of enabling a better distinction between individual nodes.

We demonstrate this in Fig. 8. First simple example is a (topological) circle, which is twisted in order to spread nodes within the interior. Another example is the finan512 graph, which became a standard example in works aimed at drawing large graphs.

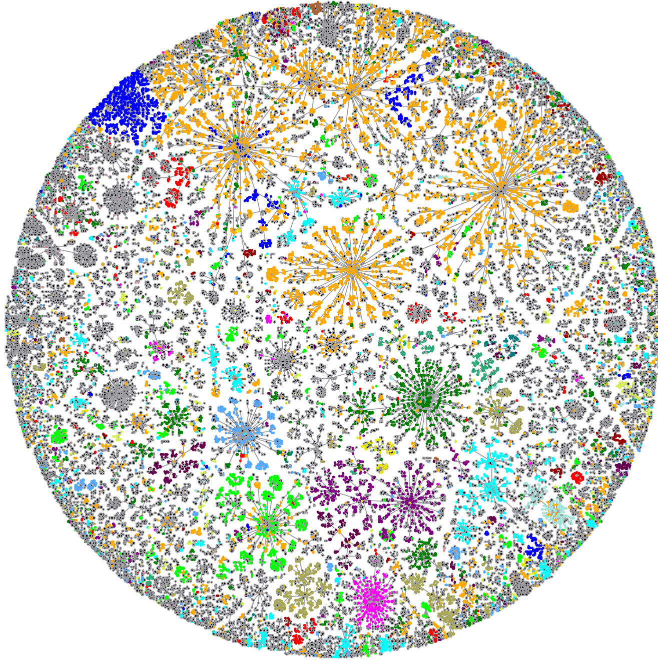


Fig. 7. An Internet map with 3743 connected components ($|V|$=33552, $|E|$=29809). Node colors indicate some known ISPs.

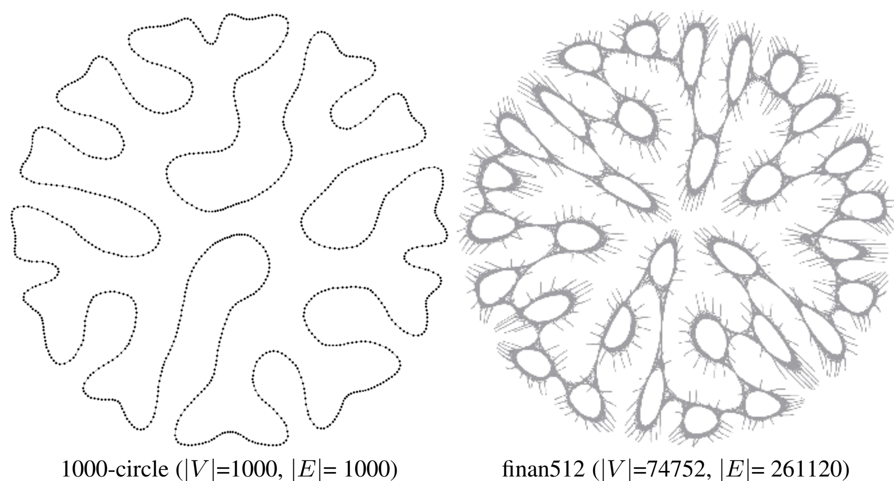
Previous works (e.g., [15,19]) placed all nodes on or close to the perimeter of a circle. On the other hand, bStress fills the interior of the circle. This enables a better view of the local details of this large graph, at the price of an inferior exhibition of symmetries. At this point, we would like to clarify that while frequently the outline of the layout is circular, this is not always the case; for example consider Fig. 9.

5.5 Distorting the Layout

The uniform spread component, $G(p)$, induces layouts where the periphery is denser than the central area. This effect can be seen in Fig. 1. Let us take a polar coordinates viewpoint, where the origin is the layout center. We observe that nodes are uniformly spread across different angular coordinates, but less so across different radial coordinates. Thus, we propose the following correction as an optional postprocessing phase.

We denote the layout density (or, sparsity) around node i by d_i. This way $d_i = 0$ for the densest possible area, while d_i is large when there is a lot of free area around i. One way to measure d_i is to set it to the average distance between i and its top k closest nodes in the layout. In our implementation, we compute a relative neighborhood graph (RNG), and define d_i as the average length of edges adjacent to i in the RNG.

We sort all nodes by their radial coordinates, which are distances from the center. Then, we smooth the computed densities, by averaging densities of nodes with similar



1000-circle ($|V|$=1000, $|E|$= 1000) finan512 ($|V|$=74752, $|E|$= 261120)

Fig. 8. bStress tends to fill the interior of a circle

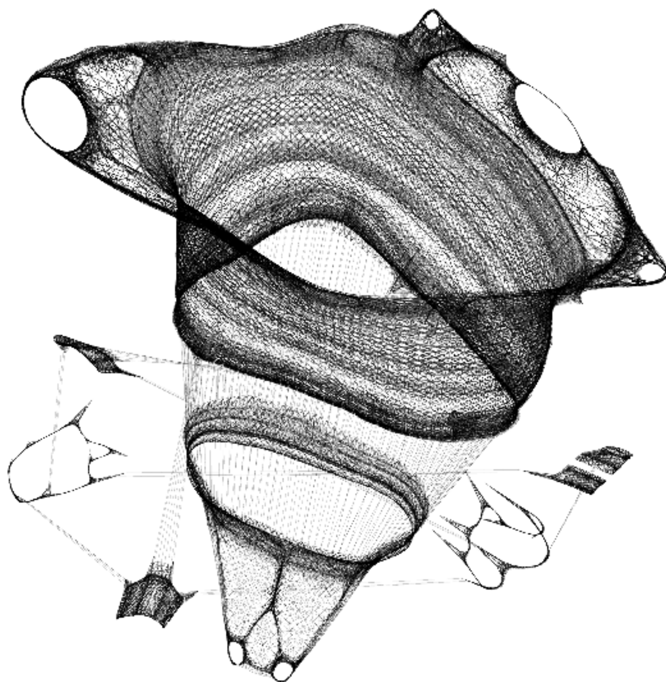


Fig. 9. The gearbox graph [12] ($|V|$=107624, $|E|$=3250488)

radial coordinates; see Sec. 6 of [8] for a similar procedure. Finally, for each node i, which comes immediately after node j in the sorted order, we modify the gap in radial coordinates between i and j by multiplying it by $1/d_i$. Thus, we shrink gaps in sparse areas, while widening gaps in dense areas.

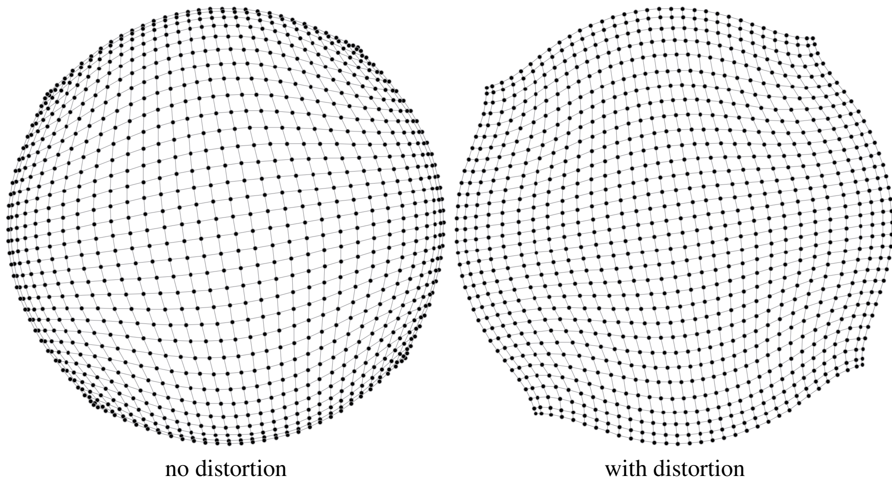


Fig. 10. The effect of post-processing the layout of a 32×32 grid with a radial distortion that makes node distribution more uniform

We include this distortion in our default settings, as it takes a negligible time, and occasionally leads to a modest improvement of layout appearance. A simple example is a square grid, whose layout improves when applying the distortion as shown in Fig. 10.

6 Conclusions

The binary stress model leads to unique graph layouts characterized by uniform distribution of nodes within a circular area. This is particularly beneficial for large graphs, where efficient utilization of the drawing area becomes vital. In addition, the model is capable of producing decent layouts even for graphs with low connectivity, where scant adjacency information cannot define a useful layout on its own. Computationally, it combines some of the benefits of both the stress and the electric-spring model, facilitating a simple, yet effective optimization procedure that scales well for very large graphs. We believe that it should coexist as a viable option along more familiar models.

References

1. Barnes, J.E., Hut, P.: A hierarchical $O(NlogN)$ force calculation algorithm. Nature 324(4), 446–449 (1986)
2. Borg, I., Groenen, P.: Modern Multidimensional Scaling: Theory and Applications. Springer, Heidelberg (1997)
3. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall, Englewood Cliffs (1999)
4. Eades, P.: A heuristic for graph drawing. Cong. Numer. 42, 149–160 (1984)
5. Freivalds, K., Dogrusoz, U., Kikusts, P.: Disconnected graph layout and the polyomino packing approach. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 378–391. Springer, Heidelberg (2002)

6. Fruchterman, T.M.G., Reingold, E.: Graph drawing by force-directed placement. Software-Practice Experience 21(11), 1129–1164 (1991)
7. Gansner, E., Koren, Y., North, S.: Graph drawing by stress majorization. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 239–250. Springer, Heidelberg (2005)
8. Gansner, E., Koren, Y., North, S.: Topological fisheye views for visualizing large graphs. IEEE Trans. Vis. Comput. Graph. 11(4), 457–468 (2005)
9. Hachul, S., Junger, M.: Drawing large graphs with a potential-field-based multilevel algorithm. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 285–295. Springer, Heidelberg (2005)
10. Hall, K.M.: An r-dimensional quadratic placement algorithm. Management Science 17(3), 219–229 (1970)
11. Hu, Y.F.: Efficient high quality force-directed graph drawing. The Mathematica Journal 10(1), 37–71 (2005)
12. Hu, Y.F.: A gallery of large graphs, `http://www.research.att.com/~yifanhu/GALLERY/GRAPHS`
13. Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Information Processing Letters 31(1), 7–15 (1989)
14. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)
15. Koren, Y.: Graph drawing by subspace optimization. In: Eurographics / IEEE TCVG Symposium on Visualization, pp. 65–74 (2004)
16. Kruskal, J., Seery, J.: Designing network diagrams. In: First General Conference on Social Graphics, pp. 22–50 (1980)
17. Quigley, A., Eades, P.: FADE: Graph drawing, clustering and visual abstraction. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 197–210. Springer, Heidelberg (2001)
18. Tutte, W.T.: How to Draw a Graph. Proc. London Math. Soc. s3-13(1), 743–767 (1963)
19. Walshaw, C.: A multilevel algorithm for force-directed graph drawing. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 171–182. Springer, Heidelberg (2001)