

Chapter 19

Algorithms for Visualizing Large Networks

Yifan Hu

AT&T Labs Research

19.1	Introduction	525
19.2	Algorithms for Drawing Large Graphs	528
19.2.1	Spring-Electrical Model	529
19.2.1.1	Fast Force Approximation	530
19.2.1.2	Multilevel Approach	532
19.2.1.3	An Open Problem: More Robust Coarsening Schemes	534
19.2.2	Stress and Strain Models	536
19.2.2.1	Stress Model	537
19.2.2.2	Strain Model (Classical MDS)	538
19.2.2.3	MDS for Large Graphs	539
19.2.3	High-Dimensional Embedding	541
19.2.4	Hall's Algorithm	542
19.3	Examples of Large Graph Drawings	543
19.4	Conclusions	544
	Acknowledgments	545
	Bibliography	545

19.1 Introduction

Graphs are often used to encapsulate relationship between objects. Graph drawing enables visualization of such relationships. The usefulness of this visual representation is dependent on whether the drawing is aesthetic. While there are no strict criteria for aesthetics of a drawing, it is generally agreed, for example, that such a drawing has minimal edge crossing, with vertices evenly distributed in the space, connected vertices close to each other, and symmetry that may exist in the graph preserved.

One of the earliest example of graph drawing is probably the Mill (also known as Morris) game boards, found in the thirteenth century *The Book of*

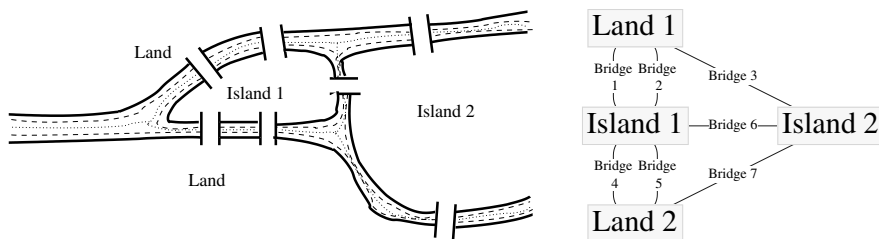


FIGURE 19.1: An illustration of the seven bridges of Königsberg (left) and the graph representation (right).

Games, produced under the direction of Alfonso X (1221-1284), King of Castile (Spain). Other examples of early graph drawing are drawings of family trees, and trees of virtues and vices.

In 1735, Euler presented and later published his famous paper [1] on the Seven Bridges of Königsberg. The problem was to find a walk through the city of Königsberg, which include two islands connected to the mainland by seven bridges (Figure 19.1). The walk must cross each bridge once and only once. Euler proved that such a walk is not possible. Euler's paper laid the foundation of graph theory. Interestingly, however, Euler did not provide a drawing of the graph representation of the problem itself in the paper.

In the eighteenth century, the Irish physicist, astronomer, and mathematician, William Hamilton, invented the Icosian game. To play the game, one has to visit every node of a dodecahedral graph once, and back to the starting node. Every edge can be passed at most once. In 1857 he sold the game to a London game dealer for 25 Pounds (about 3000 US dollars in today's money after taking inflation into account). The dealer made two versions; unfortunately, neither sold well, probably because the game was so simple that even a child could succeed. Around the same time, British mathematician Arthur Cayley wrote a paper on the enumeration of tree diagrams. In the paper he pioneered the concept of a tree, which is now fundamental to computer sciences. The paper also contains drawings of trees.

Between the eighteenth and nineteenth centuries, graph drawing also appeared in areas outside of mathematics. For example, in crystallography and chemistry, graph drawings were used to illustrate molecular structures. In 1878, the English mathematician Sylvester introduced the concept of a *graph* for the first time.

From the mid-twentieth century, with the demands from the military, transportation and communication, and science and technology, a great number of graph theory problems emerged. Often it is helpful to be able to visualize a graph. However drawing a complex graph by hand is time consuming, if not impossible, for large graphs. Therefore automatic generation of graph drawings became of interest, and were facilitated by the ever increasing computing power.

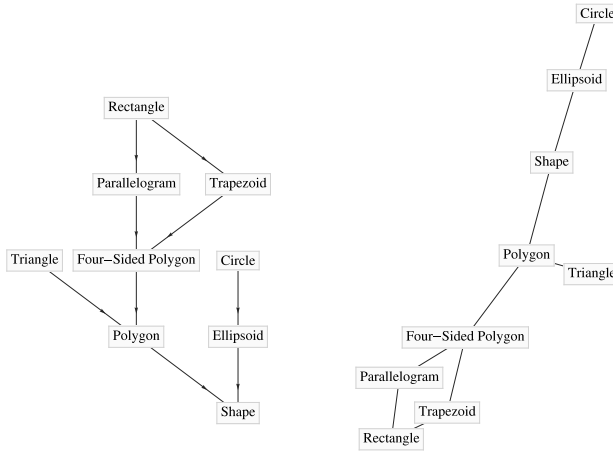


FIGURE 19.2: A graph drawn in hierarchical (left) and undirected (right) styles.

In 1963, Tutte [2] proposed an algorithm to draw planar graphs by fixing nodes on a face and placing the rest of the nodes at the barycenters of their neighbors. In 1981, Sugiyama et al. [3] proposed a method for the hierarchical drawing of directed graphs. Eades [4] proposed a force-directed algorithm in 1984, and Kamada and Kawai [5] the spring embedding algorithm in 1989. These algorithms were further improved [6, 7], and some of them applied to large graphs [8, 9, 10, 11, 12, 13] later.

Depending on the applications and properties of the graph to be visualized, there are many styles of graph drawing. For example, for planar graphs, a planar embedding is more appropriate. For general graphs, there are also two broad styles of drawing: straight line drawing, where each edge is represented by a straight line, and orthogonal drawing, where each edge is represented by lines segments that are either horizontal or vertical. Finally, for directed graphs where it is important to show the direction of the edges, a hierarchical drawing style can be used. Figure 19.2 shows a graph drawn using the hierarchical style (left) and the undirected straight-edged style (right).

In this chapter, we look at algorithms for visualizing large networks. The term large is relative to the computing power and memory available. In this paper, by large, we mean graphs of more than a few thousand vertices. Such large graphs occur in areas such as Internet mapping, social networks, biological pathways and genealogy. Large networks bring unique issues. For example, the complexity of the algorithm becomes very important. The ability of the algorithm in escaping from local minimum and achieving a globally optimal drawing is also vital. We present some of the algorithms suitable for large graphs. We shall limit our attention to straight edge drawing of undirected

graphs, mostly because scalable algorithms for these have been more intensely investigated. Our emphasis is on algorithms, with a tilt toward issues relating to combinatorial computing. We do not attempt to give a comprehensive review of the literature and history, nor of software and visualization systems; for these the reader is referred to [14, 15].

We note that not all graph can be aesthetically embedded into two or three-dimensional space. For example, many Internet graphs are of “small-world” nature [16]. Such graphs are difficult to layout in Euclidean space, and often require an interactive system to comprehend and explore. A number of techniques can be brought to bear on such graphs, including visual abstraction [17, 18], fisheye-like distortion and hyperbolic layout [18, 19, 20, 21, 22]. These topics are not within the scope of this chapter, but are nevertheless very important parts of large graph visualization.

19.2 Algorithms for Drawing Large Graphs

We use $G = \{V, E\}$ to denote an undirected graph, with V the set of vertices and E the set of edges. Denote by $|V|$ and $|E|$ the number of vertices and edges, respectively. If vertices i and j form an edge, we denote that $i \leftrightarrow j$, and call i and j neighboring vertices. We denote by x_i the current coordinates of vertex i in d -dimensional Euclidean space. Typically $d = 2$ or 3 .

The aim of graph drawing is to find x_i for all $i \in V$ so that the resulting drawing gives a good visual representation of the connectivity information between vertices. One way to solve this problem is to turn the graph drawing problem into that of finding a minimal energy configuration of a physical system. Within this framework, two popular methods, the spring-electrical model [4, 7], and the stress model [5], are the most well-known.

We first discuss the spring-electrical model, and the multilevel approach and force approximation techniques that make this model suitable for large graphs. We note that the multilevel approach is not limited to the spring-electrical model, but for convenience we introduce it in the context of this model. We then discuss the stress model and classical MDS (strain model), and look at attempts to apply these models to large graphs. Finally we present high-dimensional embeddings and Hall’s algorithm for large graph layout.

Within the graph drawing literature, the name “force-directed algorithm” has often been used with the both spring-electrical and the spring models. Strictly speaking, the name “force-directed algorithm” is appropriate when we talk about a procedure that involves calculating forces exerted on vertices, moving them along the direction of the forces, and repeating until the system comes to an equilibrium state. Therefore we shall only use the term “force-directed algorithm” when referring to this iterative procedure.

19.2.1 Spring-Electrical Model

The spring-electrical model [4] represents the graph drawing problem by a system of electrically charged vertices attracted to each other by springs; vertices also repel each other via electrical forces. Here, following [7], the attractive spring force exerted on vertex i from its neighbor j is proportional to the squared distance between these two vertices,

$$F_a(i, j) = -\frac{\|x_i - x_j\|^2}{K} \frac{x_i - x_j}{\|x_i - x_j\|}, \quad i \leftrightarrow j,$$

where K is a parameter related to the nominal edge length of the final layout. The repulsive electrical force exerted on vertex i from any vertex j is inversely proportional to the distance between these two vertices,

$$F_r(i, j) = \frac{K^2}{\|x_i - x_j\|} \frac{x_i - x_j}{\|x_i - x_j\|}, \quad i \neq j.$$

The energy of this physical system [23] is

$$E(x) = \sum_{i \leftrightarrow j} \|x_i - x_j\|^3 / (3K) - \sum_{i \neq j} K^2 \ln(\|x_i - x_j\|),$$

with its derivatives a combination of the attractive and repulsive forces. We note that there are other variations of this force model. See, e.g., [4, 24].

The spring-electrical model can be solved with the aforementioned force-directed algorithm by starting from a random layout, calculating the combined attractive and repulsive forces on each vertex, and moving the vertices along the direction of the force for a certain step length. This process is repeated, with the step length decreasing every iteration, until the layout stabilizes. The following algorithm starts with a random, or user supplied, initial layout x .

Algorithm 1: Force-directed algorithm

- ForceDirectedAlgorithm(G, x, tol) {
 - *converged* = *FALSE*;
 - *step* = initial step length;
 - while (*converged* equals *FALSE*) {
 - * $x^0 = x$;
 - * for $i \in V$ {
 - $f = 0$;
 - for $(j \leftrightarrow i)$ $f := f + F_a(i, j)$;
 - for $(j \neq i, j \in V)$ $f := f + F_r(i, j)$;
 - $x_i := x_i + step * (f / \|f\|)$;
 - * }
 - * *step* := *update_steplength*(*step*, x, x^0);

```

    * if ( $\|x - x^0\| < tol * K$ ) converged = TRUE;
  - }
  - return x;

• }

```

This procedure can be enhanced by an adaptive step length updating scheme [10, 25], and usually works well for small graphs.

For large graphs, this simple iterative procedure is not sufficient to overcome the many local minima that often exist in the space of all possible layouts. Instead, a multilevel approach has to be used (Section 19.2.1.2). Furthermore, a nested space partitioning data structure is needed to approximate the all-to-all electrical force so as to reduce the quadratic complexity to $O(|V| \log |V| + |E|)$ (Section 19.2.1.1). Combining these two powerful techniques results in efficient implementations of the spring-electrical model [9, 10] that are capable of handling graphs of millions of vertices and edges [26].

19.2.1.1 Fast Force Approximation

Each iteration of the force-directed algorithm (Algorithm 1) involves two loops. The outer loop iterates over each vertex. Of the two inner loops, the latter involves calculation of all-to-all repulsive forces, and $O(|V|)$ force calculations are needed for every vertex. Thus the overall complexity is $O(|V|^2)$.

The repulsive force calculation resembles the n -body problem in physics, which is well-studied. One of the widely used techniques to approximate the repulsive forces in $O(n \log n)$ time with good accuracy, but without ignoring long range forces, is to treat groups of far away vertices as supernodes, using a suitable data structure [27]. This idea was adopted by Tunkelang [12] and Quigley [11]. They both used an quadtree (2D) or octree (3D) data structure.

For simplicity, hereafter we use the term quadtree exclusively, which should be understood as octree for 3D. A quadtree data structure is constructed by first forming a square that encloses all vertices. This is the level 0 square. This square is subdivided into 4 squares if it contains more than 1 vertex, and forms the level 1 squares. This process is repeated until each square contains no more than 1 vertex. Figure 19.3 (left) shows a quadtree on the `jagmesh1` graph.

The quadtree forms a recursive grouping of vertices, and can be used to efficiently approximate the repulsive force in the spring-electrical model. The idea is that in calculating the repulsive force on a vertex i , if a group of vertices, S , lies in a square that is sufficiently “far” from i , the whole group can be treated as a supernode. Otherwise we traverse down the hierarchy and examine the four sibling squares. The supernode is assumed to be situated at the center of gravity of the group, $x_S = (\sum_{j \in S} x_j) / |S|$. The repulsive force on vertex i from this supernode is

$$f_r(i, S) = \frac{K^2 |S|}{\|x_i - x_S\|} \frac{x_i - x_S}{\|x_i - x_S\|}.$$

It remains to define what “far” means. Following [11, 12], the supernode S is far away from vertex i , if the width d_S of the square that contains the supernode is small, compared with the distance between the supernode and the vertex i ,

$$\frac{d_S}{\|x_i - x_S\|} \leq \theta. \quad (19.1)$$

This inequality is called the Barnes-Hut opening criterion, and was originally formulated by Barnes and Hut [27]. Here $\theta \geq 0$ is a parameter. The smaller the value of θ , the more accurate the approximation to the repulsive force, and the larger the number of force calculations. A typical value that works well in practice is $\theta = 1.2$.

The quadtree data structure allows efficient identification of all the supernodes that satisfy (19.1). The process starts from the level 0 square. Each square is checked, and recursively opened, until the inequality (19.1) is satisfied. Figure 19.3 (right) shows all the supernodes (the squares) and the vertices these supernodes consist of, with reference to vertex i located at the top-middle part of the graph. In this case there are 936 vertices, and 32 supernodes.

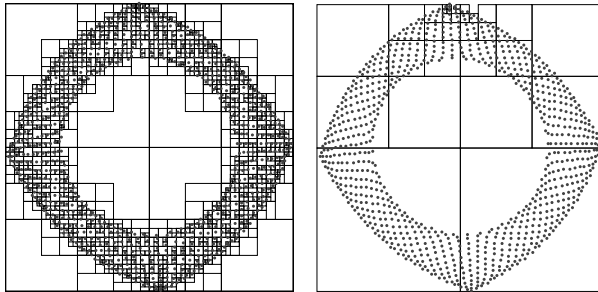


FIGURE 19.3: An illustration of the quadtree data structure. Left: the overall quadtree. Right: supernodes with reference to a vertex at the top middle part of the graph, with $\theta = 1$.

Under a reasonable assumption [27, 28] of the distribution of vertices, it can be proved that building the quadtree takes a time complexity of $O(|V| \log |V|)$. Finding all the supernodes with reference to a vertex i can be done in a time complexity of $O(\log |V|)$. Overall, using a quadtree structure to approximate the repulsive force, the complexity for each iteration of the force-directed Algorithm 1 is reduced from $O(|V|^2)$ to $O(|V| \log |V|)$. This force approximation scheme can be further improved by considering force approximation at supernode-supernode level instead of vertex-supernode level [29].

A force approximation algorithm with the same $O(|V| \log |V|)$ complexity but that is independent of the distribution of vertices is the multipole method [30], Hachul and Jünger applied this force approximation to graph drawing [9].

19.2.1.2 Multilevel Approach

While fast approximation of long range forces resolves the quadratic complexity of the force-directed algorithm for the spring-electrical model, it does not change the fact that the algorithm repositions one vertex at a time, without a “global view” of the layout. Due to the fact that this physical system of springs and electrical charges has many local minimum configurations, applying the force directed algorithm directly to a random initial layout is unlikely to give an optimal final layout. A multilevel approach can overcome this limitation. In this approach, a sequence of successively smaller graphs are generated, each captures the essential connectivity information of its parent. Global optimal layout can be found much more easily on a small graph, which are then used as a starting layout for its parent. From this initial layout, further refinement is carried out to achieve the optimal layout for the parent.

A multilevel approach has been used in many large-scale combinatorial optimization problems, such as graph partitioning [31, 32, 33], matrix ordering [34, 35], the traveling salesman problem [36], and was proved to be a very useful meta-heuristic tool [37]. A multilevel approach was later used in graph drawing [13, 38, 39, 40]. Note that a multilevel approach is not limited to the spring-electrical model, but for convenience we are introducing it in the context of this model.

A multilevel approach has three distinctive phases: coarsening, coarsest graph layout, and prolongation and refinement. In the coarsening phase, a series of coarser and coarser graphs, $G^0 = G, G^1, \dots, G^l$, are generated, each coarser graph G^{k+1} encapsulates the information needed to layout its “parent” G^k , while containing fewer vertices and edges. The coarsening continues until a graph with only a small number of vertices is reached. The optimal layout for the coarsest graph can be found cheaply. The layout on the coarser graphs are recursively prolonged to the finer graphs, with further refinement at each level.

Graph coarsening and initial layout is the first phase in the multilevel approach. There are a number of ways to coarsen an undirected graph. One often used method is based on edge collapsing (EC) [31, 32, 33]. In this scheme, a maximal independent edge set (MIES) is selected. This is a maximal set of edges, with no edges incident to the same vertex. The vertices correspond to this edge set form a maximal matching. Each edge, and its corresponding pair of vertices, are coalesced into a new vertex. [Figure 19.4 \(a\)](#) illustrates MIES and the result of coarsening using edge collapsing.

Alternatively, coarsening can be performed based on a maximal independent vertex set (MIVS) [41]. This is a maximal set of vertices such that no

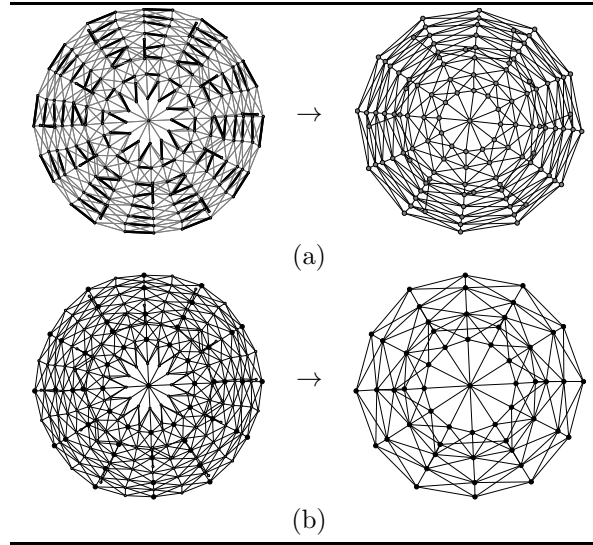


FIGURE 19.4: An illustration of graph coarsening: (a) Left: original graph with 229 vertices. Edges in a maximal independent edge set are thickened. Right: a coarser graph with 115 vertices resulted from coalescing thickened edges; (b) Left: original graph with 229 vertices. Vertices in a maximal independent vertex set are darkened. Right: a coarser graph with 55 vertices resulted from the maximal independent vertex set.

two vertices in the set are connected by an edge in the graph. Edges of the coarser graph are formed through linking two vertices in the maximal independent vertex set by an edge if their distance apart is no greater than three. Figure 19.4 (b) illustrates MIVS and the result of coarsening using maximal independent vertex set.

Coarsest graph layout is carried out at the end of the recursive coarsening process. Coarsening is performed repeatedly until the graph is very small; at that point we can layout the graph using a suitable algorithm, for example, the force-directed Algorithm 1. Because the graph on the coarsest level is very small, it is likely that it can be laid out optimally.

The prolongation and refinement step is the third phase in a multilevel procedure. The layout on the coarser graphs are recursively interpolated to the finer graphs, with further refinement at each level.

Row “spring electrical” of Figure 19.5 shows drawings of two graphs using this multilevel force-directed algorithm [10]. The drawings are of good quality for both `jagmesh1`, a mesh-like graph, and `1138_bus`, a sparser graph. The spring-electrical model does suffer slightly from “warping effect” [42]. For example, for the `jagmesh1`, vertices are closer together near the bound-

ary, compared to the interior of the mesh. This effect can be mitigated using post-processing techniques [42].

19.2.1.3 An Open Problem: More Robust Coarsening Schemes

The multilevel approach can work efficiently, provided that the coarsening scheme is able to generate a coarsened graph with many fewer vertices than its parent graph. The aforementioned multilevel algorithm was found to work well for a lot of graphs from the graph drawing literature [10]. However, when applied to the University of Florida Sparse Matrix Collection [43], we found that, for some matrices, the coarsening scheme could not coarsen sufficiently, and the multilevel scheme has to be terminated prematurely, which results in poor drawings. An example is shown in Figure 19.6. On the left of the figure is the sparsity pattern of the matrix `gupta1`. From this plot we can conclude that this matrix describes a graph of three groups of vertices: those represented by the top 1/3 of the rows in the matrix plot, the middle 1/3 of the rows, and the rest. Vertices in each group are all connected to a selected few in that group; these links are seen as dense horizontal and vertical bars in the matrix plot. At the same time, vertices in the top group are connected to those in middle group, which in turn are connected to those in the bottom group, as represented by the off-diagonal lines parallel to the diagonal. However, the graph drawing in the middle of Figure 19.6 shows none of these structures.

This graph exemplifies many of the problematic graphs: they contain star-graph like substructures, with a lot of vertices all connected to a few vertices. Such structures pose a challenge to the usual graph coarsening schemes. These schemes are not able to coarsen graphs like that adequately. For example, Figure 19.7 (left) shows such a graph, with $k = 10$ vertices on the outskirts all connected to two vertices at the center. Because of this structure, any MIES can only contain two edges (the two thick edges in Figure 19.7). When the end vertices of the MIES are merged at the center of each edge, the resulting graph has only two fewer vertices. Therefore if k is large enough, the coarsening can be arbitrarily slow ($k/(k + 2) \rightarrow 1$ as $k \rightarrow \infty$).

One solution proposed [43] is to find vertices that share the same neighbors. These vertices are matched in pairs. The usual MIES scheme is then used to match the remaining unmatched vertices. Finally the matched vertices are merged to get the coarsened graph. The scheme is able to overcome the slow coarsening problem associated with graphs having star-graph like substructures. Applying this scheme to the graph in Figure 19.8 (left) resulted in a graph with 1/2 the number of vertices (Figure 19.8 right). With this new coarsening scheme, we are able to layout many more graphs aesthetically. For example, when applied to the `gupta1` matrix, the drawing at Figure 19.6 (right) reveals the correct visual structures as we expected, with three groups of vertices, each connected to a few within the group, and a linear connectivity relation among the groups. This new coarsening scheme is able to handle many of the graphs MIES or MIVS based coarsening schemes

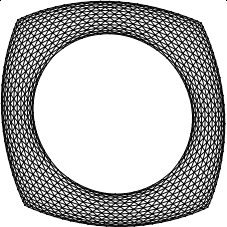
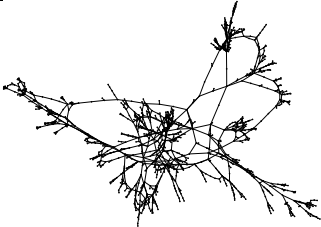
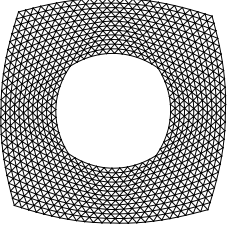

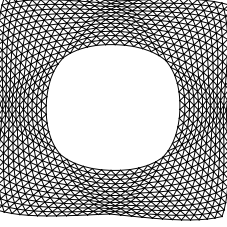
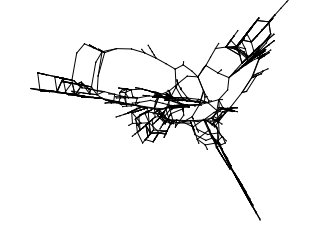
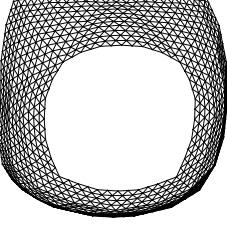
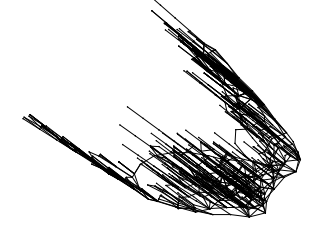
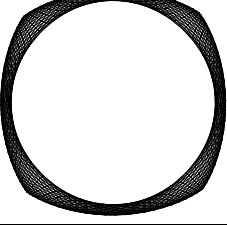
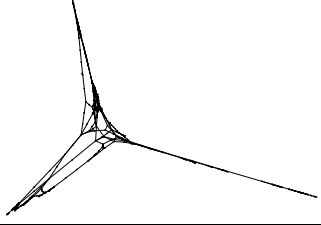
Algorithms	jagmesh1	1138_ bus
spring electrical		
stress		
classical MDS		
HDE		
Hall's		

FIGURE 19.5: An overview of all algorithms described in the chapter, applied to two graphs, jagmesh1 and 1138_bus.

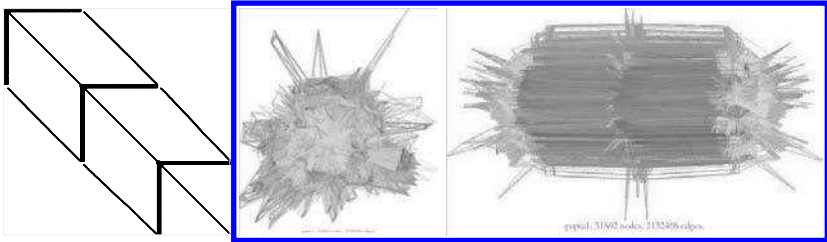


FIGURE 19.6: Matrix plot of `gupta1` matrix (left) and the initial graph drawing (middle). After applying the new coarsening scheme, the graph drawing reflects the structure of the matrix much better (right).

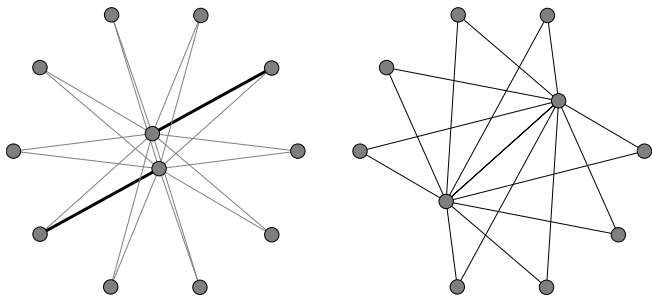


FIGURE 19.7: A maximal independent edge set based coarsening scheme fails to coarsen sufficiently a star-graph like structure: a maximal independent edge set (thick edges) (left); when merging the end vertices of the edge set at the middle of these edges, the resulting coarsened graph only has 2 fewer vertices (right).

fail to. But a more general and robust coarsening scheme that works for all graphs is still an open problem. A promising route for investigation may be the coarsening schemes associated with algebraic multigrid (AMG) [35].

19.2.2 Stress and Strain Models

While the spring-electrical model is scalable and can layout graphs of millions of nodes in minutes, it does have the limitation of not coping well when edges have predefined lengths. It is possible to assign weaker attractive force and stronger repulsive force for longer edges, but such treatment is not as principled and direct as the following stress model.

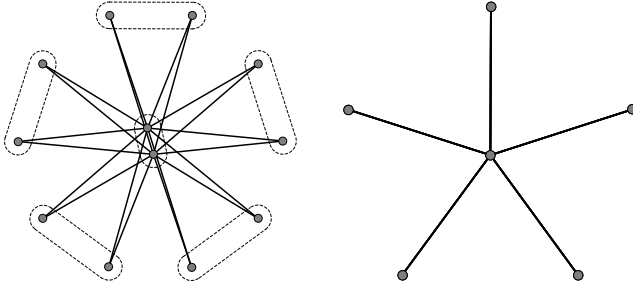


FIGURE 19.8: Matching and merging vertices with the same neighborhood structure (left, with dashed line enclosing matching vertex pairs) resulted in a new graph (right) with $1/2$ the number of vertices.

19.2.2.1 Stress Model

The stress model assumes that there are springs connecting all pairs of vertices of the graph, with the ideal spring length equal to the predefined edge length. The energy of this spring system is

$$\sum_{i \neq j} w_{ij} (\|x_i - x_j\| - d_{ij})^2, \quad (19.2)$$

where d_{ij} is the ideal distance between vertices i and j , and w_{ij} is a weight factor, typically $1/d_{ij}^2$. The layout that minimizes the above stress energy is an optimal layout of the graph according to this model.

The stress model has its root in Multidimensional Scaling (MDS) [44, 45]. Note that typically we only know the ideal distance between vertices that share an edge, which is usually taken to be one for graphs without predefined edge length. Alternatively, it has been proposed to set the edge length equal the total number of non-common neighbors of the two end vertices [46]. For other vertex pairs, one way to define d_{ij} is to take it as the shortest distance between vertex i and j . The practice of taking the shortest graph distance as the ideal edge length date back at least to 1980 in social network layout [47], and in graph drawing using classical MDS [45], but is often attributed to Kamada and Kawai [5].

There are several ways to minimize (19.2). A force-directed algorithm (Algorithm 1) can be used. The repulsive/attractive force exerted on vertex i from the spring between vertices i and j is

$$F(i, j) = -w_{ij} (\|x_i - x_j\| - d_{ij}) \frac{x_i - x_j}{\|x_i - x_j\|}, \quad i \neq j. \quad (19.3)$$

Stress-majorization: a stress-majorization technique [46] can be employed to solve the stress model. Consider the cost function (19.2),

$$\sum_{i \neq j} w_{ij} (\|x_i - x_j\| - d_{ij})^2 = \sum_{i \neq j} (w_{ij} \|x_i - x_j\|^2 - 2d_{ij} w_{ij} \|x_i - x_j\| + w_{ij} d_{ij}^2)$$

On the right hand side, the first and third terms are either constant or quadratic with regard to x , except the second one. Using the Cauchy-Schwartz inequality, $(x_i - x_j)^T (y_i - y_j) \leq \|x_i - x_j\| \|y_i - y_j\|$, we can bound the cost function by

$$g(x, y) = \sum_{i \neq j} \left(w_{ij} \|x_i - x_j\|^2 - 2d_{ij} w_{ij} \frac{(x_i - x_j)^T (y_i - y_j)}{\|y_i - y_j\|} + w_{ij} d_{ij}^2 \right),$$

with the bound tight when $y = x$. The idea of stress majorization is to minimize a sequence of quadratic function $g(x, y^k)$, with $y^0 = x^0$ the initial layout, and subsequent y^k the result of minimizing $g(x, y^{k-1})$, $k = 1, 2, \dots$

The minimum of the quadratic function $g(x, y)$ is derived by setting $\partial_{x_i} g(x, y) = 0$, giving

$$L_w x = L_{w,d} y \quad (19.4)$$

where the weighted Laplacian matrix L_w has elements

$$(L_w)_{ij} = \begin{cases} \sum_{i \neq l} w_{il}, & i = j \\ -w_{ij}, & i \neq j \end{cases}$$

and matrix $L_{w,d}$ has elements

$$(L_{w,d})_{ij} = \begin{cases} \sum_{i \neq l} w_{il} d_{il} / \|y_i - y_l\|, & i = j \\ -w_{ij} d_{ij} / \|y_i - y_j\|, & i \neq j \end{cases}$$

In summary, the process of finding a minima of (19.2) becomes that of solving a series of linear systems (19.4), with the solution x served as y in the next iteration. This iterative process is found to be quite robust, although for large graphs, it still benefits from a good initial layout. Row “stress” of [Figure 19.5](#) gives drawings of the stress model. It performed very well on both graphs.

19.2.2.2 Strain Model (Classical MDS)

The strain model, also known as classical MDS [48], predates the stress model. Classical MDS tries to fit the inner product of positions, instead of the distance between points. Specifically, assume that the final embedding is centered around the origin:

$$\sum_{i=1}^{|V|} x_i = 0.$$

Furthermore, assume that in the ideal case, the embedding fits the distance exactly:

$$\|x_i - x_j\| = d_{ij}. \quad (19.5)$$

It is then easy to prove [49] that the product of the positions, $b_{ij} = x_i^T x_j$, can be expressed as the squared and double centered distance,

$$b_{ij} = x_i^T x_j = -1/2 \left(d_{ij}^2 - \frac{1}{|V|} \sum_{k=1}^{|V|} d_{kj}^2 - \frac{1}{|V|} \sum_{k=1}^{|V|} d_{ik}^2 + \frac{1}{|V|^2} \sum_{k=1}^{|V|} \sum_{l=1}^{|V|} d_{kl}^2 \right). \quad (19.6)$$

In real data, it is unlikely that we can find an embedding that fits the distances perfectly, hence assumption (19.5) does not stand. But we would still expect that b_{ij} is a good approximation of $x_i^T x_j$. Therefore we try to find an embedding that minimizes the difference between the two,

$$\min_X \|X^T X - B\|_F, \quad (19.7)$$

where X is the $|V| \times d$ dimensional matrix of x_i 's, B is the $|V| \times |V|$ symmetric matrix of b_{ij} 's, and $\|\cdot\|_F$ is the Frobenius norm. If the eigen-decomposition of B is $B = Q^T \Lambda Q$, then the solution to (19.7) is $X = \Lambda_d^{1/2} Q$, where Λ_d is the diagonal matrix of Λ , with all but the d largest eigenvalues on the diagonal set to zero. Because the strain model does not fit the distance directly, graph drawings given by solving this model are not as satisfactory as those using the stress model [47], but it can be used as a good starting point for the stress model. Row “classical MDS” of Figure 19.5 gives drawings using classical MDS. It performed well on the mesh like `jagmesh1` graph, but not so well on the sparser `1138_bus` graph, where vertices cling close to each other, making some details of the graph unclear.

19.2.2.3 MDS for Large Graphs

In a typical usage of the stress or strain models, the ideal distance between all pairs of vertices has to be calculated, which requires an all-pair shortest path calculation. Using Johnson's algorithm, this needs $O(|V|^2 \log |V| + |V||E|)$ computation, and a storage of $O(|V|^2)$. Therefore for very large graphs, this formulation is computationally expensive and memory prohibitive. A number of strategies [8, 46, 50] have been proposed to approximately minimize (19.2) or (19.7).

A **multiscale algorithm** [38, 51] applies the multilevel approach in solving the stress model. In the GRIP algorithm [38], graph coarsening is carried out through vertex filtration, an idea similar to the maximal independent vertex set. A sequence of vertex sets, $V^0 = V \subset V^1 \subset V^2, \dots, \subset V_L$, is generated. However, coarser graphs are not constructed explicitly. Instead, a vertex set

V^k at level k of the vertex set hierarchy is constructed so that distance between vertices is at least $2^{k-1} + 1$. On each level k , the stress model is solved by a force-directed procedure. The spring force (19.3) on each vertex $i \in V^k$ is calculated by considering a neighborhood $N^k(i)$ of this vertex, with $N^k(i)$ the set of vertices in level k , chosen so that the total number of vertices in this set is $O(|E|/|V^k|)$. Thus the force calculation on each level can be done in time $O(|E|)$. It was proved that with this multilevel procedure and the localized force calculation algorithm, for a graph of bounded degree, the algorithm has close to linear computational and memory complexity.

LandmarkMDS [50] approximates the result of classical MDS by choosing $k \ll |V|$ vertices as landmarks, and calculating a layout of these vertices using the classical MDS, based on distances among these vertices. The k landmarks are chosen to be well dispersed in the graph. One possibility is to use a MaxMin strategy where the first landmark is randomly chosen, then each subsequent landmark is chosen to be furthest away from the previous landmarks, which is a well-known 2-approximation to the k -center problem. The positions of the rest of vertices are then calculated by placing these vertices at the weighted barycenter, with weights based on distances to the landmarks. So essentially classical MDS is applied to a $k \times k$ submatrix of the $|V| \times |V|$ matrix B . The complexity of this algorithm is $O(k|V| + k^2)$, and only $O(k|V|)$ distances need to be stored.

PivotMDS [8], on the other hand, takes a $|V| \times k$ submatrix C of B . The two eigenvectors corresponding to the largest eigenvalues of the $|V| \times |V|$ matrix CC^T are then calculated using power iterations and used as the x - and y - coordinates. By an algebraic argument, if v is an eigenvector of the $k \times k$ matrix C^TC , then

$$CC^T(Cv) = C(C^TCv) = \lambda(Cv),$$

hence Cv is an eigenvector of CC^T . Therefore PivotMDS proceeds by finding the largest eigenvectors of the smaller $k \times k$ matrix C^TC , then projects back to $|V|$ -dimensional space by multiplying them with C . Using this technique, the overall complexity is similar to LandmarkMDS, but unlike LandmarkMDS, PivotMDS utilizes distances between landmark vertices and other vertices in the matrix product. In practice PivotMDS was found to give drawings that are closer to the classical MDS than LandmarkMDS, and both are very efficient when used with a small number (e.g., $k \approx 50$) of pivots/landmarks.

It is worth pointing out that, for sparse graphs, there is a limitation in both algorithms. For example, if the graph is a tree, and pivots/landmarks are chosen to be non-leaves, then two leaf nodes that have the same parent will have exact the same distances to any of the pivots/landmarks, consequently their final positions based on these algorithms will also be the same. This problem may be alleviated by utilizing the layout given by these algorithms as an initial placement for a stress model based algorithm, but taking only a sparse set of terms in the stress function [46, 47].

19.2.3 High-Dimensional Embedding

The high-dimensional embedding (HDE) algorithm [52] finds coordinates of vertices in a k -dimensional space, then projects back to two or three dimensional space.

First, a k -dimensional coordinate system is created based on k -centers, where k -centers are chosen as in LandmarkMDS/PivotMDS (Section 19.2.2.3). The graph distances from each vertex to the k -centers form a k -dimensional coordinate system. The $|V|$ coordinate vectors form an $|V| \times k$ matrix Y , where the i -th row of Y is the k -dimensional coordinates for vertex i .

Since it is only possible to visualize in two or three dimensions, and since the coordinates may be correlated, the coordinates are projected back to two or three dimensions by suitable linear combinations that minimize correlations. To make this projection shift-invariant, Y is first normalized so that the center of gravity of the vertices is at the origin, i.e.,

$$Y := Y - \frac{1}{|V|} e e^T Y,$$

where e is the $|V|$ -dimensional vector of all 1's.

Assume we want to project the k dimensional coordinate system into 2-dimensions. Let v_1 and v_2 be two k -dimensional linear combination vectors, so that Yv_1 and Yv_2 form the x - and y - coordinates. The two linear combinations should be uncorrelated, so we take them to be orthogonal to each other:

$$v_1^T Y^T Y v_2 = 0.$$

Furthermore, each should be as far away from 0 as possible, so we want to maximize

$$\frac{v_l^T Y^T Y v_l}{\|v_l\|}, \quad l = 0, 1.$$

These can be achieved by taking v_1 and v_2 to be the two eigenvectors that correspond to the two largest eigenvalues of the $k \times k$ symmetric matrix $Y^T Y$. This process of choosing highly uncorrelated vectors out of high dimensional data is known as principal component analysis. The final x - and y - coordinates are given by Yv_1 and Yv_2 .

HDE clearly has many commonalities to PivotMDS. Each utilizes k -centers, and each finds the largest eigenvectors of a $k \times k$ dimensional matrix derived by multiplying the transpose of a $|V| \times k$ matrix with itself. The main difference is that in high-dimensional embedding this $|V| \times k$ matrix consists of distances to the k -centers, while in PivotMDS this matrix consists of distances squared and double centered (see (19.6)). High-dimensional embedding suffers from the same limitation as PivotMDS and LandmarkMDS, in that it is not able to layout sparse graphs as well as mesh-like graphs. Row “HDE”

of Figure 19.5 gives drawings using HDE. It performs particularly badly on 1138_bus graph, with vertices close to each other, obscuring many details.

19.2.4 Hall's Algorithm

In 1970, Hall [53] remarked that many sequencing and placement problems could be characterized as finding locations of points which minimize the weighted sum of square distances. In our notation, what he proposed was to minimize

$$\sum_{i \leftrightarrow j} w_{ij} \|x_i - x_j\|^2, \text{ subject to } \sum_{k=1}^{|V|} x_k^2 = 1$$

where x_i is the 1-dimensional coordinate value for vertex i . The objective function can be written as

$$\sum_{i \leftrightarrow j} w_{ij} \|x_i - x_j\|^2 = x^T L_w x,$$

with $x = \{x_1, x_2, \dots, x_{|V|}\}$. Here L_w is the weighted Laplacian matrix with elements

$$(L_w)_{ij} = \begin{cases} \sum_{(i,l) \in E} w_{il}, & i = j \\ -w_{ij}, & i \neq j \end{cases}$$

For a connected graph, the Laplacian is positive semi-definite with one eigenvalue of 0 corresponding to the trivial eigenvector of all 1's. The solution x of the minimization problem is the eigenvector corresponding to the smallest positive eigenvalue of the weighted Laplacian L_w . We can achieve a 2-dimensional layout by taking the two eigenvectors corresponding to the two smallest positive eigenvalues. Row "Hall's" of Figure 19.5 gives drawings employing Hall's algorithm. It performed reasonably well on the mesh like jagmesh1 graph, but is almost useless on the sparser 1138_bus graph.

Koren et al. [54] proposed an extremely fast algorithm for calculating the two extreme eigenvectors using a multilevel algorithm. The algorithm is called ACE (Algebraic multigrid Computation of Eigenvectors). Using this algorithm, they were able to layout graphs of millions of nodes in less than a minute. However, the fundamental weakness of Hall's algorithm on sparse graphs remains.

19.3 Examples of Large Graph Drawings

In this section we give example drawings of some large graphs. These graphs are drawn using a multilevel spring-electrical model based code, **sfdp** [10], available from **graphviz** [55].

One rich source of large graphs is the University of Florida Sparse Matrix Collection [43]. This is a collection of 2272 (as of January, 2010) sparse matrices. The largest matrix has 27 million rows and columns. Graph visualization provides a way to visualize these matrices and get a glimpse of the application underneath these matrices. Figure 19.9 gives the drawing of four graphs in this collection. For instance, by its name, the **boneS10** graph could be from a model of the bone. The porous structure can be seen clearly from the drawing. More drawings for all the matrices in the collection can be found at [26].

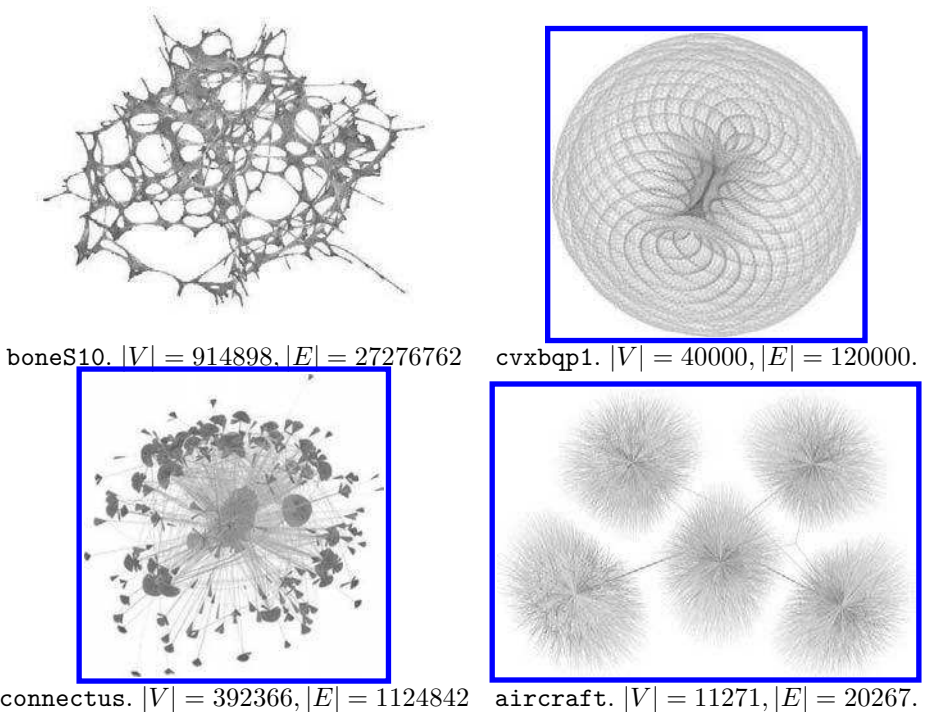


FIGURE 19.9: (See color insert.) Drawing of some graphs from the University of Florida Sparse Matrix Collection [43]. More drawings can be found at [26].

strong links with topics in algebraic graph theory, and a number of open problems remain. One of these is the need for a robust coarsening scheme (Section 19.2.1.3). There are many topics we could not possibly cover in the limited space available. For example, laying out and visualizing a dynamically changing graph remains a challenge [57]. Another challenge is how to draw and explore very complex real world graphs of a “small-world” nature [16]. This often requires not just a good layout algorithm, but also visualization techniques such as edge-bundling [58], interactive exploration systems [18, 59], and additional visual aids such as maps and bubble-sets [60, 61] – all fertile grounds for problems of a combinatorial nature.

Acknowledgments

The author would like to thank Emden Gansner and Stephen North, as well as anonymous referees, for valuable comments.

Bibliography

- [1] Euler, L., “Commentarii academiae scientiarum Petropolitanae,” *Solutio problematis ad geometriam situs pertinentis*, Vol. 8, 1741, pp. 128–140.
- [2] Tutte, W., “How to draw a graph,” *Proceedings of the London Mathematical Society*, Vol. 13, 1963, pp. 743–768.
- [3] Sugiyama, K., Tagawa, S., and Toda, M., “Methods for Visual Understanding of Hierarchical Systems,” *IEEE Trans. Systems, Man and Cybernetics*, Vol. SMC-11, No. 2, 1981, pp. 109–125.
- [4] Eades, P., “A heuristic for graph drawing,” *Congressus Numerantium*, Vol. 42, 1984, pp. 149–160.
- [5] Kamada, T. and Kawai, S., “An algorithm for drawing general undirected graphs,” *Information Processing Letters*, Vol. 31, 1989, pp. 7–15.
- [6] Gansner, E. R., Koutsofios, E., North, S. C., and Vo, K. P., “A Technique for Drawing Directed Graphs,” *IEEE Trans. Softw. Eng.*, Vol. 19, 1993, pp. 214–230.
- [7] Fruchterman, T. M. J. and Reingold, E. M., “Graph drawing by force directed placement,” *Software - Practice and Experience*, Vol. 21, 1991, pp. 1129–1164.

- [8] Brandes, U. and Pich, C., “Eigensolver methods for progressive multidimensional scaling of large data,” *Proc. 14th Intl. Symp. Graph Drawing (GD '06)*, Vol. 4372 of *LNCIS*, 2007, pp. 42–53.
- [9] Hachul, S. and Jünger, M., “Drawing large graphs with a potential field based multilevel algorithm,” *Proc. 12th Intl. Symp. Graph Drawing (GD '04)*, Vol. 3383 of *LNCIS*, Springer, 2004, pp. 285–295.
- [10] Hu, Y. F., “Efficient and High Quality Force-Directed Graph Drawing,” *Mathematica Journal*, Vol. 10, 2005, pp. 37–71.
- [11] Quigley, A., *Large scale relational information visualization, clustering, and abstraction*, Ph.D. thesis, Department of Computer Science and Software Engineering, University of Newcastle, Australia, 2001.
- [12] Tunkelang, D., *A Numerical Optimization Approach to General Graph Drawing*, Ph.D. thesis, Carnegie Mellon University, 1999.
- [13] Walshaw, C., “A Multilevel Algorithm for Force-Directed Graph Drawing,” *J. Graph Algorithms and Applications*, Vol. 7, 2003, pp. 253–285.
- [14] Batagelj, V., “Visualization of large networks,” *Encyclopedia of Complexity and Systems Science*, edited by R. A. Meyers, Springer, New York, 2009.
- [15] Krujaa, E., Marks, J., Blair, A., and Waters, R., “A Short Note on the History of Graph Drawing,” *Proc. 9th Intl. Symp. Graph Drawing (GD '01)*, Springer-Verlag, London, UK, 2002, pp. 272–286.
- [16] Watts, D. and Strogatz, S., “Collective dynamics of “small-world” networks,” *Nature*, Vol. 393, 1998, pp. 440–442.
- [17] Quigley, A. and Eades, P., “FADE: Graph Drawing, Clustering, and Visual Abstraction,” *LNCIS*, Vol. 1984, 2000, pp. 183–196.
- [18] Gansner, E. R., Koren, Y., and North, S., “Topological fisheye views for visualizing large graphs,” *IEEE Transactions on Visualization and Computer Graphics*, Vol. 11, 2005, pp. 457–468.
- [19] Herman, I., Melancon, G., and Marshall, M. S., “Graph visualization and navigation in information visualization: A survey,” *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, Vol. 6, 2000, pp. 24–43.
- [20] Lamping, J., Rao, R., and Pirolli, P., “A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies,” *SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS (CHI '95)*, ACM, 1995, pp. 401–408.

- [21] Munzner, T. and Burchard, P., “Visualizing the structure of the World Wide Web in 3D hyperbolic space,” *VRML '95: Proceedings of the first symposium on Virtual reality modeling language*, ACM, New York, USA, 1995, pp. 33–38.
- [22] Munzner, T., “Exploring Large Graphs in 3D Hyperbolic Space,” *IEEE Comput. Graph. Appl.*, Vol. 18, 1998, pp. 18–23.
- [23] Noack, A., “An Energy Model for Visual Graph Clustering,” *Proceedings of the 11th International Symposium on Graph Drawing (GD 2003)*, Vol. 2912 of *LNCS*, Springer, 2004, pp. 425–436.
- [24] Battista, G. D., Eades, P., Tamassia, R., and Tollis, I. G., *Algorithms for the visualization of Graphs*, Prentice-Hall, 1999.
- [25] Bruss, I. and Frick, A., “Fast Interactive 3-D Graph Visualization,” *LNCS*, Vol. 1027, 1995, pp. 99–11.
- [26] Hu, Y. F., “A gallery of large graphs,” <http://research.att.com/~yifanhu/GALLERY/GRAPHS/index.html>.
- [27] Barnes, J. and Hut, P., “A hierarchical $O(N \log N)$ force-calculation algorithm,” *Nature*, Vol. 324, 1986, pp. 446–449.
- [28] Pfalzner, S. and Gibbon, P., *Many-Body Tree Methods in Physics*, Cambridge University Press, Cambridge, UK, 1996.
- [29] Burton, A., Field, A. J., and To, H. W., “A cell-cell Barnes Hut algorithm for fast particle simulation,” *Australian Computer Science Communications*, Vol. 20, 1998, pp. 267–278.
- [30] Greengard, L. F., *The rapid evaluation of potential fields in particle systems*, The MIT Press, Cambridge, Massachusetts, 1988.
- [31] Gupta, A., Karypis, G., and Kumar, V., “Highly scalable parallel algorithms for sparse matrix factorization,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, 1997, pp. 502–520.
- [32] Hendrickson, B. and Leland, R., “A multilevel algorithm for partitioning graphs,” Technical Report SAND93-1301, Sandia National Laboratories, Albuquerque, NM, 1993, Also in Proceeding of Supercomputing'95 (http://www.supercomp.org/sc95/proceedings/509_BHEN/SC95.HTM).
- [33] Walshaw, C., Cross, M., and Everett, M. G., “Parallel dynamic graph partitioning for adaptive unstructured meshes,” *Journal of Parallel and Distributed Computing*, Vol. 47, 1997, pp. 102–108.
- [34] Hu, Y. F. and Scott, J. A., “A multilevel algorithm for wavefront reduction,” *SIAM Journal on Scientific Computing*, Vol. 23, 2001, pp. 1352–1375.

- [35] Safro, I., Ron, D., and Brandt, A., “Multilevel algorithms for linear ordering problems,” *J. Exp. Algorithmics*, Vol. 13, 2009, pp. 1.4–1.20.
- [36] Walshaw, C., “A Multilevel Approach to the Travelling Salesman Problem,” *Oper. Res.*, Vol. 50, 2002, pp. 862–877.
- [37] Walshaw, C., “Multilevel refinement for combinatorial optimisation problems,” *Annals of Operations Research*, Vol. 131, pp. 325–372, 2004; originally published as Univ. Greenwich Tech. Rep. 01/IM/73.
- [38] Gajer, P., Goodrich, M. T., and Kobourov, S. G., “A fast multi-dimensional algorithm for drawing large graphs,” *LNCS*, Vol. 1984, 2000, pp. 211 – 221.
- [39] Hadany, R. and Harel, D., “A multi-scale algorithm for drawing graphs nicely,” *Discrete Applied Mathematics*, Vol. 113, 2001, pp. 3–21.
- [40] Harel, D. and Koren, Y., “A Fast Multi-Scale Method for Drawing Large Graphs,” *J. Graph Algorithms and Applications*, Vol. 6, 2002, pp. 179–202.
- [41] Barnard, S. T. and Simon, H. D., “Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems,” *Concurrency: Practice and Experience*, Vol. 6, 1994, pp. 101–117.
- [42] Hu, Y. F. and Koren, Y., “Extending the spring-electrical model to overcome warping effects,” *Proceedings of IEEE Pacific Visualization Symposium*, IEEE Computer Society, 2009, pp. 129–136.
- [43] Davis, T. A. and Hu, Y. F., “University of Florida sparse matrix collection,” *in preparation*, 2009.
- [44] Kruskal, J. B., “Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis,” *Psychometrika*, Vol. 29, 1964, pp. 1–27.
- [45] Kruskal, J. B. and Seery, J. B., “Designing network diagrams,” *Proceedings of the First General Conference on Social Graphics*, U. S. Department of the Census, Washington, D.C., July 1980, pp. 22–50, Bell Laboratories Technical Report No. 49.
- [46] Gansner, E. R., Koren, Y., and North, S. C., “Graph Drawing by Stress Majorization,” *Proc. 12th Intl. Symp. Graph Drawing (GD '04)*, Vol. 3383 of *LNCS*, Springer, 2004, pp. 239–250.
- [47] Brandes, U. and Pich, C., “An experimental study on distance based graph drawing,” *Proc. 16th Intl. Symp. Graph Drawing (GD '08)*, Vol. 5417 of *LNCS*, Springer-Verlag, 2009, pp. 218–229.
- [48] Torgerson, W. S., “Multidimensional scaling: I. Theory and method,” *Psychometrika*, Vol. 17, 1952, pp. 401–419.

- [49] Young, G. and Householder, A. S., "Discussion of a set of points in terms of their mutual distances," *Psychometrika*, Vol. 3, 1938, pp. 19–22.
- [50] de Solva, V. and Tenenbaum, J. B., "Global versus local methods in non-linear dimensionality reduction," *Advances in Neural Information Processing Systems 15*, MIT Press, 2003, pp. 721–728.
- [51] Harel, D. and Koren, Y., "A Fast Multi-Scale Method for Drawing Large Graphs," *Journal of graph algorithms and applications*, Vol. 6, 2002, pp. 179–202.
- [52] Harel, D. and Koren, Y., "Graph Drawing by High-Dimensional Embedding," *lncs*, 2002, pp. 207–219.
- [53] Hall, K. M., "An r -dimensional quadratic placement algorithm," *Management Science*, Vol. 17, 1970, pp. 219–229.
- [54] Koren, Y., Carmel, L., and Harel, D., "ACE: A Fast Multiscale Eigenvectors Computation for Drawing Huge Graphs," *INFOVIS '02: Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02)*, IEEE Computer Society, Washington, DC, USA, 2002, pp. 137–144.
- [55] Gansner, E. R. and North, S., "An Open Graph Visualization System and its Applications to Software Engineering," *Software—Practice & Experience*, Vol. 30, 2000, pp. 1203–1233.
- [56] Maddison, D. R. and (eds.), K.-S. S., "The Tree of Life Web Project," <http://tolweb.org>, 2007.
- [57] Frishman, Y. and Tal, A., "Online Dynamic Graph Drawing," *proceeding of Eurographics/IEEE VGTC Symposium on Visualization (EuroVis)*, 2007, pp. 75–82.
- [58] Holten, D. and van Wijk, J. J., "Force-Directed Edge Bundling for Graph Visualization," *Computer Graphics Forum*, Vol. 28, 2009, pp. 983–990.
- [59] Moscovich, T., Chevalier, F., Henry, N., Pietriga, E., and Fekete, J., "Topology-aware navigation in large networks," *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, ACM, New York, USA, 2009, pp. 2319–2328.
- [60] Collins, C., Penn, G., and Carpendale, S., "Bubble Sets: Revealing Set Relations with Isocontours over Existing Visualizations," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 15, No. 6, 2009, pp. 1009–1016.
- [61] Gansner, E. R., Hu, Y. F., and Kobourov, S. G., "GMap: Drawing Graphs and Clusters as Map," *IEEE Pacific Visualization Symposium*, 2010, pp. 201 – 208.