Efficient, High-Quality Force-Directed Graph Drawing

Yifan Hu

We propose a graph drawing algorithm that is both efficient and high quality. This algorithm combines a multilevel approach, which effectively overcomes local minimums, with the Barnes and Hut [1] octree technique, which approximates short- and long-range force efficiently. Our numerical results show that the algorithm is comparable in speed to Walshaw's [2] highly efficient multilevel graph drawing algorithm, yet gives better results on some of the difficult problems. In addition, an adaptive cooling scheme for the force-directed algorithms and a general repulsive force model are proposed. The proposed graph drawing algorithm and others are included with *Mathematica* 5.1 and later versions in the package `DiscreteMath`Graph`Plot`.

■ 1. Introduction

Graphs are often used to encapsulate the relationship between objects. Graph drawing enables visualization of these relationships. The usefulness of the representation is dependent on the aesthetics of the drawing. While there are no strict criteria for aesthetics, it is generally agreed that minimal edge crossing, evenly distributed vertices, and depiction of graph symmetry is desirable.

This problem has been studied extensively in the literature [3] and many approaches have been proposed. In this article we concentrate on drawing undirected graphs with straight-line edges using force-directed methods [4, 5, 6, 7]. Force-directed methods, however, are one of many classes of methods proposed for straight-edge drawing. Other methods include the spectral method [8] and the high-dimensional embedding method [9].

A force-directed algorithm models the graph drawing problem through a physical system of bodies with forces acting between them. The algorithm finds a good placement of the bodies by minimizing the energy of the system. There are many variations of force-directed algorithms. The algorithm of Fruchterman and Reigold [4], which is based on the work of [5, 6], models the graph drawing problem with a system of springs between neighboring vertices of the graph,

pulling them together. At the same time, repulsive electrical forces that exist push all vertices away from each other. The algorithm of Kamada and Kawai [7], on the other hand, associates springs between all vertices, with the ideal length of a spring proportional to the graph distance of the vertices. In a force-directed algorithm, the energy of the system is typically minimized iteratively by moving the vertices along the direction of the force. This amount may be large initially, but reduces gradually based on a "cooling schedule."

There are two limiting factors in drawing large graphs for standard force-directed algorithms. The first is that the physical model typically has many local minimums, particularly so for a large graph. Starting from a random configuration, the system is likely to settle in a local minimum. This may be improved, to a limited extent, by using a slow cooling schedule at the expense of more iterations. Nevertheless, it is practically impossible to use the standard force-directed algorithms to find a good layout of very large graphs.

The second limiting factor is the computational complexity of the standard force-directed algorithms. In the algorithm of Fruchterman and Reigold [4], for any given vertex, repulsive force from all other vertices needs to be calculated. This makes the per iteration cost of the algorithm $O(|V|^2)$, with $|V|$ the number of vertices in the graph. The algorithm of Kamada and Kawai [7] requires the calculation of the graph distance among all vertices with the force based on that. Thus the algorithm not only has a computational complexity of $O(|V| \, |E|)$, with $|E|$ the number of edges in the graph but also a memory complexity of $O(|V|^2)$, although the latter can be circumvented at the cost of repeated calculations of the graph distances on the fly, instead of storing them in memory.

To overcome the first limiting factor, a multilevel approach was proposed. This idea has been successfully used in many fields, including graph partitioning [10, 11, 12] and was found to be able to overcome the localized nature of the Kernighan–Lin algorithm. Fruchterman and Reigold [4, 1148] alluded to that type of solution when, in the context of overcoming local minimums, they stated "we suspect that if we apply a multi-grid technique that allows whole portions of the graph to be moved, it might be of some help…". Harel and Koren [13], extending the earlier work of Hadany and Harel [14], proposed the so-called multiscale approach. In that approach, a sequence of coarser and coarser graphs are formed by finding the k-centers and the distance matrix associated with them. They used the Kamada and Kawai spring model [7], thus incurring high computational complexity for distance calculation and memory. Although, for the force calculation the algorithm has a computational complexity of $O(|V| \log(|V|))$, achieved by restricting force calculation to a neighborhood, thus ignoring long-range force. Gajer et al. [15] built up the multilevel of graphs by using maximal independent vertex sets, with the ith level consisting of a maximal independent vertex set such that the vertices are of distance $\geq 2^{i-1} + 1$, $i = 1, 0, \ldots$ apart. They avoided the high computational and memory complexity by calculating the graph distances on the fly and only for a restricted neighborhood, thus also ignoring the long-range force. Walshaw [2] proposed a multilevel algorithm and demonstrated that it was able to draw graphs as large as a quarter of a million vertices in a few minutes. Based on the author's earlier work in graph

partitioning, the coarser graphs are formed by finding a maximal independent edge set and collapsing these edges. The forces between vertices are based on the Fruchterman and Reigold spring-electrical model [4]. Long-range force is again ignored by restricting the force calculation to a neighborhood with a radius that decreases as one moves from coarser to finer graphs and coincides with the radius used in the original graph.

Reducing the computational cost by restricting force calculation to a neighborhood has been an often used practice, dating at least as far back as Fruchterman and Reigold [4]. Such a practice, however, comes at a cost. Because long-range forces are ignored, there is no force to evenly distribute faraway vertices. When used within the multilevel approach, Walshaw [2] argued that global untangling has been achieved on coarser graphs, thus for the final large graphs, restricting force calculation to a small neighborhood does not penalize the quality of the placement. While this is to a large extent true, for some graphs we found that the lack of long-range force did hurt at least one, if not more, of the drawings in [2].

It is possible to take account of long-range forces in an efficient way in the spring-electrical model. In this model the attractive force (the spring force) is only between neighboring vertices, while the repulsive force is global and is proportional to the inverse of the (physical) distance between vertices. The repulsive force calculation resembles the n-body problem in physics, which has been well studied. One of the widely used techniques for calculating the repulsive forces in $O(n \log(n))$ time with good accuracy, but without ignoring long-range force, is to treat groups of faraway vertices as a supernode using a suitable data structure, as in the Barnes and Hut algorithm [1]. This idea was implemented for graph drawing by both Tunkelang [16] and Quigley and Eades [17, 18]. Tunkelang combined the Barnes and Hut algorithm with a conjugate gradient method, thus per iteration computational is only $O(|V| \log(|V|))$, even though long-range forces are approximated to the required accuracy. However, the algorithm is not suitable for large graphs because the conjugate gradient is a local optimization algorithm and the number of conjugate gradient iterations increases as the size of the graph increases. Quigley and Eades [17, 18] also used the Barnes and Hut algorithm for efficient and accurate force calculation. In addition, they employed a multilevel scheme that they called hierarchical clustering. However, they used that scheme for the visual abstraction of graphs, rather than for the placement of vertices. Therefore, the algorithm was not suitable for large graphs.

In this article we propose an algorithm that is both efficient and of high quality for large graphs. The algorithm is included with *Mathematica* 5.1 and later versions in the package `DiscreteMath`GraphPlot`. We combine a multilevel approach, which effectively overcomes local minimums, with the Barnes and Hut octree algorithm, which approximates short- and long-range forces satisfactorily and efficiently. In addition, we propose an adaptive cooling scheme for the basic force-directed algorithms and a scheme for selecting the optimal depth of the octree/quadtree in the Barnes and Hut algorithm. Our numerical results show that the algorithm is competitive with Walshaw's [2] highly efficient graph drawing algorithm, yet gives better results on some of the difficult problems. We also analyze the distortion effect of the standard Fruchterman–Reigold

spring-electrical model and propose a general repulsive force model to overcome this side effect.

In Section 2, we give definitions and notations. In Section 3, we present the basic force-directed algorithms, as well as an adaptive cooling scheme. In Section 4, we briefly introduce the Barnes–Hut force calculation algorithm. In Section 5, we describe the multilevel scheme used. In Section 6, we compare the efficiency and drawings of our algorithm with that of Walshaw [2]. In Section 7, we conclude by suggesting some future works.

■ 2. Definitions and Notations

We use $G = \{V, E\}$ to denote an undirected graph, with V the set of vertices and E the set of edges. We assume the graph is connected. Disconnected graphs can be drawn by laying out each of the components separately.

If two vertices i and j form an edge, we denote that as $i \leftrightarrow j$. The coordinates of node i are denoted as x_i, and we use $\| x_i - x_j \|$ to denote the 2-norm distance between vertices i and j. We use $d(i, j)$ to denote the graph distance between vertices i and j, and we use diam(G) to denote the diameter of the graph.

The graph layout problem is one of finding a set of coordinates, $x = \{x_i \mid i \in V\}$, with $x_i \in R^2$ or $x_i \in R^3$, for 2D or 3D layout, respectively, such that when the graph G is drawn with vertices placed at these coordinates, the drawing is visually appealing.

■ 3. Force-Directed Algorithms

In this section we present the basic force-directed algorithms and analyze the characteristics of the layout given by the spring-electrical model. We will propose a general repulsive force model, as well as an adaptive step control scheme.

□ 3.1. Spring and Spring-Electrical Models

Force-directed algorithms model the graph layout problem by assigning attractive and repulsive forces between vertices and finding the optimal layout by minimizing the energy of the system.

The model of Fruchterman and Reigold [4], which we refer to as the *spring-electrical model*, has two forces. The repulsive force, f_r, exists between any two vertices i and j and is inversely proportional to the distance between them. The attractive force, f_a, on the other hand, exists only between neighboring vertices and is proportional to the square of the distance

$$\begin{aligned} f_r(i, j) &= -C K^2 / \| x_i - x_j \|, \qquad i \neq j, \qquad i, j \in V \\ f_a(i, j) &= \| x_i - x_j \|^2 / K, \qquad i \leftrightarrow j. \end{aligned} \tag{1}$$

The combined force on a vertex i is

$$f(i, x, K, C) = \sum_{i \neq j} \frac{-C K^2}{\| x_i - x_j \|^2} (x_j - x_i) + \sum_{i \leftrightarrow j} \frac{\| x_i - x_j \|}{K} (x_j - x_i). \qquad (2)$$

In these formulas, K is a parameter known as the optimal distance [4], or natural spring length [2]. The parameter C regulates the relative strength of the repulsive and attractive forces and was introduced in [2]. It is easy to see that for a graph of two vertices linked by an edge, the force on each vertex diminishes when the distance between them is equal to $K(C)^{1/3}$. The total energy of the system can be considered as

$$\text{Energy}_{\text{se}}(x, K, C) = \sum_{i \in V} f^2(i, x, K, C),$$

where x is the vector of coordinates, $x = \{x_i \mid i \in V\}$.

From a mathematical point of view, changing the parameters K and C does not actually change the minimal energy layout of the graph but merely scales the layout, as the following theorem shows.

Theorem 1. Let $x^* = \{x_i^* \mid i \in V\}$ minimizes the energy of the spring-electrical model $\text{Energy}_{\text{se}}(x, K, C)$, then sx^* minimizes $\text{Energy}_{\text{se}}(x, K', C')$, where $s = (K'/K)(C'/C)^{1/3}$. Here K, C, K' and C' are all positive real numbers.

Proof: This follows simply by the relationship

$$\begin{aligned} f(i, x, K, C) &= \sum_{i \neq j} \frac{-C K^2}{\| x_i - x_j \|^2} (x_j - x_i) + \sum_{i \leftrightarrow j} \frac{\| x_i - x_j \|}{K} (x_j - x_i) \\ &= \left(\frac{C}{C'}\right)^{2/3} \frac{K}{K'} \left(\sum_{i \neq j} \frac{-C' (K')^2}{\| s x_i - s x_j \|^2} (s x_j - s x_i) \right. \\ &\qquad \left. + \sum_{i \leftrightarrow j} \frac{\| s x_i - s x_j \|}{K'} (s x_j - s x_i) \right) \\ &= \left(\frac{C}{C'}\right)^{2/3} \frac{K}{K'} f(i, s x, K', C'), \end{aligned}$$

where $s = (K'/K)(C'/C)^{1/3}$. Thus,

$$\text{Energy}_{\text{se}}(x, K, C) = \left(\frac{C}{C'}\right)^{4/3} \left(\frac{K}{K'}\right)^2 \text{Energy}_{\text{se}}(s x, K', C').$$

Even though the parameters K and C do not have any bearing on the optimal layout from a mathematical point of view, from an algorithmic point of view, if an iterative algorithm (see Algorithm 1) is applied to minimize the energy from an initial layout, choosing a suitable K or C to reflect the range of the initial position will help the convergence to the optimal layout. Throughout this article, unless otherwise specified, we fix $C = 0.2$ as in [2], but vary K for this purpose.

One intrinsic feature of graph drawings by the spring-electrical model is that vertices in the periphery tend to be closer to each other than those in the center, even for a uniform mesh. We call this the *peripheral effect*. For example, Figure 1 shows a mesh of 100 vertices laid out using the spring-electrical model, with vertices near the outside boundary clearly closer to each other than those near the center.

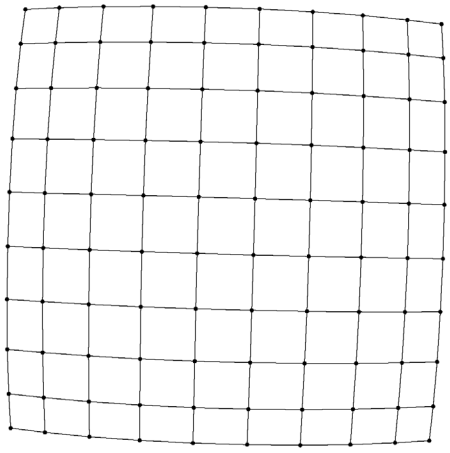

Figure 1. A 10×10 regular mesh laid out using the spring-electrical model. This graph and the others in this article were drawn with the *Mathematica* `GraphPlot` command, which is discussed in Section 6.

This peripheral effect is more profound for graphs with large diameter. We studied this effect for a line graph, with 100 vertices linked in a line. The vertices are numbered 1 to 100, with vertex 50 and 51 in the middle of the line. We find an accurate layout under the spring-electrical model by finding the root of a system of equations $f(i, x, K, C) = 0$ (see equation (2)) using *Mathematica*'s `FindRoot` function, instead of the usual force-directed iterative algorithm, because the latter is far less accurate. We set the parameters $K = 1$ and $C = 1$. Theorem 1 shows that the exact values of these two parameters are not important.

Figure 2 shows the distribution of edge lengths. As can be seen, the edge lengths of the 99 edges vary from 4.143 at the middle to 1.523 at the sides, with the ratio of the longest length to the shortest 4.143 / 1.523 = 2.72.

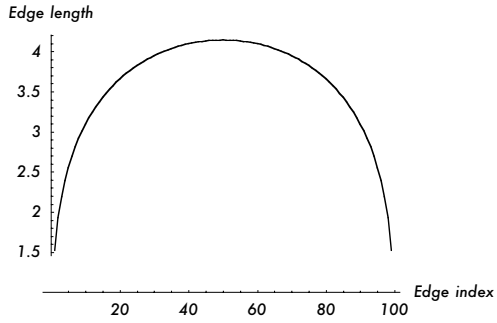

Figure 2. Distribution of edge lengths for a line graph of 100 vertices and 99 edges, laid out using the spring-electrical model.

The reason for this distortion effect at the periphery is the strong long-range force that decays slowly as the distance increases. While typically this strong long-range force does not interfere with the aesthetics of the layout, some applications, such as tree graphs, tend to suffer more. For these classes of graphs, the following general repulsive force model can be used.

$$f_r(i, j) = -C K^{1+p} / \| x_i - x_j \|^p, \qquad i \neq j, \qquad i, j \in V, \qquad p > 0. \quad (3)$$

The larger the parameter p, the weaker the long-range repulsive force. However, too large a value of p, thus too weak a long-range force, could cause the graph that should be spread out to crease instead. We found that $p = 2$ works well. Figure 3 shows the peripheral effect against the size of the line graph, for both the general model (4) with $p = 2$ and $p = 3$, and the Fruchterman and Reigold force model (1), which corresponds to the general model with $p = 1$. As can be seen, the general model with $p > 1$ reduces the peripheral effect significantly.

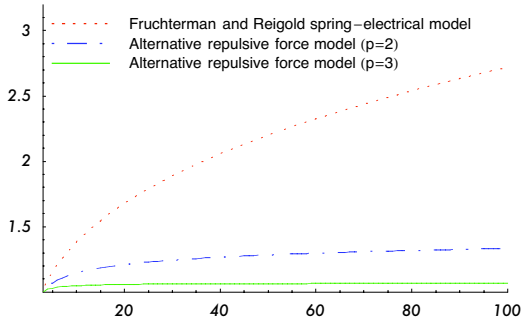

Figure 3. A plot of the ratio between the largest and smallest edge lengths for line graphs of size 3 to 100 vertices.

In the force model of Kamada and Kawai [7], which we will refer to as the *spring model*, springs are attached between any two pairs of vertices, with the ideal

length of a spring proportional to the graph distance between these two vertices. Thus both the repulsive and attractive forces are expressed as

$$f_r(i, j) = f_a(i, j) = \| x_i - x_j \| - d(i, j), \qquad i \neq j, \qquad i, j \in V \tag{4}$$

and the spring energy is

$$\text{Energy}_s(x) = \sum_{i \neq j, i, j \in V} (\| x_i - x_j \| - d(i, j))^2. \tag{5}$$

The spring model does not suffer from the peripheral effect of the spring-electrical model. It can, however, suffer from its relatively weak repulsive forces (see Section 6.3). Furthermore, as discussed later, in the spring-electrical model, the long-range force between all vertices can be well approximated by grouping vertices together using the octree/quadtree technique. This is, however, not possible in the spring model.

□ 3.2. An Adaptive Cooling Scheme

The energy of both the spring-electrical and the spring models can be minimized iteratively by moving the vertices along the direction of forces exerted on them. The following force-directed algorithm iteratively minimizes the system energy, with f_a and f_r as defined in (1) or (5). The algorithm starts with a random, or user-supplied, initial layout.

- ForceDirectedAlgorithm(G, x, tol) {
  - converged = FALSE;
  - step = initial step length;
  - Energy = Infinity
  - while (converged equals FALSE) {
    - $x^0 = x$;
    - Energy^0 = Energy; Energy = 0;
    - for $i \in V$ {
      - $f = 0$;
      - for $(j \leftrightarrow i)$ $f := f + \frac{f_a(i,j)}{\| x_j - x_i \|} (x_j - x_i)$;
      - for $(j \neq i, j \in V)$ $f := f + \frac{f_r(i,j)}{\| x_j - x_i \|} (x_j - x_i)$;
      - $x_{i:} = x_i + \text{step} * (f / \| f \|)$;
      - Energy := Energy + $\| f \|^2$;
    - }
    - step := update_steplength (step, Energy, Energy^0);
    - if (| $|x - x^0|$ | < K tol) converged = TRUE;
  - }
  - return x;
- }

Algorithm 1. An iterative force-directed algorithm.

In the algorithm, tol > 0 is a termination tolerance. The algorithm stops when a change in the layout between subsequent iterations is less than K to l.

As in [2], we update the layout of a vertex i as soon as the force for this vertex is calculated, instead of waiting until forces for all vertices have been calculated. This improves the convergence of the iterative procedure, much like the fact that among stationary iterative linear system solvers, the Gauss–Seidel algorithm is often faster than the Jacobi algorithm.

In Algorithm 1, it is necessary to update the step length step. The "cooling schedule" used in most expositions (e.g., [19]) of force-directed algorithms allows large movements (large step length) at the beginning of the iterations, but the step length reduces as the algorithm progresses. Walshaw [2] used a simple scheme,

$$\text{step} := t\,\text{step}, \tag{6}$$

with $t = 0.9$. We found this to be adequate in the refinement phase of a multilevel force-directed algorithm. However, for an application of a force-directed algorithm from a random initial layout, an adaptive step length update scheme is more successful in escaping from local minimums. This adaptive scheme is motivated by the trust region algorithm for optimization [20], where step length can increase as well as reduce, depending on the progress made. Here we measure progress by the decrease in system energy.

- function update_steplength (step, Energy, Energy^0)
- if (Energy < Energy^0) {
  - progress = progress + 1;
  - if (progress > = 5) {
    - progress = 0;
    - step := step / t;
  - }
- } else {
  - progress = 0;
  - step := t step;
- }

In the preceding algorithm, progress is a static variable that is initialized to zero and parameter $t = 0.9$. The idea of this algorithm is that the step length is kept unchanged if energy is being reduced and increased to step / t if the energy is reduced more than five times in a row. We only reduce the step length if energy increases.

Compared with the simple step length update scheme (6), the adaptive scheme is much better in escaping from local minimums. Figure 4 shows the result of applying 70 iterations of Algorithm 1 to the jagmesh1 graph with 936 vertices from MatrixMarket (math.nist.gov/MatrixMarket). The adaptive scheme is clearly much better and is able to untangle the graph a lot further than the simple scheme.

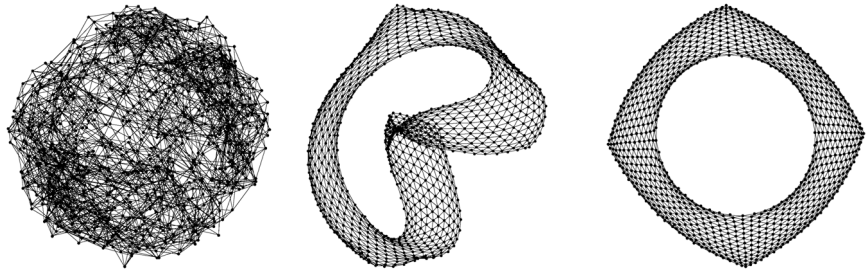

Figure 4. Comparing adaptive and simple step length update schemes on jagmesh1 after 70 iterations: simple scheme (left); adaptive scheme (middle); what the layout should look like (right).

■ 4. Barnes–Hut Force Calculation

Each iteration of Algorithm 1 involves two loops. The outer loop iterates over each $i \in V$. Of the two inner loops that calculate the attractive and repulsive forces, the latter is the most expensive and loops over each $j \neq i$, $j \in V$. 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 calculate the repulsive forces in $O(n \log n)$ time with good accuracy, but without ignoring long-range forces, is to treat groups of faraway vertices as supernodes, using a suitable data structure [1]. This idea was adopted by Tunkelang [16] and Quigley [18]. Both used an octree (3D) or quadtree (2D) data structure. In principal, other space decomposition methods can also be used. For example, Pulo [21] investigated recursive Voronoi diagrams.

For simplicity, hereafter we use the term octree exclusively, which should be understood as quadtree in the context of 2D layout. An octree data structure is constructed by first forming a square (or cube in 3D) that encloses all vertices. This is the level 0 square. This square is subdivided into four squares (or eight cubes) if it contains more than one vertex and forms the level 1 squares. This process is repeated until level L, where each square contains no more than one vertex. Figure 5 (left) shows an octree on the jagmesh1 graph.

The octree 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 cluster of vertices, S, lies in a square that is "far" from i, the whole group can be treated as a supernode. The supernode is assumed to situate at the center of gravity of the cluster, $x_S = \left(\sum_{j \in S} x_j\right) / |S|$. The repulsive force on vertex i from this supernode is

$$f_r(i, S) = -|S| C K^2 / \| x_i - x_S \| .$$

However, we need to define what "far" means. Following [16, 18], we define the supernode S to be faraway from vertex i, if the width of the square that contains the cluster is small, compared with the distance between the cluster and the vertex i,

$$\frac{d_S}{\| x_i - x_S \|} \leq \theta. \tag{7}$$

Here d_S is the width of the square that the cluster lies in, and $\theta \geq 0$ is a parameter. The smaller the value of θ, the more accurate the approximation to the repulsive force and the more computationally expensive it is. We found that $\theta = 1.2$ is a good compromise and will use this value throughout the article. This inequality (7) is called the Barnes–Hut opening criterion [1].

The octree data structure allows efficient identification of all the supernodes that satisfy (7). The process starts from the level 0 square. Each square is checked and recursively opened until the inequality (7) is satisfied. Figure 5 (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 we have 32 supernodes.

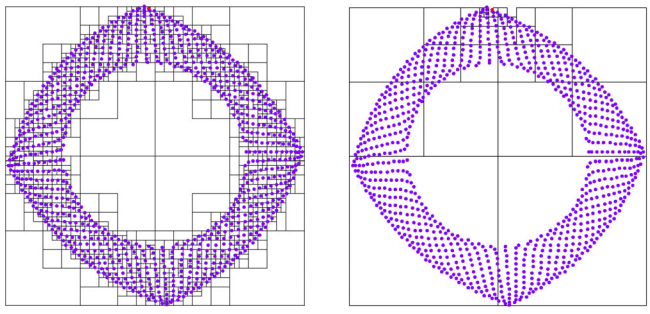

Figure 5. An illustration of the octree data structure: the overall octree (left); supernodes with reference to a vertex at the top-middle part of the graph, with $\theta = 1$ (right).

Under reasonable assumption [1, 22] of the distribution of vertices, it can be proved that building the octree takes a time complexity of $O(|V| \log(|V|))$. Finding all the supernodes with reference to a vertex i can also be done in a time complexity $O(\log(|V|))$. Overall, by using an octree structure to approximate the repulsive force, the complexity for each iteration of Algorithm 1 under the spring-electrical model is reduced from $O(|V|^2)$ to $O(|V| \log(|V|))$.

We only build the octree structure once every outer loop. Note that the positions of the vertices are actually updated continuously within the loop; therefore, as they move about, some vertices may not be in the squares where they are supposed to lie. However, we found that this does not cause a problem in practice. Consequently, we observed that construction of the tree structure only takes a fraction of the total time, and the majority of the time in Algorithm 1 is spent in finding the supernodes, as well as in the force calculations.

For very large graphs, it may happen that some of the vertices are very close to each other. Therefore, if we keep subdividing the squares without a limit, we may end up with a tree structure with one or more very deep branches. This will make the algorithm less efficient. In particular, a disproportionately large amount of time will be spent in searching through the tree to find supernodes. It is therefore necessary to limit the number of levels in the octree. We denote this limit max_tree _level. Even if a square at level max_tree _level still has multiple vertices, it is not subdivided further. We call such a square a dense leaf (of the octree).

However, it is difficult to decide a priori how many levels should be allowed. If we set the max_tree _level too small, we will have many vertices in the same square as at the last level. This increases the average number of supernodes, because when identifying supernodes needed to approximate the repulsive force on a vertex i, if a dense leaf happens to be close to i, each vertex on the dense leaf has to be treated individually as a supernode since no subgrouping is available. In the extreme case when max_tree _level = 0, every vertex belongs to one dense leaf, and there are $|V|-1$ supernodes that correspond to every vertex i. On the other hand, although a large value for max_tree _level reduces the average number of supernodes, it increases the number of squares that need to be traversed due to the deep branches.

We use an adaptive scheme to automatically find the optimal max_tree _level. This is essentially a one-dimensional optimization problem with the variable being max_tree _level and the objective function the CPU time of each outer iteration of Algorithm 1, which consists largely of the time to transverse the octree and the time in the repulsive force calculation. So one way to find the optimal max_tree _level is to measure the CPU time of the outer loop and increase/decrease max_tree _level by one each time until the bottom of a valley is located. However, CPU time measurement can fluctuate and such a scheme may cause a different max_tree _level from run to run. This in turn gives a different layout between runs, which is undesirable. Instead, we use

$$h(\text{max\_tree\_level}) = \text{counts} + \alpha\,\text{ns} \tag{8}$$

as the objective function, where counts is the total number of squares traversed, ns is the total number of supernodes found during one outer iteration, and α is a parameter chosen so that (8) gives the best estimate of the CPU time. Through numerical experiments, we found that α in the range of 1.5 to 2.0 gives very good correlation to CPU time measurement. Thus we used $\alpha = 1.7$. The adaptive scheme starts with max_tree _level = 8. After one outer iteration, we set max_tree _level = 9. Then, depending on whether the estimated CPU time increases or decreases, we try a smaller or larger depth. If we ever hit a depth already tried, we end the procedure and use the depth corresponding to the smallest estimated CPU time. Typically, we found that the procedure converges within 3-4 outer iterations, and the max_tree _level located is very near the optimal value. For smaller graphs of a few thousand vertices, typically max_tree _level settles down at around 8, while for very large graphs, max_tree _level can go as high as 11.

■ 5. The Multilevel Algorithm

While approximation of long-range force using octree data structure greatly reduces the complexity of Algorithm 1, it does not change the fact that the algorithm repositions one vertex at a time, instead of laying out a whole region as a unit. Large graphs typically have many local minimal energy configurations, and such an algorithm is likely to settle to one of the local minimums. The adaptive step length control scheme we introduce goes some way toward a better layout, but as Figure 4 shows, is still insufficient toward a global minimum.

The multilevel approach has been used in many large-scale combinatorial optimization problems, such as graph partitioning [11, 12, 23], matrix ordering [24], the traveling salesman problem [25], and has proven to be a very useful meta-heuristic tool [26]. The multilevel approach was also used in graph drawing [2, 13, 14]. In particular, Walshaw [2] was able to lay out graphs with up to 225,000 vertices in a few minutes, and largely of good quality.

The 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^1 , … , G^l, are generated. The aim is for each coarser graph G^{k+1} to encapsulate the information needed to lay out 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 layouts on the coarser graphs are recursively prolonged to the finer graphs, with further refinement at each level. Hereafter, we use a superscript to denote the level. For example, x^k is the coordinates of vertices in the level k graph G^k, $k = 0, \ldots, l$.

□ 5.1. Graph Coarsening

There are a number of ways to coarsen an undirected graph. One frequently used method is based on edge collapsing (EC) [11, 12, 23], in which pairs of adjacent vertices are selected and each pair is coalesced into one new vertex. Each vertex of the resulting coarser graph has an associated weight, equal to the number of original vertices it represents. Each edge of the coarser graph also has a weight associated with it. Initially, all edge weights are set to 1. During coarsening, edge weights are unchanged unless both merged vertices are adjacent to the same neighbor. In this case, the new edge is given a weight equal to the sum of the weights of the edges it replaces. The edges to be collapsed are usually selected using *maximal matching*. This is a maximal set of edges, no two of which are incident to the same vertex. For undirected graph partitioning, heavy-edge matching has been found to work well. Here, the idea is to preferentially collapse heavier edges. When looking through a neighbor list for an unmatched vertex, an edge with the largest weight is selected. In the context of graph drawing, Walshaw [2] chose to keep vertex weights in the coarser graphs as uniform as possible by matching a vertex with a neighbor with the smallest vertex weight. We found that both heavy-edge matching and matching with a vertex of smallest weight give graph layouts of similar quality and used the former in this article. Figure 6 illustrates a graph (left) and the result (middle) of coarsening using EC.

Other coarsening methods have been proposed. In [27], a maximal independent vertex set (MIVS) of a graph is chosen as the vertices for the coarser graph. An independent set of vertices is a subset of the vertices such that no two vertices in the subset are connected by an edge in the graph. An independent set is maximal if the addition of an extra vertex always destroys the independence. Edges of the coarser graph are formed by linking two vertices in MIVS by an edge if their distance apart is no greater than 3. Figure 6 illustrates a graph (left) and the result (right) of coarsening using MIVS.

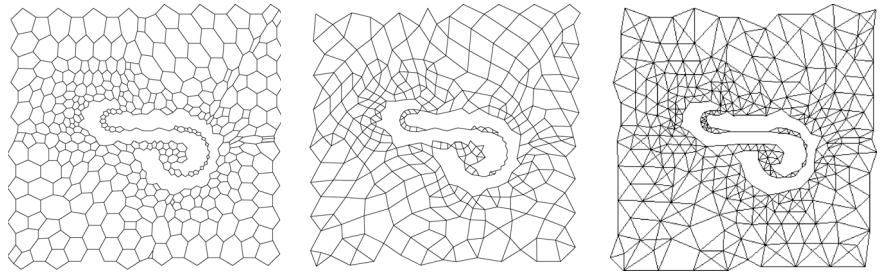

Figure 6. An illustration of graph coarsening: original graph with 788 vertices (left); a coarser graph with 403 vertices resulted from EC (center); a coarser graph with 332 vertices resulted from MIVS (right).

We have implemented both EC and MIVS coarsening schemes for our multilevel graph drawing algorithm. Notice that with EC, the coarse graph always has more than 50% of the vertices of the original graph. For graphs with a high average degree, it may happen that the number of vertices in the coarser graph, G^{i+1}, may be very close to the number of vertices in the original graph, G^i. This can significantly increase the complexity of the multilevel algorithm. Therefore, we will stop coarsening if there are only two vertices in the graph or

$$\frac{|V^{i+1}|}{|V^i|} > \rho. \tag{9}$$

We used $\rho = 0.75$. Here V^i is the vertices in G^i.

On the other hand, for a connected graph, the MIVS coarsening usually, though not necessarily, results in a coarser graph with less than 50% of the number of vertices of the original graph. In our experience, for graph layout, EC tends to give slightly better results than MIVS, probably because it coarsens less aggressively. However, MIVS is usually faster, due to the lower complexity. To overcome the complexity issue of EC, we propose a third scheme, HYBRID. This scheme uses EC whenever possible, however, if the threshold (9) is breached, we use MIVS coarsening instead.

□ 5.2. Initial Layout on the Coarsest Graph

On the coarsest level, we lay out the graph using Algorithm 1 combined with the adaptive step length control scheme. For MIVS and HYBRID coarsening schemes, the coarsest graph has only two vertices, thus a random placement of the two vertices would be sufficient.

□ 5.3. The Refinement Step

The layouts on the coarser graphs are recursively prolonged to the finer graphs, with further refinement at each level.

If graph $G^{i+1} = \{V^{i+1}, E^{i+1}\}$ was derived using EC from $G^i = \{V^i, E^i\}$, the position of a vertex $u \in V^{i+1}$ is given to the two vertices $v, w \in V^i$ that collapse into u. If G^{i+1} was derived using MIVS from G^i, then a vertex $v \in V^i$ either inherits the position if it is in the MIVS, or v must have one or more neighbors in MIVS, in which case the position of v is the average of the positions of these neighbors.

Once the prolongation is carried out to give an initial layout for G^i, this layout is refined using Algorithm 1. As in [2], if in the initial layout two vertices happen to be at the same position, as could be the case with EC-based coarsening, a random perturbation is done to separate them. Because the initial layout is derived from the layout of a coarser graph, this layout is typically already well placed globally and what is required is just some local adjustment. Therefore, we found that it is preferable to use a conservative step length update scheme. The simple scheme (6) works well.

Although the initial layout in graph G^i prolongated from the coarser graph G^{i+1} is usually globally well positioned, a naive application of Algorithm 1 would cause large movement of vertices, thus potentially destroying the useful information inherited. For example, in the context of the spring model, the physical distance between two vertices u and v in the initial layout in G^i is roughly the same as the graph distance of the corresponding vertices in G^{i+1}, that is

$$\| x_u^i - x_v^i \| \approx d_{G^{i+1}}(u', v'),$$

where u' and v' are two vertices in the coarser graph G^{i+1} that corresponds to u and v. However, to minimize the energy on level i, we want $\| x_u - x_v \|$ to be as close to the graph distance of them in G^i, that is, we want

$$\| x_u^i - x_v^i \| \approx d_{G^i}(u, v).$$

Typically $d_{G^{i+1}}(u', v')$ is much smaller than $d_{G^i}(u, v)$. If the initial layout is used as is, then to achieve the minimal energy, the graph has to be expanded by the ratio between these two distances through a serious of large moves, which is inefficient and could lose some information in the initial good layout.

Therefore, for a multilevel algorithm based on the spring model, we scale the initial coordinates by the ratio of the pseudo-diameters between the two subsequent levels of graphs,

$$\gamma = \text{diam}(G^i) / \text{diam}(G^{i+1}). \tag{10}$$

For the spring-electrical model, Walshaw [2] suggested keeping the coordinates unchanged but reducing the natural spring length K^i to

$$K^i = K^{i+1} / \gamma,$$

with $\gamma = \sqrt{7/4}$. Walshaw derived this value based on examining a graph with four vertices. We use instead (10) and found it to be equally effective. On the coarsest level, we set K^l to be the average edge length of the initial random layout, as in [2].

To avoid the $O(|V|^2)$ complexity of the repulsive force calculation in the spring-electrical model, Walshaw [2], following Fruchterman and Reigold [4], cut off the contributions from faraway vertices. Only repulsive forces from vertices within a certain radius were considered. Specifically, on the ith level, for vertex v, repulsive forces from vertex u were ignored when $\| x_u^i - x_v^i \| > R^i$. If a small radius R^i is chosen, then repulsive force over even a short distance is ignored. This can cause faraway regions to collapse into each other due to the lack of force to spread them out. On the other hand, a larger radius R^i is expensive. In the extreme, $R^i = \infty$ gives us the $O(|V^i|^2)$ complexity. Walshaw [2] proposed the radius R^i

$$R^i = 2(i+1)K^i.$$

Notice that the radius is larger, relative to the natural spring length, on the coarser graphs, for which a large R^i value would not be too costly due to the smaller sizes of the graphs. The effect of this cutoff, which would otherwise be more profound, was largely dampened because of the multilevel approach. Overall, the power of the multilevel approach means that the good quality global layout achieved on coarser graphs was inherited by the finer graphs, and a small radius on the finest graphs usually worked well. Nevertheless, we found that for some graphs, particularly those with a hollow interior, such as the graph finan512 in Section 6, the adverse effect of ignoring long-range repulsive force was obvious. With our use of octree data structure, we no longer need to use a cutoff radius for the spring-electrical model. Nevertheless we set a cutoff radius of

$$R^i = r(i+1)K^i, \tag{11}$$

and by default we set $r = \infty$, which is equivalent to not using a cutoff radius at all. This, however, also allows us to have a finite r value to experiment with. For the spring model, to avoid calculation of the all-to-all distance matrix needed to compute the force (5), we used a similar strategy, by only calculating the distance of two vertices u and v and their attractive/repulsive forces, if

$$d(u, v) \le r(i+1), \tag{12}$$

with a default r value of 4.

□ 5.4. The Multilevel Algorithm

We present the overall multilevel algorithm in the following. In the algorithm, $n^i = |V^i|$ is the number of vertices in the ith level graph G^i. x^i is the coordinate vector for the vertices in V^i. We represent G^i by a symmetric matrix G^i, with the entries of the matrix the edge weights. The prolongation operator from G_{i+1} to G_i is also represented by a matrix P^i, of dimension $n^i \times n^{i+1}$. For details on the implementation of the multilevel process, and some examples, see [24]. The starting point is the original graph, $G_0 = G$.

function MultilevelLayout (G^i, tol)

- Coarsest graph layout
  - if ($n^{i+1} <$ MinSize or $n^{i+1} / n^i > \rho$) {
    - $x^{i:}$ = random initial layout
    - x^i = ForceDirectedAlgorithm(G^i , x^i, tol)
    - return x^i
  - }
- The coarsening phase:
  - set up the $n^i \times n^{i+1}$ prolongation matrix P^i
  - $G^{i+1} = P^{i\,T}\ G^i\ P^i$
  - x^{i+1} = MultilevelLayout(G^{i+1}, tol)
- The prolongation and refinement phase:
  - prolongate to get initial layout: $x^i = P^i\ x^{i+1}$
  - refinement: x^i = ForceDirectAlgorithm(G^i , x^i, tol)
  - return x^i

Algorithm 2. A multilevel force-directed algorithm.

In Algorithm 2, coarsening will stop if the graph is too small, $n^{i+1} <$ MinSize, or there is not enough coarsening, $n^{i+1} / n^i > \rho$. We used MinSize = 2 and $\rho = 0.75$.

■ 6. Numerical Results

In this section we demonstrate the drawings using our algorithms on some large examples and also compare our algorithms with those of Walshaw [2], both in terms of efficiency and quality of layout.

The algorithms are implemented in *Mathematica* 5.1, under the `GraphPlot` package. To load the package, evaluate

```
<< DiscreteMath`GraphPlot`;
```

This package has many graph theory functions. Among them, `GraphPlot` and `GraphPlot3D` draw graphs in 2D and 3D, respectively. Details of the package can be found in the *Mathematica* Help Browser.

The details for the algorithms we will demonstrate and the names they are denoted by follow. We also give the *Mathematica* command corresponding to each of the algorithms, where we use `gr` to represent the graph.

MSE(r) (Multilevel Spring-Electrical Model)

Multilevel Algorithm 2 with HYBRID coarsening scheme, octree data structure, and repulsive/attractive force given by (1). The cutoff radius for the repulsive force calculation is (11), with $r = \infty$ by default. The *Mathematica* command for MSE(∞) is

```
GraphPlot[gr, Method→{SpringElectricalModel, Tolerance→0.01}]
```

The *Mathematica* command for MSE(r) ($r < \infty$) is

```
GraphPlot[gr, Method→{SpringElectricalModel, Tolerance→0.01,
InferentialDistance→r}]
```

MSE(r , p) (Multilevel Spring-Electrical Model with General Repulsive Force)

Multilevel Algorithm 2 with HYBRID coarsening scheme, octree data structure, and repulsive/attractive force given by (1). The cutoff radius for the repulsive force calculation is (11), with $r = \infty$ by default. The difference from MSE(r) is that the general repulsive force model (4) is used with parameter p. MSE(r) is the same as MSE(r, 1). The *Mathematica* command for MSE(∞, 1) is

```
GraphPlot[gr, Method→{SpringElectricalModel, Tolerance→0.01}]
```

The *Mathematica* command for MSE(r, 1) ($r < \infty$) is

```
GraphPlot[gr, Method→{SpringElectricalModel, Tolerance→0.01,
InferentialDistance→r}]
```

The *Mathematica* command for MSE(r, p) ($r < \infty$, $p \neq 1$) is

```
GraphPlot[gr, Method→{SpringElectricalModel, Tolerance→0.01,
InferentialDistance→r, RepulsiveForcePower→-p}]
```

MS(r) (Multilevel Spring Model)

Multilevel Algorithm 2 with HYBRID coarsening scheme, octree data structure, and repulsive/attractive force given by (5). The cutoff radius for the distance and force calculation is (12), with $r = 4$ by default. The *Mathematica* command for MS(∞) is

```
GraphPlot[gr, Method→{SpringModel, Tolerance→0.01}]
```

The *Mathematica* command for MS(r) ($r < \infty$) is

```
GraphPlot[gr, Method→{SpringModel, Tolerance→0.01, InferentialDistance→
r}]
```

SE (Spring-Electrical Model)

Algorithm 1 with octree data structure and repulsive/attractive force given by (1). The *Mathematica* command for SE is

```
GraphPlot[gr, Method→{SpringElectricalModel, Tolerance→0.01,
RecursionMethod→None}]
```

MLFDP

The multilevel force-directed placement algorithm (MLFDP) from [2].

For all the preceding algorithms, we used a tolerance of tol = 0.01 to be comparable with Walshaw [2]. All our results are from a 3.0 GHz Pentium 4 machine with 2 GB of memory and a 512 KB cache under a Linux operating system. Our code is written in C and compiled with gcc using compilation flag "gcc -O2".

□ 6.1. Comparison with Walshaw

In this section we compare our algorithm with that of Walshaw [2].

Table 1 lists a set of nine test problems from [2]. Some of these problems originate from engineering applications for which there is a known layout. Table 2 gives the CPU time for MSE(2) and MSE(∞), as well as the CPU time for the multilevel force-directed placement algorithm (MLFDP) from [2]. To see the extra cost of the multilevel approach, we also include the CPU time for the single level spring-electrical model, SE, in the last column of the table. The set of nine problems were chosen to be the same as those in Table 3 of [2]. We draw all the graphs in 2D and use these layouts for all the subsequent figures. However to be able to compare them with [2], we also laid out the last four graphs in 3D, even though sierpinski10 is really a 2D graph and, as Figure 12 shows, we can layout in 2D just fine.

Notice that Walshaw's CPU results were for a 1 GHz machine, 1/3 of the clock speed of our machine. However, based on our experience of clock speed and actual performance, we would expect the performance of our machine to be less than three times of Walshaw's.

Walshaw's MLFDP algorithm should be somewhat similar to our MSE(2), because both employ a similar cutoff radius. There are, however, some differences. MSE(2) uses the octree data structure and searches through the structure to decide if a cluster of vertices can form a supernode or should be excluded because it is outside the cutoff radius. So we have two approximations. In terms of CPU time, forming supernodes reduces the number of repulsive force calculations, but searching through the octree data structure is more expensive than the regular mesh-like data structures used by Walshaw to implement the cutoff radius. Therefore, overall, we expect the complexity of MLFDP and MSE(2) to be comparable. This is confirmed in Table 2. MLFDP and MSE(2) indeed do have quite comparable CPU time in general, although MSE(2) is notably faster on finan512, while MLFDP is notably faster on dime20. MSE(∞), which does not ignore long-range force, is only on average 21% more expensive than MSE(2).

Comparing multilevel algorithm MSE(∞) with its single level counterpart SE, it is seen that MSE(∞) is no more than twice as expensive, in fact for most graphs the difference is small. It is surprising that for sierpinski10, SE is more expensive than MSE(∞) for both the 2D and 3D layouts! A careful examination of the innermost loops of the force calculations and octree code reveals that MSE(∞) takes between two to three times as many operations as SE. The abnormality in CPU time is due to poor cache performance of the octree code in SE. SE starts from a random layout and never quite gets to a good layout for large graphs. When looping over vertices in the natural order to find their supernodes in the octree code, the locations of the vertices are unpredictable. The multilevel algorithm MSE(∞), on the other hand, always starts from a good initial layout. For most of the graphs in Table 1, the natural vertex ordering is such that if two vertices are close in their indices, they also tend to be close in their physical locations in the original layout. Thus in the octree code, roughly the same squares tend to be examined again and again when finding supernodes for consecutive vertices. This means that cache performance is very good for multilevel algorithm MSE(∞), but very poor for single level algorithm SE. This analysis is confirmed when we randomly shuffle the vertices. With such a random ordering, CPU timings for SE stay roughly the same, while the CPU timings for MSE(∞) increase about twice.

The preceding analysis may suggest that the multilevel algorithm is susceptible to poor initial vertex indexing. However, this is not true. First, large graphs tend to have good natural ordering. Second, poor ordering can be easily remedied by preordering the vertices using a suitable ordering algorithm that is inexpensive. For example, we used the METIS [10] nested dissection algorithm to order previously randomly shuffled graphs, using the adjacency matrix of the graphs, and then applied the multilevel algorithm to the resulting graphs. We found that the CPU timings of MSE(∞) on these preordered graphs are very comparable to those in Table 2. In fact for dime20, the CPU time is reduced from 290.6 to 252.3 with nested dissection preordering! Nested dissection preordering, however, has no effect on the cache performance of SE, due to its poor initial and subsequent layouts. It is possible to improve SE by ordering the vertices using a nested dissection based on the physical locations of the vertices, but since SE is not a good graph drawing algorithm anyway, we will not pursue this further here.

Graph	$\lvert V \rvert$	$\lvert E \rvert$	Avg. Degree	Diameter	Graph Type
c-fat500-10	500	46627	186.5	4	Random clique test
4970	4970	7400	3.	106	2 D dual
4 elt	15606	45878	5.9	102	2 D nodal
finan512	74752	261120	7.	87	Linear programming
dime20	224843	336024	3.	1179	2 D nodal
data	2851	15093	10.6	79	3 D nodal
add32	4960	9462	3.8	28	32-bit adder
sierpinski10	88575	177147	4.	1024	2 D fractal
mesh100	103081	200976	3.9	203	3 D dual

Table 1. Description of test problems.

	Size		CPU				
Graph	$\lvert V \rvert$	$\lvert E \rvert$	MLFDP*	MSE (2)	MSE (∞)	MS (4)	SE
2 D							
c-fat500-10	500	46627	5.6	0.37	0.37	0.82	0.2
4970	4970	7400	6.4	2.5	2.7	10.9	1.8
4 elt	15606	45878	24.3	9.4	11.7	102.9	9.2
finan512	74752	261120	363.8	56.6	59.8	3714.9	60.
dime20	224843	336024	264.3	195.5	290.6	1984.6	277.7
data	2851	15093	–	1.1	1.2	17.8	1.3
add32	4960	9462	–	3.1	3.3	44.4	2.6
sierpinski10	88575	177147	–	44.1	65.1	146.8	75.6
mesh100	103081	200976	–	91.6	109.4	5807.8	89.5
3 D							
data	2851	15093	6.6	2.3	2.4	33.	1.6
add32	4960	9462	12.5	6.2	7.1	230.7	3.2
sierpinski10	88575	177147	136.7	64.7	100.9	317.2	114.9
mesh100	103081	200976	431.1	158.	204.	6431.3	138.2

Table 2. CPU time (in seconds) for some force-directed algorithms. *: MLFDP data from [2], was for a 1 GHz Pentium III. All other times are for a 3 GHz Pentium 4. –: data not available.

As we would expect, MS(4) is very slow. This is because the spring model seeks to lay out vertices to have a physical distance equal to the graph distance. It is not obvious how to extend the octree methodology to the spring model. We can certainly work out the average graph distance of a cluster of vertices to another vertex, but to do so we still have to find the individual graph distances first. Therefore, no saving is achieved. A cutoff radius does allow us to reduce the $O(\lvert V \rvert^2)$ complexity. However, we found that for good drawing quality, we have to use a relatively large radius. This is probably because, unlike the spring-electrical model, the spring model does not have a strong repulsive force and with the cutoff radius the repulsive force is weakened further. At a cutoff

radius of 4, a large number of vertices are included, making the algorithm quite costly.

In terms of drawing quality, MSE(2) performs comparably to MLFDP, with MSE(∞) the best. Both MSE(2) and MLFDP ignore long-range forces. However, the multilevel process enables them to inherit global information from coarser graphs, thus in most cases both still give good quality drawings. Nevertheless, for some problems, the adverse effect can be seen.

□ 6.2. Comparison of Drawings

In the following, we give drawings of the graphs in Table 1. Our drawings of c-fat500-10 are the same as in [2] and are thus not included here. All our drawings are done in 2D, as we found that 2D drawings give us good representation.

Figure 7 (left) gives drawings of 4970 using MSE(∞). In this case the mesh around three corners is cluttered compared with the drawing in [2]. We believe this is due to the peripheral effect discussed in Section 3.1, which is reduced when there is a cutoff radius, as in Figure 10(b) of [2], and in MSE(2) (middle). An alternative way to reduce the peripheral effect is to explicitly use a weaker repulsive force model (3), as the drawing by MSE(∞, 2) (right) shows.

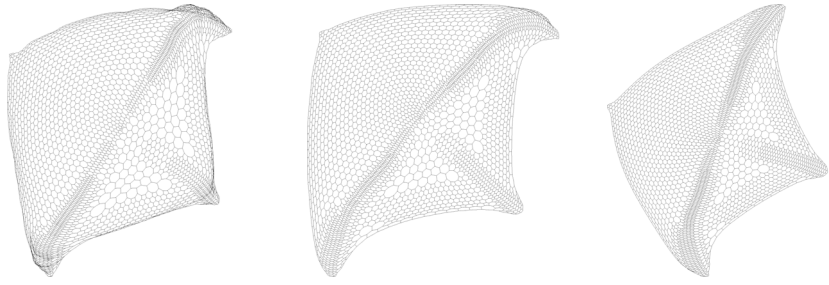

Figure 7. Drawings of 4970 by MSE(∞) (left), MSE(2) (middle), and weaker repulsive force model MSE(∞, 2) (right).

Figure 8 (left) gives the drawing of finan512 using MSE(∞). This drawing is more appealing than Figure 13 of [2]. In that drawing, the circle is elongated, with the "knobs" flat and close to the circle. We believe this is due to the effect of ignoring the long-range repulsive force, so that the circle does not have enough force to make it rigid and rounded, and the knobs do not have enough force to push them out. We observed a similar side effect when we looked at the drawing given by MSE(2) in Figure 8 (right), where the circle is twisted, although it does draw the knobs well.

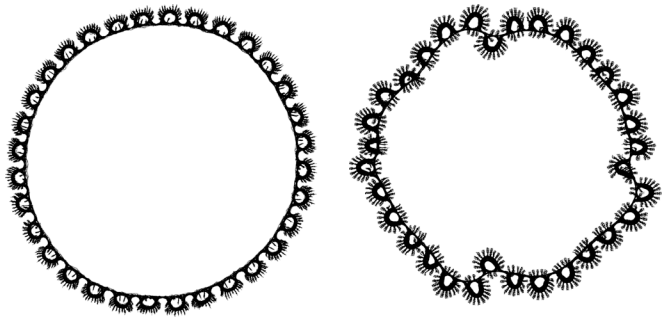

Figure 8. Drawings of finan512 by MSE(∞) (left) and MSE(2) (right).

Figure 9 shows the drawings of dime20. The drawings are somewhat different from Figure 14(b) of [2]. In that drawing, the tip we see in Figure 9 at the top of the larger hole protrudes through the outer rim. Also, in [2], this large hole was replaced by a figure eight. Comparing the drawings of MSE(∞) and MSE(2), the drawing by MSE(2) has thicker outer rims, because of the weakened repulsive force, and thus reduced peripheral effect.

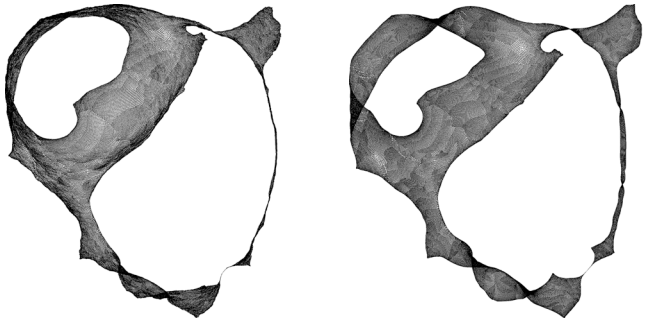

Figure 9. Drawings of dime20 by MSE(∞) (left) and MSE(2) (right).

Figure 10 gives the drawings of add32. The drawings are different from Figure 16(a) in [2] in that they occupy a larger area. Comparing the drawings of MSE(∞) and MSE(2), the latter is "fluffier" in that the branches extend to occupy more space. This is another example of the peripheral effect of a strong repulsive force in MSE(∞). We found that for tree-type applications, drawings by MSE(∞) tend to have leaves and some branches clinging to the main branches. MSE(2) suffers less because of the weakened repulsive force due to the cutoff radius. For these type of applications, it is often better to apply the general repulsive force model (3) with a weaker force given by $p > 1$. Figure 11 shows drawings with $p = 2$ (MSE(∞, 2)) and $p = 3$ (MSE(∞, 3)). In our view they give more details.

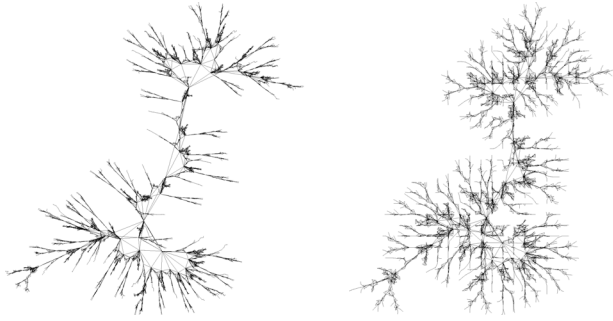

Figure 10. Drawings of add32 by MSE(∞) (left) and MSE(2) (right).

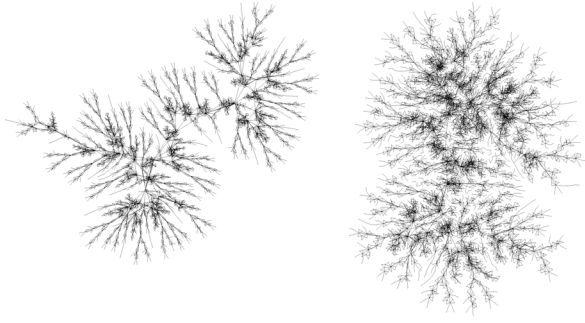

Figure 11. Drawings of add32 with a weaker repulsive force by MSE(∞, 2) (left) and MSE(∞, 3) (right).

Figure 12 gives the drawings of sierpinski10. In [2] Walshaw uses a 3D layout since he found 2D layout unsatisfactory. We found our 2D layout to be good, particularly MSE(2). MSE(∞) demonstrates again the peripheral effect. The strong repulsive force pushes some of the vertices out. The bottom of Figure 12 shows the result of using a general repulsive force model (3) with weaker force given by $p = 2$, which does not suffer from the peripheral effect.

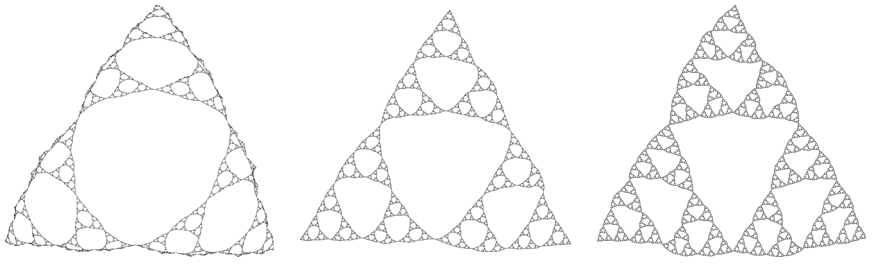

Figure 12. Drawings of sierpinski10 by MSE(∞) (left), MSE(2) (middle), and MSE(∞, 2) (right).

Figure 13 gives drawings of mesh100. Compared with the drawing in Figure 18(b) of [2], our drawings bear a closer resemblance to the actual mesh given in Figure 18(a) of [2].

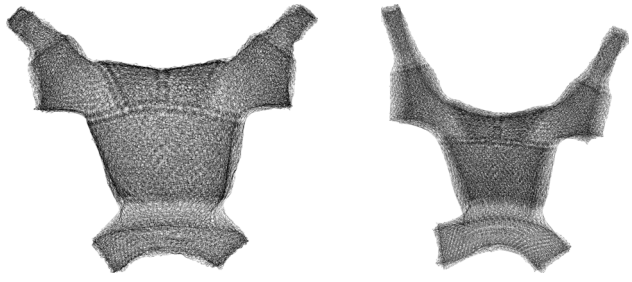

Figure 13. Drawings of mesh100 by MSE(∞) (left) and MSE(2) (right).

Overall, our algorithms give comparable drawings to those in [2]. For difficult graphs, notably finan512 and sierpinski10, we achieve more appealing drawings. The general repulsive force model (3) offers choices to overcome the peripheral effect and can sometimes give more appealing drawings. As a further example, Figure 14 shows a good alternative drawing of finan512, by using a slightly weaker repulsive force. A comparison with Figure 8 (left), shows that without a very strong repulsive force to push them outward, some of the "spikes" now point inward. We found that further weakening of the repulsive force would cause the circle not to be rounded, much like what happens in Figure 8 (right).

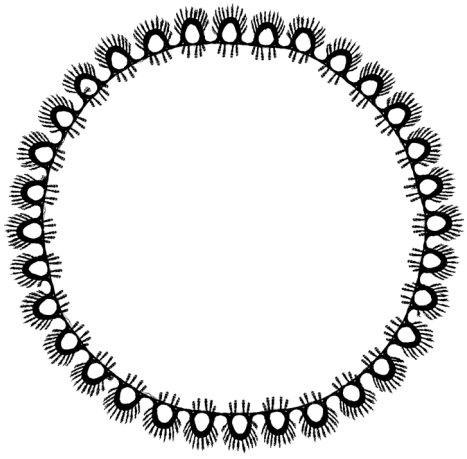

Figure 14. An alternative drawing of finan512 by MSE(∞, 1.3).

□ 6.3. Comparing the Spring-Electrical Model with the Spring Model

In addition to efficiency considerations that favor the spring-electrical model, we also found that compared with the spring model, it gives good drawings for most graphs. The spring model, on the other hand, works particularly well for graphs originated from uniform or near-uniform meshes, albeit requiring more CPU time. The spring model works well for such graphs because it is possible to lay them out so that the physical distance is very close to the graph distance of vertices.

For example, Figure 4 (right) gives a drawing of jagmesh1 using MSE(∞). It is seen that closer to the outer boundary, the peripheral effect of the spring-electrical model is obvious. This effect can be reduced using a weakened repulsive force, as Figure 15 (left) shows using MSE(∞, 2). However, the spring model, MS(4), gives probably the most appealing drawing, as shown in Figure 15 (right). The same happens to sierpinski10 (Figure 16) and mesh100 (Figure 17). Both come very close to what the graphs look like in their original layout, in Figure 17(a) and Figure 18(a) of [2]. MS(4) also draws the data graph quite well, compared with MSE(∞), as shown in Figure 18.

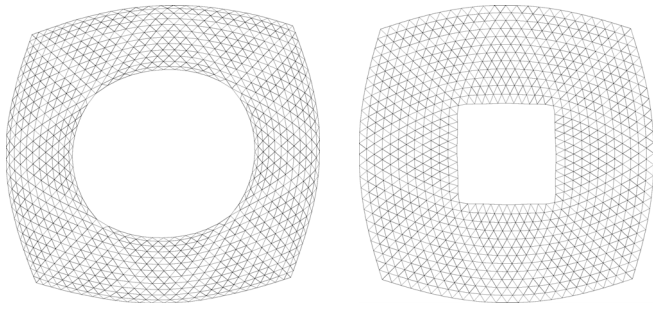

Figure 15. Alternative drawings of jagmesh1 of Figure 4 by MSE(∞, 2) (left) and MS(4) (right).

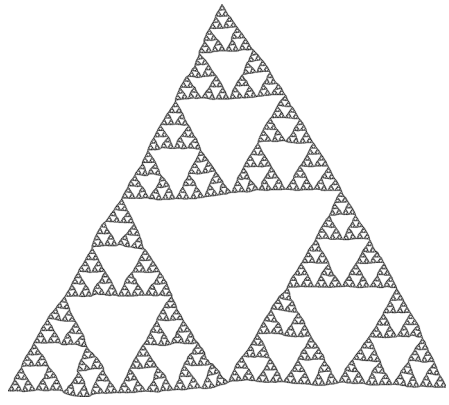

Figure 16. A drawing of sierpinski10 using MS(4).

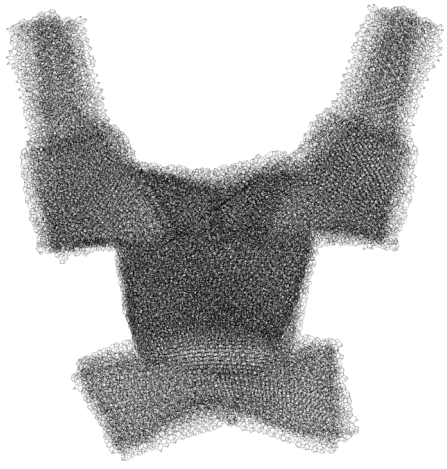

Figure 17. A drawing of mesh100 using MS(4).

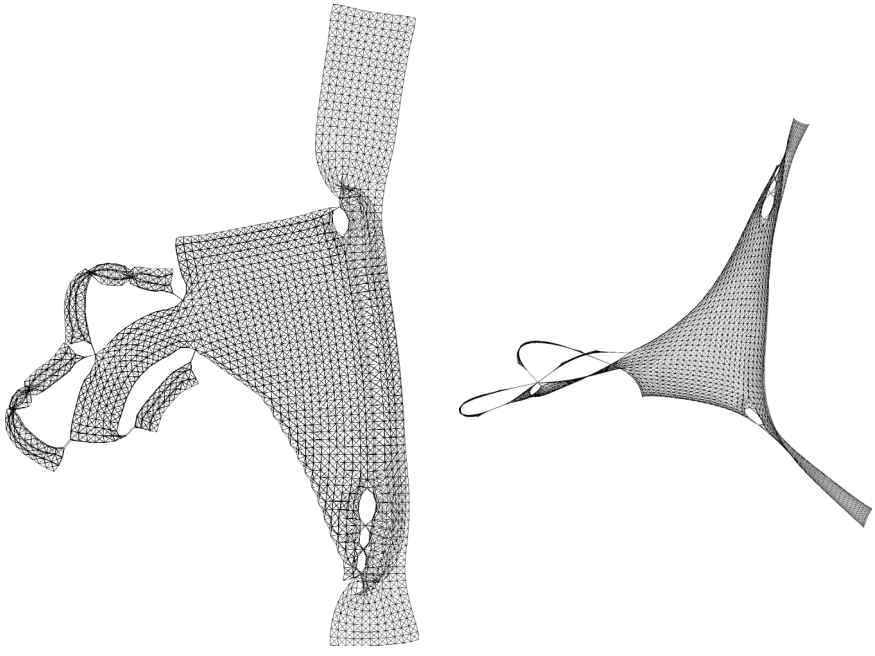

Figure 18. Drawings of data using MS(4) (left) and MSE(∞) (right).

However, for graphs that come from a locally refined mesh, the spring model works poorly. For example, Figure 19 shows the drawing of 4elt by MS(4) (right) and MSE(∞). It is clear that MS(4) strives to draw the graph as uniformly as

possible, but since this is not possible for such a highly refined graph, and in the absence of a strong long-range repulsive force, a lot of foldings occur near the highly refined regions. Therefore, overall we favor the multilevel SE algorithm for its efficiency and general good quality of drawings.

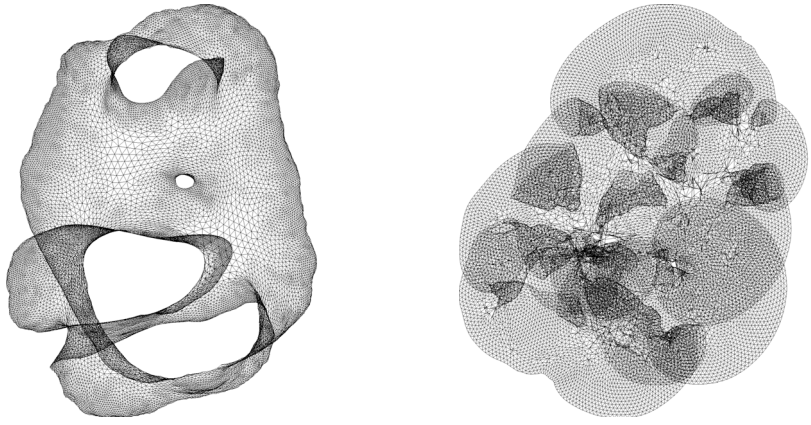

Figure 19. Drawings of 4elt by MSE(∞) (left) and MS(4) (right).

□ 6.4. Further Examples

In this section we demonstrate our algorithms with further examples from the University of Florida Sparse Matrix Collection (www.cise.ufl.edu/research/sparse/matrices). Three of the graphs (skirt, bodyy6 and pwt) have known layouts. Table 3 describes the graphs and the CPU time taken to lay these out in 2D.

Graph	$\|V\|$	$\|E\|$	Diameter	Graph Type	CPU
skirt	12598	91961	98[1]	NASA matrix	8.4
bodyy6	19366	57421	122	NASA matrix	14.4
pwt	36519	144794	262[2]	NASA matrix	25.0
pkustk01	22044	478668	26	Beijing botanical exhibition hall	14.3
pkustk02	10800	399600	33	Feiyue twin tower building	9.5

Table 3. Problem description and CPU time (in seconds) for some graphs. [1]: skirt has seven components; four of them are nontrivial and have diameters 98, 68, 68, and 39, respectively. [2]: pwt has 57 components, 56 components are just a single vertex.

Figure 20 shows the original layout of the skirt graph. It is surprising to us that this mesh actually consists of seven components; three of these are just an isolated vertex. Of the four nontrivial components, one is the main tube and nozzle, another is the skirt/struts at the bottom of Figure 20 (right). The other two

components are the two "rings" connecting the main tube/nozzle and the skirt. They are seen in Figure 20 (right) as one thick horizontal belt.

Figure 21 shows the drawings of the tube/nozzle and the skirt/struts. The tube/nozzle has a much finer mesh near the nozzle end, thus the drawing has the nozzle part expanded. The drawing of the skirt/struts is interesting. In the drawing, two struts protruding out of Figure 21 (left) are drawn separated from three pieces of the skirt, revealing the weak linkage between the struts and the pieces of skirts.

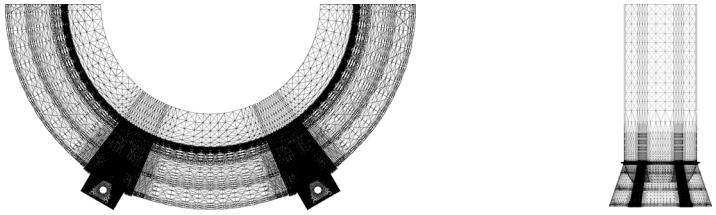

Figure 20. Original layout of skirt: two views.

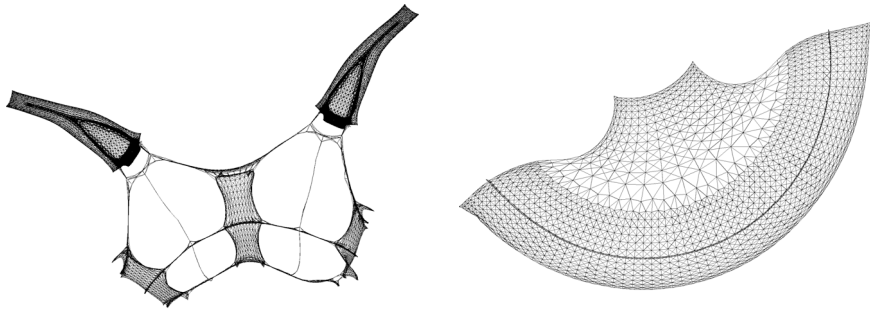

Figure 21. Drawings of two components of skirt by MSE(∞): the skirt/struts (left); the tube/nozzle (right).

Figure 22 (left) shows the original layout of a highly refined mesh. The mesh is highly refined around the middle void and to its right. The drawing algorithm, in its effort to draw edges as uniformly as possible, turned the mesh inside out (Figure 22, right). The hole in the middle is actually the middle void in the original mesh. The original mesh is so highly refined to the right of the middle void that the drawing shows a folding. However, contrary to our intuition, using algorithms MSE(∞, 2) or MSE(2), both having a weaker repulsive force, removes the folding (Figure 23). So it seems the folding is due to the strong repulsive force of the spring-electrical model, another example of the peripheral effect. The original mesh is nearly symmetric and all the drawings also exhibit good symmetry.

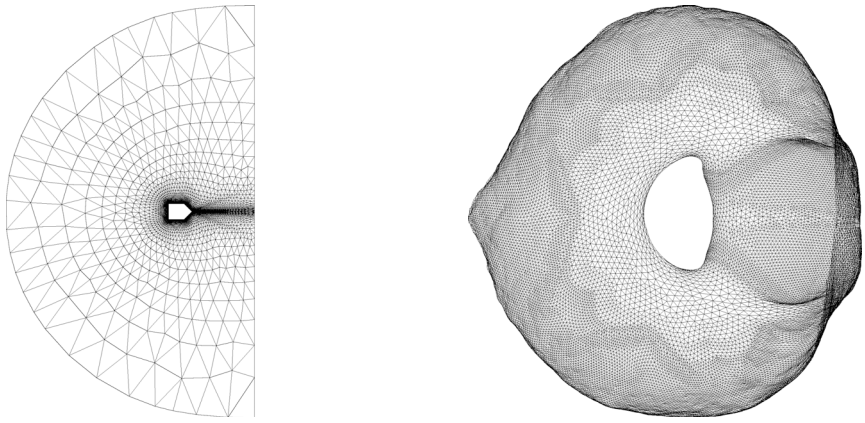

Figure 22. Original layout of bodyy6 (left) and drawing by MSE(∞) (right).

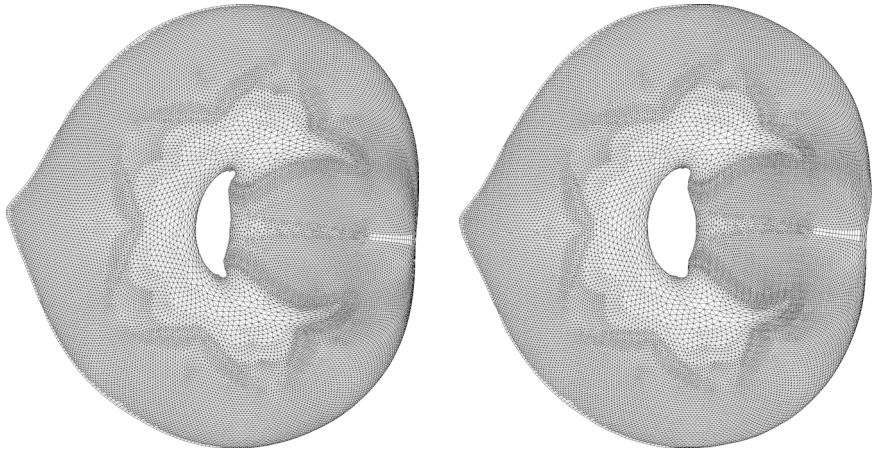

Figure 23. Drawings of bodyy6 using MSE(∞, 2) (left) and MSE(2) (right).

Figure 24 (left) shows the pwt mesh, which is probably a mesh for a pressured wind tunnel. The drawing by MSE(∞) corresponds to the original layout well. The large chamber in the original mesh has a mesh density similar to the pipe and is thus indistinguishable from the pipe in the drawing. In Figure 25 close-up views of the smaller chamber in the original layout and our corresponding drawing are given. The drawing depicts the details well.

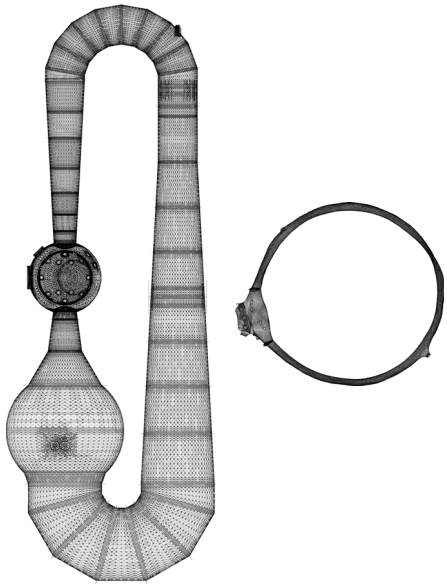

Figure 24. Original layout of pwt (left) and a drawing by MSE(∞) (right).

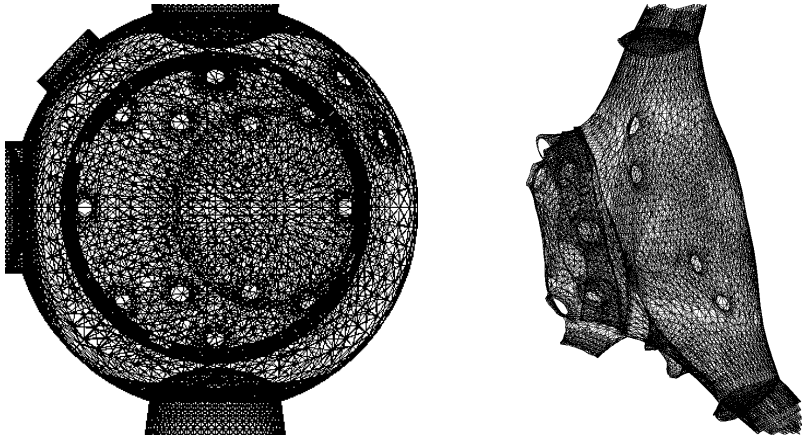

Figure 25. Close-up view of the original layout of pwt (left) and a drawing by MSE(∞) (right).

Finally, Figure 26 shows drawings of pkustk01 and pkustk02. These two graphs have high average degrees (43 and 74, respectively). However, judging by the drawings, our algorithms performed very well.

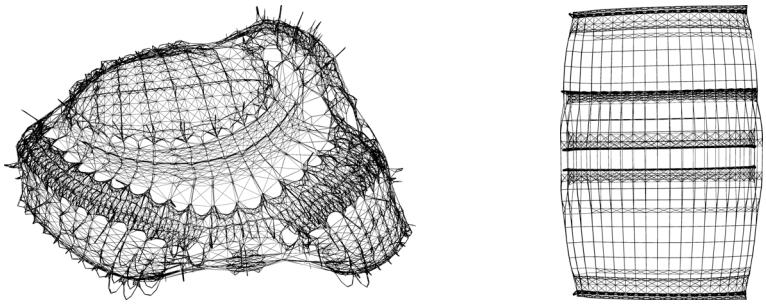

Figure 26. Drawings of pkustk01 (left) and pkustk02 (right) using MSE(∞).

■ 7. Conclusions and Future Work

In this article we proposed an algorithm that uses a multilevel approach to find global optimal layouts and the octree technique to approximate short- and long-range forces satisfactorily and efficiently. These two techniques were each proposed earlier for graph drawing [2, 16, 18], but as far as we are aware, were never combined to form one powerful algorithm for large-scale graph drawing. A number of practical techniques, including adaptive step and octree depth control, and a hybrid coarsening scheme, were introduced for the algorithm to work effectively. This algorithm is demonstrated to be both efficient and of high quality for large graphs, competitive to Walshaw's [2] highly successful graph drawing algorithm, yet gives better drawings on some difficult problems.

We also proposed a general repulsive force model to overcome the peripheral effect of the spring-electrical model. Finally, we compared the spring-electrical model with the spring model and demonstrated examples where the spring model may be suitable.

Both the multilevel approach and the octree data structure do have limitations. For example, both the EC coarsening scheme and the MIVS-based coarsening scheme would not work effectively on star graphs (a graph with one vertex connected to all other vertices and no two other vertices connected). The former would coarsen too slowly thus having unacceptable complexity, while the latter would coarsen too fast and not preserve the graph information on the coarser graphs. The parameter θ in the Barnes and Hut octree algorithm is empirically fixed (to 1.2 in all our drawings). We experienced one case where this value gives an artifact only corrected with a smaller value $\theta = 0.8$. It may be possible to correct this artifact without changing the θ value by adding a random offset to the first square of the octree. These limitations remain a topic for further investigations. However, in general the proposed algorithm performed extremely well on a range of graphs from different application areas, a small number of which were shown.

The graph drawing algorithm presented in this article is in the *Mathematica* 5.1 release (November 2004).

After the completion of this article, our attention was drawn to an independent work by Hachul and Jünger [28]. In that paper, the multilevel approach is combined with the multipole expansion technique [29] to give a $O(|V|\log(|V|)+|E|)$ algorithm. The efficiency and quality of the algorithm in that paper appear to be at the same level as this article, although many details differ, including the multilevel scheme and the force approximation.

■ Acknowledgments

The author would like to thank Chris Walshaw for providing the graphs in Table 1 and Andrew A. de Laix for bringing space decomposition techniques to my attention.

■ References

[1] J. Barnes and P. Hut, "A Hierarchical *O*(*n* log *n*) Force-Calculation Algorithm," *Nature*, **324**(4), 1986 pp. 446–449.

[2] C. Walshaw, "A Multilevel Algorithm for Force-Directed Graph Drawing," *Journal of Graph Algorithms and Applications*, **7**(3), 2003 pp. 253–285.

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

[4] T. M. J. Fruchterman and E. M. Reigold, "Graph Drawing by Force-Directed Placement," *Software—Practice and Experience*, **21**(11), 1991 pp. 1129–1164.

[5] P. Eades, "A Heuristic for Graph Drawing," *Congressus Numerantium*, **42**, 1984 pp. 149–160.

[6] N. R. Quinn and M. A. Breuer, "A Force Directed Component Placement Procedure for Printed Circuit Boards," *IEEE Transactions on Circuits and Systems*, **CAS-26**(6), 1979 pp. 377–388.

[7] T. Kamada and S. Kawai, "An Algorithm for Drawing General Undirected Graphs," *Information Processing Letters*, **31**(1), 1989 pp. 7–15.

[8] Y. Koren, L. Carmel, and D. Harel, "Drawing Huge Graphs by Algebraic Multigrid Optimization," *Multiscale Modeling and Simulation*, **1**(4), 2003 pp. 645–673.

[9] D. Harel and Y. Koren, "Graph Drawing by High-Dimensional Embedding," *Journal of Graph Algorithms and Applications*, **8**(2), 2004 pp. 195–214.

[10] G. Karypis and V. Kumar, "Multilevel *k*-way Partitioning Scheme for Irregular Graphs," *Journal of Parallel and Distributed Computing*, **48**(1), 1998 pp. 96–129.

[11] B. Hendrickson and R. Leland, "A Multilevel Algorithm for Partitioning Graphs," *Technical Report* SAND93-1301, Albuquerque, NM: Sandia National Laboratories, 1993. Also in *Proceeding of Supercomputing'95 (SC95)*, San Diego, CA www.supercomp.org/sc95/proceedings/509_BHEN/SC95.HTM.

[12] C. Walshaw, M. Cross, and M. G. Everett, "Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes," *Journal of Parallel and Distributed Computing*, **47**(2), 1997 pp. 102–108.

[13] D. Harel and Y. Koren, "A Fast Multi-Scale Method for Drawing Large Graphs," *Journal of Graph Algorithms and Applications*, **6**(3), 2002 pp. 179–202.

[14] R. Hadany and D. Harel, "A Multi-Scale Algorithm for Drawing Graphs Nicely," *Discrete Applied Mathematics*, **113**(1), 2001 pp. 3–21.

[15] P. Gajer, M. T. Goodrich, and S. G. Kobourov, "A Fast Multi-Dimensional Algorithm for Drawing Large Graphs," *Lecture Notes in Computer Science*, **1984**, New York: Springer-Verlag, 2000 pp. 211–221.

[16] D. Tunkelang, "A Numerical Optimization Approach to General Graph Drawing," Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1999.

[17] A. Quigley and P. Eades, "Fade: Graph Drawing, Clustering, and Visual Abstraction," *Lecture Notes in Computer Science*, **1984**, New York: Springer-Verlag, 2000 pp. 183–196.

[18] A. Quigley, "Large Scale Relational Information Visualization, Clustering, and Abstraction," Ph.D. thesis, Department of Computer Science and Software Engineering, University of Newcastle, Australia, 2001.

[19] R. Davison and D. Harel, "Drawing Graphs Nicely Using Simulated Annealing," *ACM Transactions on Graphics*, **15**(4), 1996 pp. 301–331.

[20] R. Fletcher, *Practical Methods of Optimization,* 2nd ed., New York: John Wiley & Sons, 2000.

[21] K. J. Pulo, "Recursive Space Decompositions in Force-Directed Graph Drawing Algorithms," in *Proceedings of the Australian Symposium on Information Visualisation (InVis.au 2001)*, Sydney, Australia (P. Eades and T. Pattison, eds.), *Conferences in Research and Practice in Information Technology Series,* **9**, Darlinghurst, Australia: Australian Computer Society, 2001 pp. 95–102.

[22] S. Pfalzner and P. Gibbon, *Many-Body Tree Methods in Physics*, New York: Cambridge University Press, 1996.

[23] A. Gupta, G. Karypis, and V. Kumar, "Highly Scalable Parallel Algorithms for Sparse Matrix Factorization," *IEEE Transactions on Parallel and Distributed Systems*, **8**(5), 1997 pp. 502–520.

[24] Y. F. Hu and J. A. Scott, "A Multilevel Algorithm for Wavefront Reduction," *SIAM Journal on Scientific Computing*, **23**(4), 2001 pp.1352–1375.

[25] C. Walshaw, "A Multilevel Approach to the Travelling Salesman Problem," *Operations Research*, **50**(5), 2002 pp. 862–877.

[26] C. Walshaw, "Multilevel Refinement for Combinatorial Optimisation Problems," *Annals of Operations Research*, **131**, 2004 pp. 325–372; originally published as *University of Greenwich Technical Report* 01/IM/73, 2001.

[27] S. T. Barnard and H. D. Simon, "A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems," *Concurrency: Practice and Experience*, **6**(2), 1994 pp. 101–117.

[28] S. Hachul and M. Jünger, "Drawing Large Graphs with a Potential-Field-Based Multilevel Algorithm," in *Proceedings of the Twelfth International Symposium on Graph Drawing (GD 2004)*, New York (J. Pach, ed.), *Lecture Notes in Computer Science*, **3383**, New York: Springer-Verlag, 2004 pp. 285-295.

[29] L. F. Greenguard, *The Rapid Evaluation of Potential Fields in Particle Systems,* ACM Distinguished Dissertations Series, Cambridge, MA: The MIT Press, 1988.

Yifan Hu
Senior Developer
Wolfram Research, Inc.
100 Trade Center Drive
Champaign, IL 61820, USA
yifanhu@wolfram.com