Efficient and High Quality Force-Directed Graph Drawing

Yifan Hu*
Wolfram Research Inc
100 Trade Center Drive
Champaign, IL 61820, USA

Abstract

We propose a graph drawing algorithm that is both efficient and of high quality. This algorithm combines a multilevel approach which effectively overcomes local minimums, with Barnes and Hut [2] octree technique which approximates short and long range force efficiently. Our numerical results show that the algorithm is competitive in speed to Walshaw's [26] highly efficient multilevel graph drawing algorithm, yet gives better drawing 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.

1 Introduction

Graphs are often used to encapsulate the 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, and with symmetry that may exist in the graph depicted. This problem has been studied extensively in the literature (e.g., [3]), and many approaches were proposed. In this paper we concentrate on drawing undirected graphs with straight-line edges, using force-directed methods (e.g., [7, 5, 22, 14]). Force-direct methods however are one of many classes of methods proposed for straight edge drawing. Other methods include, for example, the spectral method [16] and the high dimensional embedding method [11].

A force-directed algorithm models the graph drawing problem by 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

*Email: yifanhu@wolfram.com

many variations of force-directed algorithms. The algorithm of Fruchterman and Reigold [7], which is based on the work of [5, 22], models the graph drawing problem by a system of springs between neighboring vertices of the graph, which pull the vertices together. At the same time, repulsive electrical forces that exist push all vertices away from each other. The algorithm of Kamada and Kawai [14], 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 they subject to. This amount may be large initially, but reduces gradually, based on a "cooling schedule".

There are two limiting factors for standard force-directed algorithms in drawing large graphs. 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 [7], 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 [14] requires the calculation of the graph distance among all vertices, and 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 calculation of the graph distances on the fly, instead of storing them in the memory.

To overcome the first limiting factor, multilevel approach was proposed. This idea was successfully used in many fields, including that of graph partitioning (e.g., [15, 12, 27]) and was found to be able to overcome the localized nature of the Kernighan-Lin algorithm. An idea of that kind for use in graph drawing was alluded to in Fruchterman and Reigold [7], where, in the context of overcoming local minimum, 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 [10], extending earlier work of Hadany and Harel [9], proposed the so-called multi-scale approach. In that approach, a sequence of coarser and coarser graphs are formed by finding the k-centers, and the distance matrix associated with the k-centers. They used the Kamada and Kawai spring model [14], thus incurring a high computational complexity of $O(|V||E|)$ for distance calculation and memory complexity of $O(|V|^2)$. 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* [17] built up the multilevel of graphs by using maximal independent vertex sets, with the i-th level consisting of a maximal independent vertex set such that the vertices are of distance $\geq 2^{i-1}+1, i = 1, 0, ...$ 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 [26] proposed a multilevel algorithm and demonstrated that it was able to draw graphs as large as a quarter of 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 force between vertices are based on the Fruchterman and Reigold spring-electrical model [7]. 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 by Fruchterman and Reigold 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 [7]. Such a practice however comes at a cost. Because long range forces are ignored, there is no force to evenly distribute far away vertices. When used within multilevel approach, Walshaw [26] 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, however, we found that the lack of long range force did hurt at least one, if not more, of the drawings in [26].

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 $O(nlog(n))$ time with good accuracy, but without ignoring long range force, is to treat groups of far away vertices as a supernode, using a suitable data structure, as in the Barnes and Hut algorithm [2]. This idea was implemented for graph drawing by both Tunkelang [23], and Quigley and Eades [21, 20]. 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, although the algorithm is not suitable for large graphs because conjugate gradient is a local optimization algorithm, and that the number of conjugate gradient iterations increases as the size of the graph increases. Quigley and Eades [21, 20] also used the Barnes and Hut algorithm for efficient and accurate force calculation. In addition, they employed a multilevel scheme, what 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 paper we propose an algorithm that is both efficient and of high quality for large graphs. We combine 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 octree/quadtree in the Barnes and Hut algorithm. Our numerical results show that the algorithm is competitive with Walshaw's [26] highly efficient graph drawing algorithm, yet gives better drawings 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.

The rest of this paper is organized as follows. 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 [26]. We conclude the paper by suggesting some future works in Section 7.

2 Definitions and Notations

We use $\mathcal{G} = \{V, E\}$ 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 $diam(\mathcal{G})$ 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 $\mathcal{G}$ 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 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 [7], also know as 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

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

The combined force on a vertex i is therefore

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

In the above formulas, K is a parameter known as the optimal distance [7], or natural spring length [26]. The parameter C regulates the relative strength of the repulsive and attractive forces, and was introduced in [26]. 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

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

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

From 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 spring-electrical model $Energy_{se}(x,K,C)$, then sx^* minimizes $Energy_{se}(x,K',C')$, 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

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

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

$$Energy_{se}(x,K,C) = \left(\frac{C}{C'}\right)^{4/3} \left(\frac{K}{K'}\right)^2 Energy_{se}(sx,K',C').$$

□

Even though the parameters K and C do not have any bearing on the optimal layout from Mathematical point of view, from algorithmic point of view, if an

iterative algorithm (see Algorithm 1 later) 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 paper, unless otherwise specified, we fix $C = 0.2$ as in [26], but vary K for this purpose.

One intrinsic feature of graph drawings by the spring-electrical model is that vertices in the peripheral tends to be closer to each other than those in the center, even for a uniform mesh. We call this *peripheral effect*. For example, Figure 1 shows a mesh of 100 vertices laid out using the Fruchterman and Reigold model [7], with vertices near the outside boundary clearly closer to each other than those near the center.

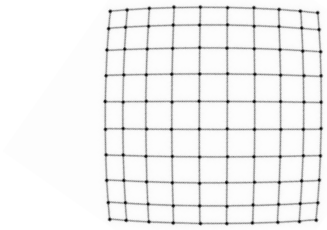

Figure 1: A 10 x 10 regular mesh laid out using the spring-electrical model

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 accurate layout under 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 using 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 $4.143/1.523 = 2.72$.

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

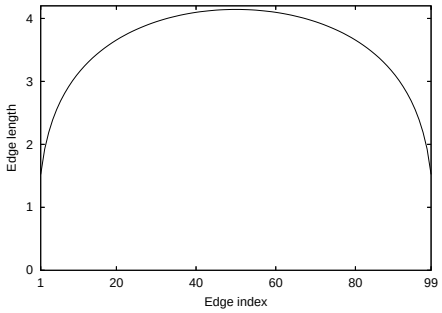

Figure 2: Distribution of edge length for a line graph of 100 vertex and 99 edges, laid out using Fruchterman and Reigold model.

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

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 (3) with $p = 2$ and $p = 3$, and the Fruchterman and Reigold force model (1), which corresponding to the general model with $p = 1$. As can be seen, the general model with $p > 1$ reduces the peripheral effect significantly.

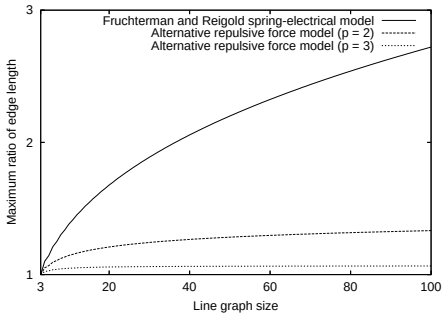

Figure 3: The maximal ratios of edge length for line graphs of size 3 to 100 vertices.

In force model of Kamada and Kawai [14], also know as 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

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

and the spring energy is

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

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 (4). The algorithm starts with a random, or user supplied, initial layout.

Algorithm 1: An iterative force-directed algorithm

- ForceDirectedAlgorithm($\mathcal{G}, x, tol$) {
  - $converged = FALSE$;
  - $step =$ 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 $(j \neq i,\ j \in V)$ $f := f + \frac{f_r(i,j)}{||x_j - x_i||}(x_j - x_i)$;
      - $x_i := x_i + step * (f/||f||)$;
      - $Energy := Energy + ||f||^2$;
    - }
    - $step := update\_steplength(step, Energy, Energy^0)$;
    - if $(||x - x^0|| < K\ tol)$ $converged = TRUE$;
  - }

  - return x;
- }

As in [26], 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 improved the convergence of the iterative procedure, much like the fact that among stationary iterative linear system solvers, Gauss-Seidel algorithm is often faster than Jacobi algorithm.

In the above iterative algorithm, it is necessary to update the step length *step*. The "cooling schedule" used in most expositions (e.g., [4]) of force-directed algorithms allow large movements (large step length) at the beginning of the iterations, but the step length reduces as the algorithm progresses. Walshaw [26] used a simple scheme,

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

with $t = 0.9$. We found this to be adequate in the refinement phase of multilevel a force-directed algorithm. However for an application of force direct 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 [6], 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 above, *progress* is a static variable that is initialized to zero, and parameter $t = 0.9$. The idea of the above algorithm is that the steplength is kept unchanged if energy is being reduced, and increased to $step/t$ if the energy is reduced more than 5 times in a row. We only reduce the steplength if energy increases.

Compared with the simple step length update scheme (6), the adaptive scheme is much better in escaping from local minimum. Figure 4 shows the result of applying 70 iterations of Algorithm 1 to the jagmesh1 graph with 936 vertices from MatrixMarket (`http:///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.

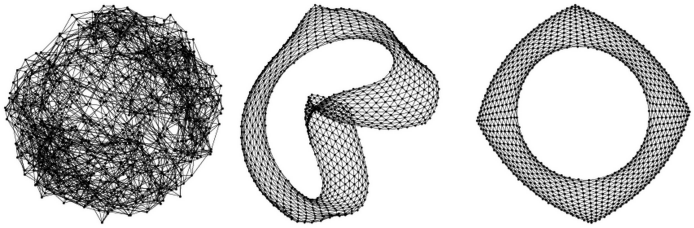

Figure 4: Comparing adaptive and simple steplength update schemes on jagmesh1 after 70 iterations. Left: simple scheme; middle: adaptive scheme; right: what the layout should look like.

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 technique to calculate the repulsive forces in $O(nlogn)$ 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 [2]. This idea was adopted by Tunkelang [23] and Quigley [20], both used an octree (3D) or quadtree (2D) data structure. In principal, other space decomposition methods can also be used. For example, Pulo [19] 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 4 squares (or 8 cubes) if it contains more than 1 vertex, and forms the level 1 squares. This process is repeated until level L, where each square contains no more than 1 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, $x_S = (\sum_{j \in S} x_j)/|S|$. The repulsive force on vertex i from this supernode is

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

It remains to define what "far" means. Following [23, 20], we define the supernode S to be far away 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,

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

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 paper. This inequality (7) is called the Barnes-Hut opening criterion, and was originally formulated by Barnes and Hut [2].

The octree data structure allows efficient identification of all the supernodes that satisfies (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.

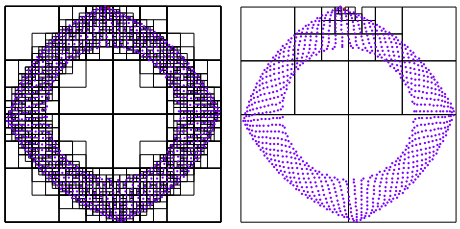

Figure 5: An illustrate of the octree data structure. Left: the overall octree. Right: supernodes with reference to a vertex at the top middle part of the graph, with $\theta = 1$.

Under reasonable assumption [2, 18] 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, using 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 that 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 take a fraction of the total time, and the majority of the time in Algorithm 1 is spend 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 sub-dividing 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 not as efficient as it could be. In particular, a disproportional large amont 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 lie in the same square 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 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 different max_tree_level from run to run, thus in turn gives different layout between runs, which is undesirable. Instead, we use

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

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 steplength control scheme we introduce goes in some way toward a better layout, but as Figure 4 shows, is still insufficient toward a global minimum.

Multilevel approach has been used in many large scale combinatorial optimization problems, such as graph partitioning [8, 12, 27], matrix ordering [13], the travel salesman problem [25], and is proved to be a very useful meta-heuristic tool [24].

Multilevel approach was also used in graph drawing [9, 10, 26]. In particular, Walshaw [26] was able to layout graphs up to 225,000 vertices in a few minutes, and largely to good quality.

The multilevel approach has three distinctive phases: coarsening, coarsest graph layout, and finally, prolongation and refinement. In the coarsening phase, a series of coarser and coarser graphs, $\mathcal{G}^0, \mathcal{G}^1, \ldots, \mathcal{G}^l$, are generated, the aim is for each coarser graph $\mathcal{G}^{k+1}$ to encapsulate the information needed to layout its "parent" $\mathcal{G}^k$, 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. Hereafter, we use superscript to denote the level. E.g., x^k is the coordinates of vertices in the level k graph $\mathcal{G}^k$, $k = 0, \ldots, l$.

5.1 Graph coarsening

There are a number of ways to coarsen an undirected graph. One often used method is based on edge collapsing (EC) [12, 8, 27], 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 one. During coarsening,

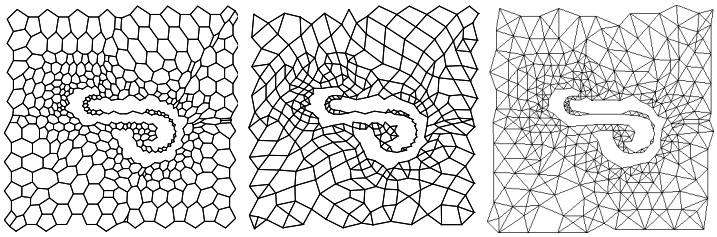

Figure 6: An illustration of graph coarsening: original graph with 788 vertices (left); a coarser graph with 403 vertices resulted from edge-clasping (center); a coarser graph with 332 vertices resulted from maximal independent vertex set (right).

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 a *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 [26] choose to keep vertex weights in the coarser graphs as uniform as possible by match a vertex with a neighboring with the smallest vertex weight. We found that both heavy-edge matching and matching with vertex of smallest weight give graph layout of similar quality, and used the former in this paper. Figure 6 illustrates a graph (left) and the result (middle) of coarsening using edge collapsing.

Other coarsening methods have been proposed. In [1], 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 the maximal independent vertex set by an edge if their distance apart is no greater than three. Figure 6 illustrates a graph (left) and the result (right) of coarsening using maximal independent vertex set.

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, $\mathcal{G}^{i+1}$, may be very close to the number of vertices in the original graph, $\mathcal{G}^i$. 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 $\mathcal{G}^i$.

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 threshold (9) is breached, we use MIVS coarsening instead.

5.2 Initial layout on the coarsest graph

On the coarsest level, we layout the graph using Algorithm 1 combined with the adaptive steplength 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 layout on the coarser graphs are recursively prolonged to the finer graphs, with further refinement at each level.

If graph $\mathcal{G}^{i+1} = \{V^{i+1}, E^{i+1}\}$ was derived using EC from $\mathcal{G}^i = \{V^i, E^i\}$, 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 $\mathcal{G}^{i+1}$ was derived using MIVS from $\mathcal{G}^i$, 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 $\mathcal{G}^i$, this layout is refined using Algorithm 1. As in [26], 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 apart. 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 steplength update scheme. The simple scheme (6) works well.

However, although the initial layout in graph $\mathcal{G}^i$ prolongated from the coarser graph $\mathcal{G}^{i+1}$ 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 $\mathcal{G}^i$ is roughly the same as the graph distance of the corresponding vertices in $\mathcal{G}^{i+1}$, 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 $\mathcal{G}^{i+1}$ 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 $\mathcal{G}^i$, 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 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 subsequence levels of graphs,

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

For spring-electrical model, Walshaw [26] 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 4 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 [26].

To avoid the $O(|V|^2)$ complexity of the repulsive force calculation in the spring-electrical model, Walshaw [26], following Fruchterman and Reigold [7], cut off the contributions from far away vertices. Only repulsive forces from vertices within a certain radius were considered. Specifically, on the i-th 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 far away 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 [26] 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 cut-off, which would otherwise be more profound, was largely damped 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 (4), we used a similar strategy, by only calculating the distance of two vertices u and v, and their attractive/repulsive forces, if

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

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 i-th level graph $\mathcal{G}^i$. x^i is the coordinate vector for the vertices in V^i. We represent $\mathcal{G}^i$ by a symmetric matrix G^i, with the entries of the matrix the edge weights. Prolongation operator from $\mathcal{G}_{i+1}$ to $\mathcal{G}_i$ 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 [13]. The starting point is the original graph, $\mathcal{G}_0 = \mathcal{G}$.

Algorithm 2: a multilevel force-directed algorithm
function MultilevelLayout $(\mathcal{G}^i, tol)$

- *Coarsest graph layout*
  - *if* $(n^{i+1} < MinSize$ *or* $n^{i+1}/n^i > \rho)$ {
    - $x^i :=$ *random initial layout*
    - x^i =*ForceDirectedAlgorithm*$(\mathcal{G}^i, x^i, tol)$
    - *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:* $x^i = ForceDirectedAlgorithm(\mathcal{G}^i, x^i, tol)$

– *return* x^i

In the above algorithm, 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 that of Walshaw [26], both in terms of efficiency and quality of layout. These are the details for the algorithms we will demonstrate, and the short names they are denoted by.

- MSE(r) (Multilevel Spring Electrical model): Multilevel Algorithm 2 with HYBRID coarsening scheme, octree data structure, and repulsive/attractive force gives by (1). The cutoff radius for the repulsive force calculation is (11), with $r = \infty$ by default.
- 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 gives by (1). The cutoff radius for the repulsive force calculation is (11), with $r = \infty$ by default. The difference to MSE(r) is that the general repulsive force model (3) is used with parameter p. MSE(r) is the same as MSE($r, 1$).
- MS(r) (Multilevel Spring model): Multilevel Algorithm 2 with HYBRID coarsening scheme, octree data structure, and repulsive/attractive force gives by (4). The cutoff radius for the distance and force calculation is (12), with $r = 4$ by default.
- SE (Spring Electrical model): Algorithm 1 with octree data structure and repulsive/attractive force gives by (1).

For all the above algorithms, we used a tolerance of $tol = 0.01$ to be comparable with Walshaw [26]. All our results are from a 3.0 GHz Petium 4 machine with 2 GB of memory and 512 KB of cache, under Linux operating system. Our code is written in C and compiled with gcc using compilation flag "gcc -O2".

6.1 Comparison with Walshaw [26]

In this section we compare our algorithm with that of Walshaw [26].

Table 1 lists a set of 9 test problems from [26]. 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 direct placement algorithm MLFDP from [26]. To

see the extra cost of 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 9 problems were chosen to be the same as those in Table 3 of [26]. We draw all the graphs in 2D and use these layouts for all the subsequent figures. However to be able to compare with [26], we also laid out the last 4 graphs in 3D, even though sierpinsji10 graph is really a 2D graph and as Figure 12 shows, we can layout in 2D just fine.

Notice that Walshaw's CPU results was for a 1 GHz machine, 1/3 of the clock speed of our machine. However based on our experience of clock speed of computers and their actual performance, we would expect that the performance of our machine to be less than 3 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 of the cutoff radius. So we have two approximations, that of forming a supernode, and that of cutting off far away vertices. 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 counter part 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 graph, SE is more expensive than MSE(∞) for both 2D and 3D layout! A careful examination reveals that in terms of number of operations in the inner most loops of force calculations and octree code, MSE(∞) does take between 2-3 times more operations compared with SE. The abnormality in CPU time is due to poor cache performance of the octree code in SE. The reason for this poor performance is that 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 thus are unpredictable. The multilevel algorithm MSE(∞), on the other hand, always start from a good initial layout. Because for most of the graphs we have 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. Therefore, the good initial layout in the multilevel algorithm means that vertices with close indices also tend to be close in their physical locations. 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 stays roughly the same, while the CPU timings for MSE(∞) becomes about twice larger.

The above analysis may suggest that the multilevel algorithm is susceptible to poor initial vertex indexing. However this is not true, because firstly, large graphs tend to have good natural ordering. Secondly, poor ordering can be easily remedied by preordering the vertices using a suitable ordering algorithm that is inexpensive. For example, we used METIS [15] 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 the multilevel algorithm MSE(∞) on these preordered graphs are very comparable to those in Table 2. In fact for dime20 graph, the CPU time is reduced from 290.6 to 252.3 with nested disection preordering! Nested dissection ordering however has no effect on the cache performance of the single level force-directed algorithm SE, due to its poor initial and subsequence layouts. It is possible to improve SE by ordering the vertices using a nested disection based on physical locations of the vertices, but since SE is not a good graph drawing algorithm anyway, we will not pursue this further here.

Table 1: Description of test problems

graph	$\|V\|$	$\|E\|$	avg. degree	diameter	graph type
c-fat500-10	500	46627	186.5	4	random clique test
4970	4970	7400	3.0	106	2D dual
4elt	15606	45878	5.9	102	2D nodal
finan512	74752	261120	7.0	87	linear programming
dime20	224843	336024	3.0	1179	2D nodal
data	2851	15093	10.6	79	3D nodal
add32	4960	9462	3.8	28	32-bit adder
sierpinski10	88575	177147	4.0	1024	2D fractal
mesh100	103081	200976	3.9	203	3D dual

As we would expect, the multilevel spring model algorithm MS(4) is very slow. This is because the spring model seeks to layout vertices to have a physical distance equals 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 out the individual graph distances first, therefore no saving is achieved. A cutoff radius does allow us to reduce the $O(|V|^2)$ complexity, however we found that for good drawing quality, we have to use a relatively large radius, 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

Table 2: CPU time (in seconds) for some force-directed algorithms

	size		CPU*				
graph	$\|V\|$	$\|E\|$	MLFDP*	MSE(2)	MSE(∞)	MS(4)	SE
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
4elt	15606	45878	24.3	9.4	11.7	102.9	9.2
finan512	74752	261120	363.8	56.6	59.8	3714.9	60.0
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
data	2851	15093	6.6	2.3	2.4	33.0	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	158.0	204.0	6431.3	138.2

*: MLFDP data from [26], that was for a 1 GHz Pentium III. All other timings are all for a 3 GHz Pentium 4.
-: data not available.

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 ignores 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 in the next section.

6.2 Comparison of drawings

In the following, we give drawings of those graphs in Table 1. Our drawings of c-fat500-10 graph are the same as in [26] and are thus not included here. All our drawings are done in 2D as we found that 2D drawings give us good enough representation.

Figure 7 (left) gives drawings of 4970 graph using MSE(∞). In this case the mesh around three corners is cluttered compared with the drawing in [26]. 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 [26], 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($\infty, 2$) (bottom) shown.

Figure 8 (left) gives the drawing of finan512 graph using MSE(∞). This

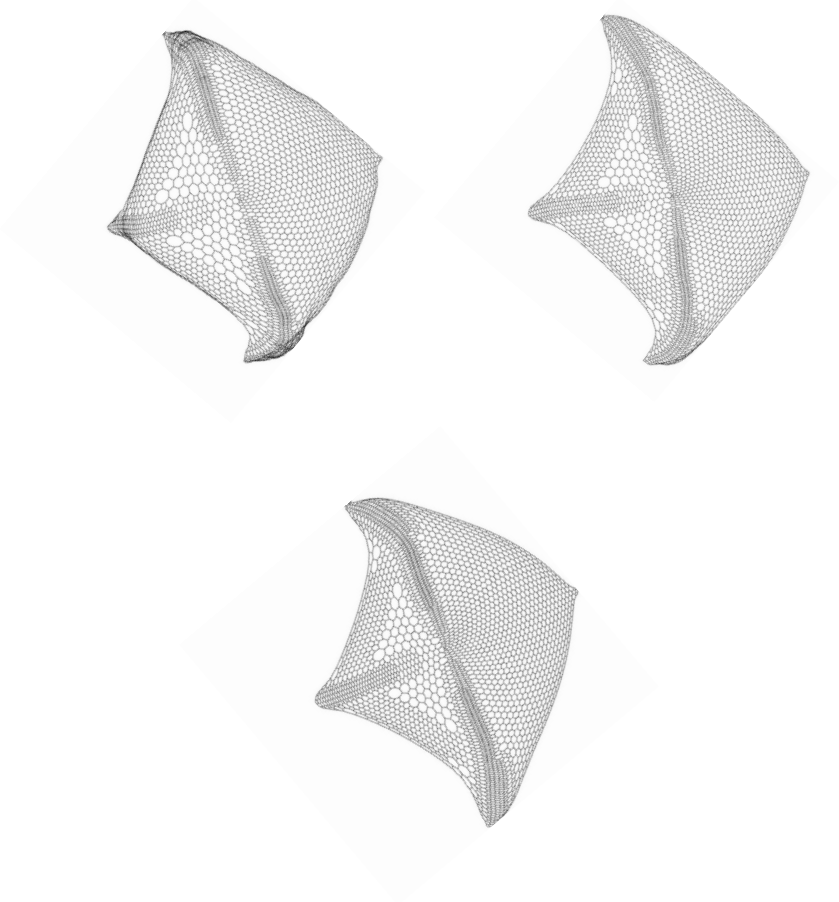

Figure 7: Drawings of 4970 graph by MSE(∞) (top left), MSE(2) (top right) and weaker repulsive force model MSE($\infty, 2$) (bottom)

drawing is more appealing than Figure 13 of [26]. In that drawing, the circle is elongated, with the "knobs" flat and close to the circle. We believe this was 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 look at the drawing given by MSE(2) in Figure 8 (right), where the circle is seen twisted, although it does draw the "knobs" well, compared with [26].

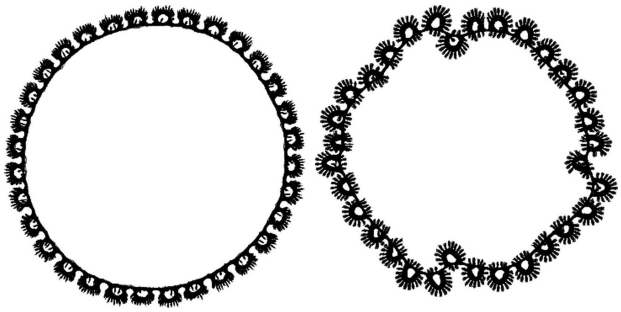

Figure 8: Drawings of finan512 graph by MSE(∞) (left) and MSE(2) (right)

Figure 9 shows the drawings of dime20 graph. The drawings are somewhat different from Figure 14(b) of [26]. In that drawing, the tip we see in Figure 9 at the top of the larger hole protrudes through the outer rim. Also, in that drawing, this large hole was replaced by an "8" shape. Comparing the drawings of MSE(∞) and MSE(2), the drawing by MSE(2) has thicker outer rims, because of the weakened repulsive force, thus reduced peripheral effect.

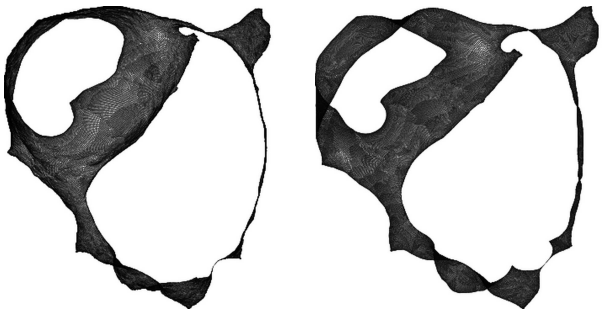

Figure 9: Drawings of dime20 graph by MSE(∞) (left) and MSE(2) (right)

Figure 10 gives the drawings of add32 graph. The drawings are different from Figure 16(a) in [26] in that it occupies 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 of applications, drawings by MSE(∞) tend to have leaves and some branches cling 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 gives by $p > 1$. Figure 11 shows drawings with $p = 2$ (MSE($\infty, 2$)) and $p = 3$ (MSE($\infty, 3$)). In our view they give more details.

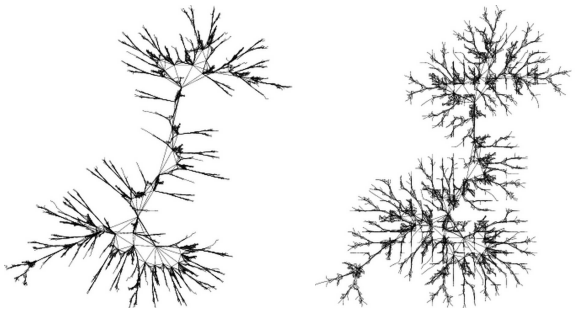

Figure 10: Drawings of add32 graph by MSE(∞) (left) and MSE(2) (right)

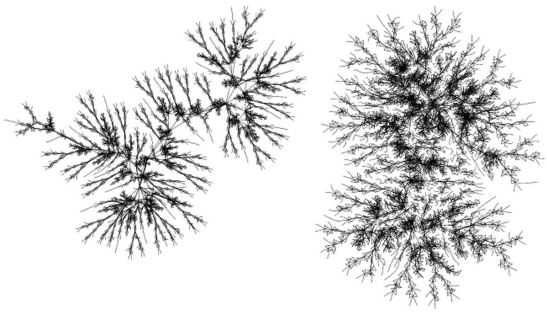

Figure 11: Drawings of add32 graph with a weaker repulsive force by MSE($\infty, 2$) (left) and MSE($\infty, 3$) (right)

Figure 12 gives the drawings of sierpinski10 graph. In [26] Walshaw has to layout the graph in 3D since he found 2D layout unsatisfactory. We found our 2D layout to be good, particular MSE(2). MSE(∞) demonstrates again the peripheral effect. The strong repulsive force pushes some of the vertices out. Figure 12 (bottom) 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 13 gives drawings of mesh100 graph. Compared with drawing in Figure 18(b) of [26], our drawings bear a closer resemblance to the actual mesh

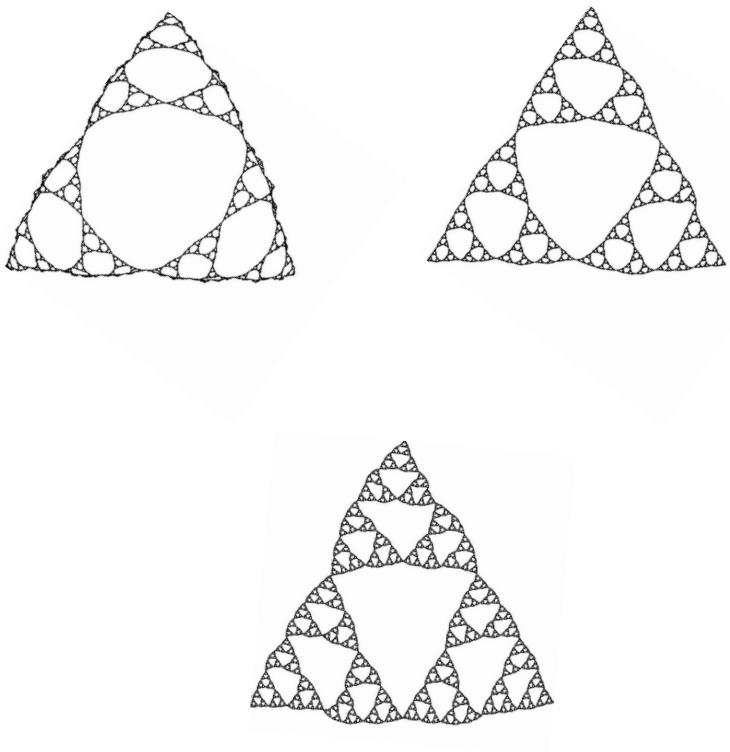

Figure 12: Drawings of sierpinski10 graph by MSE(∞) (top left), MSE(2) (top right) and MSE($\infty, 2$) (bottom)

given in Figure 18(a) of [26].

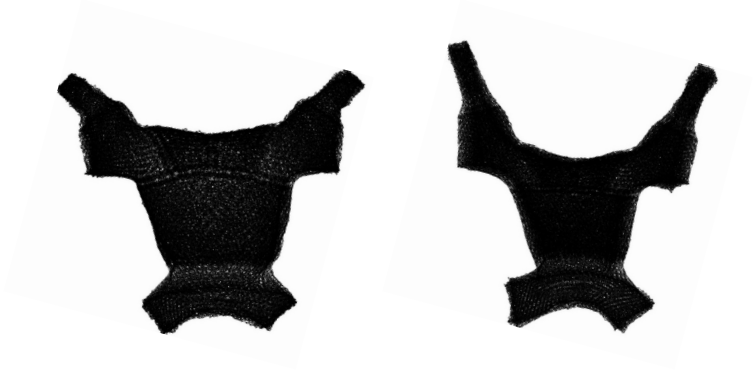

Figure 13: Drawings of mesh100 graph by MSE(∞) (left), MSE(2) (right) and MSE($\infty, 2$)

Overall, it is seen that our algorithms, give comparable drawings to those in [26]. 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 graph, by using a slightly weaker repulsive force. Compared with Figure 8 (left), it is seen that without a very strong repulsive force to push them outward, some of the "spikes" now point inward to the center of the circle. We found that further weakening of 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 graph by MSE($\infty, 1.3$)

6.3 Comparing the spring electrical model with the spring model

In addition to efficiency considerations that favors the spring electrical model, we also found that the spring electrical model, compared with the spring model, 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 reason that the spring model works well for such graphs is that for these graphs, 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 drawing of jagmesh1 graph 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) shown using MSE($\infty, 2$). 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 [26]. MS(4) also draws the data graph quite well, compared with MSE(∞), as shown in Figure 18.

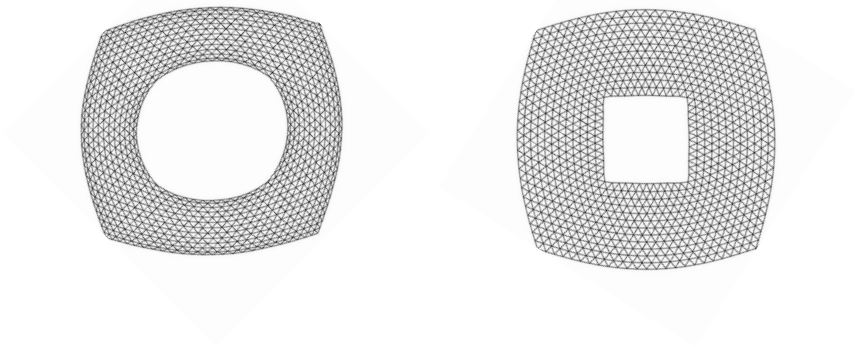

Figure 15: Alternative drawings of jagmesh1 graph of Figure 4 by MSE($\infty, 2$) (left), and MS(4) (right)

However, for graphs that come from a locally refined mesh, spring model works poorly. For example, Figure 19 shows the drawing of 4elt graph by MS(4) (right) and MSE(∞). It is clear that MS(4) strives to draw the graph as uniform a possible, but since this is not possible for such a highly refined graph, and in the absent of strong long range repulsive force, a lot of foldings occurred near the highly refined regions. Therefore, overall we favor multilevel spring electrical algorithm for its efficiency and general good quality of drawings.

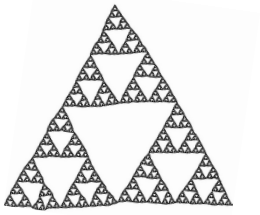

Figure 16: A drawing of sierpinski10 graph using MS(4)

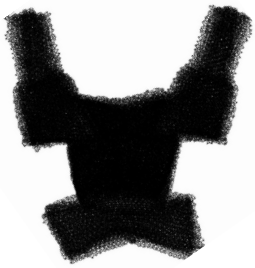

Figure 17: A drawing of mesh100 graph using MS(4)

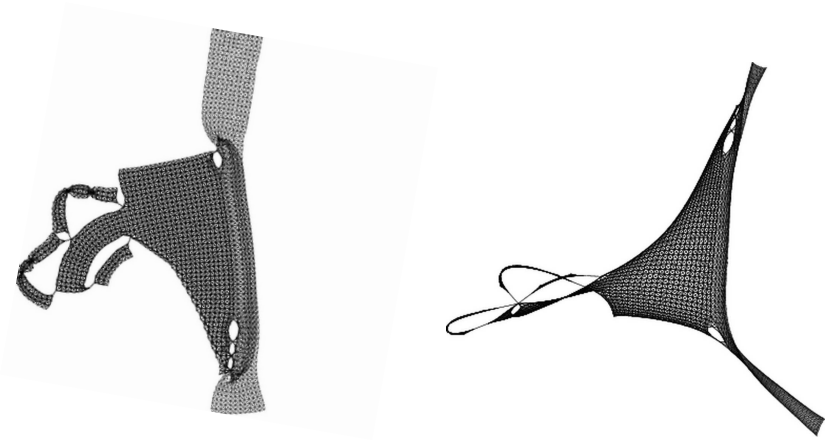

Figure 18: Drawings of data graph using MS(4) (left) and MSE(∞) (right)

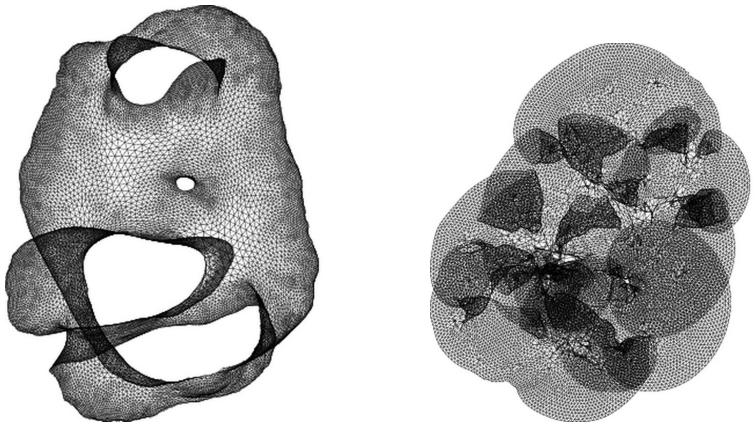

Figure 19: Drawings of 4elt graph by MSE(∞) (left), and MS(4) (right)

6.4 Further examples

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

Table 3: Problem description and CPU time (in seconds) for some graphs

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

[1]: skirt has 7 components, 4 of them non-trivial and have diameters 98, 68, 68 and 39, respectively.
[2]: pwt has 57 components, 56 out of these 57 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 7 components, 3 of these are just an isolated vertex. Of the four non-trivial components, one is the main tube and nozzle, another is the skirt/structs 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/structs. 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/structs is interesting. In the drawing, two struts praising out of Figure 21 (left) are drawn separated from three pieces of the skirt, revealing the weak linkage among the structs and the pieces of skirts.

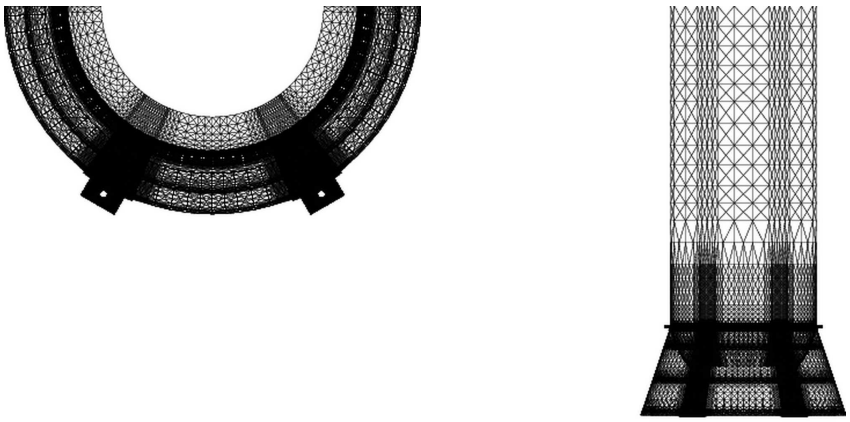

Figure 20: Original layout of skirt graph: two views

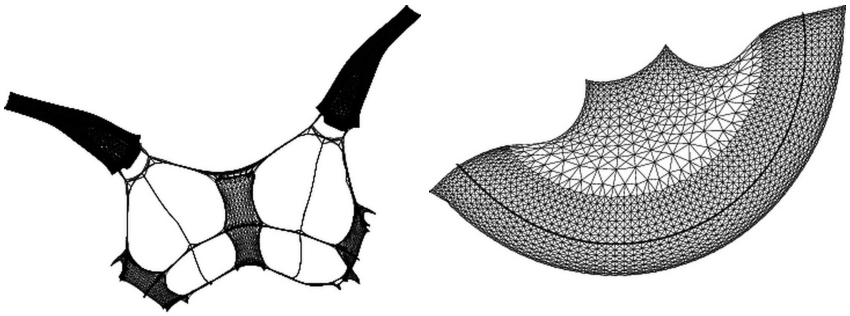

Figure 21: Drawings of two components of skirt graph by MSE(∞), Left: the skirt/structs; right: the tube/nozzle

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 uniform 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$(\infty, 2)$ or MSE(2), both having a weaker repulsive force, remove the folding (Figure 23). So it seems the folding is due to the strong repulsive force of the Fruchterman Reigold model, another example of the peripheral effect. The original mesh is nearly symmetric, all the drawings also exhibits good symmetry.

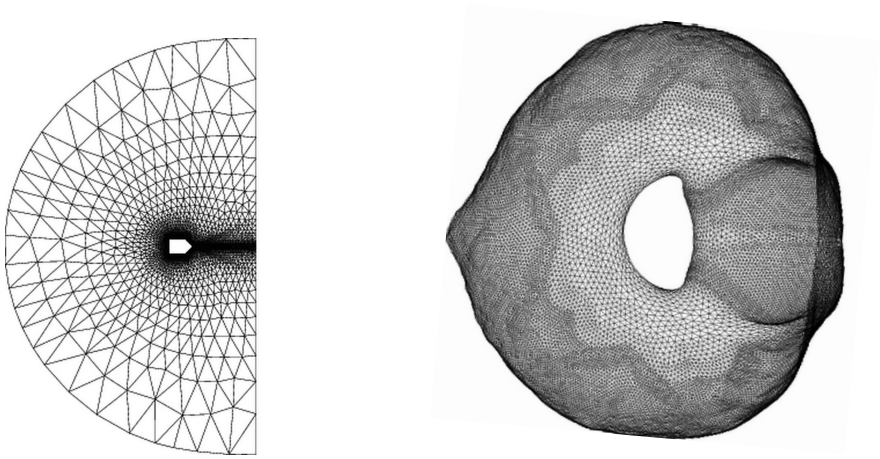

Figure 22: Original layout of bodyy6 graph (left), and drawing by MSE(∞) (right).

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, it is seen that the drawing depicted the details well.

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

7 Conclusions and Future Work

In this paper we proposed an algorithm that uses 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 [26, 23, 20], but as far as we are aware, were never combined together to form one powerful algorithm for large scale graph drawing. A number of practical techniques, including adaptive step

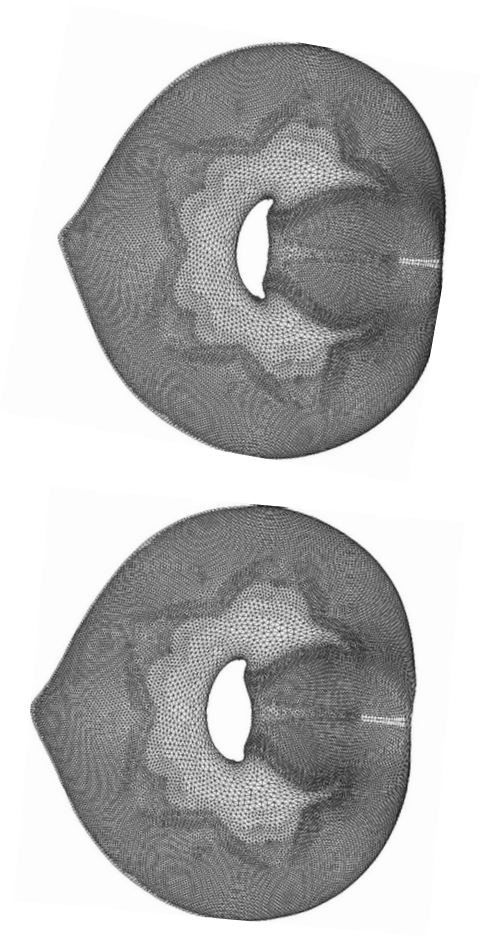

Figure 23: Drawings of bodyy6 graph using MSE($\infty, 2$) (top), and MSE(2) (bottom).

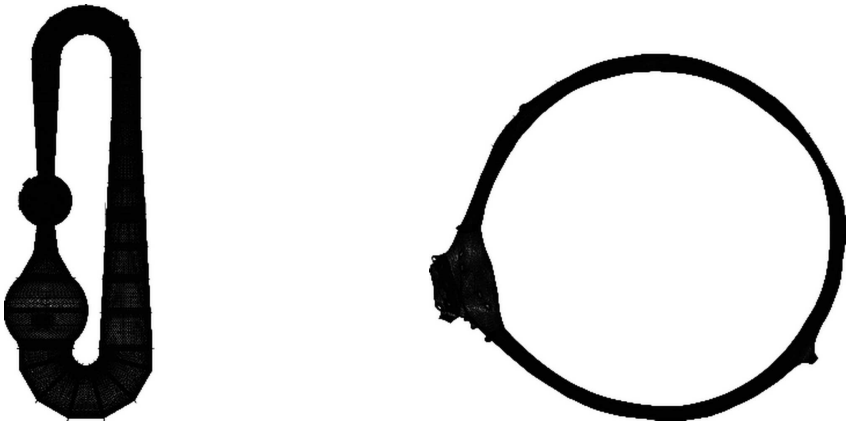

Figure 24: Original layout of pwt graph (left), and a drawing by MSE(∞) (right).

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

Figure 26: Drawings of pkustk01 graph (top), and pkustk02 graph (bottom) using MSE(∞) .

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 [26] highly successful graph drawing algorithm, yet gives better drawings on some difficult problems.

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

Both the multilevel approach and the octree data structure do have limitations. For example, both the edge-collapsing coarsening scheme, and the maximal independent vertex set 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 are connected to each other), because the former would coarsen too slowly thus having unacceptable complexity, while the latter would coarsen too fast and does not preserve the graph information on the coarser graphs. The parameter θ in Barnes and Hut octree algorithm is empirically fixed (to 1.2 in all our drawings in this paper). We have 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 these were shown in this paper.

It is intended that the graph drawing algorithm in this paper will be included in a future release of Mathematica.

8 Acknowledgments

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

References

[1] S. T. Barnard and H. D. Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6:101–117, 1994.

[2] J. Barnes and P. Hut. A hierarchical O(NlogN) force-calculation algorithm. *Nature*, 324:446–449, 1986.

[3] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Algorithms for the visualization of Graphs.* Prentice-Hall, 1999.

[4] R. Davison and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Trans. Graphics*, 15:301–331, 1996.

[5] P. Eades. A heuristic for graph drawing. *Congressus Nutnerantiunt*, 42:149–160, 1984.

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

[7] T. M. J. Fruchterman and E. M. Reigold. Graph drawing by force directed placement. *Software - Practice and Experience*, 21:1129–1164, 1991.

[8] A. Gupta, G. Karypis, and V. Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, 5:502–520, 1997.

[9] R. Hadany and D. Harel. A multi-scale algorithm for drawing graphs nicely. *Discrete Applied Mathematics*, 113:3–21, 2001.

[10] D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *J. Graph Algorithms and Applications*, 6:179–202, 2002.

[11] D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. *To appear in J. Graph Algorithms and Applications*, 2004.

[12] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories, Allbuquerque, NM, 1993. Also in Proceeding of Supercomputing'95 (`http://www.supercomp.org/sc95/proceedings/509_BHEN/SC95.HTM`).

[13] Y. F. Hu and J. A. Scott. A multilevel algorithm for wavefront reduction. *SIAM Journal on Scientific Computing*, 23:1352–1375, 2001.

[14] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.

[15] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[16] Y. Koren, L. Carmel, and D. Harel. Drawing huge graphs by algebraic multigrid optimization. *Multiscale Modeling and Simulation*, 1:645–673, 2003.

[17] M. T. Goodrich P. Gajer and S. G. Kobourov. A fast multi-dimensional algorithms for drawing large graphs. *Lecture Notes In Computer Science*, 1984:211 – 221, 2000.

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

[19] K. J. Pulo. Recursive space decompositions in force-directed graph drawing algorithms. In P. Eades and T. Pattison, editors, *Australian symposium on Information visualisation, December 01, 2001, Sydney, Australia. Conferences in Research and Practice in Information Technology Series.*, volume 9, pages 95–102. Australian Computer Society Inc., 2001.

[20] A. Quigley. *Large scale relational information visualization, clustering, and abstraction.* PhD thesis, Department of Computer Science and Software Engineering, University of Newcastle, Australia, 2001.

[21] A. Quigley and P. Eades. Fade: Graph drawing, clustering, and visual abstraction. *Lecture Notes in Computer Science*, 1984:183–196, 2000.

[22] N. Quinn and M. Breur. A force directed component placement procedure for printed circuit boards. *IEEE Trans. on Circuits and Systems*, CAS-26(6):377–388, 1979.

[23] D. Tunkelang. *A Numerical Optimization Approach to General Graph Drawing.* PhD thesis, Carnegie Mellow University, 1999.

[24] C. Walshaw. Multilevel refinement for combinatorial optimisation problems. To appear in Annals Oper. Res.; originally published as Univ. Greenwich Tech. Rep. 01/IM/73, 2001.

[25] C. Walshaw. A multilevel approach to the travelling salesman problem. *Oper. Res.*, 50:862–877, 2002.

[26] C. Walshaw. A multilevel algorithm for force-directed graph drawing. *J. Graph Algorithms Appl.*, 7:253–285, 2003.

[27] C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 47:102–108, 1997.