Drawing Large Graphs with a Potential-Field-Based Multilevel Algorithm (Extended Abstract)

Stefan Hachul and Michael Jünger

Universität zu Köln, Institut für Informatik,
Pohligstraße 1, 50969 Köln, Germany
{hachul,mjuenger}@informatik.uni-koeln.de

Abstract. Force-directed graph drawing algorithms are widely used for drawing general graphs. However, these methods do not guarantee a sub-quadratic running time in general. We present a new force-directed method that is based on a combination of an efficient multilevel scheme and a strategy for approximating the repulsive forces in the system by rapidly evaluating potential fields. Given a graph $G = (V, E)$, the asymptotic worst case running time of this method is $O(|V| \log |V| + |E|)$ with linear memory requirements. In practice, the algorithm generates nice drawings of graphs containing 100000 nodes in less than 5 minutes. Furthermore, it clearly visualizes even the structures of those graphs that turned out to be challenging for some other methods.

1 Introduction

Given a graph $G = (V, E)$, force-directed graph drawing methods generate drawings of G in the plane in which each edge is represented by a straight line connecting its two adjacent nodes. The computation of the drawings is based on associating G with a physical model. Then, an iterative algorithm tries to find a placement of the nodes so that the total energy of the physical system is minimal. Such algorithms are quite popular, since they are easy to implement and often generate nice drawings of general graphs. In practice, classical force-directed algorithms like [5, 12, 6, 4] are not well suited for drawing large graphs containing many thousands of vertices, since their worst case running time is at least quadratic. Significantly accelerated force-directed algorithms have been developed by [15, 14, 7, 9, 17]. These algorithms generate nice drawings of a big range of large graphs in reasonable time. Some of these methods guarantee a sub-quadratic running time in special cases or under certain assumptions but not in general. Others are not sub-quadratic in any case. Besides force-directed algorithms other very fast methods for drawing large graphs have been invented by Harel and Koren [10] and Koren et al. [13] that do not use a physical force model.

In Section 2 we sketch the most important parts of a new force-directed graph drawing algorithm that guarantees a sub-quadratic worst case running time. An excerpt of the experimental results is given in Section 3. For space restrictions, we can neither describe every basic component of this algorithm in detail, nor compare our method with the existing ones in a satisfactory way. This will be presented in the full version of this paper.

2 The Fast Multipole Multilevel Method ($\boldsymbol{FM^3}$)

We describe the most important parts of the new method that is a combination of an efficient multilevel technique with an $O(|V| \log |V|)$ approximation algorithm to obtain the repulsive forces between all pairs of nodes/particles. Other important parts like a preprocessing step that enables the algorithm to draw graphs with nodes of different sizes and a part that is designed to handle disconnected graphs are not described here. Therefore, we simply assume that the given graph G is a connected weighted graph. The edge weight of each edge represents its individual desired edge length.

2.1 The Force Model

First, we must choose a force model. This is done by identifying the nodes with charged particles that repel each other and by identifying edges with springs, like in most classical force-directed methods. If in $\mathbb{R}^2$ two charges u, v are placed at a distance d from each other, the repulsive forces between u and v are proportional to $1/d$. Our choice of the spring forces is not strictly related to physical reality. We found that choosing the spring force of an edge e to be proportional to $\log(d/\mathit{desired\_edge\_length}(e)) \cdot d^2$ gives very good results in practice.

2.2 The Multilevel Strategy

Since in classical force-directed algorithms many iterations are needed to transform an initial (random) drawing of a large graph into the final drawing, one might hope to reduce the constant factor of force-directed algorithms by using a multilevel strategy. Multilevel strategies have been introduced into force-directed graph drawing by [7, 9, 17] and share the following basic ideas: Given $G = (V, E) =: G_0$ they create a series of graphs $G_1, \ldots, G_k$ with decreasing sizes. Then, the smallest graph G_k at *level k* is drawn using (a variation of) a classical force-directed *(single-level)* algorithm. This drawing is used to get an initial layout of the next larger graph G_{k-1} that is drawn afterwards. This process is repeated until the original graph G_0 is drawn.

Unlike previous approaches, we want to design a multilevel algorithm that has provably the same asymptotic running time as the single-level algorithm that is used to draw all graphs G_i with $i = 0, \ldots, k$.

The idea of our multilevel step is as follows: First, we partition the node set of G into disjoint subsets. The induced connected subgraphs are called *solar systems*. A solar system S consists of one central *sun* node (s-node). Each of its neighbors is called *planet* node (p-node) and is also contained in S. The rest of the nodes in a solar system are called *moon* nodes (m-nodes), and each m-node is required to have graph-theoretic distance 2 to its associated s-node in G. Each m-node is assigned to its nearest neighboring p-node in S. This p-node is relabeled *planet with moon* node (*pm*-node), indicating that at least one m-node is assigned to it. Thus, the subgraph of G that is induced by a solar system has diameter at most 4.

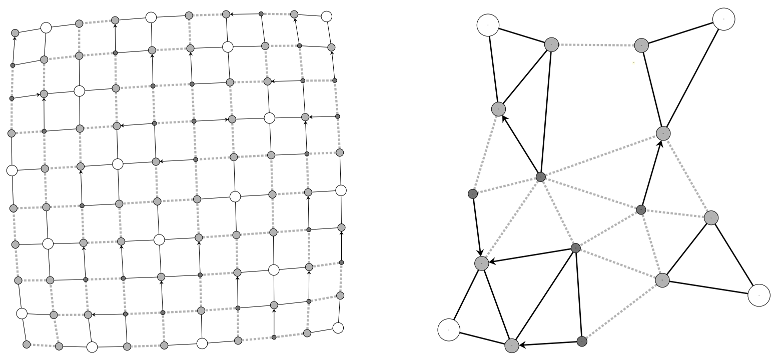

Fig. 1. (left) Drawing of $G = G_0$. (right) Drawing of G_1.

Figure 1(left) shows an example of a grid graph that is partitioned into 17 solar systems. The sun, planet, and moon nodes are represented by the white big, grey medium, and black small circles, respectively. The solid edges represent *intra* solar system edges, whereas the edges connecting nodes of two different solar systems (*inter* solar system edges) are dashed edges. The edges that connect an m-node and its assigned *pm*-node are drawn as directed edges, indicating that the m-node is assigned to this planet node.

We sketch a linear time method for constructing a solar system partitioning of a graph G that works in three steps: First, we create the sun nodes. We store a candidate set V' that is a copy of V and randomly select a first sun node s_1 from V'. Then, s_1 and all nodes that have a graph-theoretic distance at most 2 from s_1 in G are deleted from V'. We iteratively select the next sun nodes in the same way, until V' is empty and $Suns = s_1, \ldots, s_l$ is the list of all sun nodes. Second, for each $s_i \in Suns$ all its neighbors are labeled as planet nodes. Finally, there might be some nodes in V that are neither labeled as planet nodes, nor as sun nodes. These nodes are the moon nodes, and we assign each moon node to the planet node that is its nearest neighbor in G.

Given a solar system partition of the node set of $G = G_0$, we construct a smaller graph G_1 by collapsing (shrinking) the node set of each solar system into one single node and deleting parallel edges (see Figure 1(right)). The smaller graph should reflect the attributes of the bigger graph as much as possible. Therefore, we initialize the desired edge length of an edge $e_1 = (u_1, v_1)$ in G_1 as follows: Suppose that p-node u_0 belongs to the solar system S_0 with sun node s_0 in G_0 and p-node v_0 belongs to the solar system T_0 with sun node t_0 in G_0. Let us also assume that the edge $e_0 = (u_0, v_0)$ is the unique inter solar system edge connecting S_0 and T_0. Furthermore, we assume that nodes u_1 and v_1 in G_1 are obtained by collapsing S_0 and T_0. Then, we set the desired edge length of e_1 to $\textit{desired\_edge\_length}((s_0, u_0)) + \textit{desired\_edge\_length}(e_0) + \textit{desired\_edge\_length}((v_0, t_0))$. For later use, we denote the corresponding path P_0 and its length p_0. If more than one inter solar system edge in G_0 connects nodes of S_0 with nodes of T_0, we just take the average of the previously calculated desired edge lengths. The case that u_0 and/or v_0 is a moon node is treated similarly.

This partitioning and collapsing process is iterated until the smallest graph G_k contains only a constant number of nodes. Then, this graph is drawn by an algorithm that is introduced later.

Going upwards to G_{k-1}, we assign initial positions to the nodes of G_{k-1} in two steps: First, we place each sun node s of G_{k-1} at the position of its ancestor (that represents its solar system) in the drawing of G_k. Now, we place the other nodes of G_{k-1}. This is done by using information that has been generated during the collapsing process: Given u_0, v_0, s_0, t_0, p_0, and P_0 like in the example above, we place u_0 on the line connecting s_0 and t_0 at the position $\mathrm{Pos}(s_0) + \frac{\textit{desire\_edge\_length}((s_0,u_0))}{p_0}(\mathrm{Pos}(t_0) - \mathrm{Pos}(s_0))$. If u_0 belongs to more than one such path P_0, we take the barycenter of all these positions. The case that u_0 is a moon node is treated similarly.

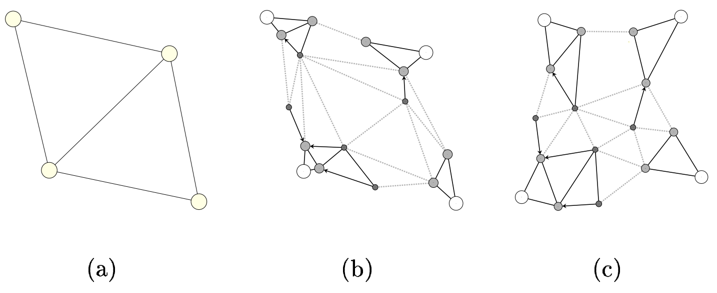

Fig. 2. (a) Drawing of G_2. (b) Initial placement of G_1. (c) Drawing of G_1.

Figure 2 demonstrates this procedure. Figure 2(a) is a drawing of the multilevel graph G_2 of Figure 1(left). Figure 2(b) is the initial position of the drawing of G_1 that is obtained from the drawing of G_2. Figure 2(c) shows G_1 that is drawn with a new force-directed single-level algorithm.

The total running time of the multilevel strategy is $t_{\mathrm{mult}}(|V|, |E|) = \sum_{i=0}^{k-1} t_{\mathrm{create}}(|V_i|, |E_i|) + \sum_{i=0}^{k-1} t_{\mathrm{init\_pl}}(|V_i|, |E_i|) + \sum_{i=0}^{k} t_{\mathrm{single}}(|V_i|, |E_i|)$. Here, t_{create}

$(|V_i|, |E_i|)$ denotes the time that is needed to create the multilevel graph G_{i+1} from G_i. $t_{\text{init\_pl}}(|V_i|, |E_i|))$ denotes the time that is needed to get an initial placement of the nodes of the multilevel graph G_i from the drawing of G_{i+1}. $t_{\text{single}}(|V_i|, |E_i|)$ is the time that the chosen single-level algorithm needs to draw G_i.

Since every node of G_i belongs to a solar system, and a solar system contains at least two nodes, G_{i+1} contains at most $|V_i|/2$ nodes. Therefore, $k \leq \log |V|$.

Let us assume that $|E_{i+1}| \leq |E_i|/2$ for all $i = 0, \ldots, k-1$. Since both $t_{create}(|V_i|, |E_i|)$ and $t_{\text{init\_pl}}((|V_i|, |E_i|)$ are linear in $|V_i| + |E_i|$ we get $\sum_{i=0}^{k-1} t_{\text{create}}$ $(|V_i|, |E_i|) + \sum_{i=0}^{k-1} t_{\text{init\_pl}}(|V_i|, |E_i|) = O(|V| + |E|)$. Furthermore, we get the following estimation on t_{single}:

$\sum_{i=0}^{k} t_{\text{single}}(|V_i|, |E_i|) \leq \sum_{i=0}^{k} t_{\text{single}}\left(\frac{|V|}{2^i}, \frac{|E|}{2^i}\right) \leq t_{\text{single}}(|V|, |E|) \sum_{i=0}^{k} \frac{1}{2^i} \leq$ $2\ t_{\text{single}}(|V|, |E|)$. The second inequality is true for sufficiently large values of $|V|$ and $|E|$, since $t_{\text{single}}(|V|, |E|) = \Omega(|V| + |E|)$. Therefore, $t_{\text{mult}}(|V|, |E|)$ and $t_{\text{single}}(|V|, |E|)$ have the same asymptotic running time.

Certainly, it cannot be guaranteed that the number of edges decreases by a factor $\frac{1}{2}$ as well. This might result in an additional factor $k = \log |V|$ on the parts of the algorithm that touch edges. However, it can be shown by an analogous argumentation that $t_{\text{mult}}(|V|, |E|)$ and $t_{\text{single}}(|V|, |E|)$ have the same asymptotic running time if $|E_{i+1}| \leq |E_i|/d$ for all $i = 0, \ldots, k-1$ and a fixed divisor $d > 1$. Therefore, it is sufficient to stop the multilevel process, whenever the algorithm has generated more than a constant number of graphs G_i that do not satisfy the inequality $|E_{i+1}| \leq |E_i|/d$ for some small $1 < d \leq 2$.

2.3 The Force Calculation Step

In order to save running time, the multilevel algorithms [7, 9, 17] use the grid-variant method of [6] or variations of [12] that are comparatively inaccurate approximative variations of the original single-level algorithms [6, 12].

Unlike this, the single-level algorithm that is used in *FM*^{3} follows the basic strategy of [15, 14] by approximating the repulsive forces between all pairs of distinct nodes/particles with high accuracy and calculating the forces induced by the edges/springs exactly. Then, these forces are added, and the nodes are moved in the direction of the resulting forces. This process is repeated a constant number of iterations. (In practice, we let the constant decrease from 300 iterations for G_k to 30 iterations for G_0, although convergence is reached even faster for many tested graphs.)

In the following, we concentrate on the calculation of the repulsive forces. Greengard [8] has invented an N-body simulation method that is based on the evaluation of the field of the potential energy of $N := |V|$ particles. This is done by evaluating multipole expansions using a hierarchical data structure called quadtree. However, Aluru et al. [1] have shown that the running time of his method depends on the particle distribution and cannot be bounded in the number of particles. They also have proven that the running time of the popular center of mass approximation method of Barnes and Hut [3] that is used in the graph drawing methods [15, 14] cannot be bounded in the number of particles. Based on the techniques and analytical tools of Greengard [8], Aluru et al. [1] have presented an $O(N \log N)$ approximative multipole algorithm that is distribution independent.

Based on the work of Greengard [8] and Aluru et al. [1], we have developed a new $O(N \log N)$ multipole method that — in practice — is faster than Aluru et al.'s [1] method. It works in two steps. Given a distribution of N particles in the plane, first a special quadtree data structure is constructed. Then, each node of the quadtree is assigned information that is used for approximating the potential energy of the system. In particular, a constant number of coefficients of a so called *multipole expansion* (to be introduced later) are associated with each tree node and are used to obtain the repulsive forces.

Construction of the reduced quadtree

Suppose that N particles are distributed on a square D and we fix a *leaf capacity* $c \geq 1$. (In practice we choose $c = 25$.) Furthermore, suppose one recursively subdivides D into four squares of equal size, until each square contains at most c particles. This process can be represented by an ordered rooted tree of maximum child degree four (with the root representing D) that is called *quadtree.* The particles are stored in the leaves of the quadtree. A *degenerate path* $P = (v_1, \ldots, v_p)$ in a quadtree is a path in which v_1 and v_p have at least 2 nonempty children and $v_2, \ldots, v_{p-1}$ each have exactly one nonempty child. A *reduced quadtree* T can be obtained from a quadtree by shrinking degenerate paths $P = (v_1, \ldots, v_p)$ to edges (v_1, v_p). Figure 3 shows an example.

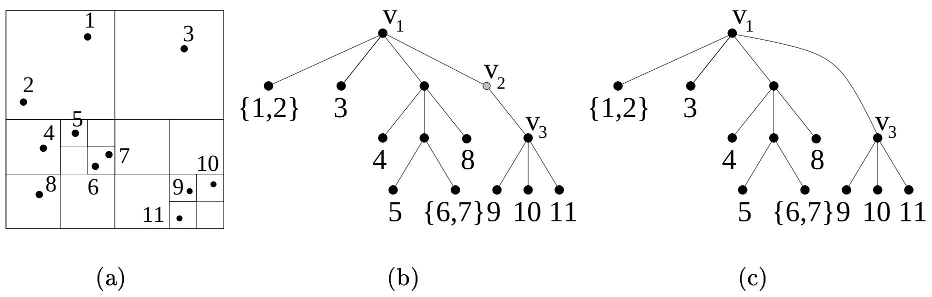

Fig. 3. (a) A distribution of $N = 11$ particles in the plane. (b) The quadtree with leaf capacity $c = 2$ associated with (a). $P = (v_1, v_2, v_3)$ is a degenerate path in the quadtree. (c) The reduced quadtree with leaf capacity $c = 2$ associated with (a).

A reduced quadtree has only $O(N)$ nodes independently of the distribution of the particles. This allows the development of a linear time method (excluding the time needed for constructing the reduced quadtree) for approximating the repulsive forces, using this structure.

Aluru et al. [1] present an $O(N \log N)$ method that constructs a reduced quadtree with $c = 1$. As can be shown by a reduction from sorting, it is neither possible to construct a quadtree nor a reduced quadtree for arbitrary distributions of the particles in $o(N \log N)$ time.

We have developed a new $O(N \log N)$ method that is omitted here for brevity, since it quite technical. Instead, we will explain another new tree construction method that is conceptionally simpler and in practice faster. But (motivated by the assumptions in [15]) it restricts the possible particle distributions: We force the particles to be placed on a large square grid with a resolution that is polynomial in N. This can be realized by rounding the x, y coordinates of each particle to integers in the range $[0, \mathcal{P}(N)]$, where $\mathcal{P}(N)$ is any whole-numbered polynomial in N of maximum degree l, and by treating pathological cases in which particles have same coordinates efficiently. In practice, it is sufficient to set $\mathcal{P}(N) = d \cdot N^2$, with a big constant d, say 1000. This bounds the depth of the reduced quadtree to $O(\log(\mathcal{P}(N)) = O(l \cdot \log N) = O(\log N)$.

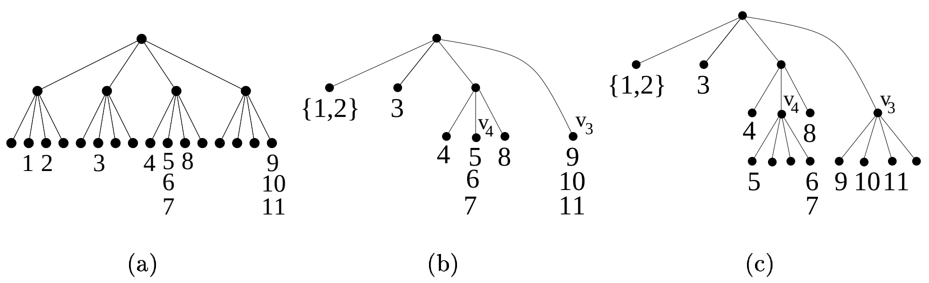

Fig. 4. Building up the reduced quadtree T with leaf capacity $c = 2$ and $N = 11$ particles for the distribution of Figure 3(a). (a) First step: Building up the complete subtree T^1. (b) Second step: Thinning out T^1. (c) Recursion: Building up the complete quadtrees $T^2(v_3)$ and $T^2(v_4)$.

First, we build up a complete truncated subtree T^1 with depth $\max\{1, \lfloor \log N / c \rfloor\}$. Then, all particles are assigned to the leaves of T^1. Since T^1 contains $O(N)$ nodes and its structure is predefined, this step can be performed in linear time. Figure 4(a) shows an example of T^1 that corresponds to the distribution of Figure 3(a). In the next step, we thin out T^1. This is done by traversing the tree bottom up and thereby calculating for each internal tree node the number of particles that are contained in the square region that it represents. This also needs time linear in N. Then, we traverse the subtree T^1 top down, delete all nodes that do not contain particles and shrink degenerate path to edges. If (during this process) we visit an internal node v that is the root of a subtree containing at most c particles, this subtree is deleted, and all the particles that were stored in the deleted subtree are assigned to v. Figure 4(b) shows the thinned out subtree T^1.

If none of the leaves of T^1 contains more than c particles, the procedure ends and $T^1 = T$ has been constructed in linear time. Otherwise, we repeat the previous steps recursively. For example, the nodes v_3 and v_4 in Figure 4(b) both contain $3 > c$ particles. Therefore, we build up complete subtrees $T^2(v_3)$ rooted at v_3 and $T^2(v_4)$ rooted at v_4. Both subtrees have depth $\max\{1, \lfloor \log 3/c \rfloor\} = 1$. Now, the particles $5, 6, 7$ are assigned to the leaves of $T^2(v_4)$ and the particles $9, 10, 11$ are assigned to the leaves of $T^2(v_3)$ (see Figure 4(c)). After thinning out $T^2(v_3)$ and $T^2(v_4)$ the desired tree T (see Figure 3(c)) is created.

What is the total running time of this approach? Building up T^1 needs $O(N)$ time. If T^1 is not the reduced quadtree, we build up subtrees $T^2(v_1), \ldots, T^2(v_k)$ for all leaves $v_1, \ldots, v_k$ of T^1 that contain more than c particles. This needs $O(N)$ time in total, since the sum of the tree nodes contained in all $T^2(v_i)$ is at most $O(N)$. Then, we possibly have to build up subtrees rooted at the leaves of the T^2 trees and so forth. Since for each $j \geq 1$ the sum of the tree nodes of all T^j is bounded above by $O(N)$, the total running time is $O(|\text{recursion\_levels}| \cdot N)$. Therefore, the running time is bounded by $O(N \log N)$. The construction of the tree needs linear time whenever $|\text{recursion\_levels}|$ is a constant.

Evaluating Multipole Expansions

Unlike the construction of the tree, the calculation of the forces — using the tree data structure — is quite complex. Therefore, we only sketch the basic ideas. The most essential part is the following theorem of Greengard [8]. First, we identify each point $p = (x, y) \in \mathbb{R}^2$ with a point $z = x + iy \in \mathbb{C}$.

Theorem 1 (Multipole Expansion) *Suppose that m charges of strengths q_i, $\{i = 1, \ldots, m\}$ are located within a circle of radius r around the center z_0. Then, for any $z \in \mathbb{C}$ with $|z - z_0| > r$, the potential Energy $\mathcal{E}(z)$ induced by the m charged particles is given by:*
$\mathcal{E}(z) = Q \log(z - z_0) + \sum_{k=1}^{\infty} \frac{a_k}{(z - z_0)^k}$ *with* $Q = \sum_{i=1}^{m} q_i$ *and* $a_k = \sum_{i=1}^{m} \frac{-q_i (z_i - z_0)^k}{k}$

The corresponding force is $\mathcal{F}(z) = (\mathrm{Re}(\mathcal{E}'(z)), -\mathrm{Im}(\mathcal{E}'(z)))$. Based on this theorem, the idea is to develop the infinite series only up to a constant index p. In practice, choosing $p = 4$ has turned out to be sufficient to keep the error of the approximation less than 10^{-2}. The resulting truncated Laurent series is called *p-term multipole expansion.* Estimations of the error and several other fundamental theorems for working with such series can be found in [8].

We demonstrate the use of this theorem for speeding up force-calculation algorithms on an example: Suppose that m particles are located within a circle C_0 of radius r with center z_0. Suppose that another m particles are located within a circle C_1 of radius r with center z_1, and let $|z_0 - z_1| > 3r$ (see Figure 5).

Computing the repulsive forces acting on each particle in C_0 due to all particles in C_1 naively would need $\Theta(m^2)$ time. Now, suppose that we first compute the coefficients of a p-term multipole expansion of the potential due to the particles in C_1. This needs $\Theta(pm)$ time. Evaluating the resulting p-term multipole expansion for all particles within C_0 needs also $\Theta(pm)$ time. Therefore, we obtain an accurate approximation of the potential energy of all particles placed in

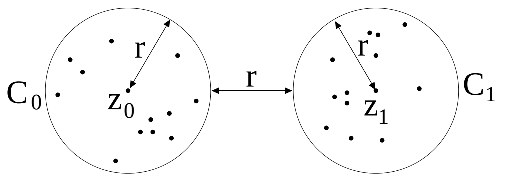

Fig. 5. An example distribution, showing the use of the Multipole Expansion Theorem.

C_0 due to the particles placed in C_1 in $\Theta(m)$ time. Since we are interested in the forces rather than the energy, we first calculate the derivative of the p-term multipole expansion before evaluating it for each particle in C_0. Since the multipole expansion is a simple Laurent series, the calculation of the derivative of the p-term multipole expansion needs only $O(p)$ additional time.

Now we sketch the idea how this theorem is used for calculating the forces: First, the p-term multipole expansions of the particles in the leaves of the reduced quadtree are calculated. Then, the reduced quadtree is traversed bottom up, and thereby p-term multipole expansions of the interior nodes are obtained. Afterwards, the tree is traversed top down, and suitable coefficients of p-term multipole expansions are used to calculate coefficients of special power series that are called *p-term local expansions*. Finally, these expansions are evaluated to obtain the repulsive forces. All these operations together take time linear in the number of particles.

To get a better impression how this algorithm really works, we refer the interested reader to [8, 1]. Our method for evaluating the multipole expansions is an extension of the method of Aluru [1] et al. for the general case in which $c \geq 1$.

It is important to note that our multipole method remains $O(N \log N)$ — even if we allow arbitrary particle positions during the computation — if we use our other tree construction method or the tree construction method of Aluru et al. [1].

3 Remarks on the Experimental Results

The method *FM*^{3} has been implemented in C++ within the framework of *AGD* [11]. We tested our method on a 2.8 GHz PC running Linux. The tested graphs are the graphs contained in the graph partitioning archive of C. Walshaw [16] with up to 200000 nodes and the biggest graphs from the AT&T graph collection [2]. Furthermore, we generated artificial graphs containing up to 100000 nodes. For example, these graphs include grid graphs, sierpinski graphs, random disconnected graphs, graphs that contain many biconnected components, graphs with a very high edge density, and graphs that contain nodes with a very high degree. The tested graphs containing less than 1000, 10000, and 100000 nodes have been drawn in less than 2, 24, and 263 seconds, respectively. Figure 6 shows example drawings that are generated by *FM*^{3}. Our practical experiments indicate that the combination of our multilevel strategy with a highly accurate force approximation algorithm increases the quality of the generated drawings.

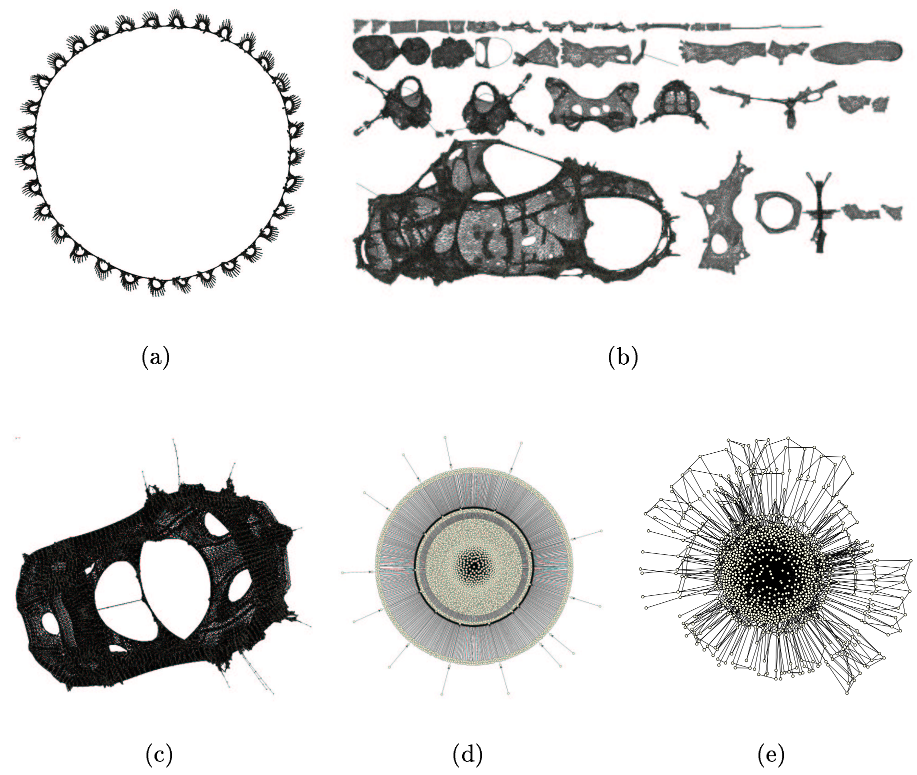

Fig. 6. (a) *finan512*: $|V| = 74752$, $|E| = 261120$, CPU-time = 158.2 seconds. (b) *fe_body*: $|V| = 44775$, $|E| = 163734$, CPU-time = 96.5 seconds. (c) *bcsstk31*: $|V| = 35588$, $|E| = 572914$, edge density = 16.1, CPU-time = 83.6 seconds. (d) *dg_1087*: $|V| = 7602$, $|E| = 7601$, maximum degree = 6566, CPU-time = 18.1 seconds. (e) *ug_380*: $|V| = 1104$, $|E| = 3231$, maximum degree = 856, CPU-time = 2.1 seconds.

4 Conclusions and Future Work

We have developed a new force-directed graph drawing algorithm (FM^3) that runs in $O(|V| \log |V| + |E|)$ time. The practical experiments demonstrate that FM^3 is very fast and creates nice drawings of even those graphs that turned out to be challenging for some other tested algorithms. This will be presented in the full version of this paper.

Bibliography

[1] S. Aluru et al. Distribution-Independent Hierarchical Algorithms for the N-body Problem. *Journal of Supercomputing*, 12:303–323, 1998.

[2] The AT&T graph collection: `www.graphdrawing.org`.

[3] J. Barnes and P. Hut. A hierarchical $\mathcal{O}(N \log N)$ force-calculation algorithm. *Nature*, 324(4):446–449, 1986.

[4] R. Davidson and D. Harel. Drawing Graphs Nicely Using Simulated Annealing. *ACM Transaction on Graphics*, 15(4):301–331, 1996.

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

[6] T. M. J. Fruchterman and E. M. Reingold. Graph Drawing by Force-directed Placement. *Software–Practice and Experience*, 21(11):1129–1164, 1991.

[7] P. Gajer et al. A Multi-dimensional Approach to Force-Directed Layouts of Large Graphs. In J. Marks, editor, *Graph Drawing 2000*, volume 1984 of *Lecture Notes in Computer Science*, pages 211–221. Springer-Verlag, 2001.

[8] L. F. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems*. ACM distinguished dissertations. The MIT Press, Cambridge, Massachusetts, 1988.

[9] D. Harel and Y. Koren. A Fast Multi-scale Method for Drawing Large Graphs. In J. Marks, editor, *Graph Drawing 2000*, volume 1984 of *Lecture Notes in Computer Science*, pages 183–196. Springer-Verlag, 2001.

[10] D. Harel and Y. Koren. Graph Drawing by High-Dimensional Embedding. In M. T. Goodrich and S. G. Kobourov, editors, *Graph Drawing 2002*, volume 2528 of *Lecture Notes in Computer Science*, pages 207–219. Springer-Verlag, 2002.

[11] M. Jünger et al. *Graph Drawing Software*, volume XII of *Mathematics and Visualization*, chapter AGD - A Library of Algorithms for Graph Drawing, pages 149–169. Springer-Verlag, 2004.

[12] T. Kamada and S. Kawai. An Algorithm for Drawing General Undirected Graphs. *Information Processing Letters*, 31:7–15, 1989.

[13] Y. Koren et al. Drawing Huge Graphs by Algebraic Multigrid Optimization. *Multiscale Modeling and Simulation*, 1(4):645–673, 2003.

[14] A. Quigley and P. Eades. FADE: Graph Drawing, Clustering, and Visual Abstraction. In J. Marks, editor, *Graph Drawing 2000*, volume 1984 of *Lecture Notes in Computer Science*, pages 197–210. Springer-Verlag, 2001.

[15] D. Tunkelang. JIGGLE: Java Interactive Graph Layout Environment. In S. H. Whitesides, editor, *Graph Drawing 1998*, volume 1547 of *Lecture Notes in Computer Science*, pages 413–422. Springer-Verlag, 1998.

[16] C. Walshaw's graph collection: `www.gre.ac.uk/~c.walshaw/partition`.

[17] C. Walshaw. A Multilevel Algorithm for Force-Directed Graph Drawing. In J. Marks, editor, *Graph Drawing 2000*, volume 1984 of *Lecture Notes in Computer Science*, pages 171–182. Springer-Verlag, 2001.