A Fast Multi-Scale Method for Drawing Large Graphs

David Harel *Yehuda Koren*
Dept. of Computer Science and Applied Mathematics
The Weizmann Institute of Science, Rehovot, Israel
http://www.wisdom.weizmann.ac.il/
harel@wisdom.weizmann.ac.il yehuda@wisdom.weizmann.ac.il

Abstract

We present a multi-scale layout algorithm for the aesthetic drawing of undirected graphs with straight-line edges. The algorithm is extremely fast, and is capable of drawing graphs that are substantially larger than those we have encountered in prior work. For example, the paper contains a drawing of a graph with over 15,000 vertices. Also we achieve "nice" drawings of 1000 vertex graphs in about 1 second. The proposed algorithm embodies a new multi-scale scheme for drawing graphs, which was motivated by the earlier multi-scale algorithm of Hadany and Harel [HH99]. In principle, it could significantly improve the speed of essentially any force-directed method (regardless of that method's ability of drawing weighted graphs or the continuity of its cost-function).

1 Introduction

A graph $G(V, E)$ is an abstract structure that is used to model a relation E over a set V of entities. Graph drawing is a conventional tool for the visualization of relational information, and its usefulness depends on its readability, that is, the capability of conveying the meaning of the diagram quickly and clearly. In recent years, many algorithms for drawing graphs automatically were proposed (the state of the art is surveyed comprehensively in [Di+99, KW01]).

We concentrate on the problem of drawing an undirected graph with straight-line edges. In this case the problem reduces to that of positioning the vertices by determining a mapping $L : V \longrightarrow \mathbb{R}^2$. A popular generic approach to this problem is the *force-directed* technique, which introduces a heuristic cost function (an *energy*) of the mapping L, which (hopefully) achieves its minimum when the layout is nice. Variants of this approach differ in the definition of the energy, and in the optimization method that finds its minimum. Some known algorithms are those of [Ea84, KK89, DH89, FR91]. Major advantages of force-directed methods are their relatively simple implementation and their flexibility (heuristic improvements are easily added), but there are some problems with them too. One severe problem is the difficulty of minimizing the energy function when dealing with large graphs. The above methods focus on graphs of up to 100 vertices. For larger graphs the convergence to a minimum, if possible at all, is very slow.

We propose a new method for drawing graphs that could in principle improve the speed of every force-directed method. We build our algorithm around the

Kamada-Kawai method, and the resulting algorithm, which is extremely fast, is capable of drawing graphs that are substantially larger than those we have encountered in prior work (although subsequent developments are reviewed in Section 6). The algorithm, which was motivated by the earlier multi-scale algorithm of Hadany and Harel [HH99], works by producing a sequence of improved approximations of the final layout. Each approximation allows vertices to deviate from their final place by an extent limited by a decreasing constant r. As a result, the layout can be computed using increasingly coarse representations of the graph, where closely drawn vertices are collapsed into a single vertex. Each layout in the sequence is generated very rapidly, by performing a local beautification on the previously generated layout.

2 Force-Directed Graph Drawing

The force-directed approach is apparently the prevalent attitude for drawing general graphs. Algorithms based on this approach consist of two components. The first is the force (or energy) model that quantifies the quality of a drawing. The second is an optimization algorithm for computing a drawing that is locally optimal with respect to this model. The resulting final layout brings the system to equilibrium, where the total force on each vertex is zero, or equivalently, the potential energy is locally minimal with respect to the vertex positions. Regarding the drawing standard, force-directed methods draw the edges as straight-line segments, so the whole issue reduces to the problem of positioning the vertices.

In this section we outline some notable work on force-directed graph drawing.

2.1 The Spring Embedder Method

The spring embedder method is the earliest viable algorithm for drawing general graphs. It was proposed by Eades [Ea84], and was later refined by Fruchterman and Reingold [FR91]. This method likens a graph to a mechanical collection of electrical charged rings (the vertices) and connecting springs (the edges). Every two vertices reject each other by a repulsive force and adjacent vertices (connected by an edge) are pulled together by an attractive force. The method seeks equilibrium of these conflicting constraints. Spring based methods are very successful with small-sized graphs of up to around 50 vertices.

Regarding the optimization algorithm, we describe the method of [FR91]. A predetermined number of sweeps is performed. In each sweep every vertex is moved in the direction of the total force exerted on it. The extent of the movement is determined by a global cooling schedule that restricts the distance a vertex can move as a decreasing function of the sweep number. Frick et al. [Fr$^+$94] have suggested several improvements on this scheme, the most notable of which is a local cooling schedule, resulting in a different extent of movement for each vertex according to its temperature. They show graphs of size 256. More recently, Tunkelang [Tu99] suggested several other refinements to the optimization method. One is to approximate the effect of many distant repulsive

forces as being exerted by a single vertex using a quad-tree, in the spirit of the Barnes-Hut method [BH86]. Another improvement is the use of the Conjugate-Gradient method for decreasing the energy. Tunkelang dealt with 1000-vertex graphs, drawn in 5-10 minutes.

2.2 Kamada and Kawai's Method

Kamada and Kawai [KK89] modelled a graph as a system of springs that act in accordance with Hooke's Law: Every two vertices are connected by a spring, whose rest length is proportional to the graph-theoretic distance between its two endpoints, and its stiffness is inversely proportional to the square of its rest length. The optimization procedure tries to minimize the total energy of the system, that is:

$$E = \sum_{v,u \in V} \frac{1}{d_{uv}^2}(l(u,v) - Ld_{uv})^2$$

where $l(u,v)$ is the length of the spring between u and v, L is the length of a single edge, and d_{uv} is the graph-theoretic distance between u and v.

Kamada and Kawai's method treats all the aesthetic criteria that the spring-embedder method addresses, and produces drawings with a similar quality. An advantage of this method is that it can be applied straightforwardly to drawing edge-weighted graphs, assuming that edge lengths have to reflect their weights.

2.3 Hadany and Harel's Multi-Scale Algorithm

The *multi-scale* approach proposed by Hadany and Harel [HH99] is an improvement of the force-directed technique, which facilitates the drawing of larger graphs. The main idea of this method is to consider a series of abstractions of the graph called *coarse graphs*, in which the combinatorial structure is significantly simplified, but important topological features are well preserved. The energy minimization is divided between these coarse graphs, in such a way that globally related properties are optimized on coarser graphs, while locally related properties are optimized on finer graphs. As a result, the energy minimization process considers only small neighborhoods at once, yielding a quick running time. The scheme of the algorithm is a recursive repetition of:

1. Perform *fine-scale* relocations of vertices that yield a locally organized configuration.

2. Perform *coarse-scale* relocations (through local relocations in the coarse representations) correcting global disorders not found in stage (1).

3. Perform *fine-scale* relocations that correct local disorders introduced by stage (2).

All actual local relocations are carried out in [HH99] by minimizing the energy defined in [KK89] using simple gradient-descent. The coarsening step

is done in a pre-processing stage by contracting edges that minimize a convex combination of three objectives (in a greedy fashion):

- The *cluster number*: The number of original vertices forming the new vertex.
- The *degree number*: The degree of the vertices forming the new vertex in the current abstraction of the graph.
- The *homotopic number*: The number of vertices (in the coarse graph) that are adjacent to both the vertices forming the edge.

The algorithm produces good results on large graphs containing hundreds of vertices, in a reasonable time.

In this paper we continue the multi-scale approach of Hadany and Harel [HH99], and introduce a simpler and much faster algorithm which produces better drawings.

3 Multi-Scale Graph Drawing

The intuition of [HH99] for beauty in graph layout is that the graph should be nice on all scales. In other words the drawing should be nice at both the micro level and the macro level. Relying only on this intuition, we will formalize the notion of *scale* relevant to the graph drawing problem. The crucial observation is that global aesthetics refer to phenomena that are related to large areas of the picture, disregarding its micro structure, which has only a minor impact on the global issue of beauty. On the other hand, local aesthetics refer to phenomena that are limited to small areas of the drawing. Following this line of thinking, we will construct a *coarse scale* of a drawing by shrinking nodes that are drawn close to each other, into a single node, obtaining a new drawing that eliminates many local details but preserves the global structure of the original drawing. An alternative view of our notion of coarsening is as an approximation of a nice layout. This approximation allows vertices to deviate from their final position by an amount limited to some constant r. As a consequence, we can unify all the vertices whose final location lies within a circle of radius r, and thus obtain the coarse scale representation.

Our presentation of the drawing scheme is preceded by some definitions:

Definition 3.1
A layout *of a graph $G(V, E)$ is a mapping of the vertices to the Euclidean space: $L_G : V \longrightarrow \mathbb{R}^2$. We often omit the subscript $_G$.*
For simplicity, we assume that there is a single optimal layout with respect to fixed set of aesthetic criteria accepted in force-directed algorithms. We term this layout nice. *The nice layout of G is denoted by L^*_G, or simply L^*.*

Definition 3.2
L is a locally nice *layout of $G(V, E)$ with respect to r, if the intersection of $L(V)$*

with every circle of radius r induces a nice layout of the appropriate subgraph of G.

Definition 3.3
L is a globally nice *layout of $G(V,E)$ with respect to r if*

$$\max_{v \in V}\{|L(v) - L^*(v)|\} < r$$

Definition 3.4
A locality preserving k-clustering *(a k-*lpc *for short) of $G(V,E)$ with respect to r is the weighted graph $G(\{V_1, V_2, \ldots, V_k\}, E', w)$, where:*

$$V = V_1 \cup V_2 \cdots \cup V_k, \quad \forall i \neq j \,:\, V_i \cap V_j = \emptyset$$
$$E' = \{\, (V_i, V_j) \mid \exists (v_i, v_j) \in E \,\wedge\, v_i \in V_i \,\wedge\, v_j \in V_j \}$$
$$w(V_i, V_j) = \frac{1}{|V_i||V_j|} \sum_{v \in V_i, u \in V_j} d_{vu}\,, \quad \forall_{(V_i,V_j)\in E'}$$

(d_{vu} is the shortest distance between u and v in G)

and for every i:

$$\max_{v,u \in V_i} \{|L^*(v) - L^*(u)|\} < r$$

i.e., all the vertices in one cluster are drawn relatively close in the nice layout. We sometimes call the vertices of a k-lpc clusters.

At first glance, this definition seems to be of a little practical value, as it refers to the unknown nice layout L^*. We will discuss this important point in the next section.

Definition 3.5
A multi-scale representation *of a graph $G(V,E)$ is a sequence of graphs G^{k_1}, G^{k_2}, $\ldots$, G^{k_l}, where $k_1 < k_2 < \cdots < k_l = |V|$, and for all $1 \leqslant i \leqslant l$: G^{k_i} is a locality preserving k_i-clustering of $G(V,E)$ with respect to r_i, where $r_1 > r_2 > \cdots > r_l = 0$.*

Remark: We naturally assume that in a nice layout of an edge-weighted graph, the lengths of the edges have to reflect their weights.

Our method relies on the ease of drawing graphs with a small number of vertices and on the following two assumptions, which formalize what we think to be amenability to multi-scale aesthetics — independence between global and local aesthetics.

Assumption 3.1
Let G_r be k-lpc of a graph G with respect to r, and let $\hat{r} \geqslant r$. L is a globally nice layout of G_r with respect to $\hat{r}$ if and only if L is a globally nice layout of G with respect to $\hat{r}$. (In G, we take $L(v) = L(V_i)$ for each $v \in V_i$).

Corollary: If L is a nice layout of a k-lpc of a graph G with respect to r then L is a globally nice layout of G with respect to r.

The intuition of Assumption 3.1 is that global aesthetics is independent of the micro structure of the graph, so the differences between the layouts of G_r and of G are bounded with r.

Assumption 3.2
If L is both a locally and a globally nice layout of a graph G with respect to r, then it is a nice layout of G.

Now we present the multi-scale drawing scheme, which draws a graph by producing a sequence of improved approximations of the final layout.

The Multi-Scale Drawing Scheme

1. Place the vertices of G randomly in the drawing area.

2. Choose an adequate decreasing sequence of radiuses $\infty = r_0 > r_1 > r_2 > \cdots > r_l = 0$.

3. *for* i=1 *to* l *do*

- **3.1** Choose an appropriate value of k_i, and construct G^{k_i}, a k_i-lpc of G w.r.t. r_i.
- **3.2** Place each vertex of G^{k_i} at the (weighted) location of the vertices of G that constitute it.
- **3.3** Locally beautify local neighborhoods of G^{k_i}.
- **3.4** Place each vertex of G at the location of its cluster (i.e., the vertex in G^{k_i}).

4. *end*

The viability of the scheme stems from the following observations:

1. In the first iteration, after step 3.3, we should have a nice layout of G^{k_1}. To guarantee this, the value of r_1 has to be large enough so that the resulting G^{k_1} will be small and can be easily drawn nicely.

2. Step 3.3 should yield a locally nice layout of G^{k_i} w.r.t. r_{i-1}. To guarantee this, we have to choose large enough neighborhood.

3. At the beginning of iteration i, we have a globally nice layout of G^{k_i} w.r.t. r_{i-1}. Hence, by Assumption 3.2, after step 3.4 we have a nice layout of G^{k_i}. This layout is a globally nice layout of $G^{k_{i+1}}$ w.r.t. r_i, by Assumption 3.1.

We remark that the choice of the multi-scale representation can be based upon either the decreasing sequence $r_1, \ldots, r_l = 0$ (as described above) or the increasing sequence $k_1, \ldots, k_l = |V|$ (as we have done in our implementation).

In Definition 3.4 we added weights to the edges of the k-lpc in order to retain the size proportions of G, which is necessary for making Assumption 3.1 valid. However, in practice, we conjecture that our scheme works well even without weighting the edges of the k-lpc, when using some variants of the spring-embedder method (e.g., those of [Ea84] and [FR91]) as the local beatification method. The reason for this is that such methods benefit significantly from the better initialization, which, while not capturing the final size of the graph, often solves many large scale conflicts correctly.

4 The New Algorithm

In order to implement a multi-scale graph drawing algorithm based on the scheme of Section 3, we have to further elaborate on two points: (1) how to find the multi-scale representation of a graph (line 3.1 of the scheme), and (2) how to devise a locally nice layout (line 3.3 of the scheme).

4.1 Finding a Multi-Scale Representation

In order to construct a multi-scale representation of a graph G based on Definition 3.5, we must find a k-lpc of G, in it vertices that are drawn close in the nice layout should be grouped together. The important question is: *How can we know which vertices will be close in the final picture, if we still do not know what the final picture looks like?* [1] Luckily we do have a heuristic that can help decide which vertices will be drawn closely. Moreover, this key decision can be made very rapidly, and is a major reason for the fast running time of our algorithm.

The heuristic is based on the observation that a nice layout of the graph should convey visually the relational information that the graph represents, so *vertices that are closely related in the graph (i.e., the graph theoretic distance is small) should be drawn close together.* This heuristic is very conservative, and all the force directed drawing algorithms use it heavily.

Employing this heuristic, we can approximate a k-lpc of G by using an algorithm for the well known *k-clustering* problem. In this problem we wish to partition V into k clusters so that the longest graph-theoretic distance between two vertices in the same cluster is minimized. In reality, we would like to identify every vertex in the cluster with a single vertex that approximates the barycenter of the cluster. Hence we use a solution to the closely related *k-center* problem, where we want to choose k vertices of V, such that the longest distance from V to these k centers is minimized. These fundamental problems arise in many areas and have been widely investigated in several papers (see e.g., [Go85] and [HS86]). Unfortunately, both problems are NP-hard, and it has been shown in [Go85] and [HS86] that unless P=NP there does not exist a $(2-\epsilon)$-approximation

[1] What is needed is only a sufficient (even if not necessary) condition that vertices are close.

algorithm for any fixed $\epsilon > 0$.[2] Nevertheless, there are various fast and simple 2-approximation algorithms for these problems.

We will approximate a k-lpc as the solution to the k-center problem, adopting a 2-approximation method mentioned in [Ho96]:

K-Centers $(G(V,E)$,$k)$
% *Goal:* Find a set $S \subseteq V$ of size k, such that $\max_{v \in V} \min_{s \in S}\{d_{sv}\}$ is minimized.
  $S \leftarrow \{v\}$ for some arbitrary $v \in V$
  for $i = 2$ to k **do**
    1. Find the vertex u farthest away from S
      (i.e., such that $\min_{s \in S}\{d_{us}\} \geqslant \min_{s \in S}\{d_{ws}\}$, $\forall w \in V$)
    2. $S \leftarrow S \cup \{u\}$
  return S

Complexity: Line 1 can be carried out in time $\Theta(|E|)$ by BFS. In our case, it can be done faster, since, as we shall see, we already have the all-pairs shortest path length (APSP) at our hands (it is needed for the local beautification). Utilizing this fact and memorizing the current distance of every vertex from S, we can implement line 1 in time $\Theta(|V|)$, yielding a total time complexity of $\Theta(k|V|)$ (without the APSP computation).

4.2 Local Beautification

We have chosen to use a variant of the Kamada and Kawai method [KK89] as our local drawing method. We found it to be very appropriate, because it relates every pair of vertices, so, when constructing a new coarse representation of the graph, we do not have to define which pairs of vertices are connected by an edge. Notice that this property of the Kamada and Kawai method has a price: it forces us to waste $\Theta(|V|^2)$ memory, even when the graph is sparse. Another advantage of the Kamada and Kawai method is that it can deal directly with weighted graphs, which is convenient in our case since the multi-scale representation of a graph contains weighted graphs.

The Energy Function: We consider the graph $G(V,E)$, where each vertex v is mapped by the layout L into a point in the plane $L(v)$ with coordinates (x_v, y_v). The distance d_{uv} is defined as the length of the shortest path in G between u and v. We define the *k-neighborhood* of v to be: $N^k(v) = \{u \in V \mid 0 \leqslant d_{uv} < k\}$. In order to find a layout with aesthetically pleasing k-neighborhoods, we use an energy function that relates the graph theoretic distance between vertices in the graph to the Euclidean distance between them in the drawing, and is defined as follows:

$$E_k = \sum_{v \in V} \sum_{u \in N^k(v)} k_{uv}(\|L(u) - L(v)\| - l d_{uv})^2$$

[2] A δ-approximation algorithm delivers an approximate solution guaranteed to be within a constant factor δ of the optimal solution.

where l is the length of a single edge, $\|L(u)-L(v)\|$ is the Euclidean distance between $L(u)$ and $L(v)$, and k_{uv} is a weighting constant that can be either $\frac{1}{d_{uv}}$ or $\frac{1}{d_{uv}^2}$.

The energy represents the normalized mean squared error between the Euclidean distance of vertices in the picture and the graph-theoretic distance. Only pairs in the same k-neighborhood are considered.

Local Minimization of the Energy: Our purpose is to find a layout that brings the energy E_k to a local minimum. The necessary condition of a local minimum is as follows:

$$\frac{\partial E_k}{\partial x_v} = \frac{\partial E_k}{\partial y_v} = 0, \quad \forall v \in V$$

To achieve this condition we iteratively choose the vertex that has the largest value of Δ_v^k, which is defined as:

$$\Delta_v^k = \sqrt{(\frac{\partial E_k}{\partial x_v})^2 + (\frac{\partial E_k}{\partial y_v})^2}$$

and move this vertex, v, by the amount of $\delta_v^k = (\delta_v^k(x), \delta_v^k(y))$. The computation of δ_v^k is carried out by viewing E_k as a function of only $L(v) = (x_v, y_v)$, and the use of a two-dimensional Newton-Raphson method. As a result, the unknowns $\delta_v^k(x)$ and $\delta_v^k(y)$ are found by solving the following pair of linear equations:

$$\frac{\partial^2 E_k}{\partial x_v^2}\delta_v^k(x) + \frac{\partial^2 E_k}{\partial x_v \partial y_v}\delta_v^k(y) = -\frac{\partial E_k}{\partial x_v}$$
$$\frac{\partial^2 E_k}{\partial y_v \partial x_v}\delta_v^k(x) + \frac{\partial^2 E_k}{\partial y_v^2}\delta_v^k(y) = -\frac{\partial E_k}{\partial y_v}$$

The interested reader can find further details in [KK89].

Algorithms for Locally Nice Layout

The following algorithm, which is based on [LSD] and as mentioned in [Br+95], is a modification of [KK89]. The algorithm computes a nice layout of every k-neighborhood of a graph:

LocalLayout($d_{V\times V}$, L, k, $Iterations$)
% *Goal:* Find a locally nice layout L by beautifying k-neighborhoods
% $d_{V\times V}$: all-pairs shortest path length
% L: initialized layout
% k: radius of neighborhoods
for $i = 1$ to $Iterations \cdot |V|$ **do**
1. Choose the vertex v with the maximal Δ_v^k
2. Compute δ_v^k by solving the above mentioned equations
3. $L(v) \leftarrow L(v) + (\delta_v^k(x), \delta_v^k(y))$
end

A typical value of the parameter $Iterations$ is 4.

Complexity: An efficient implementation of line 1 is by storing the first derivatives of E_k in a *binary heap*, sorted by the values of Δ_v^k. (After the movement of each vertex v, the first derivatives of vertices in $N^k(v)$ are updated.) Hence, the computation time of line 1 is $\Theta(\log |V|)$. The computation time of δ_v^k (line 2) is $\Theta(|N^k(v)|)$, which is a constant for a bounded degree graph. These computations are carried out $Iterations \cdot |V|$ times, so the overall time complexity for a bounded degree graph is $\Theta(|V| \log |V|)$.

This algorithm differs from the one of Kamada and Kawai [KK89] in its single loop structure, and in the use of a predefined number of iterations, instead of basing termination on the value of Δ_v^k. This results in much faster termination when dealing with large graphs. We recommend this as an improvement (in a stand-alone sense) of the Kamada and Kawai method. We have tested it and have found it far superior for larger graphs. The recommended value of $Iterations$ when running this algorithm as a stand alone (not as a part of the multi-scale algorithm) is 10, at least. Performance can be improved by executing most of the iterations (the first ones) without choosing the vertex with the largest value of Δ_v^k, but iterating through all the vertices in some order.

4.3 The Multi-Scale Drawing Algorithm

We now describe the full algorithm:

Layout$G(V, E)$
% *Goal:* Find L, a nice layout of G
% *Constants:*
% Rad[= 7] — determines radius of local neighborhoods
% $Iterations$[= 4] — determines number of iterations in local beautification
% $Ratio$[= 3] — ratio between number of vertices in two consecutive levels
% $MinSize$[= 10] — size of the coarsest graph
  Compute the all-pairs shortest path length: $d_{V\times V}$
  Set up a random layout L
  $k \leftarrow MinSize$
  while $k \leqslant |V|$ **do**
    $centers \leftarrow$ **K-Centers**$(G(V, E)\ , k)$
    $radius = \max_{v\in centers} \min_{u\in centers}\{d_{vu}\} \cdot Rad$
    LocalLayout ($d_{centers\times centers}$, $L(centers)$,$radius$,$Iterations$)
    for every $v \in V$ **do**
      $L(v) \leftarrow L(center(v)) + rand$
    $k \leftarrow k \cdot Ratio$
  return L

Comments: Inside the for-loop, the call $center(v)$ returns the center that is closest to v. We add a small random noise $(0, 0) < rand < (1, 1)$, because our local beautification algorithm performs badly when the vertices are initialized to the same point. One might try to improve the global aesthetics by iteratively repeating the body of the while-loop (for each fixed k) a number of times that decreases with k (e.g., $\frac{|V|}{|k|}$ times).

Complexity: The overall asymptotical complexity is determined by the computation of the all-pairs shortest path length (APSP) (line 1), which we implemented by initiating a BFS from every vertex. It thus takes time $\Theta(|V||E|)$. Experimental running times are given in Table 1. The space complexity is $\Theta(|V|^2)$, since we have to memorize the all-pairs shortest path length matrix.

5 Examples

This section contains examples of the results of our algorithm. The implementation is in C++, and runs on a Pentium III 1GHz PC. Table 1 gives the actual running times of the algorithm on the graphs given here. We should mention that our implementation is far from being efficient, and an optimized version will probably do better. For all the executions, we have used the default parameters: $Iterations = 4$, $Ratio = 3$, $MinSize = 10$. Regarding, Rad, its default value – 7 was sufficient for achieving reasonable results for all the graphs, except the full binary tree. However, we have found for several graphs, that we can get "smoother" results when increasing Rad. Hence, for these graphs we give also results with $Rad > 7$, as indicated in the table. As can be seen in the table,

almost all the running time is dedicated for computing the all-pairs shortest path length (APSP), and for performing local beautification.

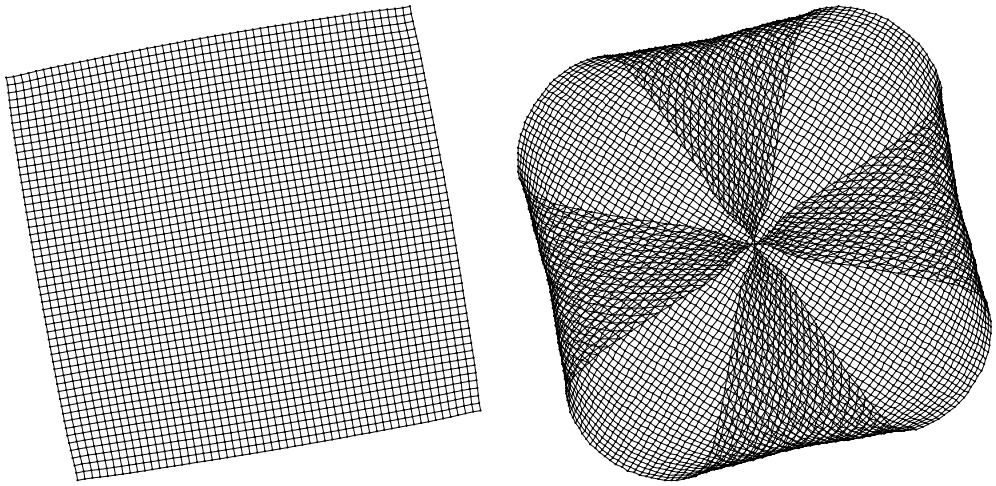
Graph Name	**Figure**	**|V|**	**|E|**	**Rad**	**APSP time**	**Beautification time**	**Overall time**
Grid 32×32		1024	1984	7	0.3	0.7	1.08
Grid 55×55	1(a)	3025	5940	7	1.45	4.11	5.87
Grid 80×80		6400	12640	7	12.11	15.153	29.39
Grid 100×100		10000	19800	7	38.27	35.11	76.22
Folded Grid 80×80	1(b)	6400	12642	7	12.11	15.34	29.16
				20	12.11	34.81	49.22
Torus 64×16	2(a)	1024	2048	7	0.3	0.81	1.16
				15	0.3	1.94	2.28
Torus 160×40	2(b)	6400	12800	7	12.13	15.6	29.57
				15	12.13	27.06	41.27
Sierpinski depth 8	3	9843	19683	7	39.36	29.39	71.45
				15	39.36	38.59	80.83
Random Sparse Grid	4(b)	6400	9480	7	12.58	13.75	28.11
Random Sparse Torus	5(b)	1600	2133	7	0.42	1.41	2.02
Full Binary Tree	6	1023	1022	16	0.13	3.59	3.73
				18	0.13	4.72	4.88
Partial Grid	7	3025	5156	7	1.41	3.85	5.56
				35	1.41	20.31	22.04
3elt	8	4720	13722	7	6.13	10.28	17.73
				20	6.13	31.75	39.71
Crack	10	10240	30380	7	49.61	22.05	73.34
4elt	9	15606	45878	7	129.98	143.67	285.45
				20	129.98	236.33	383.06

Table 1: Running time (in seconds; non-optimized) of the various components of the algorithm

The layouts shown in Figures 1–3 demonstrate applying our algorithm to regular graphs. We were particularly happy with the graphs in Figures 4 and 5, since the "correct" (grid or torus) appearance is retained despite the partiality of information available to the algorithms.

Drawing the full binary trees in Figure 6 was harder. Force-directed methods behave better on bi-connected graphs, since there are forces that work in many ways and directions. They perform quite badly on trees. Our method succeeded in finding a nice layout of the trees only when it considered the entire graph as one in its "local" beautification stages. The problem is that the local beautification scheme is based only on neighborhoods in the graph-theoretic sense, but in deep kinds of trees, leaves should show up close to each other even if they are far apart in the graph-theoretic sense.

We mention that even in laying out full binary trees our method has two advantages when compared to Kamada and Kawai's method [KK89]. First, its running time is still faster; about 5sec for a 1023-vertex tree, compared with 15sec of Kamada-Kawai's method (implemented efficiently, as discussed in Section 4.2), on account of the relatively few beautification iterations. Second,



(a) (b)

Figure 1: (a) 55x55 (3025-vertex) square grid; (b) 80x80 (6400-vertex) square grid with each two opposite corners connected

the resulting picture (Figure 6(a)) is almost planar and has no global distortions, in contrast to Kamada-Kawai's method, which will be trapped in many local minima, and will always result in many edge crossings (see Figure 6(c)).

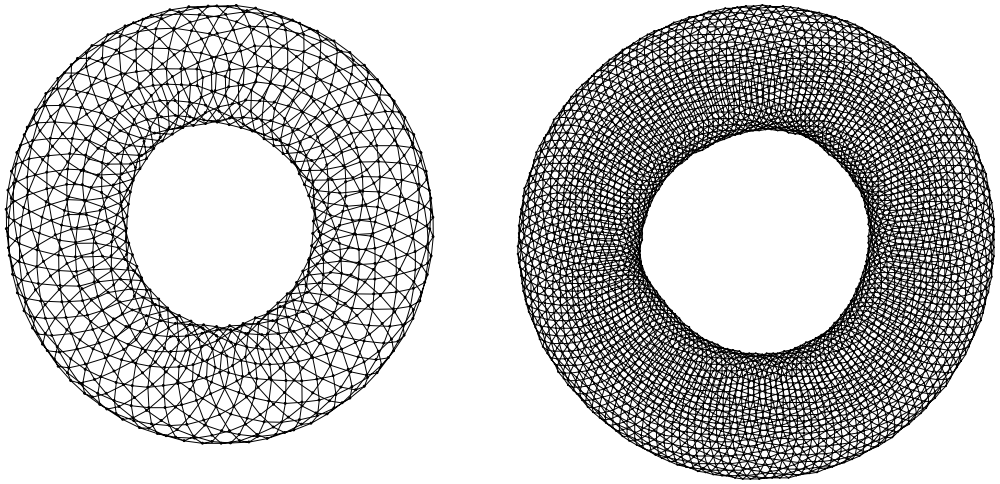
Figure 7, which is a grid with some of the horizontal edges removed, illustrates a similar problem. In Figure 7(a), we set things up so that the local beautification procedure considered larger than regular neighborhoods, as this is the only way to take into account together the vertices of two consecutive "lines" to make the resulting picture planar. On the other hand, since in Figure 7(b) the local beautification procedure considered neighborhoods that were too small, we get the overlapping.

In Figures 8 and 9 we present "real-life" graphs taken from Francois Pellegrini's Scotch graph collection, at: `http://www.labri.u-bordeaux.fr/Equipe/PARADIS/Member/pelegrin/graph/`. In Figure 10 we present the beautiful Crack graph taken from the collection of Jordi Petit, at: `http://www.lsi.upc.es/~jpetit/MinLA/Experiments/`.

6 Related Work

The multi-scale approach to graph drawing was first proposed in [HH99], as described in Section 2.

The major technical improvement of our approach over [HH99] is a more intuitive and easier to implement coarsening step, which is the source of its much better running time. Our method eliminates the entire costly pre-processing



(a) (b)

Figure 2: Toruses: (a) 64x16 (1024-vertex) (b) 160x40 (6400-vertex)

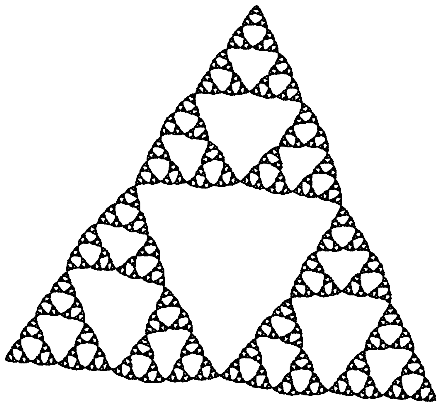
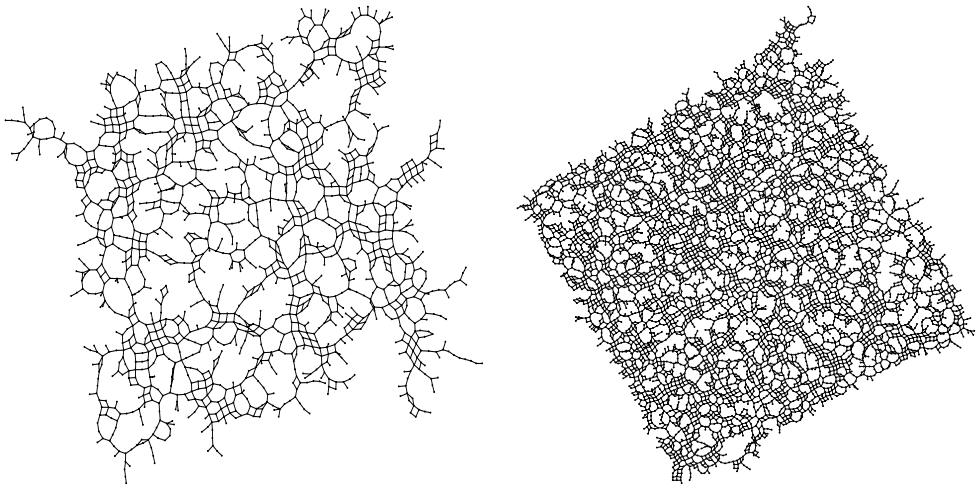
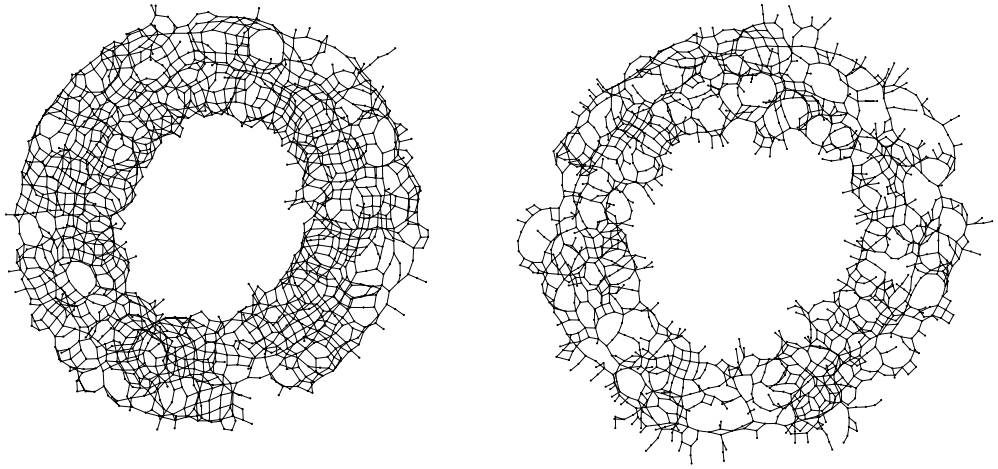


Figure 3: A depth 8 Sierpinski (9843 vertices)



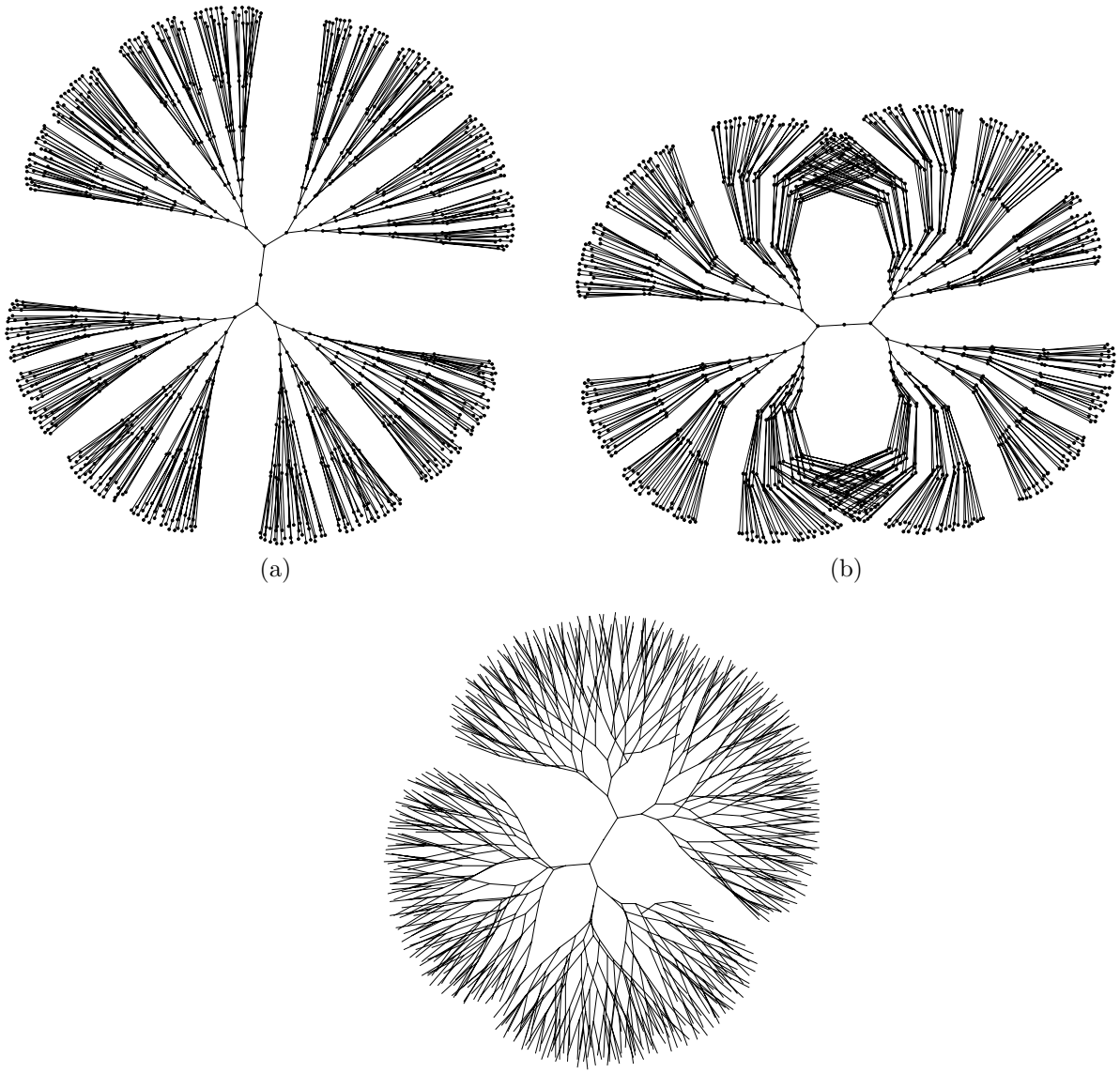
(a) (b)

Figure 4: (a) 40x40 (1600-vertex) grid with $\frac{1}{3}$ of the edges omitted at random; (b) 80x80 (6400-vertex) grid with $\frac{1}{4}$ of the edges omitted at random



(a) (b)

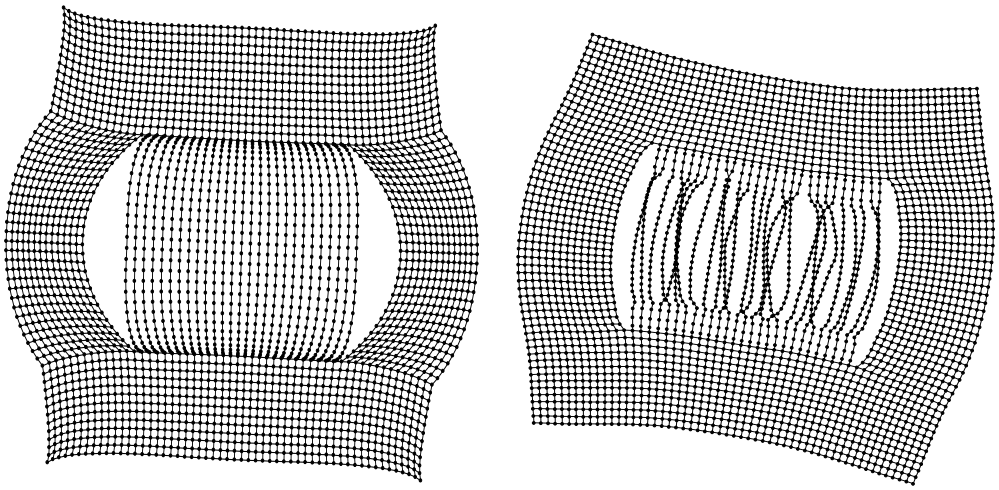
Figure 5: 80x20 (1600-vertex) torus with $\frac{1}{5}$ and $\frac{1}{3}$ of the edges omitted at random



(a) (b)

(c)

Figure 6: Full binary trees: (a,b) 1023-vertices, depth 9; In (a) beautification considered the whole graph at once, while in (b) neighborhoods of radius 16 were considered at once; (c) is the result of the Kamada-Kawai algorithm. Notice that unlike our algorithm, Kamada-Kawai results with bad "global" edge crossings (i.e., near the root).



(a) (b)

Figure 7: 55x55 (3025-vertex) partial grid: (a) beautification considered larger than normal neighborhoods, of radius 35 (b) beautification considered usual neighborhoods of radius 7

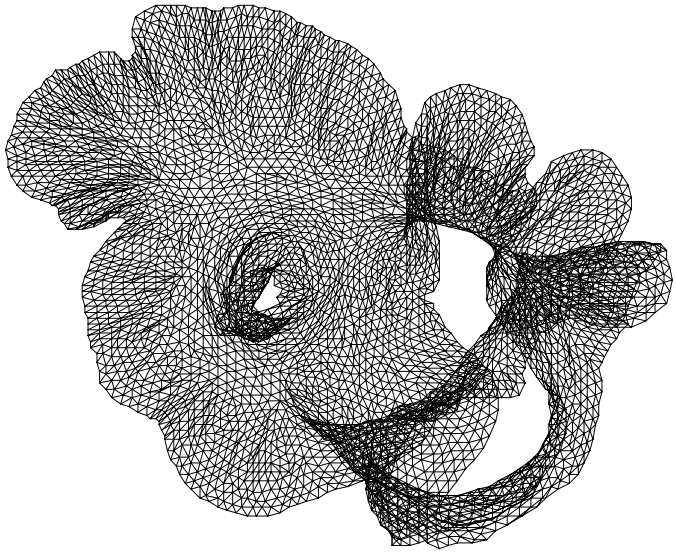


Figure 8: The graph 3elt from the Scotch collection; $|V| = 4720, |E| = 13722$

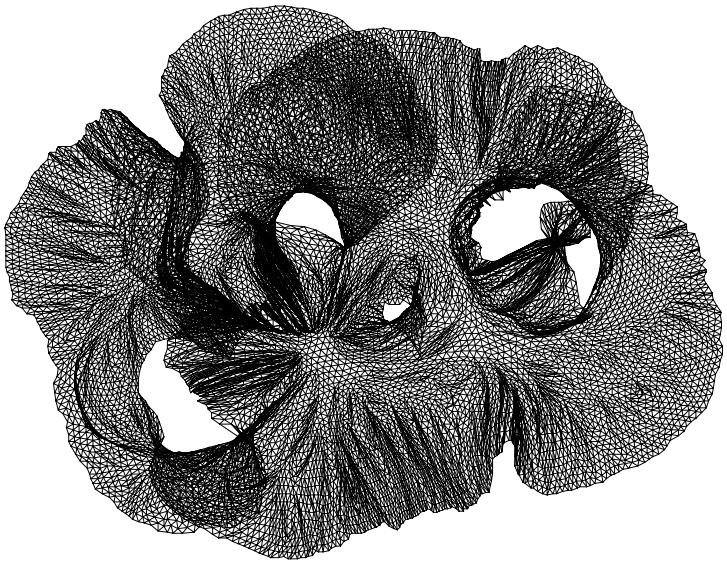


Figure 9: The graph 4elt from the Scotch collection; $|V| = 15606, |E| = 45878$

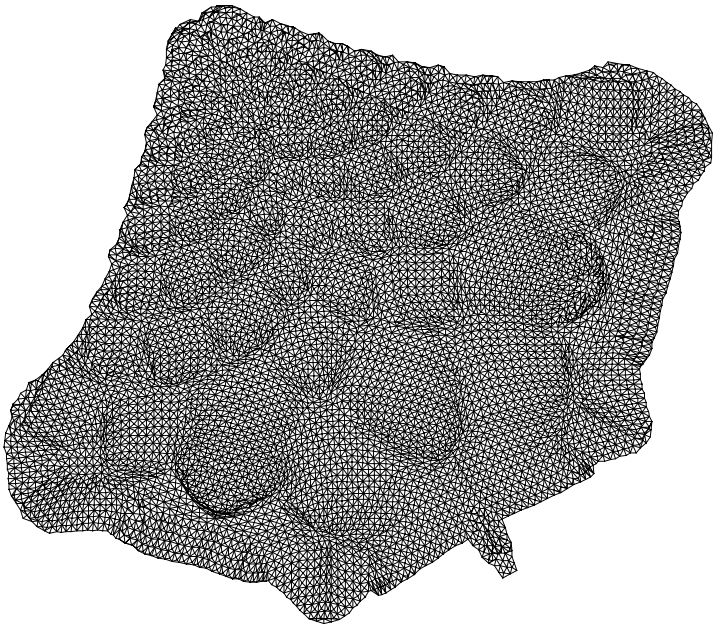


Figure 10: The graph Crack; $|V| = 10240, |E| = 30380$

step of [HH99] (with the exception of the computation of $d_{V\times V}$, which is part of the local beautification process). Practically all the coarsening overhead in our algorithm is carried out in a negligible fraction of the overall execution time.

Another advantage of our algorithm over that of [HH99] is its simpler schematic structure. Ours starts by beautifying the coarsest-scale representation and proceeds by improving the finer-scale representations until reaching the original graph. The Hadany-Harel method starts with the original graph, continues until it reaches the coarsest level, and then proceeds back to the original graph. Eliminating this alone results in a 50% improvement in beautification time, by making only half of the local beautification steps.

In addition, the beautification process of [HH99] does not directly change the layout, but performs a *relative* change of the layout, since every vertex is located anew relative to its previous location. This imposes a strong inter-dependency between all the multi-scale representations of the graph. As a result, one has to introduce weights on the edges in order to make it possible to link the layouts of the different scales. In contrast, our scheme makes a *direct* change to the graph's layout at each level of the beautification process. As a consequence, the dependencies between the various representations are weaker, and we can use a multi-scale representation with unweighted edges, which while not capturing the final size of the graph often solves many large scale conflicts correctly. As a result of this, we can potentially utilize other force-directed methods as our local beautification procedure, a fact that might possibly yield several additional benefits.

Following the appearance of our technical report [HK99], two other multi-scale (or "multi-level") methods were presented at the Graph Drawing 2000 conference together with ours, claiming to have certain advantages over ours. We now describe what seem to be their most important characteristics.

The method of Gajer et al. [G$^+$00] introduces several changes to our algorithm. The most notable one is that the graph theoretic distances between vertices are not stored in a $|V|\times|V|$ table, but are computed dynamically, when needed. This improves the storage requirements of our algorithm, making it suitable for dealing with larger graphs. Other attributes of [G$^+$00] are: drawing in higher dimensions and optimizing the energy using gradient-descent, where extent of movement is limited by a local temperature (like [Fr$^+$94]). [G$^+$00] reports on drawings of 16,000-vertex graphs in around 40 seconds.

The method of Walshaw [Wa00] is a multi-level extension of Fruchterman's and Reingold's spring embedder algorithm [FR91]. An advantage of working with the forces of [FR91] (compared to the energy of Kamada and Kawai) is that the method of [FR91] does not require the computation (and storage) of the all-pairs shortest path length. As a result, Walshaw [Wa00] can deal with huge graphs of 100,000 vertices in 10–20 minutes, which is very impressive.

Another method for drawing large graphs is that of Quigley and Eades [QE00]. Their method is an improvement of the spring embedder algorithm. Unlike the multi-scale approach, they consider the whole graph at once. However, in order to accelerate computation, the nodes are clustered in a *quad-tree* structure, which reflects actual spatial proximity. The repulsive forces between

a vertex and a cluster of distant vertices are approximated as a single force, what significantly lessens the number of forces in the system. There seem to be some connection with the work of Tunkelang [Tu99] mentioned in Section 2 above. We have not conducted tests to compare the running time of the multi-scale approach with that of [QE00]. However, the multi-scale approach optimizes several different energies on several levels, thus apparently capturing the various scales of beauty. Moreover, the multi-scale method may be faster and more robust, since it is not sensitive to the initial placement that is derived from coarser scales.

Recently, we have developed two new methods [K+01, HK02] for drawing huge graphs. They can draw 10^5-vertex graphs in few seconds and 10^6-vertex graphs in less than a minute, and thus appear to be significantly faster than previously known methods. The method in [K+01] draws a graph by quickly calculating eigenvectors of the Laplacian matrix associated with it, using a special algebraic multigrid technique, which is a rather accurate variant of the multi-scale paradigm.

The method in [HK02] is somewhat more novel and has two phases: first embed the graph in a very high dimension (say, 50) and then project it onto the 2-D plane using Principal Components Analysis. The high-dimensional embedding is done very rapidly using a simple algorithm that utilizes the freedom given by the many dimensions. This method possesses a useful inherent capability to exhibit the graph in various dimensions, and an effective means for interactive exploration of large graphs.

7 Conclusions and Future Work

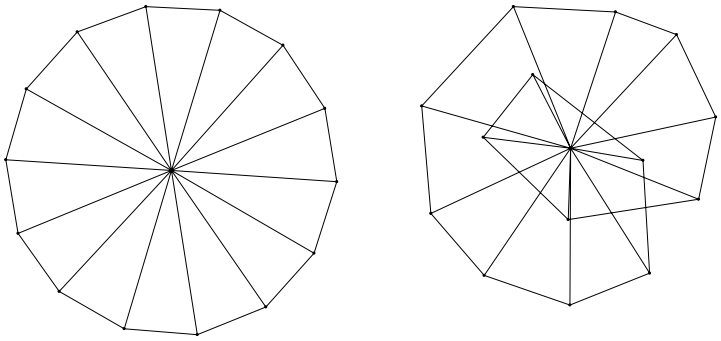
We have presented a new multi-scale approach for drawing graphs nicely, and have suggested a useful formulation for the desired properties of the coarse graphs. Our algorithm is able to deal extremely well and extremely fast with certain classes of large graphs.

The algorithm was designed for speed and simplicity and does not require explicit representations of coarse graphs.

A more powerful and general implementations of the multi-scale drawing scheme would overcome some limitations of our algorithm. We can improve the time and space complexity of our algorithm, by discarding the all-pairs shortest path length computation. This would enable the algorithm to deal with larger graphs. Another improvement lies in the construction of the coarse scale representation, for which we used a simple heuristic. For many graphs this heuristic is fine, but for some graphs it may not be enough. For example consider graphs with a tiny diameter, like the graph of Figure 11, whose diameter is 2. Our heuristic will totally fail on such a graph, since the distances between each pair of vertices are roughly the same, and it is thus unable to differentiate different clusters.

Our algorithm uses the heuristic of Kamada and Kawai [KK89], which does not punish edge-crossings explicitly. It may be interesting to see how the ex-

plicit consideration of crossings, as is done, e.g., in [DH89], will be reflected in the results of a multi-scale method such as ours. This could be of particular importance in drawing less regular-looking (that is, more random-looking) graphs.



Result of Fruchterman-Reingold Result of our method

Figure 11: Bad example: a graph with a degenerate diameter

Acknowledgements

We would like to thank the referees for their helpful comments.

References

[BH86] J. Barnes and P. Hut, "A hierarchical $O(N \log N)$ Force-Calculation Algorithm", *Nature*, **324** (1986), 446–449.

[Br+95] F.J. Brandenburg, M. Himsolt and C. Rohrer, "An Experimental Comparison of Force-Directed and Randomized Graph Drawing Algorithms", *Proceedings of Graph Drawing '95*, Lecture Notes in Computer Science, Vol. 1027, pp. 76–87, Springer Verlag, 1995.

[Di+99] G. Di Battista, P. Eades, R. Tamassia and I.G. Tollis, *Algorithms for the Visualization of Graphs*, Prentice-Hall, 1999.

[DH89] R. Davidson and D. Harel, "Drawing Graphs Nicely Using Simulated Annealing", *ACM Trans. on Graphics* **15** (1996), 301–331. (Preliminary version: Technical Report, The Weizmann Institute of Science, 1989.)

[Ea84] P. Eades, "A Heuristic for Graph Drawing",*Congressus Numerantium* **42** (1984), 149–160.

[Fr+94] A. Frick, A. Ludwig and H. Mehldau, "A Fast Adaptive Layout Algorithm for Undirected Graphs", *Proceedings of Graph Drawing 1994*, Lecture Notes in Computer Science, Vol. 894, pp. 389–403, Springer Verlag, 1995.

[FR91] T.M.G. Fruchterman and E. Reingold, "Graph Drawing by Force-Directed Placement", *Software-Practice and Experience* **21** (1991), 1129–1164.

[G+00] P. Gajer, M. T. Goodrich, and S. G. Kobourov, "A Multi-dimensional Approach to Force-Directed Layouts of Large Graphs", *Proceedings of Graph Drawing 2000*, Lecture Notes in Computer Science, Vol. 1984, pp. 211–221, Springer Verlag, 2000.

[Go85] T. Gonzalez, "Clustering to Minimize the Maximum Inter-Cluster Distance", *Theoretical Computer Science* **38** (1985), 293–306.

[HH99] R. Hadany and D. Harel, "A Multi-Scale Method for Drawing Graphs Nicely", *Discrete Applied Mathematics*, **113** 3-21 (2001). (Also, *Proc. 25th Inter. Workshop on Graph-Theoretic Concepts in Computer Science* (WG '99), Lecture Notes in Computer Science, Vol. 1665, pp. 262–277, Springer Verlag, 1999.)

[HK99] D. Harel and Y. Koren, "A Fast Multi-Scale Method for Drawing Large Graphs", Technical Report MCS99-21, Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, 1999.

[HK02] D. Harel and Y. Koren, "A Multi-dimensional Approach to Force-Directed Layouts of Large Graphs", to appear in *Proceedings of Graph Drawing 2002*, Lecture Notes in Computer Science, Springer Verlag.

[Ho96] D. S. Hochbaum (ed.), *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, 1996.

[HS86] D. S. Hochbaum and D.B. Shmoys, "A Unified Approach to Approximation Algorithms for Bottleneck Problems", *J. Assoc. Comput. Mach.* **33** (1986), 533–550.

[KK89] T. Kamada and S. Kawai, "An Algorithm for Drawing General Undirected Graphs", *Information Processing Letters* **31** (1989), 7–15.

[KW01] M. Kaufmann and D. Wagner (Eds.), *Drawing Graphs Methods and Models*, Lecture Notes in Computer Science, Vol. 2025, Springer Verlag, 2001.

[K+01] Y. Koren, L. Carmel and D. Harel "ACE: A Fast Multiscale Eigenvectors Computation for Drawing Huge Graphs", Technical Report MCS01-17, The Weizmann Institute of Science, 2001. Available at: `www.wisdom.weizmann.ac.il/reports.html`. To appear in *Proceedings of IEEE Symposium on Information Visualization (InfoVis 2002).*

[LSD] The LSD library, available from the Graphlet website at `http://www.fmi.uni-passau.de/Graphlet/download.html`

[QE00] A. Quigley and P. Eades, "FADE: Fraph Drawing, Clustering, and Visual Abstraction", *Proceedings of Graph Drawing 2000*, Lecture Notes in Computer Science, Vol. 1984, pp. 183–196, Springer Verlag, 2000.

[Tu99] D. Tunkelang, *A Numerical Optimization Approach to General Graph Drawing*, Ph.D. Thesis, Carnegie Mellon University, 1999.

[Wa00] C. Walshaw, "A Multilevel Algorithm for Force-Directed Graph Drawing", *Proceedings of Graph Drawing 2000*, Lecture Notes in Computer Science, Vol. 1984, pp. 171–182, Springer Verlag, 2000.