A Fast Multi-Scale Method for Drawing Large Graphs

David Harel and Yehuda Koren

Dept. of Applied Mathematics and Computer Science
The Weizmann Institute of Science, Rehovot, Israel
{harel,yehuda}@wisdom.weizmann.ac.il

Early version appeared as Weizmann Institute Tech. Report MCS99-21, 1999.
An abridged version appeared in *Proc. Working Conf. on Advanced Visual Interfaces (AVI2000)*, ACM Press, May 2000, pp. 282-285

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 of substantially larger size than any other algorithm we are aware of. For example, the algorithm achieves optimal drawings of 1000 vertex graphs in about 2 seconds. The paper contains graphs with over 6000 nodes. The proposed algorithm embodies a new multi-scale scheme for drawing graphs, which was motivated by the recently published multi-scale algorithm of Hadany and Harel [7]. It can 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 [2]).

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. Various 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 [4], [10], [3] and [5]. 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 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 can significantly 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 of substantially larger size than any other algorithm we are aware of. The algorithm, which was motivated by the recent multi-scale algorithm of Hadany and Harel [7], 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 Multi-Scale Graph Drawing

The intuition of [7] 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 21
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 22
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 23
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 24
A locality preserving k-clustering *(a k-*lpc *for short) of $G(V,E)$ with respect to r is the weighted graph $G(\{V_1, V_2, \dots, V_k\}, E', w)$, where:*

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

(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 25
A multi-scale representation *of a graph $G(V,E)$ is a sequence of graphs G^{k_1}, G^{k_2}, ... ,G^{k_l}, where $k_1 < k_2 < \dots < k_l = |V|$, 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 $r_1 > r_2 > \dots > r_l = 0$.*

Remark: We naturally assume that in a nice layout of a 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 21
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 21 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 22
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. Not too large, however, because we want to draw it optimally by a standard method. A sufficiently dense choice of the sequence of r_i's will do.
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 22, 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 21.

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 24 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 21 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 [4] and [5]) 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.

3 The New Algorithm

In order to implement a multi-scale graph drawing algorithm based on the scheme of Section 2, 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).

3.1 Finding a Multi-Scale Representation

In order to construct a multi-scale representation of a graph G based on Definition 25, 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 represent, 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., [6] and [9]). Unfortunately, both problems are NP-hard, and it has been shown in [6] and [9] that unless P=NP there does not exist a $(2-\epsilon)$-approximation algorithm for any fixed $\epsilon > 0$.[2] Nevertheless, there are various fast and simple 2-approximation algorithms for these problems.

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

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

We will approximate a k-lpc as the solution to the k-center problem, adopting a 2-approximation method mentioned in [8]:

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.

1. $S \leftarrow \{v\}$ for some arbitrary $v \in V$
2. *for* $i = 2$ *to* k *do*
   2.1 Find the vertex u farthest away from S
   (i.e., such that $\min_{s \in S}\{d_{us}\} \geqslant \min_{s \in S}\{d_{ws}\} \quad , \forall w \in V$)
   2.2 $S \leftarrow S \cup \{u\}$
3. *return* S
4. *end*

Complexity: Line 2.1 can be carried out in time $\Theta(|E|)$ by BFS. In our case, it can be done faster, since, as we shall see, we have the all-pairs shortest distance 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 2.1 in time $\Theta(|V|)$, yielding a total time complexity of $\Theta(k|V|)$.

3.2 Local Beautification

We have chosen to use a variant of the Kamada and Kawai method [10] 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 Functional: 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 aestheticaly pleasing k-neighborhoods, we use an energy functional 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:

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

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, which is defined as:

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

and move this vertex, v, by the amount of (δ_x^v, δ_y^v). The computation of (δ_x^v, δ_y^v) 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 δ_x^v and δ_y^v are found by solving the following pair of linear equations:

$$\begin{aligned} \frac{\partial^2 E_k}{\partial x_v^2}\delta_x^v + \frac{\partial^2 E_k}{\partial x_v \partial y_v}\delta_y^v &= -\frac{\partial E_k}{\partial x_v} \\ \frac{\partial^2 E_k}{\partial y_v \partial x_v}\delta_x^v + \frac{\partial^2 E_k}{\partial y_v^2}\delta_y^v &= -\frac{\partial E_k}{\partial y_v} \end{aligned}$$

The interested reader can find further details in [10].

Algorithms for Locally Nice Layout

The following algorithm, which is from [11] and as mentioned in [1], is a modification of [10]. The algorithm computes a nice layout of every k-neighborhood of a graph:

LocalLayout ($d_{V \times V}$ (all-pairs shortest dist.), L (initialized layout))
Goal: Find a locally nice layout L by beautifying k-neighborhoods

1. *for* i=1 *to* $Iterations \cdot |V|$ *do*
   1.1 Choose the vertex v with the maximal Δ_v
   1.2 Compute δ_x^v and δ_y^v by solving the above mentioned equations
   1.3 $L(v) \leftarrow L(v) + (\delta_x^v, \delta_y^v)$
2. *end*

A typical value of the parameter $Iterations$ is 4. (When running this algorithm as a stand alone (not as a part of the multi-scale algorithm) the value should be 10, at least.)

Complexity: The computation time of step 1.1 is $\Theta(|V|)$, since we memorize the first derivatives of E_k, and update them in time $\Theta(|N^k(v)|)$ after the movement of each vertex v. The computation time of δ_x^v and δ_y^v is $\Theta(|N^k(v)|)$. These computations are carried out $Iterations \cdot |V|$ times, so the overall time complexity is $\Theta(|V|^2)$. We can select the vertex v with the maximal Δ_v in constant time, by making this selection through sampling constant sized subsets of V, without serious harm to the quality of the results. Moreover, if the degree of G is bounded, $\Theta(|N^k(v)|)$ is constant. As a result, the overall time complexity in the bounded degree case is $\Theta(|V|)$.

In the full version of the paper we describe an alternative beautification method, which eliminates the choice of the vertex with the maximal Δ_v.

3.3 The Multi-Scale Drawing Algorithm

We now describe the full algorithm:

Layout $(G(V, E))$
Goal: Find L, a nice layout of G

1. Compute the all-pairs shortest distance $(d_{V \times V})$
2. Set up a random layout L
3. $k \leftarrow Threshold$
4. *while* $k \leqslant |V|$ *do*
   4.1 $Centers \leftarrow$ **K-Centers**$(G(V, E)\ , k)$
   4.2 **LocalLayout** $(d_{Centers \times Centers},\ L(Centers))$
   4.3 *for* every $v \in V$ *do*
      4.3.1 $L(v) \leftarrow L(center(v)) + \xi$
   4.4 $k \leftarrow k \cdot Ratio$
5. *end*

Comments: In line 4.3.1, we assume that the call $center(v)$ returns the center that is closest to v. We add a small random noise $(0, 0) < \xi < (1, 1)$, because our local beautification algorithm performs badly when the vertices are initialized to the same point. $Threshold$ and $Ratio$ are constants, with typical values of 10 and 3, respectively. One might try to improve the global aesthetics by iteratively repeating lines 4.1–4.3 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 distance (line 1), which we implemented by initiating a BFS from every vertex. It thus takes time $\Theta(|V||E|)$. For 1000-vertex sparse graphs, the local beautification (line 4.2) consumes 80-90% of the running time, and the computation of the all-pairs shortest distance consumes the remaining 10-20%. All the other parts of the algorithm consume only negligible time. As the size of the graphs becomes larger, the computation of the all-pairs shortest distance becomes more dominant. The space complexity is $\Theta(|V|^2)$, since we have to memorize the all-pairs shortest distance matrix.

4 Examples

This section contains examples of the results of our algorithm. The implementation is in C++, and runs on a Pentium III 533Mhz PC. We set the **LocalLayout** procedure to beautify neighborhoods of radius $7 \times appr$, where *appr* is the approximate distance between neighbor vertices in the current representation. The constant *iterations* was set to be 4. Typical execution time with these parameters is about 2sec for 1000-vertex graphs, under 12sec for 3000-vertex graphs, and 60sec for 6000-vertex graphs. Well optimized code will probably do better in the 6000-vertex case.

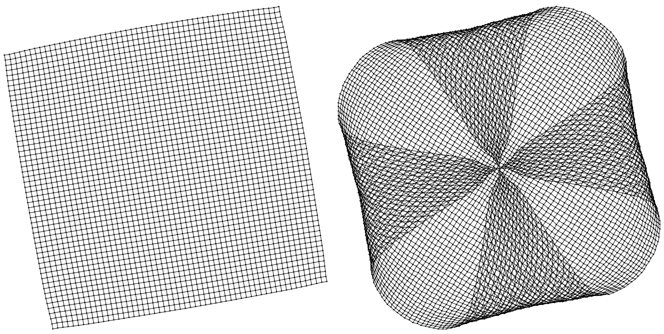
As can be seen the layouts are extremely natural looking; almost "optimal" in aesthetics. The virtually perfect layouts shown in Figures 1– 5 are typical of the power of the algorithm. The graphs in Figures 6 and 7 are particularly impressive as the "correct" (grid or torus) appearance is retained despite the partiality of information available to the algorithms.

Drawing the full binary trees in Figure 8 took more than the normal time. 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 [10]. First, its running time is still much faster (about 10sec for the 1023-vertex tree), on account of the relatively few beautification iterations. Second, the resulting picture 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.

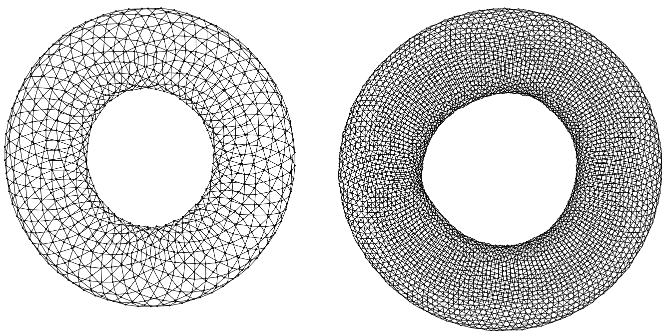
Figure 9, which is a grid with some of the horizontal edges removed, illustrates a similar problem. In Figure 9(a), we set things up so that the local beautification procedure considered larger than regular neighborhoods, as this is the only way to take into account together the vertices of two consecutive "lines" to make the resulting picture planar. On the other hand, since in Figure 9(b) the local beautification procedure considered neighborhoods that were too small, we get the overlapping.

It is interesting to mention that the fact that Figure 9(a) involves more global considerations by taking into account larger neighborhoods is not only an advantage. In fact, these global considerations sometimes come at the expense of important local aesthetics, resulting in overcrowded clusters of vertices. In general, multi-scale aesthetics have a fundamental advantage over regular aesthetics in that they separate global considerations from local ones. Hence, a multi-scale layout algorithm serves not only as a way to minimize a complex energy-function rapidly, but often also as a better general approach to graph drawing. Mixing the global and the local aesthetics, is a *compromise* that we can adopt when we can-not distinguish the primitive clusters of the graph and are willing to pay dearly in time for considering larger clusters. For example, consider the cylinder of Figure 5. In Figure 5(a), the beautification procedure considered neighborhoods of radius 4, and in Figure 5(b) larger neighborhoods were considered. Because the correct multi-scale structure of the graph was identified, Figure 5(a) is superior to Figure 5(b), in which the structure of the small circles is not apparent.



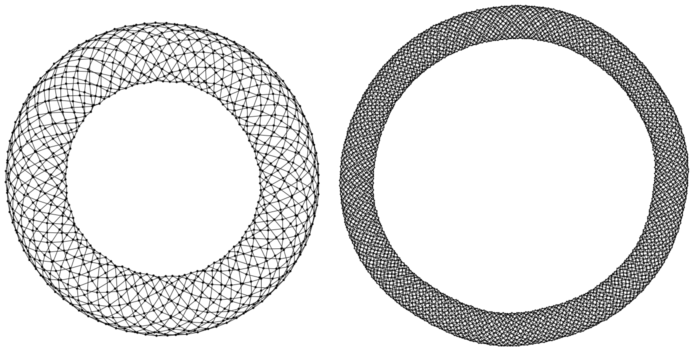
(a) (b)

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



(a) (b)

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



(a) (b)

Fig. 3. Cayley graphs of the rings: (a) Z_{1000} with generators ± 9 and ± 11 (b) Z_{6000} with generators ± 13 and ± 17

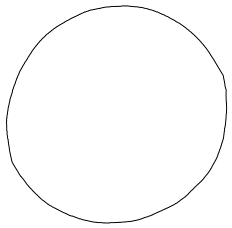
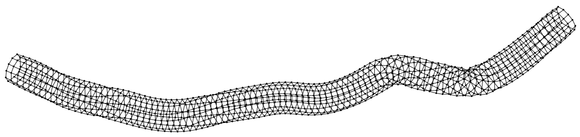
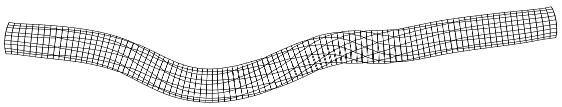


Fig. 4. 1000-vertex circle

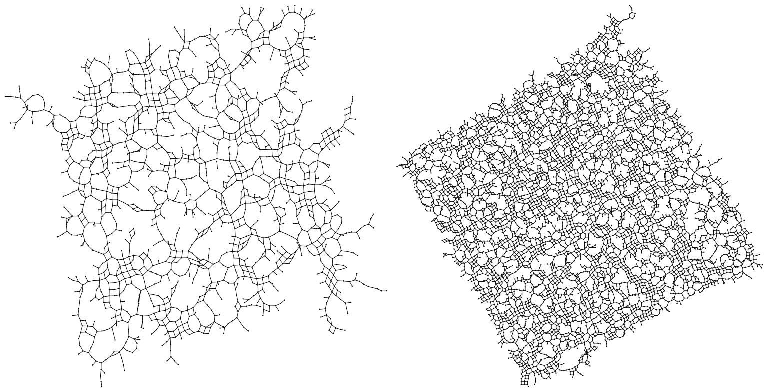


(a)



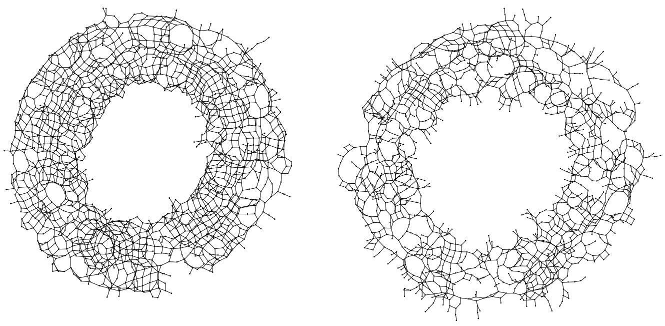
(b)

Fig. 5. 100x10 (1000-vertex) cylinder; beautification considered neighborhoods of radius 4 in (a) and 10 in (b)



(a) (b)

Fig. 6. (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)

Fig. 7. 80x20 (1600-vertex) torus with $\frac{1}{5}$ and $\frac{1}{3}$ of the edges omitted at random

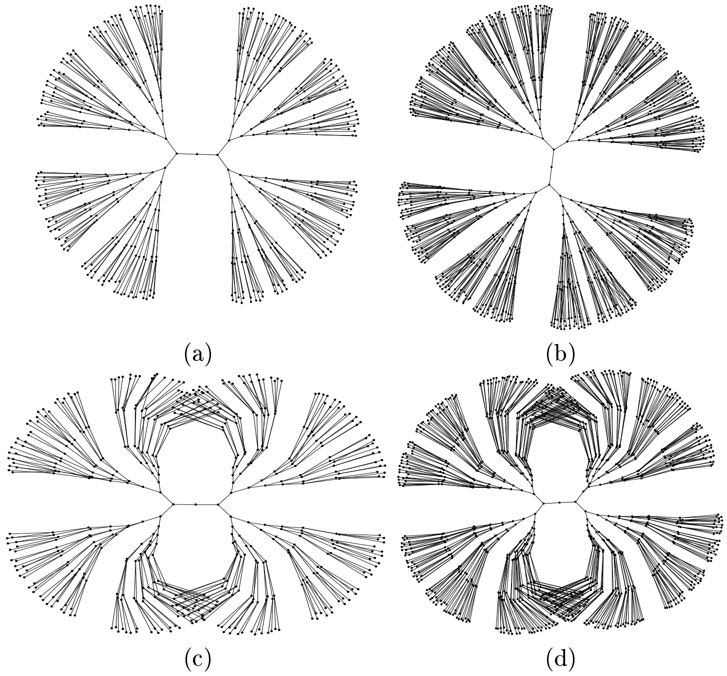


Fig. 8. Full binary trees: (a,c) 511-vertices, depth 8; (b,d) 1023 vertices, depth 9; beautification considered neighborhoods of radius 16, 18, 14 and 16, respectively; in (a) and (b) this is the whole graph.

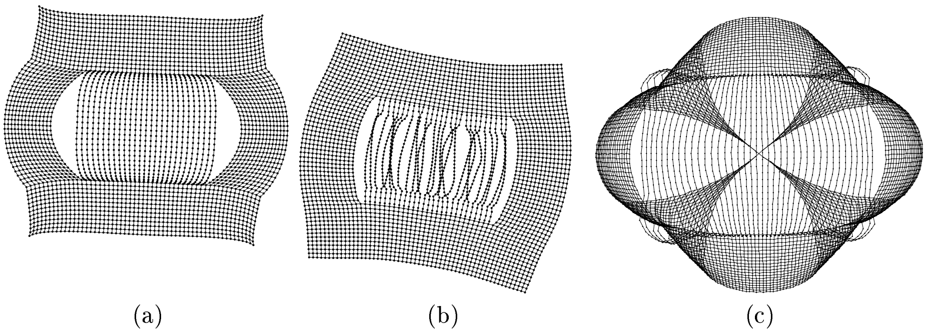


Fig. 9. (a,b) 55x55 (3025-vertex) sparse grid: (a) beautification considered larger than normal neighborhoods, of radius 35 (b) beautification considered usual neighborhoods of radius 7; (c) 80x80 (6400-vertex) sparse grid with each two opposite corners connected

5 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 coarsenings. Our algorithm is able to deal extremely well and extremely fast with 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 will overcome some limitations of our algorithm. We can change the algorithm to require only linear time and space, by discarding the all-pairs shortest distance computation (and not relying any more on the Kamada-Kawai method). This will enable the algorithm to deal with graphs of over 10,000 vertices. 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. Our heuristic may fail on such graphs, since the distances between each pair of vertices are roughly the same, and it is thus unable to distinguish between different clusters. We are aware of better heuristics that are more complicated, and a new implementation of the multiscale scheme is underway.

References

1. Brandenburg, F.J., Himsolt, M., and Rohrer, C., "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.
2. Di Battista, G., Eades, P., Tamassia, R. and Tollis, I.G., *Algorithms for the Visualization of Graphs*, Prentice-Hall, 1999.
3. Davidson, R., and Harel, D., "Drawing Graphs Nicely Using Simulated Annealing", *ACM Trans. on Graphics* **15** (1996), 301–331.
4. Eades, P., "A Heuristic for Graph Drawing",*Congressus Numerantium* **42** (1984), 149–160.
5. Fruchterman, T.M.G., and Reingold, E., "Graph Drawing by Force-Directed Placement", *Software-Practice and Experience* **21** (1991), 1129–1164.
6. Gonzalez, T., "Clustering to Minimize the Maximum Inter-Cluster Distance", *Theoretical Computer Science* **38** (1985), 293–306.
7. Hadany, R., and Harel, D., "A Multi-Scale Method for Drawing Graphs Nicely", *Discrete Applied Mathematics*, in press, 2000. (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.)
8. Hochbaum, D. S. (ed.), *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, 1996.
9. Hochbaum, D.S., and Shmoys, D. B, "A Unified Approach to Approximation Algorithms for Bottleneck Problems", *J. Assoc. Comput. Mach.* **33** (1986), 533–550.
10. Kamada, T., and Kawai, S., "An Algorithm for Drawing General Undirected Graphs", *Information Processing Letters* **31** (1989), 7–15.
11. The LSD library, available from the Graphlet website at http://www.fmi.uni-passau.de/Graphlet/download.html