A Fast Multi-Scale Method for Drawing Large Graphs

[Extended Abstract]

David Harel
Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science
Rehovot, Israel
harel@wisdom.weizmann.ac.il

Yehuda Koren
Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science
Rehovot, Israel
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 of substantially larger size than any other algorithm we are aware of. For example, the algorithm achieves optimal drawings of 1000 vertex graphs in less than 3 seconds. The paper contains graphs with over 6000 nodes. The proposed algorithm embodies a new multi-scale scheme for drawing graphs, which can significantly improve the speed of essentially any force-directed method.
Graphs have become an important part of recently proposed user interfaces, hence the relevance of this paper to work on interface issues.

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 [1]). The central objective of a graph drawing algorithm is to assign a location for each node and a route for each edge, so that the resulting picture, which is called the graph's *layout*, will be "nice".

We concentrate on the problem of drawing an undirected graph with straight-line edges. In this case the problem reduces to the problem 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 [3, 7, 2, 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 30-40 vertices. For larger graphs the convergence to the minimum, if possible at all, is very slow.

Given the recent popularity of graph representations in user interfaces, using traditional force-directed methods for graphs of a more than a few dozen vertices is inappropriate due to the slow running time. For such interactive applications, fast algorithms are a must.

In this paper we offer a new scheme for drawing graphs, that facilitates very rapid drawing of graphs with thousands of vertices, several orders of magnitude more than current force-directed methods.

Our method is based on a general *multi-scale* approach, which was first used for graph drawing by Hadany and Harel [6]. The main idea of this approach is to consider much smaller abstractions of the graph, which approximate its coarse structure, for achieving a global arrangement of the graph, and to accomplish local beutification by optimizing small neighborhoods (using traditional methods).

2. MULTI-SCALE GRAPH DRAWING

The intuition of [6] 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. A 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 two definitions:

DEFINITION 2.1.
A layout *of a graph $G(V, E)$ is a mapping of the vertices to the Euclidean space: $L : V \longrightarrow \mathbb{R}^2$.*
For simplicity, we assume that there is a single optimal layout with respect to a 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 2.2.
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.

The Multi-Scale Drawing Scheme

1. Place the vertices of G randomly in the drawing area.
2. Choose an adequate decreasing sequence of radiuses $r_1 > r_2 > \dots > r_l = 0$.
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*

In the following section, we briefly describe an implementation of this method which is optimized for running quickly with the beautification method of Kamda-Kawai [7].

3. THE NEW ALGORITHM

When implementing our multi-scale drawing scheme one confronts the necessity of a reliable algorithm for deciding whether two vertices will be drawn close in the final nice layout. This decision is crucial in the construction of a k-lpc of a graph. 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?* 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. Unfortunately, both problems are NP-hard, and in addition it has been shown that unless P=NP there does not exist even a $(2 - \epsilon)$-approximation algorithm for any fixed $\epsilon > 0$. Nevertheless, there are various fast and simple 2-approximation algorithms for these problems.

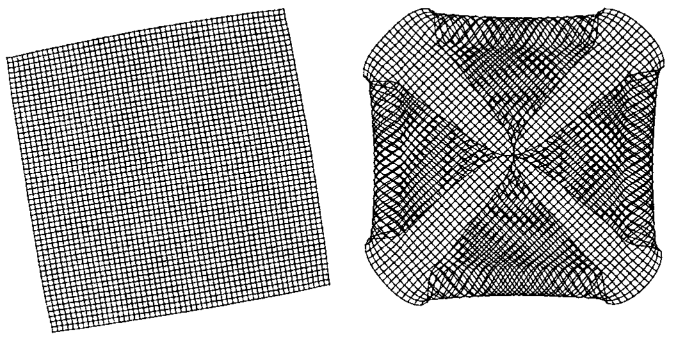
We will approximate a k-lpc as the solution to the k-center problem, adopting a 2-approximation method mentioned in [4]. Since in the preprocessing step the matrix of the all pairs shortest path is computed (it is needed for the beutification algorithm), the 2-approximation of the k-centers can be carried out in time $\Theta(k|V|)$.

We have chosen to use a variant of the Kamada and Kawai method [7] as our local drawing method. The local drawing method considers only a limited neighborhood of each node (e.g., only very "close" vertices).

We now describe the full algorithm:

The Multi-Scale Drawing Algorithm
Goal: Find L, a nice layout of $G(V, E)$

```
1. Compute the all-pairs shortest path length ($d_{V \times V}$)
2. Set up a random layout $L$
3. $k \leftarrow$ Threshold
   /* Threshold = 10 */
4. while $k \leqslant |V|$ do
   4.1 Centers $\leftarrow$ K-Centers($G(V, E)$ ,$k$)
   4.2 LocalLayout ($d_{Centers \times Centers}$, $L(Centers)$)
```



(a) (b)

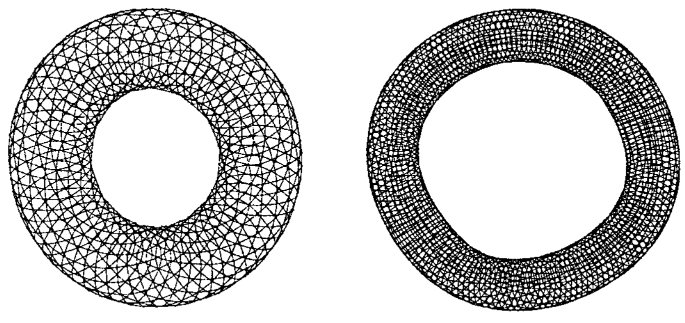
Figure 1: (a) 55x55 (3025-vertex) square grid; (b) the same grid with opposite corners connected

```
4.3 for every $v \in V$ do
    4.3.1 $L(v) \leftarrow L(center(v)) + \xi$
          /* $center(v)$ is the center closest to $v$, $\xi$ is a
          small random noise */
4.4 $k \leftarrow k \cdot Ratio$
    /* $Ratio = 3$ */
5. end
```

4. EXAMPLES

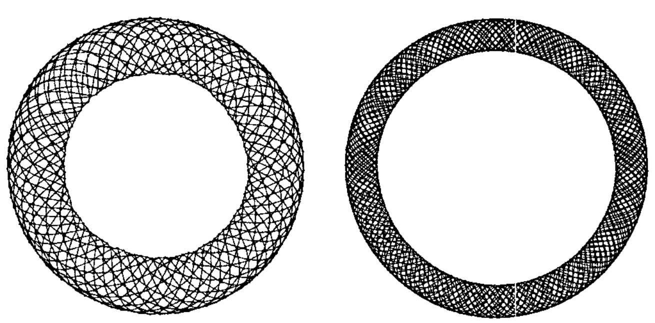
This section contains examples of the results of our algorithm. The implementation is in C++, and runs on a Pentium II 450Mhz PC. Typical execution time with these parameters is about 3sec for 1000-vertex graphs, under 20sec for 3000-vertex graphs, and about 2 minutes for 6000-vertex graphs.

As can be seen the layouts are extremely natural looking; almost "optimal" in aesthetics. The virtually perfect layouts shown in Figures 1–6 are typical of the power of the algorithm. The graphs in Figures 8 and 9 are particularly impressive as the "correct" (grid or torus) appearance is retained despite the partiality of information available to the algorithms.



(a) (b)

Figure 2: Toruses: (a) 64x16 (1024-vertex) (b) 150x20 (3000-vertex)



(a) (b)

Figure 3: Cayley graphs: (a) Z_{1000} with generators ± 9 and ± 11; (b) Z_{3000} with generators ± 11 and ± 13

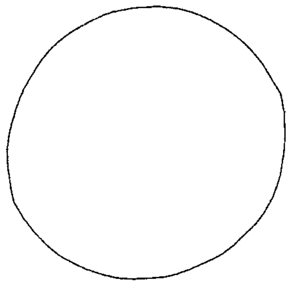


Figure 4: 1000-vertex circle

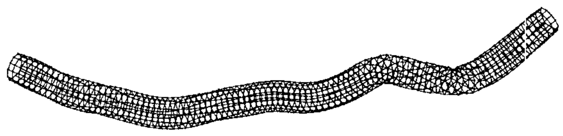


Figure 5: 100x10 (1000-vertex) cylinder

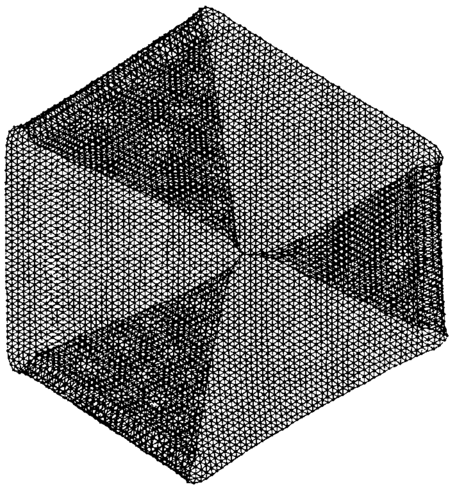


Figure 6: 6105-vertex triangular grid with connected corners

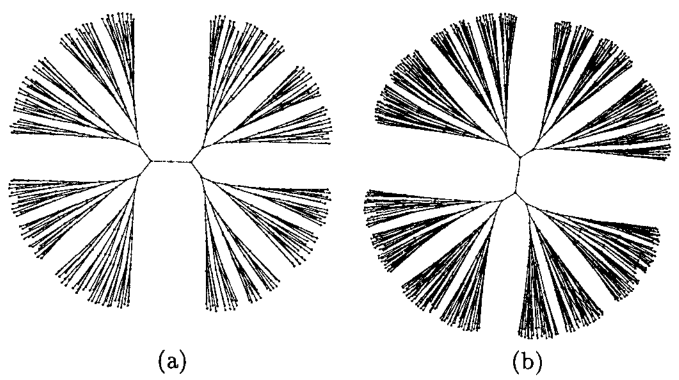


Figure 7: Full binary trees: (a) 511-vertices, depth 8; (b) 1023 vertices, depth 9

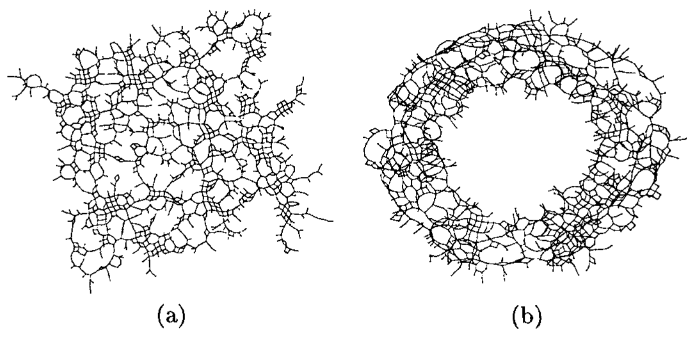


Figure 8: (a) 40x40 (1600-vertex) grid with $\frac{1}{3}$ of the edges omitted at random; (b) 80x20 (1600-vertex) torus with $\frac{1}{3}$ of the edges omitted at random

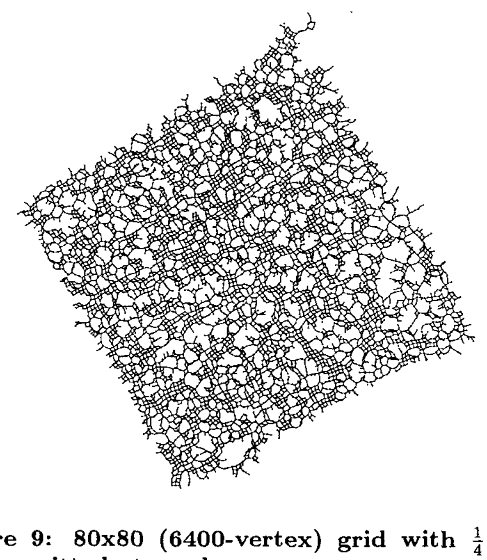


Figure 9: 80x80 (6400-vertex) grid with $\frac{1}{4}$ of the edges omitted at random

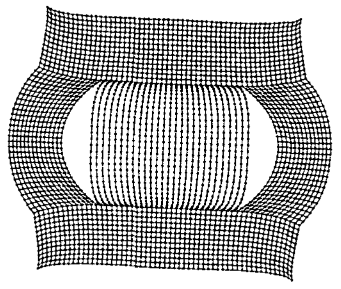


Figure 10: 55x55 (3025-vertex) sparse grid

5. CONCLUSIONS AND FUTURE WORK

We have presented a new multi-scale approach for drawing graphs nicely. Our algorithm is able to deal extremely well and extremely fast with very large graphs.

The algorithm was optimized for speed, and hence we used a simple heuristic for the construction of the coarse scale representation of the graph. For many graphs our method is fine, but for some graphs, especially those with a tiny diameter, the heuristic may not be enough. We are aware of better while more time consuming heuristics, and a new implemntation of the multiscale scheme is underway.

6. REFERENCES

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

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

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

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

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

[6] R. Hadany and D. Harel. A multi-scale method for drawing graphs nicely. In *Proc. 25th Inter. Workshop on Graph-Theoretic Concepts in Computer Science*, 1999.

[7] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(5):7–15, 1989.