GRIP: Graph dRawing with Intelligent Placement*

Pawel Gajer[1] and Stephen G. Kobourov[2]

[1] Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218
[2] Department of Computer Science
University of Arizona
Tucson, AZ 85721

Abstract. This paper describes a system for Graph dRawing with Intelligent Placement, GRIP. The system is designed for drawing large graphs and uses a novel multi-dimensional force-directed method together with fast energy function minimization. The system allows for drawing graphs with tens of thousands of vertices in under a minute on a mid-range PC. To the best of the authors' knowledge GRIP surpasses the fastest previous algorithms. However, speed is not achieved at the expense of quality as the resulting drawings are quite aesthetically pleasing.

1 Introduction

The GRIP system is based on the algorithm of Gajer, Goodrich, and Kobourov [6]. It is written in C++ and OpenGL and uses an adaptive Tcl/Tk interface. Given an abstract graph, GRIP produces a drawings in two and three dimensions either directly or by projecting higher dimensional drawings into 2D or 3D space. GRIP follows a number of force-directed drawing tools [1,3,4,5,7,8] but uses a novel intelligent placement approach to drawing in higher dimensions. A fast energy minimization function combined with the use of a simple vertex filtration allows GRIP to draw graphs with tens of thousands of vertices in under one minute.

An overview of the system and its three main phases is given in Fig. 1. Starting with a graph $G = (V, E)$, we first create a maximal independent set (MIS) filtration $\mathcal{V}: \ V = V_0 \supset V_1 \supset \ldots \supset V_k \supset \emptyset$ of the set V of vertices of G, so that $k = O(\log n)$, and $|V_k| = 3$. A filtration $\mathcal{V}$ of V is called a *maximal independent set filtration* if V_1 is a maximal independent set of G, and each V_i is a maximal subset of V_{i-1} so that the graph distance between any pair of its elements is at least $2^{i-1} + 1$. The *graph distance* between a pair of vertices is defined as the length of the shortest path between them in the graph. Note that the size of a maximal independent set filtration is $\log \delta(G)$, where $\delta(G)$ is the diameter of the graph. We can ensure that the last set has exactly three elements by modifying the last one or two sets in the filtration.

* This research partially supported by NSF under Grant CCR-9625289, and ARO under grant DAAH04-96-1-0013.

J. Marks (Ed.): GD 2000, LNCS 1984, pp. 222–228, 2001.

Fig. 1. An overview of the algorithm. Given a graph G, the algorithm proceeds phases. The first phase creates a MIS filtration. The second and third phases use the filtration sets $V_k, V_{k-1}, \ldots, \mathcal{V}_0$ to repeatedly add more vertices and refine the drawing.

Given a graph G, GRIP produces its drawing in three distinct stages: filtration, initial placement, and refinement. First, we generate an initial embedding of V_k. Since $|V_k| = 3$ the three vertices can be placed in $\mathbb{R}^n$ using their graph distances. Then, we add the vertices of V_{k-1} that are not in V_k, placing them initially at the positions determined by their graph distances to a subset of the elements of V_k. The positions of the vertices in V_{k-1} are modified using a force-directed layout method. This process of adding new vertices and refining their positions is repeated for $V_{k-2}, \ldots, V_1, V_0$. The refined positions of the elements of V_0 constitute the final layout of the vertices of G. Note that we only draw vertices of G up to this point. When all the vertices have been placed we draw the edges of G as straight line segments connecting their endpoints.

2 Building MIS Filtrations

A maximal independent set filtration of a graph G can be obtained by computing the distances between all pairs of vertices of G. A problem with this strategy is that the running time of the all-pairs shortest path algorithm is $\Omega(nm)$ and the storage complexity is $\Omega(n^2)$, e.g., see [2]. When dealing with graphs with tens of thousand of vertices, both the running time and the space complexity of the all-pairs shortest path algorithms pose serious problems.

Our solution is based on the observation that to construct MIS filtration we do not need the distances between all the pairs of vertices. Moreover, the information that has been used to construct $|V_i|$ is not needed to construct $|V_{i+1}|$. Therefore, we use the following "create then destroy" strategy for construction of MIS filtrations. Suppose we have already constructed set V_i. To create V_{i+1}, we build for each vertex of V_i a breadth-first search (BFS) tree up to depth 2^i, but store in it only elements of V_i. Note that this is all we need to build V_{i+1}.

In the process of creating V_{i+1} we may need to build many BFS trees, but we destroy them immediately once they have been used, so that by the time we enter the next phase (of building V_{i+2}), all memory has been freed. Note that as i decreases, the number of vertices for which we have to perform a BFS calculation increases, but at the same time the depth to which we have to build these BFS trees decreases as well. The storage required for this strategy is $\max_i \sum_{v \in V_i} |\mathtt{bfs}_{2^i}(v, V_i)|$, where $|\mathtt{bfs}_{2^i}(v, V_i)|$ is the number of elements of V_i that belong to the BFS tree of v of depth 2^i. The time complexity for this strategy in the case of bounded degree graph G is $\Theta(\sum_{i=0}^{k} \sum_{v \in V_i} |\mathtt{bfs}_{2^i}(v)|)$, where $|\mathtt{bfs}_{2^i}(v)|$ is the number of the BFS tree of v of depth 2^i. Clearly, if we

build a complete BFS tree for each vertex of G, then the running time and space complexity of this procedure even in the bounded degree case would be $O(n^2)$. Our tests indicate that the running time and storage cost of the above MIS filtration construction procedure (in the bounded degree case) is less than quadratic as we only construct partial BFS trees and destroy them right away. In all of our experiments, the time spent creating the MIS filtration was less than 3% of the total running time, see Fig 4(a) and Fig 4(b), respectively.

We store a MIS filtration of a graph of n vertices in an array `misFlt` of size n so that the first $|V_k|$ entries in the `misFlt` array are the elements of V_k. The next $|V_{k-1}|$ entries in the array are the elements of V_{k-1}, followed by $|V_{k-2}|$ entries in the `misFlt` array of the elements of V_{k-2}, all the way to V_1. To keep track of where one set ends and another one begins we store the indices indicating the borders of different level sets of the filtration in a separate array `misBorder` of size $\log \delta(G)$. Thus the space complexity for storing a MIS filtration is $n + \log \delta(G)$. The same method can be applied to any filtration.

3 Initial Placement and Refinement of V_i

The second and third phase of the algorithm are the placement and refinement stages, respectively. In the i^{th} placement stage, the vertices of set V_i are intelligently placed in $\mathbb{R}^n$. In the i^{th} refinement stage a local force-directed method is used to obtain better positions for the vertices of V_i. After the placement and refinement phases for V_i have been completed, the process is repeated for V_{i-1}, V_{i-2}, all the way to V_1.

Consider the general placement case. Suppose the refinement and placement phases for V_i have been completed and we want to start the placement phase for V_{i-1}. All the vertices in V_i are also in V_{i-1}, since $V_{i-1} \supset V_i$ as defined by the construction of the filtration. Thus we are only concerned with the placement of the vertices in V_{i-1} that are not in V_i. The idea behind the intelligent placement is that every vertex t is placed "close" to its optimal position as determined by the graph distances from t to several already placed vertices. The intuition is that if we can place the vertices close to their optimal positions from the very beginning, then the refinement phases need only a few iterations of a local force-directed method to reach minimal energy state.

For example, the following "three closest to t vertices" strategy starts by setting $\texttt{pos}[t]$ to the barycenter $(\texttt{pos}[u] + \texttt{pos}[v] + \texttt{pos}[w])/3$ of u, v, and w, the three vertices closest to t. This is followed by a force-directed modification of the position vector of t with the energy function E calculated only at the three points u, v, w. This makes the procedure very fast, and in our tests it produced good results, see Fig. 2. More details about the placement algorithm can be found in [6].

While the refinement is calculated using a force-directed method, it is important to note that the forces are calculated locally, see Fig. 3(a). For each level of the filtration V_i, we perform $\texttt{rounds}(i)$ updates of the vertex positions, where $\texttt{rounds}(i)$ is a scheduling function which can be specified at the beginning of the

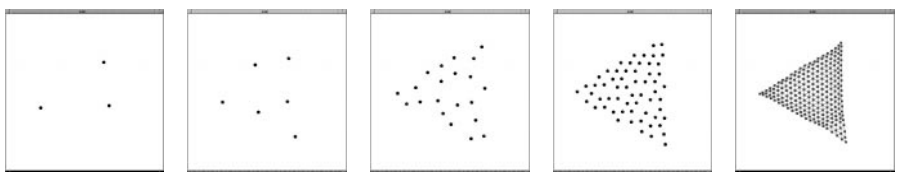

Fig. 2. Drawing of the vertices in the filtration sets. Here $\mathcal{V} : V_0 \supset V_1 \supset V_2 \supset V_3 \supset V_4$. The sizes of the sets are 231, 60, 21, 6, 3, respectively. The process begins with a placement for V_4, followed by V_3, etc. Note that edges are drawn only when all the vertices are placed.

execution. Typically, $5 \leq \texttt{rounds}(i) \leq 30$. At all levels of the filtration except the last one, the displacement vector $\texttt{disp}[v]$ of v is set to a local Kamada-Kawai force vector,

$$\overrightarrow{\texttt{F}}_{\texttt{KK}}(v) = \sum_{u \in N_i(v)} \left(\frac{\texttt{dist}_{\mathbb{R}^n}(u,v)}{\texttt{dist}_G(u,v) \cdot \texttt{edgeLength}^2} - 1 \right) (\texttt{pos}[u] - \texttt{pos}[v]).$$

For the last level of the filtration, $V_0 = V$, when all the vertices have been placed, we set the displacement vector to a local Fruchterman-Reingold force vector,

$$\overrightarrow{\texttt{F}}_{\texttt{FR}}(v) = \sum_{u \in \texttt{Adj}(v)} \frac{\texttt{dist}_{\mathbb{R}^n}(u,v)^2}{\texttt{edgeLength}^2} (\texttt{pos}[u] - \texttt{pos}[v]) + \\ + \sum_{u \in N_i(v)} s \frac{\texttt{edgeLength}^2}{\texttt{dist}_{\mathbb{R}^n}(u,v)^2} (\texttt{pos}[v] - \texttt{pos}[u]),$$

Here, $\texttt{dist}_{\mathbb{R}^n}(u,v)$ is the Euclidean distance between $\texttt{pos}[u]$ and $\texttt{pos}[v]$, and $\texttt{dist}_G(u,v)$ is the graph distance between u and v. In the above equations, $\texttt{edgeLength}$ is the unit edge length, $\texttt{Adj}(v)$ is the set of vertices adjacent to v, and s is a small scaling factor which is set to 0.05 in our program. Note that for a vertex $v \in G$ the force calculation is performed over a restriction $N_i(v)$ of the vertices of G. Each vertex neighborhood $N_i(v)$ contains a constant number of vertices closest to v which belong to V_i. Thus only a constant number of vertices which are near vertex v are used to refine v's position. This is why we call this type of force calculation local.

The local temperature $\texttt{heat}[v]$ of v is a scaling factor of the displacement vector for vertex v. The algorithm for determining the local temperature is in Fig. 3(b). To speed up the calculation, we maintain two auxiliary arrays $\texttt{oldDisp}$ and $\texttt{oldCos}$, where $\texttt{oldDisp}[v]$ is the previous displacement vector for v , and $\texttt{oldCos}[v]$ is the previous value of the cosine of the angle between $\texttt{oldDisp}[v]$ and $\texttt{disp}[v]$. When a displacement vector of v is calculated for the first time, $\texttt{heat}[v]$ is set to a default value $\texttt{edgeLength}/6$. There are three cases for determining the local temperature: (i) if either $\texttt{oldDisp}[v]$ or $\texttt{disp}[v]$ is a zero vector, then the value of $\texttt{heat}[v]$ does not change; (ii) if v is oscillating around some stationary point or if it is moving in the same direction we add to it a factor $\texttt{heat}[v] * (1 + \cos * r * s)$; (iii) in all other cases we add a factor of $\texttt{heat}[v] * (1 + \cos * r)$.

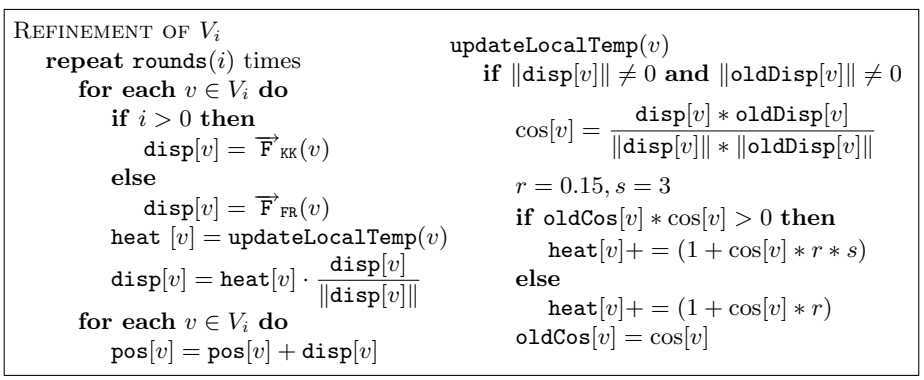

Fig. 3. Pseudocode for (a) the refinement phase and (b) the local temperature calculation.

4 Implementation

The GRIP system is written in C++ with OpenGL and uses a flexible Tcl/Tk interface. The system can generate several typical classes of graphs parametrized by their number of vertices, e.g. paths, cycles, square meshes, triangular meshes, and complete graphs. GRIP also contains generators for complete n-ary trees, random graphs with parametrized density, and knotted triangular and rectangular meshes. Different types of tori, as well as cylinders and Moebius bends can be generated with parametrized thickness and length. Finally, Sierpinski graphs in 2 and 3 dimensions (Sierpinski triangles and Sierpinski pyramids, respectively) are also available. In addition to the set of graphs that GRIP can generate, other graphs can be read from a file in several standard formats. The running times for the MIS creation and for the entire drawing process can be seen in Fig 4(a) and Fig. 4(b), respectively.

The parameters discussed in this paper can be changed via GRIP's flexible interface, thus allowing for experimentation with different scheduling functions, scaling parameters, filtrations, etc. There are controls for the drawing dimension and the drawing speed. The drawings produced by default are three dimensional, interactive, and use color and shading to aid three dimensional perception. For faster drawings the interactive display can be turned off and only the final drawing is shown. The size and colors of the vertices and edges can also be modified. Several drawings produced by GRIP are included in Fig. 5–10.

References

1. I. Bruß and A. Frick. Fast interactive 3-D graph visualization. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes Computer Science*, pages 99–110. Springer-Verlag, 1996.
2. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

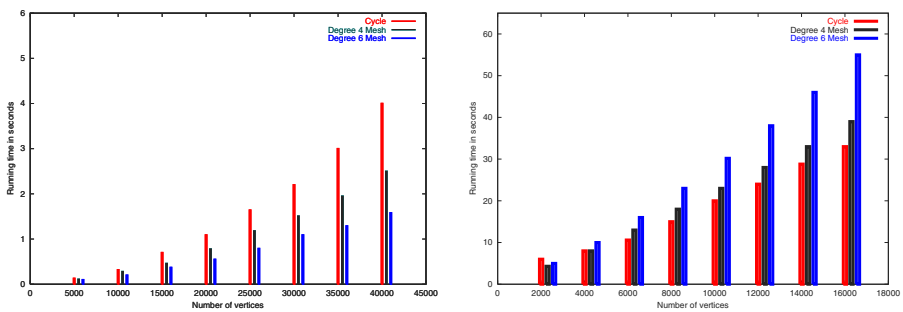

Fig. 4. (a) The left chart shows the running times for construction of a MIS filtration for cycles, and meshes of degree 4 and 6. (b) The right chart shows the total running time for the same graphs.

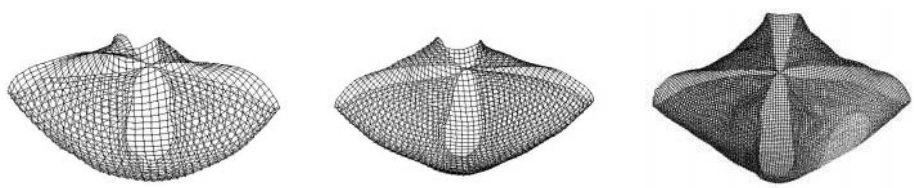

Fig. 5. Knotted rectangular (degree 4) meshes of 1600, 2500, and 10000 vertices.

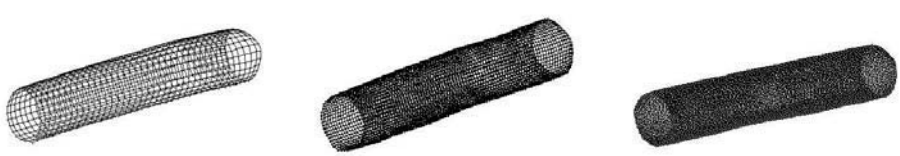

Fig. 6. Cylinders of 1000, 4000, and 10000 vertices.

3. R. Davidson and D. Harel. Drawing graphics nicely using simulated annealing. *ACM Trans. Graph.*, 15(4):301–331, 1996.
4. A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, LNCS 894, pages 388–403, 1995.
5. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Softw. – Pract. Exp.*, 21(11):1129–1164, 1991.
6. P. Gajer, M. T. Goodrich, and S. G. Kobourov. A multi-dimensional approach to force-directed layouts of large graphs. In *To appear in Proceedings of the 8th Symposium on Graph Drawing*, 2000.
7. D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. Technical Report MCS99-21, The Weizmann Institute of Science, Rehovot, Israel, 1999.
8. T. Kamada and S. Kawai. Automatic display of network structures for human understanding. Technical Report 88-007, Department of Information Science, University of Tokyo, 1988.

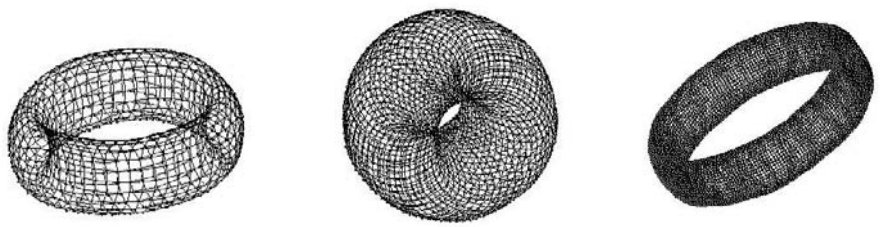

Fig. 7. Tori of various length and thickness: 1000, 2500, and 10000 drawn in four dimensions and projected down to three dimensions.

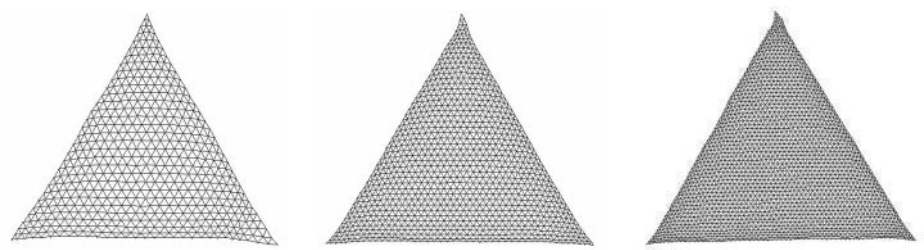

Fig. 8. Triangular (degree 6) meshes of 496, 1035, and 2016 vertices.

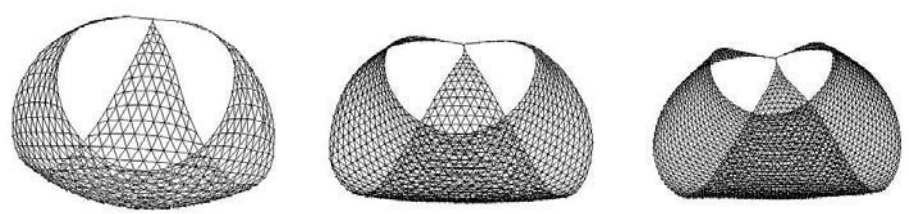

Fig. 9. Knotted triangular (degree 6) meshes of 496, 1035, and 2016 vertices.

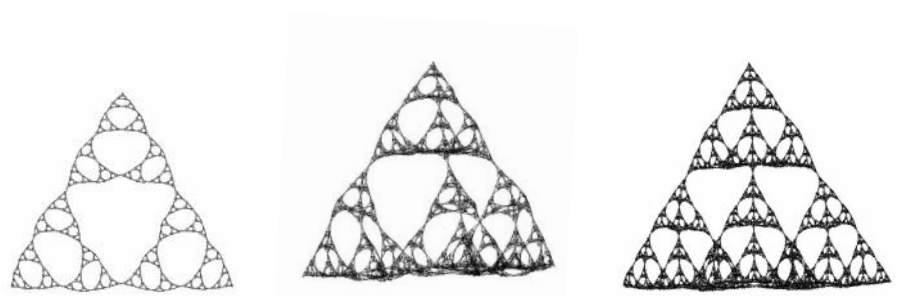

Fig. 10. Sierpinski graphs in 2D and 3D (a) 2D Sierpinski of depth 6 (1095 vertices); (b) 3D Sierpinski of depth 5 (2050 vertices); (c) 3D Sierpinski of depth 6 (8194 vertices).