A Multi-dimensional Approach to Force-Directed Layouts of Large Graphs[⋆]

Pawel Gajer[1], Michael T. Goodrich[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. We present a novel hierarchical force-directed method for drawing large graphs. The algorithm produces a graph embedding in an Euclidean space $\mathbb{E}$ of any dimension. A two or three dimensional drawing of the graph is then obtained by projecting a higher-dimensional embedding into a two or three dimensional subspace of $\mathbb{E}$. Projecting high-dimensional drawings onto two or three dimensions often results in drawings that are "smoother" and more symmetric. Among the other notable features of our approach are the utilization of a maximal independent set filtration of the set of vertices of a graph, a fast energy function minimization strategy, efficient memory management, and an intelligent initial placement of vertices. Our implementation of the algorithm can draw graphs with tens of thousands of vertices using a negligible amount of memory in less than one minute on a mid-range PC.

1 Introduction

Graphs are common in many applications, from data structures to networks, from software engineering to databases. Typically, small graphs are drawn manually so that the resulting picture best shows the underlying relationships. The task of drawing graphs by hand becomes more challenging as the complexity of the graphs increases. Graph drawing tools have been the focus of the graph drawing community for at least the last two decades, see [5,6] for a comprehensive reviews of the field. Numerous algorithms have been developed for drawing special classes of graphs such as trees and planar graphs. There are fewer general purpose graph drawing algorithms, however. Force-directed methods are the methods of choice for drawing general graphs. Substantial interest in force-directed methods stems from their conceptual simplicity, applicability to general graphs, and aesthetically pleasing drawings.

With few exceptions, most existing automated systems have trouble dealing with graphs of thousands of vertices. In this paper we present a new algorithm

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

which allows for drawing simple undirected graphs with tens of thousands of vertices in under a minute. Even larger graphs can be displayed using this algorithm in conjunction with a fisheye view [14,19,22] or a multi-level display algorithm [7] which would allow us to accommodate graphs with more vertices than the number of pixels of the display device. However, the effectiveness of fisheye and multi-level views depends on a good recursive clustering, which in turn depends on a good initial embedding of the graph. Creating a good embedding for large graphs has been prohibitively expensive using existing algorithms. Our algorithm allows us to create excellent initial embeddings in very reasonable times; hence, it can be used either by itself or as a preprocessing step to the above large-graph layout methods. The key features of the algorithm are: (1) intelligent initial placement of vertices; (2) multi-dimensional drawing; (3) a simple recursive coarsening scheme; (4) fast energy function minimization; (5) space and time efficiency.

The rest of this paper is organized as follows: In Section 2 we review previous work in visualization of large graphs and force-directed algorithms for automated graph drawing. In Section 3 we describe our algorithm and in Section 4 we present some concluding remarks.

2 Previous Work

2.1 Visualization of Large Graphs

Visualizing large graphs presents unique problems which require non-orthodox solutions. Drawings that display the entire graph have the advantage of showing global graph structure. For large graphs such drawings become impractical as the limited resolution of display devices makes details hard to discern. Partially drawing graphs allows for display of larger graphs but fails to convey their global structure. Two other approaches to visualization of large graphs are of particular interest: fisheye views and multi-level displays. Fisheye views [14,19, 22] show an area of interest quite large and detailed while showing other areas successively smaller and in less detail. Multi-level views allow us to view large graphs at multiple abstraction levels. A natural realization of such multiple level representations is a 3D drawing with each level drawn on a plane at a different z-coordinate, and with the clustering structure drawn as a tree in 3D.

The multi-level display algorithms are introduced by Eades and Feng [9] in the context of visualization for clustered graphs. Compound and clustered graphs are studied in [10,11,20,23]. Creating a graph clustering based on binary space partitions and using it to display large graphs was introduced by Duncan, Goodrich, and Kobourov [7]. The quality of the resulting multi-level drawings depends on the initial embedding of the graph in the plane.

2.2 Force-Directed Algorithms

The force-directed placement algorithm of Quinn and Breur [21] and the spring embedder of Eades [8] are among the first practical algorithms for graph drawing.

In the latter algorithm the graph is modeled as a physical system of rings and springs. Classical force-directed methods start from a random embedding of a graph and utilize standard optimization methods to find a minimum of an energy function of their choice. The use of an energy function E is a characteristic feature of force-directed layout algorithms. It is used to assign to each embedding $\rho : G \to \mathbb{R}^n$ of a graph G in some Euclidean space $\mathbb{R}^n$ (typically $n = 2$ or $n = 3$) a non-negative number $E(\rho)$. Force-directed methods are based on the premise that minima of reasonably chosen energy functions produce aesthetically pleasing graph drawings. The main differences between force-directed algorithms are in the choice of energy function and the methods for its minimization. Examples of force-directed algorithm include the algorithms of Kamada and Kawai [18], Davidson and Harel [4], Fruchterman and Reingold [13], and Frick *et al* [3,12].

The main problem with most standard force-directed algorithms is their inability to draw large graphs. Even the best classical algorithms can draw graphs with a maximum of only several hundred vertices. When presented with a computationally expensive graph algorithm, a standard approach is to associate with the graph a hierarchy of graphs. The needed computation is done starting with the smallest graph in the hierarchy, then proceeding to larger and larger graphs and using at each stage the results of the previous computation. This strategy has been brought to the area of force-directed graph drawing from particle physics [1,2] in the multi-scale algorithm of Hadany and Harel [16]. In [17] Harel and Koren introduce several simplifications to the algorithm resulting in faster drawings and allowing for larger graphs.

However, as one of the underlying steps of the algorithm in [17], all-pairs shortest paths are computed, which is both time and space expensive. The quadratic space complexity incurred by the matrix of distances between vertices of the graph is another problem when we draw large graphs. Other computationally expensive procedures include the clustering procedure for a construction of a hierarchy of graphs and the Newton-Raphson optimization method for scaling the displacement vectors. Finally, the algorithm in [17] creates drawings in 2D and as it is based on the Newton-Raphson method, extending it to 3D considerably slows down the algorithm. The algorithm described in the next section addresses the above problems and introduces several new features.

3 The Algorithm

3.1 Algorithm Overview

The pseudo-code for the algorithm can be seen in Fig. 1. In the first stage we create a filtration of the set of vertices of the given graph and set up the scheduling function `nbrs`(), described in Sections 3.2 and 3.3, respectively. The main for-loop runs through all levels of the filtration, starting at V_k. At stage i for each vertex $v \in V_i - V_{i+1}$ we find sets $N_i(v), N_{i-1}(v), \ldots, N_0(v)$ and find an initial position `pos`$[v]$ of v. The vertex neighborhood $N_i(v)$ is a set of `nbrs`(i) closest to v elements of V_i. The method for determining $N_i(v), N_{i-1}(v), \ldots, N_0(v)$ and for determining the initial positions are in Sections 3.3 and 3.4, respectively.

```
Main Algorithm
    create a filtration $\mathcal{V}: V_0 \supset V_1 \supset \ldots \supset V_k \supset \emptyset$
    set up scheduling function nbrs()
    for $i = k$ to 0 do
        for each $v \in V_i - V_{i+1}$ do
            find vertex neighborhood $N_i(v), N_{i-1}(v), \ldots, N_0(v)$
            find initial position pos[$v$] of $v$
        repeat rounds times
            for each $v \in V_i$ do
                compute local temperature heat[$v$]
                disp[$v$] $\leftarrow$ heat[$v$] $\cdot \overrightarrow{F}_{N_i}(v)$
            for each $v \in V_i$ do
                pos[$v$] $\leftarrow$ pos[$v$] + disp[$v$]
    add all edges $e \in E$
```

Fig. 1. After creating the vertex filtration and setting up the scheduling function the algorithm processes each filtration set, starting with the smallest one. Here `pos`$[v]$ is a point in $\mathbb{R}^n$ corresponding to vertex v and `rounds` is a small constant. In the refinement stage `heat`$[v]$ is scaling factor for the displacement vector `disp`$[v]$, which in turn is computed over a restriction $N_i(v)$ of the vertices of G.

The refinement stage is repeated `rounds` times, where `rounds` is a small constant. Within the refinement stage, the displacement vector `disp`$[v]$ of v is set to a local Kamada-Kawai force vector. Here local means that the force vector $\overrightarrow{F}_{N_i}(v)$ is computed over v's vertex neighborhood $N_i(v)$ rather than over all vertices in G. The displacement vector is scaled by a local temperature factor `heat`$[v]$. In Section 3.5 we describe the process of calculating `heat`$[v]$.

3.2 Vertex Set Filtrations

Faced with the problem of drawing a large graph, it is natural to associate with it a hierarchy of graphs and produce a drawing starting with the smallest graph in the hierarchy, and drawing larger and larger graphs using at each stage the previous drawing. Two important properties of such a hierarchy are its depth and the distribution of vertices. A constant depth hierarchy implies that as we go from one level to the next, more than a constant fraction of the vertices are added and this makes the drawing of the old level insufficient for placement of new vertices. On the other hand, a linear depth hierarchy is too time consuming to traverse. Thus, logarithmic depth is highly desirable. The effectiveness of this scheme is also dependent on the uniformity of the distribution of the vertices at all levels of the hierarchy. The hierarchy of graphs can be thought of as containing different levels of abstraction of the underlying graph. Uniform distribution of the vertices implies more accurate levels of abstraction which in turn implies better drawings on each level.

Hadany and Harel [16] create a hierarchy of graphs based on the cluster number, the degree number, and the homotopic number. Harel and Koren [17]

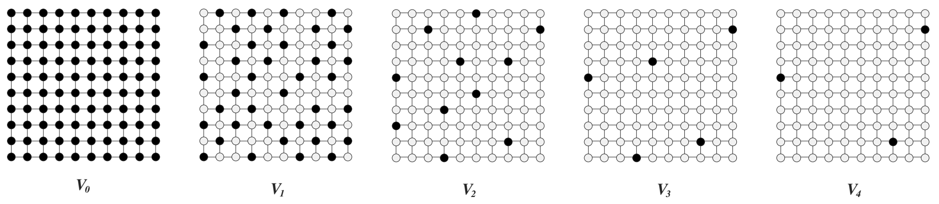

Fig. 2. An example of a MIS filtration. Here the underlying graph $G = (V, E)$ is a rectangular mesh of size 10×10. The dark vertices are included in the filtration. Here $V = V_0$, V_1 is a standard maximal independent set, V_2 is a maximal subset of V_1 so that the distances between its elements are at least $2^1 + 1 = 3$, and so on.

use a simpler method to create the hierarchy of graphs, which relies on a 2-approximation of the k-centers problem. The algorithm of [17] begins by producing a *graph centers (GC) filtration* $V = V_0 \supset V_1 \supset \ldots \supset V_k \supset \emptyset$ of the set V of vertices of the graph G, with $|V_i| = c \cdot x^{k-i}$, where $x > 1$ and $c = |V_k|$ is a constant. A cluster of vertices closest to each center is created for each center and on every level. A set of weighted edges is computed between elements of V_i, so that the weights correspond to the number of edges between the elements of the corresponding clusters. Thus the GC filtration together with the edges forms a hierarchy of graphs.

While having proper graphs on each level is necessary in many applications utilizing graph hierarchies, in the context of graph drawing we can save time and space by using just a filtration of the vertex set. Note that in a filtration there are no edges but only vertices. As we already pointed out, logarithmic depth and "uniform" filtrations are best for graph drawing purposes. We have developed and tested one specific such filtration that we call a *maximal independent set (MIS) filtration* and we use it in this algorithm.

Recall that $S \subset V$ is an *independent set* of a graph $G = (V, E)$ if no two elements of S are connected by an edge of G. Equivalently, S is an independent set of G if the graph distance between any two elements of S is at least two. The *graph distance* between two vertices is defined as the length of the shortest path between them in the graph. A maximal independent set filtration of G is a family of sets $V = V_0 \supset V_1 \supset \ldots \supset V_k \supset \emptyset$, such that each V_i is a maximal subset of V_{i-1} for which the graph distance between any pair of its elements is at least $2^{i-1} + 1$, see Fig. 2.

There are several advantages of MIS filtrations over GC filtrations. First, the number of vertices in the sets V_is in the case of MIS filtrations is controlled by the geometry of the graph, whereas in the graph centers filtration the sizes are arbitrarily set by the user. Moreover, we can build a MIS filtration using little time and space, whereas graph centers filtrations require knowledge about the distances between all pairs of vertices and the all-pairs shortest path is a serious time and space bottleneck when dealing with large graphs.

MIS filtrations can be constructed as follows. Suppose we constructed an order i independent set V_i of G. To construct V_{i+1} let $V^* = V_i$ be an auxiliary set of vertices from which we will draw elements of V_{i+1}. Take a random element $v_0 \in V^*$ out of V^*, and place it in V_{i+1}. Next remove all elements of V^* whose graph distance to v_0 is less than or equal to 2^i. This distance factor is important in ensuring that vertices are well distributed and in guaranteeing small depth of the filtration. Choose another element v_1 of V^*, and remove from V^* the chosen vertex and all vertices whose distance to v_1 is less than or equal to 2^i. Place v_1 in V_{i+1}. Repeat this procedure until V^* is empty. Note that the set V_1 produced by this procedure is an ordinary maximal independent set of G. An example of a maximal independent set filtration is shown in Fig. 2.

The construction of a MIS filtration stops at level k so that $2^k > \delta(G)$, where $\delta(G)$ is the diameter of G. Therefore, each MIS filtration has depth $O(\log \delta(G))$. MIS filtrations provide excellent distribution of the vertices by construction, a property needed for high quality filtrations.

3.3 Finding Vertex Neighborhoods $N_i(v)$

One of the key ideas of the hierarchical force-directed graph layout method is that at each stage of the construction a force-directed position refinement method is applied to a given layer V_i of a filtration only locally. More precisely, for a given energy function E and $v \in V_i$, the gradient of E at $\texttt{pos}[v]$ is computed not for E but for the restriction of E to some neighborhood $N_i(v)$ of v in V_i. Utilization of a good filtration of V and a local position refinement strategy are the key means of escaping a quadratic lower bound for space and time complexity of the classical force-directed methods.

This section describes a procedure of constructing $N_i(v)$ sets and the definition of the scheduling function $\texttt{nbrs}()$. Intuitively, at each stage of the hierarchical graph drawing strategy we should be getting a better and better approximation of the final drawing of the graph. Ideally, at the last stage, when we perform a force-directed local refinement of the position of each vertex v of the graph, it should be enough to take $N_0(v)$ to be the set of adjacent vertices of v. The time complexity of this last stage calculation is $c \cdot \sum_{v \in V} N_0(v) = c \cdot n \cdot \texttt{avgDeg}(G)$, where $\texttt{avgDeg}(G)$ is the average degree of G. We would like to make $c \cdot n \cdot \texttt{avgDeg}(G)$ an upper bound for the complexity of calculations at each stage of graph drawing construction. Therefore, we set $\texttt{nbrs}(i) = \Theta(\frac{\texttt{avgDeg}(G) \cdot n}{|V_i|})$.

Suppose $\mathcal{V}$ is a logarithmic depth filtration of the set V of vertices of G. The calculation of the sets $N_i(v), N_{i-1}(v), \ldots, N_0(v)$ is performed for each element $v \in V$ only once, when it is added to a set of already placed vertices, see Fig. 1. We require that $N_k(v)$ contains $\Theta(\texttt{nbrs}(k))$ elements for each $k = i, i-1, \ldots, 0$. Therefore, the space complexity of this strategy is bounded above by

$$\sum_{i=0}^{k} |V_i - V_{i+1}| \left(\texttt{nbrs}(1) + \texttt{nbrs}(2) + \cdots + \texttt{nbrs}(i)\right). \qquad (1)$$

Since $V_{i+1} \subset V_i$, we have $|V_i - V_{i+1}| = |V_i| - |V_{i+1}|$, and after simplifications (1) takes the form

$$\sum_{i=0}^{k} |V_i|\texttt{nbrs}(i) \leq c_0 \sum_{i=0}^{k} |V_i| \frac{\texttt{avgDeg}(G) \cdot n}{|V_i|} = c_0 \sum_{i=0}^{k} \texttt{avgDeg}(G) \cdot n = $$
$$= c_0 \texttt{avgDeg}(G) \cdot (k+1)n. \tag{2}$$

Similarly we can show that there exists a positive constant c_1 so that equation (1) is greater than $c_1\texttt{avgDeg}(G) \cdot (k+1)n$. Thus, the storage complexity of the above strategy for finding $N_i(v), N_{i-1}(v), \ldots, N_0(v)$ for all $v \in V$ is $\Theta(\texttt{avgDeg}(G)kn)$. If G is of bounded degree, then $\Theta(\texttt{avgDeg}(G)kn) = \Theta(kn)$, where $k = \log n$ for a GC filtration, and $k = \log \delta(G)$ for a MIS filtration.

Let the *depth of a vertex*, $\texttt{depth}(v)$, with respect to $\mathcal{V}$ be the largest d, such that $v \in V_d$. The sets $N_i(v), N_{i-1}(v), \ldots, N_0(v)$ are created by repeated application of a breadth-first search algorithm. A new vertex with depth d is placed in each of $N_j(v)$, for $j \leq d$, if $N_j(v)$ is not full already. The process stops when all $N_j(v)$s are full. Note that the running time of this procedure is bounded above by

$$\sum_{i=1}^{k} |V_i|(1 \cdot \texttt{nbrs}(1) + 2 \cdot \texttt{nbrs}(2) + \cdots + i \cdot \texttt{nbrs}(i)). \tag{3}$$

As in the case of the expression (1), (3) is equal to

$$\sum_{i=0}^{k} i|V_i|\texttt{nbrs}(i) \leq c_0 \sum_{i=0}^{k} i|V_i| \frac{\texttt{avgDeg}(G) \cdot n}{|V_i|} = c_0 \sum_{i=0}^{k} i\texttt{avgDeg}(G) \cdot n = $$
$$= c_0 \texttt{avgDeg}(G) \cdot \frac{(k+1)k}{2} n. \tag{4}$$

Similarly we can show that there exists a positive constant c_1 so that equation (3) is greater than $c_1\texttt{avgDeg}(G) \cdot \frac{(k+1)k}{2} n$. The time complexity of this strategy for finding $N_i(v), N_{i-1}(v), \ldots, N_0(v)$ for all $v \in V$ is $\Theta(\texttt{avgDeg}(G)k^2n)$. If G is of bounded degree, then $\Theta(\texttt{avgDeg}(G)k^2n) = \Theta(k^2n)$, where $k = \log n$ for a GC filtration, and $k = \log \delta(G)$ for a MIS filtration.

3.4 Initial Placement of Vertices

Most graph drawing algorithms begin by placing all the vertices of the graph randomly in the plane or in 3D. In this algorithm we have adopted a different approach in that we add vertices to the current drawing one at a time and only after we have found a suitable place for them. Here we describe the process in two dimensional space, but in practice it can be done in any Euclidean space $\mathbb{E}$. Recall that in the first step of the algorithm we compute a filtration $\mathcal{V} = V_0 \supset V_1 \supset \ldots \supset V_k \supset \emptyset$. If necessary, we modify the last one or two sets of the filtration so that the last one has exactly three elements, $V_k = \{u, v, w\}$.

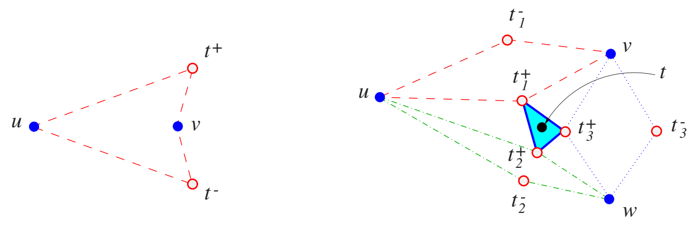

Fig. 3. Initial placement for a new vertex t; darkly shaded vertices have already been placed. (a) Given two vertices in $\mathbb{R}^2$, there are up to two possible places for t, based on its graph distance to u and v. (b) Using three vertices in $\mathbb{R}^2$ results in a better placement.

We start the process of drawing G by placing u, v, and w as follows: we find a triangle with endpoints given by $\texttt{pos}[u], \texttt{pos}[v], \texttt{pos}[w]$, so that $\texttt{d}_{\mathbb{R}^2}(u,v) = \texttt{d}_G(u,v)$, $\texttt{d}_{\mathbb{R}^2}(v,w) = \texttt{d}_G(v,w)$, $\texttt{d}_{\mathbb{R}^2}(w,u) = \texttt{d}_G(w,u)$, where $\texttt{d}_{\mathbb{R}^2}(u,v)$ is the Euclidean distance between $\texttt{pos}[u]$ and $\texttt{pos}[v]$, and $\texttt{d}_G(u,v)$ is the graph distance between u and v.

In general, after refining the positions of the vertices in V_i, we need to find initial positions for the vertices in $V_{i-1} - V_i$. Once all vertices in V_{i-1} are placed their positions are refined, and we proceed to the next level. This two-stage process continues until all vertices have been drawn. A natural way to place a new vertex given the placement of several others is to use the graph distance from the new vertex to several of its closest neighbors that have already been placed. We base our placement strategy on this simple idea.

Suppose that we are looking for a place for a new vertex $t \in V_{i-1} - V_i$. Furthermore, suppose that we know two vertices $u, v \in V_i$ which have already been placed. Then using their position vectors, $\texttt{pos}[u]$ and $\texttt{pos}[v]$, and the graph distances $\texttt{d}_G(u,t)$ and $\texttt{d}_G(v,t)$, it is straightforward to find a position $\texttt{pos}[t]$ of t in the plane so that $\texttt{d}_{\mathbb{R}^2}(u,t) = \texttt{d}_G(u,t)$, $\texttt{d}_{\mathbb{R}^2}(v,t) = \texttt{d}_G(v,t)$, as shown in Fig. 3(a). This idea can be generalized so that three or more already placed vertices are used to determine the location of new vertices. For each vertex $t \in V_{i-1} - V_i$ we find its three closest neighbors $u, v, w \in V_i$, see Fig. 3(b). Since u, v and w have already been placed we can obtain a suitable place for t by solving the following system of equations for u, v, w, and t

$$\begin{cases} (x - x_u)^2 + (y - y_u)^2 = \texttt{d}_G(u,t)^2 \\ (x - x_v)^2 + (y - y_v)^2 = \texttt{d}_G(v,t)^2 \\ (x - x_w)^2 + (y - y_w)^2 = \texttt{d}_G(w,t)^2, \end{cases}$$

where $\texttt{pos}[u] = (x_u, y_u)$, $\texttt{pos}[v] = (x_v, y_v)$, $\texttt{pos}[w] = (x_w, y_w)$, $\texttt{pos}[t] = (x, y)$. Since this system of equations is over-determined and may not have any solutions, we solve the following three pairs of equations instead:

$$\begin{cases} \texttt{d}_{\mathbb{R}^2}(u,t) = \texttt{d}_G(u,t) \\ \texttt{d}_{\mathbb{R}^2}(v,t) = \texttt{d}_G(v,t) \end{cases} \qquad \begin{cases} \texttt{d}_{\mathbb{R}^2}(v,t) = \texttt{d}_G(v,t) \\ \texttt{d}_{\mathbb{R}^2}(w,t) = \texttt{d}_G(w,t) \end{cases} \qquad \begin{cases} \texttt{d}_{\mathbb{R}^2}(u,t) = \texttt{d}_G(u,t) \\ \texttt{d}_{\mathbb{R}^2}(w,t) = \texttt{d}_G(w,t) \end{cases}$$

Solving these three systems of quadratic equations we obtain up to six different solutions. We choose the three closest to each other, call them t_1^+, t_2^+, t_3^+, and place t are their barycenter: $\texttt{pos}[t] = (t_1^+ + t_2^+ + t_3^+)/3$, see Fig. 3(b).

3.5 Local Temperature Calculations

A common problem with most force-directed algorithms is determining the scaling factor of the displacement vector at each phase. Clearly, in the early iterations vertices should move farther than in the last iteration, but coming up with a schedule for scaling the displacement vector that works well for most graphs is generally difficult. One of the reasons for this difficulty is that initially the vertices are placed at random and as a result can be arbitrarily far from their final position. As a result of the intelligent placement of vertices in our algorithms, this is much less of a problem. The local temperature $\texttt{heat}[v]$ of v is simply a scaling factor of the displacement vector $\texttt{disp}[v]$ of v. One particular implementation is considered in detail in [15] but regardless of the specifics of the implementation, the time complexity for updating the local temperature for each v is constant and thus the total time complexity for local temperature calculations is linear.

3.6 Multi-dimensional Drawing

One of the major advantages of a simple local temperature calculation is that unlike the Newton-Raphson and the majority of other classical optimization methods, it works with minor changes in any dimension. In order to obtain an embedding of a graph in $\mathbb{R}^n$, we can simply make $\texttt{pos}[v]$ an n dimensional vector. A problem with drawings in dimensions higher than three is that they cannot be trivially displayed. An obvious solution to this problem is to find a projection from $\mathbb{R}^n$ into $\mathbb{R}^3$ or $\mathbb{R}^2$.

Consider the case in which a four dimensional drawing is projected down to three dimensions. The projection method described below generalizes to higher dimensions as well. We begin by taking a random vector e_0' in $\mathbb{R}^4$ and normalizing it $\mathsf{e}_0 = \frac{\mathsf{e}_0'}{\|\mathsf{e}_0'\|}$. Next we find three vectors $\mathsf{e}_1', \mathsf{e}_2', \mathsf{e}_3' \in \mathbb{R}^4$ so that $\mathsf{e}_0, \mathsf{e}_1', \mathsf{e}_2', \mathsf{e}_3'$ are linearly independent in $\mathbb{R}^4$. We find these vectors by repeatedly choosing a random vector and checking if it is independent from the previous ones until we have four vectors. We then use the Gram-Schmidt orthogonalization process to produce an orthonormal basis $\mathsf{e}_0, \mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_3$ of $\mathbb{R}^4$ using $\mathsf{e}_0, \mathsf{e}_1', \mathsf{e}_2', \mathsf{e}_3'$. The three vectors $\mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_3$ span a 3 dimensional subspace S of $\mathbb{R}^4$ which is perpendicular to the vector e_0. The orthogonal projection $\rho : \mathbb{R}^4 \to S$ from $\mathbb{R}^4$ onto S in the direction of the vector e_0 is given by the formula $\rho(v) = v - (\mathsf{e}_0, v) * \mathsf{e}_0$, where (e_0, v) is the scalar product between e_0 and v. Yet to display v on the screen using OpenGL, we need the coordinates (v_1, v_2, v_3) of the projection $\rho(v)$ of v onto S with respect to the basis vectors $\mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_3$. We get these by a simple scalar product calculation $v_1 = (\mathsf{e}_1, v)$, $v_2 = (\mathsf{e}_2, v)$, $v_3 = (\mathsf{e}_3, v)$.

The above procedure easily generalizes to higher dimensions. Our experiments with 4D drawings yield better results than regular three dimensional drawings. In particular, note the problems with the drawings of the Moebius bend

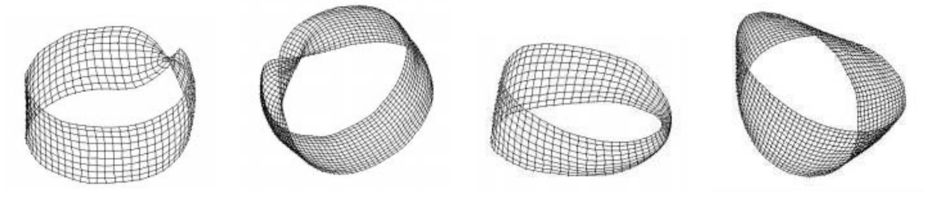

Fig. 4. (a-b) Moebius bands on 600 and 1500 vertices drawn in 3D. Note the rough twists.(c-d) The same graphs but drawn in 4D and projected in 3D.

directly in 3D in Fig. 4(a-b) and the improved drawings when the same graphs are drawn in 4D and projected to 3D in Fig. 4(c-d).

3.7 Space and Time Complexity

Main Theorem. *If G is a graph of bounded degree and $\mathcal{V}$ is a GC filtration or a MIS filtration of the set V of vertices of G, then the time complexity of our algorithm, after constructing $\mathcal{V}$, is $\Theta(n \cdot k^2)$ and the space required is $\Theta(n \cdot k)$, where $k = \log n$ if $\mathcal{V}$ is a GC filtration, and $k = \log \delta(G)$ if $\mathcal{V}$ is a MIS filtration.*

Proof. The proof of the theorem follows from the fact that after building a filtration $\mathcal{V}$, all parts of the algorithm take linear time and space, except the procedure for finding $N_i(v), N_{i-1}(v), \ldots, N_0(v)$ for each element v of V. Thus both time and space complexity of the algorithm is determined by the time and space complexity of the procedure for finding $N_i(v)$s. In Section 3.3, we showed that the time required for finding the sets $N_i(v)$ is $\Theta(n \cdot k^2)$ and the space required is $\Theta(n \cdot k)$, which concludes the proof.

4 Conclusion

We have presented a novel algorithm for drawing large graphs. The algorithm employs a vertex filtration together with intelligent placement of vertices and fast energy minimization. The algorithm produces drawings in two, three, and higher dimensions in sub-quadratic time and space. While the algorithm works very well for sparse graphs and graphs of low degree, it does not produce high quality drawings for all graphs. In particular, well-connected graphs pose significant challenges as the vertex filtrations become very shallow.

References

1. A. Brandt. Multilevel computations of integral transforms and particle interactions with oscillatory kernels. *Computer Physics Communications*, 65:24–38, 1991.
2. A. Brandt. Multigrid methods in lattice field computations. *Nucl. Phys. B*, 26:137–180, 1992. Proc. Suppl.
3. I. Bruß and A. Frick. Fast interactive 3-D graph visualization. In *Graph Drawing (Proc. GD '95)*, pages 99–110, 1995.

4. R. Davidson and D. Harel. Drawing graphics nicely using simulated annealing. *ACM Trans. Graph.*, 15(4):301–331, 1996.
5. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry: Theory and Applications*, 4:235–282, 1994.
6. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, Englewood Cliffs, NJ, 1999.
7. C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Balanced aspect ratio trees and their use for drawing very large graphs. In *Proceedings of the 6th Symposium on Graph Drawing*, pages 111–124, 1998.
8. P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
9. P. Eades and Q. Feng. Multilevel visualization of clustered graphs. In *Proceedings of the 4th Symposium on Graph Drawing (GD '96)*, pages 101–112, 1996.
10. P. Eades, Q. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In *Proceedings of the 4th Symposium on Graph Drawing (GD '96)*, pages 113–128, 1996.
11. Q. Feng, R. F. Cohen, and P. Eades. How to draw a planar clustered graph. In *Procs. of the 1st Annual International Conference on Computing and Combinatorics (COCOON '95)*, pages 21–31, 1995.
12. 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.
13. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Softw. – Pract. Exp.*, 21(11):1129–1164, 1991.
14. G. W. Furnas. Generalized fisheye views. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI '86)*, pages 16–23, 1986.
15. P. Gajer and S. G. Kobourov. GRIP: Graph dRawing with Intelligent Placement. In *To appear in Proceedings of the 8th Symposium on Graph Drawing*, 2000.
16. R. Hadany and D. Harel. A multi-scale algorithm for drawing graphs nicely. In *Proc. 25th International Workshop on Graph Teoretic Concepts in Computer Science (WG'99)*, 1999.
17. 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.
18. 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.
19. K. Kaugars, J. Reinfelds, and A. Brazma. A simple algorithm for drawing large graphs on small screens. In *Graph Drawing (GD '94)*, pages 278–281, 1995.
20. S. C. North. Drawing ranked digraphs with recursive clusters. In *Graph Drawing 93, Proceedings of the First International Workshop on Graph Drawing*, Sept. 1993.
21. N. Quinn and M. Breur. A force directed component placement procedure for printed circuit boards. *IEEE Transactions on Circuits and Systems*, CAS-26(6):377–388, 1979.
22. M. Sarkar and M. H. Brown. Graphical fisheye views. *Communications of the ACM*, 37(12):73–84, 1994.
23. K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):876–892, 1991.