Available online at www.sciencedirect.com
SCIENCE DIRECT®

Computational Geometry 29 (2004) 3–18

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

A multi-dimensional approach to force-directed layouts of large graphs ☆

Pawel Gajer [a], Michael T. Goodrich [b], Stephen G. Kobourov [c,*]

[a] *Department of Computer Science, Johns Hopkins University, USA*
[b] *Department of Information and Computer Science, University of California, Irvine CA, USA*
[c] *Department of Computer Science, University of Arizona, USA*

Available online 18 May 2004

Communicated by I. Streinu

Abstract

We present a novel hierarchical force-directed method for drawing large graphs. Given a graph $G = (V, E)$, the algorithm produces an embedding for G 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 $\mathbb{E}$. Such projections typically result in drawings that are "smoother" and more symmetric than direct drawings in 2D and 3D.

In order to obtain fast placement of the vertices of the graph our algorithm employs a multi-scale technique based on a maximal independent set filtration of vertices of the graph. While most existing force-directed algorithms begin with an initial random placement of all the vertices, our algorithm attempts to place vertices "intelligently", close to their final positions. Other notable features of our approach include a fast energy function minimization strategy and efficient memory management. 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 550 MHz Pentium PC.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Large graph drawing; Multi-scale method; High-dimensional embedding; Force-directed method

1. Introduction

Graphs are common in many applications, from compilers to networks, from software engineering to databases. Typically, small graphs are drawn manually so that the resulting picture best shows

☆ This research partially supported by NSF under Grant CCR-9625289, and ARO under grant DAAH04-96-1-0013. A preliminary version of this paper appeared in the Proceedings of the 8th Annual Symposium on Graph Drawing, 2000.
* Corresponding author.
E-mail address: kobourov@cs.arizona.edu (S.G. Kobourov).

0925-7721/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comgeo.2004.03.014

the underlying relationships. The task of drawing graphs by hand becomes more challenging as the complexity and size of the graphs increases. Graph drawing tools have been the focus of the graph drawing community for at least the last two decades; see [11,30] for comprehensive reviews of the graph drawing field and [44] for work in information visualization. Numerous algorithms have been developed for drawing special classes of graphs such as trees and planar graphs. There are few general purpose graph drawing algorithms, however. Force-directed methods are often the methods of choice for drawing general graphs. Substantial interest in force-directed methods stems from their conceptual simplicity, applicability to general graphs, and typically aesthetically pleasing results.

Automated graph drawing tools can rarely guarantee optimal drawings. Thus such tools usually attempt to optimize a set of goals which tend to produce nice drawings. Typical goals include small area, even distribution of vertices, minimizing edge crossings, etc. Depending on the application, the goals are ranked in order of importance and often only one or two are used in the drawing algorithm. It is not uncommon that different aesthetic criteria can be contradictory.

With few exceptions, current automated systems cannot deal with graphs of tens of thousands of vertices. Meanwhile, it is common for the graphs to be visualized to have more vertices than the number of pixels on conventional displays. Such massive graphs occur naturally in the many areas such as networking, telecommunications, and databases. The majority of drawing tools attempt to display an entire graph, with each vertex and edge explicitly depicted. This approach is impracticable for large graphs, for example when the number of vertices exceeds the number of pixels on the display device. In the case of very large graphs different techniques are called for.

In this paper we present a new algorithm which can draw simple undirected graphs with tens of thousands of vertices in under a minute. Even larger graphs can be displayed using the `GRIP` system in conjunction with a fisheye view [21,31,40] or the multi-level display algorithms of Eades and Feng [14]. Large graphs can be visualized using clustering based on a binary space partition (BSP) together with either fisheye views or multi-level displays as shown in [12,32]. The BSP-based clustering approach allows for the effective visualization of very large graphs. However, the effectiveness of the above algorithms depends on a good recursive clustering, which in turn depends on a good initial embedding of the graph. Creating a good embedding has been prohibitively expensive using existing algorithms. Our algorithm allows us to create excellent initial embeddings in very reasonable times. The key features of the algorithm are:

- intelligent initial placement of vertices,
- multi-dimensional drawing,
- a simple recursive coarsening scheme,
- fast energy function minimization,
- space and time efficiency.

The rest of this paper is organized as follows. In Section 2 we review some of the previous work in three dimensional drawing, visualization of large graphs, and force-directed algorithms for automated graph drawing. In Section 3 we describe our algorithm and introduce maximal independent set filtrations, intelligent placement of vertices, and multi-dimensional drawing. In Section 4 we discuss possible modifications of the algorithm. Also included are several drawings obtained by the `GRIP` layout system [22], which is based on our algorithm.

2. Previous work

2.1. Drawing in three dimensions

Although the majority of the work in graph drawing is in two dimensional graph layout, there have been several algorithms and tools designed for three dimensional graph drawing. The additional degree of freedom sometimes allows for more natural representations, and there is growing evidence which shows that the human brain can comprehend increasingly complex structures if they are displayed as objects in three dimensional space [45,46]. Existing work in three dimensional (3D) graph drawing algorithms focuses on algorithms for special kinds of graphs, for example the algorithms of Cohen et al. [7]. Drawing general graphs in 3D using the force-directed approach is studied by Fruchterman and Reingold [20], and Monien et al. [34]. Other recent 3D drawing algorithms include Bruß and Frick [5], Cruz and Twarog [9], and Ostry [35].

In the context of orthogonal drawings, 3D point-drawing algorithms were developed by Eades, Symvonis and Whitesides [16] and Papakostas and Tollis [36]. 3D orthogonal box-drawings were studied by Biedl [2] and multi-dimensional orthogonal graph drawings are presented by Wood [47].

2.2. Visualization of large graphs

Visualizing large graphs presents unique problems and requires unorthodox solutions. Drawings that display the entire graph have the advantage of showing the global structure of the graph. 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 [21,31,40] show an area of interest quite large and detailed while showing peripheral areas successively smaller and in less detail. Multi-level views [12,14,17,32] 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.

Multi-level display algorithms are introduced in the context of visualization for clustered graphs by Eades and Feng [14] and Feng [17]. Compound and clustered graphs are studied by Sugiyama and Misue [33,41], by Eades et al. [15], and Feng et al. [18]. The above algorithms assume that the clustering of the graph is given. Creating a graph clustering based on binary space partitions and using it to display large graphs was introduced by Duncan et al.[12] and Kobourov [32]. The quality of the resulting multi-level drawings of [12] often depends on the initial embedding of the graph in the plane. The algorithm presented in this paper 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 these large-graph layout methods. Quigley [38] studies the quality of the abstract graph views obtained from clustering techniques by formally measuring the representational differences between abstract views and the underlying graph.

2.3. Force-directed algorithms

The force-directed placement algorithm of Quinn and Breur [39] and the spring embedder of Eades [13] are among the first practical algorithms for the drawing of general graphs. 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 the energy function of their choice. A characteristic feature of force-directed layout algorithms is the use of a *cost* (or *energy*) function E, which assigns 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.

The energy minimization algorithm of Kamada and Kawai [29] uses the Newton–Raphson method for improved drawings. The simulated annealing method of Davidson and Harel [10] is another flexible force-directed algorithm. Fruchterman and Reingold [20] use a slightly different heuristic which results in a faster algorithm. The algorithms of Bruß and Frick [5] and Frick et al. [19] add the notion of local temperature to further speed up the drawing process. A force-directed method can also be used to draw graphs with node labels as shown by Gansner and North [23].

The classic force-directed algorithm produces excellent results for small graphs. However, the algorithm has two major drawbacks. The first of these drawbacks is that the force-directed algorithm does not scale well with size. Large graphs present a problem for even the best existing graph drawing algorithms because these algorithms generally cannot handle more than about a hundred vertices. For larger graphs, the basic algorithm often fails to arrive to a minimum of the energy function and arithmetic precision also becomes a problem. The second drawback is the poor running time of force-directed algorithms. A typical implementation of a force-directed algorithm runs in phases. In each phase new locations are computed for all the vertices. A phase runs in $O(n^2 + m)$ time, where n is the number of vertices and m the number of edges of the graph. The number of phases is typically linear in the number of vertices or edges leading to overall running time of $O(n^3)$ or $O(n^4)$. This poses serious problems when dealing with graphs of tens of thousands of vertices.

Several new algorithms for drawing large graphs were presented at the 8th Symposium on Graph Drawing. Harel and Koren [27] present a multi-scale scheme that computes a simpler graph hierarchy. Walshaw [43] describes a different multilevel algorithm, based on graph coarsening, refinement of the layout on each layer, and interpolation of the results onto the next level. The n-body simulation method of Quigley and Eades [37] uses the Barnes–Hut [1] hierarchical space decomposition method. An earlier implementation of the Barnes–Hut method was used for graph drawing by Tunkelang in JIGGLE [42]. JIGGLE also temporarily increases the degrees of freedom by starting with layouts in higher dimensional space. A method similar to our intelligent placement is described in the context of incremental drawing by Cohen in [6].

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 performed by 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 [3,4] in the multi-scale algorithm of Hadany and Harel [26]. In [27] Harel and Koren introduce several simplifications to the previous algorithm resulting in faster drawings and allowing for larger graphs. With their beautiful drawings of graphs with 3,000 vertices they mark a new chapter in the area of force-directed graph drawings.

However, as one of the underlying steps of the algorithm in [27], all-pairs shortest paths are computed, which is both time and space expensive. Using a binary heap implementation the all-pairs shortest paths problem can be solved in $O(nm \log n)$ time, and using Fibonacci heaps, in $O(n^2 \log n + nm)$ time, e.g. see [8]. In addition, the quadratic space complexity incurred by the matrix of distances between vertices of the graph also quickly becomes an obstacle for drawing 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 [27] 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 novel features.

3. The algorithm

3.1. Algorithm overview

In the remainder of this paper when we refer to a "graph" we assume a simple, undirected and unweighted graph, unless specified otherwise. The algorithm begins by creating a filtration of the set of vertices V of the graph, $\mathcal{V}$: $V_0 \supset V_1 \supset \cdots \supset V_k \supset \emptyset$. Next the vertices in the smallest filtration set V_k are placed in their initial positions, using their graph distance as an approximation to their optimal Euclidean distance. Here the *graph distance* between two vertices is defined as the length of the shortest path between them in the graph. The current positions are then refined using a small number of force-directed refinement rounds. The same process is repeated with the vertices in $V_{k-1}, V_{k-2}, \ldots, \mathcal{V}_0$. Thus, there are k phases in the main algorithm and each phase has a placement stage and a refinement stage.

The pseudo-code for the algorithm can be seen in Fig. 1. After creating the vertex set filtration the algorithm starts adding a few vertices at a time. The main **for**-loop runs through all levels of the filtration, starting at V_k, and performs placement and refinement stages. At phase i for each vertex $v \in V_i - V_{i+1}$ we find a family of neighborhood sets $N_i(v), N_{i-1}(v), \ldots, N_0(v)$. We use the neighborhood sets to find an initial position `pos`$[v]$ of v. The vertices in $N_i(v)$ are the closest neighbors to v from the set V_i. The graph distances between v and the vertices in $N_i(v)$ are used to determine the placement for v. The exact

```
MAIN ALGORITHM
    create a filtration $\mathcal{V}$: $V_0 \supset V_1 \supset \cdots \supset V_k \supset \emptyset$
    **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[$v$] $\cdot \vec{F}_{N_i}(v)$
            **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.

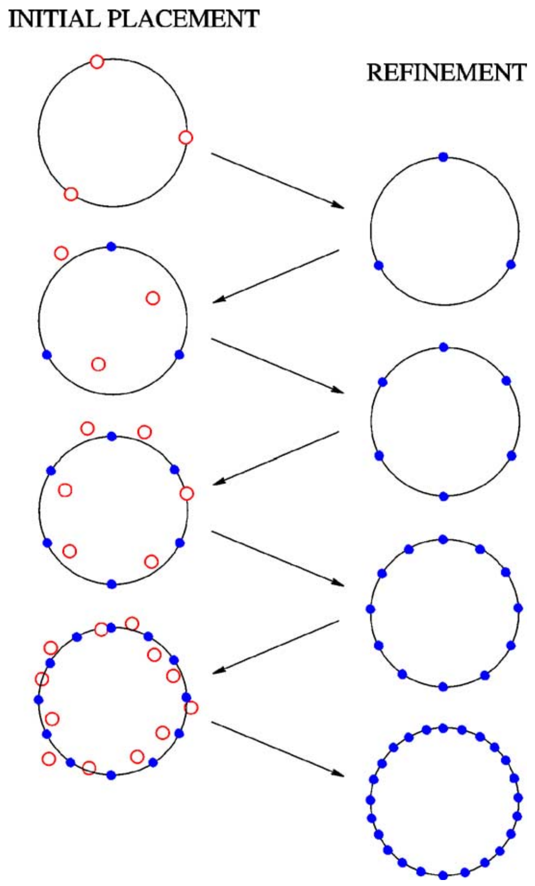

Fig. 2. A hierarchy of coarsened graphs for the cycle of 24 vertices, G. Initially placed vertices are light and vertices that have already been refined are darkly shaded. The left column shows the drawing at the beginning of each phase, just after the new vertices were added. Their initial positions are calculated based on the positions of their nearest neighbors from the previous layer of the filtration $\mathcal{V}$. The right column shows how the positions of newly added vertices change after applying the force-directed local refinement. Note that the vertices in the left column are originally placed "close" to their eventual refined positions.

methods for determining the neighborhood sets $N_i(v), N_{i-1}(v), \ldots, N_0(v)$ and for determining the initial vertex positions are in Sections 3.3 and 3.4, respectively.

The refinement phase is repeated `rounds` times, where `rounds` is a small constant.[1] In the refinement stage vertices are perturbed using a local force-directed algorithm. The displacement vector `disp`$[v]$ of v is set to a local Kamada–Kawai force vector. Here *local* means that the force vector $\vec{F}_{N_i}(v)$ 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`$[v]$. More details about the process of calculating `heat`$[v]$ can be found in Section 3.5. Finally, once all the vertices have been placed, the edges of the graph are added. Fig. 2 illustrates the drawing stages of the algorithm for a cycle graph.

[1] In principle, `rounds` can be set to be a constant function, but our experiments with a linearly increasing function with values from 5 to 15 yield better results.

3.2. Vertex set filtrations

When trying to draw a large graph, it is natural to associate with it a hierarchy of graphs and produce the drawing starting with the smallest graph in the hierarchy and drawing larger and larger graphs so that at each stage we use the previous drawing. Two important properties of such a hierarchy are:

- the depth of the hierarchy (number of levels),
- the distribution of the vertices in the levels.

A shallow hierarchy (e.g. constant depth) implies that as we go from one level to the next, more than a constant fraction of the vertices are added. Usually this means that information from the old level is not sufficient to create a good drawing on the new level. On the other hand, a deep hierarchy (e.g. linear in the number of vertices) is too time consuming to traverse. Thus, logarithmic depth in the number of vertices is highly desirable.

The effectiveness of a multi-scale scheme like this also depends 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 [26] create a hierarchy of graphs based on the cluster number, the degree number, and the homotopic number. Harel and Koren [27] use a simpler method to create the hierarchy of graphs, which relies on the k-clusters problem. Since solving the k-clusters problem or the related k-centers problem is NP-hard [25,28], their algorithm uses a straightforward 2-approximation algorithm for the k-centers problem to create a GC filtration. The algorithm begins by producing a *graph centers* (*GC*) *filtration* $V = V_0 \supset V_1 \supset \cdots \supset V_k \supset \emptyset$ 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.

The creation of a GC filtration proceeds as follows. Pick a random element v of V and add it to V_k. Find a vertex farthest away from all the vertices in V_k and add it to V_k. Continue this process until V_k has c elements. Suppose we have already found V_i, $1 \leqslant i \leqslant k$. To find the next set V_{i-1} let $V_{i-1} = V_i$ and again continue adding to V_{i-1} the elements that are farthest away from V_{i-1} until $|V_{i-1}| = c \cdot x^{k-i+1}$. When V_1 is completed we have a GC filtration of V.

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 highly desirable 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. Recall that the *graph distance* between two vertices is the length of the shortest path between them in the graph. A maximal independent set filtration of G is a family of sets

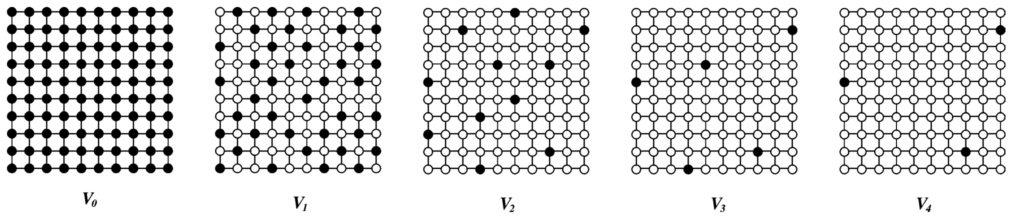

Fig. 3. An example of a MIS filtration. Here the underlying graph $G = (V, E)$ is a rectangular mesh of size 10×10. The dark vertices are included in the filtration. Here $V = V_0$, V_1 is a standard maximal independent set, V_2 is a maximal subset of V_1 so that the distances between its elements are at least $2^2 = 4$, and so on.

$V = V_0 \supset V_1 \supset \cdots \supset V_k \supset \emptyset$, 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 greater than or equal to 2^i; see Fig. 3.

Since the maximum independent set problem is NP-hard [24], we use maximal independent sets instead. Conceptually, MIS filtrations can be constructed as follows. Let $V^* = V$, take a random vertex $v \in V^*$ and add it to V_1. Remove v and all of its neighbors from V^* and repeat until no more vertices can be chosen. Suppose we constructed an order i independent set V_i of G. To construct V_{i+1} let $V^* = V_i$ and take a random vertex $v \in V^*$ out of V^*, and place it in V_{i+1}. Next remove from V^* all vertices whose graph distance to v 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 w of V^*, and remove from V^* the chosen vertex and all vertices whose distance to w is less than or equal to 2^i. Place w in V_{i+1}. Repeat this procedure until V^* is empty. An example of a maximal independent set filtration is shown in Fig. 3.

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.

The three different filtrations considered have their advantages and disadvantages. The number of vertices in the MIS filtration sets is controlled by the topology of the graph, whereas in the graph centers filtration the sizes are arbitrarily set by the user. While we cannot guarantee sub-quadratic time and space, MIS filtrations are faster to create and use very little space compared to GC 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. A good filtration of V and an efficient local position refinement strategy are the key means of achieving a sub-quadratic lower bound for space and time complexity of our method.

This section describes a procedure of constructing the neighborhood sets, $N_i(v)$, and the definition of the function $\texttt{nbrs}(i)$ which determines the size of $N_i(v)$. Intuitively, each stage of the hierarchical graph drawing strategy should result in a closer 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 and c is a constant. 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

$$\texttt{nbrs}(i) = \Theta\left(\frac{\texttt{avgDeg}(G) \cdot n}{|V_i|}\right).$$

Suppose $\mathcal{V}$ is a logarithmic depth filtration of the set V of vertices of G. The calculation of the sets $N_k(v), N_{k-1}(v), \ldots, N_0(v)$ 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 $N_i(v)$ contains $\Theta(\texttt{nbrs}(i))$ elements for each $i = k, k-1, \ldots, 0$. Therefore, the space complexity of this strategy is bounded above by

$$\sum_{i=0}^{k} |V_i - V_{i+1}|\bigl(\texttt{nbrs}(1) + \texttt{nbrs}(2) + \cdots + \texttt{nbrs}(i)\bigr). \tag{1}$$

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

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

Similarly we can show that there exists a positive constant c_1 so that Eq. (1) is greater than $c_1\texttt{avgDeg}(G) \cdot (k+1)n$. Thus, the storage complexity of the above strategy for finding $N_k(v), N_{k-1}(v), \ldots, N_0(v)$ 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 $N_k(v), N_{k-1}(v), \ldots, N_0(v)$ 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 $j \leqslant d$, if $N_j(v)$ is not already full. The process stops when all $N_j(v)$s are full. Note that the running time of this procedure is bounded above by

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

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

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

Similarly we can show that there exists a positive constant c_1 so that Eq. (3) is greater than $c_1\texttt{avgDeg}(G) \cdot \frac{(k+1)k}{2}n$. The time complexity of this strategy for finding $N_k(v), N_{k-1}(v), \ldots, N_0(v)$ 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 $\mathcal{V} = V_0 \supset V_1 \supset \cdots \supset V_k \supset \emptyset$. If necessary, we modify the last one or two sets of the filtration so that the last one has exactly three elements, $V_k = \{u, v, w\}$. We assume that G is connected (if not, each connected component can be drawn independently). 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

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

where $\texttt{dist}_{\mathbb{R}^2}(u,v)$ 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 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 $\texttt{dist}_G(u,t)$ and $\texttt{dist}_G(v,t)$, it is straightforward to find a position $\texttt{pos}[t]$ of t in the plane so that

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

as shown in Fig. 4(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 $u, v, w \in V_i$ via a BFS; see Fig. 4. 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

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

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

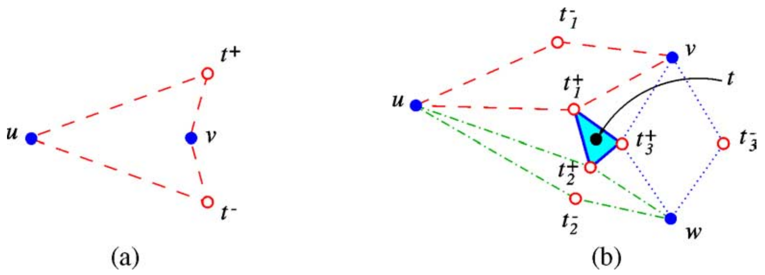

Fig. 4. Initial placement for a new vertex t; darkly shaded vertices have already been placed. (a) Given two vertices in $\mathbb{R}^2$, there are up to two possible places for t, based on its graph distance to u and v. (b) Using three vertices in $\mathbb{R}^2$ in a similar fashion results in a better placement for t.

equations instead

$$\begin{cases} \mathtt{dist}_{\mathbb{R}^2}(u,t) = \mathtt{dist}_G(u,t), \\ \mathtt{dist}_{\mathbb{R}^2}(v,t) = \mathtt{dist}_G(v,t), \\ \mathtt{dist}_{\mathbb{R}^2}(v,t) = \mathtt{dist}_G(v,t), \\ \mathtt{dist}_{\mathbb{R}^2}(w,t) = \mathtt{dist}_G(w,t), \\ \mathtt{dist}_{\mathbb{R}^2}(u,t) = \mathtt{dist}_G(u,t), \\ \mathtt{dist}_{\mathbb{R}^2}(w,t) = \mathtt{dist}_G(w,t). \end{cases}$$

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 at their barycenter: $\mathtt{pos}[t] = (t_1^+ + t_2^+ + t_3^+)/3$; see Fig. 4(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. This approach is similar to that used in the GEM system [5]. The local temperature $\mathtt{heat}[v]$ of v is simply a scaling factor of the displacement vector $\mathtt{disp}[v]$ of v. One particular implementation is considered in detail in [22] 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 $\mathtt{pos}[v]$ 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 $\mathsf{e}_0 = \mathsf{e}'_0/\|\mathsf{e}'_0\|$. Next we find three vectors $\mathsf{e}'_1, \mathsf{e}'_2, \mathsf{e}'_3 \in \mathbb{R}^4$ so that e_0, e'_1, e'_2, 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 e_0, e_1, e_2, e_3 of $\mathbb{R}^4$ using e_0, e'_1, e'_2, e'_3. The three vectors e_1, e_2, e_3 span a three-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 e_1, e_2, 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. For any $m > 3$, we find a projection of $\mathbb{R}^m$ onto some three-dimensional subspace S of $\mathbb{R}^m$ by specifying $m - 3$ linearly independent vectors $\mathsf{e}'_0, \mathsf{e}'_1, \ldots, \mathsf{e}'_{m-4}$ (generalized projection directions), and complete them to a basis $\mathsf{e}'_0, \mathsf{e}'_1, \ldots, \mathsf{e}'_{m-1}$ of $\mathbb{R}^m$. Next, using the Gram–Schmidt orthogonalization process we create an orthonormal basis $\mathsf{e}_0, \mathsf{e}_1, \ldots, \mathsf{e}_{m-1}$ of $\mathbb{R}^m$. The last three vectors e_{m-3}, e_{m-2}, e_{m-1} form a basis of a three-dimensional subspace S of $\mathbb{R}^m$, and the coordinates (v_1, v_2, v_3) of the orthogonal projection of any $v \in \mathbb{R}^m$ onto S are given by the formula $v_1 = (\mathsf{e}_{m-3}, v)$, $v_2 = (\mathsf{e}_{m-2}, v)$, $v_3 = (\mathsf{e}_{m-3}, v)$.

Our experiments with four dimensional drawings yield results that are noticeably different from regular three dimensional drawings. In particular, note the problems with the drawings of the Moebius strip directly in 3D in Fig. 5 and the much better quality drawings of the same graphs drawn in 4D and projected to 3D in Fig. 6.

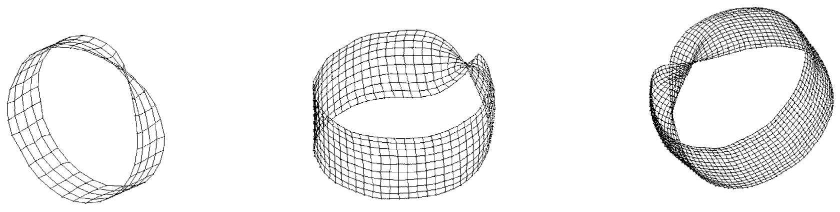

Fig. 5. Moebius strips on 150, 300 and 1500 vertices drawn in directly 3D. Note the rough “twists”.

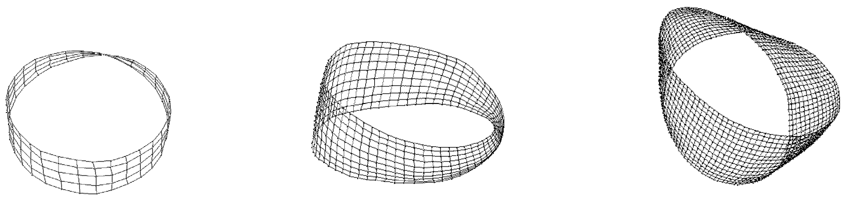

Fig. 6. The same Moebius strips as in Fig. 5 but drawn in 4D and projected in 3D. Note the smooth twists.

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 $N_k(v), N_{k-1}(v), \ldots, N_0(v)$ 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 the neighborhood sets $N_i(v)$. 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. $\square$

4. Conclusion and future work

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. One of the problems that remains to be addressed concerns the running time and space complexity for the creation

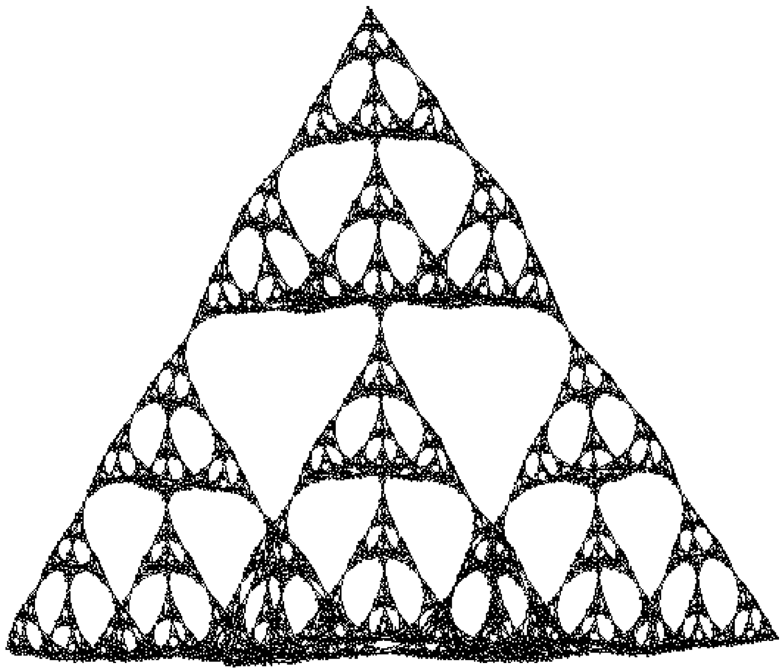

Fig. 7. This drawing of the Sierpinski pyramid was created with the `GRIP` system, which is based on the algorithm described in this paper. The graph contains 8,194 vertices and the drawing took 22 seconds on a 550 MHz Pentium processor.

of the maximal independent set filtration. While our tests indicate that the running time and space required are sub-quadratic in the number of vertices in the graph, this remains to be proved. 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 and graphs with small diameter pose significant challenges as the vertex filtrations become very shallow. Also, our algorithm works best on sparse graphs. While the majority of large graphs that need to be visualized have low average degree, sometimes the maximum degree can be as big as $O(n)$. An algorithm for general graphs with sub-quadratic time and space complexity would be highly desirable.

The algorithm described in this paper is used in the design of the GRIP system (**G**raph d**R**awing with **I**ntelligent **P**lacement) [22] which produced the drawings in Figs. 5 and 6. We include one more example of an interesting class of graphs called Sierpinski graphs. The drawing of the Sierpinski pyramid of the 6th order, which contains 8194 vertices, was produced using GRIP in 22 seconds on a 550 MHz Pentium processor; see Fig. 7.

Acknowledgements

We would like to thank the anonymous referees for the useful comments and suggestions.

References

[1] J. Barnes, P. Hut, A hierarchical $O(N \log N)$ force calculation algorithm, Nature 324 (1986) 446–449. Technical Report.
[2] T.C. Biedl, Three approaches to 3D-orthogonal box-drawings, in: Proceedings of the 6th Symposium on Graph Drawing (GD), in: Lecture Notes in Computer Science, vol. 1547, Springer-Verlag, Berlin, 1998, pp. 30–43.
[3] A. Brandt, Multilevel computations of integral transforms and particle interactions with oscillatory kernels, Comput. Phys. Comm. 65 (1991) 24–38.
[4] A. Brandt, Multigrid methods in lattice field computations, Nucl. Phys. B 26 (1992) 137–180. Proc. Suppl.
[5] I. Bruß, A. Frick, Fast interactive 3-D graph visualization, in: F.J. Brandenburg (Ed.), Proceedings of the 3rd Symposium on Graph Drawing (GD), Lecture Notes Computer Science, vol. 1027, Springer-Verlag, Berlin, 1996, pp. 99–110.
[6] J.D. Cohen, Drawing graphs to convey proximity: An incremental arrangement method, ACM Trans. Computer–Human Interaction 4 (3) (1997) 197–229.
[7] R.F. Cohen, P. Eades, T. Lin, F. Ruskey, Three-dimensional graph drawing, Algorithmica 17 (1997) 199–208.
[8] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.
[9] I.F. Cruz, J.P. Twarog, 3d graph drawing with simulated annealing, in: F.J. Brandenburg (Ed.), Proceedings of the 3rd Symposium on Graph Drawing (GD), Lecture Notes Computer Science, vol. 1027, Springer-Verlag, Berlin, 1996, pp. 162–165.
[10] R. Davidson, D. Harel, Drawing graphics nicely using simulated annealing, ACM Trans. Graph. 15 (4) (1996) 301–331.
[11] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, Graph Drawing: Algorithms for the Visualization of Graphs, Prentice Hall, Englewood Cliffs, NJ, 1999.
[12] C.A. Duncan, M.T. Goodrich, S.G. Kobourov, Balanced aspect ratio trees and their use for drawing large graphs, J. Graph Alg. Appl. 4 (2000) 19–46.
[13] P. Eades, A heuristic for graph drawing, Congr. Numer. 42 (1984) 149–160.
[14] P. Eades, Q. Feng, Multilevel visualization of clustered graphs, in: Proceedings of the 4th Symposium on Graph Drawing (GD), Lecture Notes in Computer Science, vol. 1190, Springer-Verlag, Berlin, 1996, pp. 101–112.
[15] P. Eades, Q. Feng, X. Lin, Straight-line drawing algorithms for hierarchical graphs and clustered graphs, in: Proceedings of the 4th Symposium on Graph Drawing (GD), Lecture Notes in Computer Science, vol. 1190, Springer-Verlag, Berlin, 1996, pp. 113–128.

[16] P. Eades, A. Symvonis, S. Whitesides, Two algorithms for three dimensional orthogonal graph drawing, in: S. North (Ed.), Proceedings of the 4th Symposium on Graph Drawing (GD), Lecture Notes in Computer Science, vol. 1190, Springer-Verlag, Berlin, 1996, pp. 139–154.
[17] Q. Feng, Algorithms for drawing clustered graphs, PhD Thesis, Department of Computer Science and Software Engineering, University of Newcastle, 1997.
[18] Q. Feng, R.F. Cohen, P. Eades, How to draw a planar clustered graph, in: Proc. 1st Annual International Conference on Computing and Combinatorics (COCOON '95), 1995, pp. 21–31.
[19] A. Frick, A. Ludwig, H. Mehldau, A fast adaptive layout algorithm for undirected graphs, in: R. Tamassia, I.G. Tollis (Eds.), Proceedings of the 2nd Symposium on Graph Drawing (GD), Lecture Notes in Computer Science, vol. 894, Springer-Verlag, Berlin, 1995, pp. 388–403.
[20] T. Fruchterman, E. Reingold, Graph drawing by force-directed placement, Softw. – Pract. Exp. 21 (11) (1991) 1129–1164.
[21] G.W. Furnas, Generalized fisheye views, in: Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'86), 1986, pp. 16–23.
[22] P. Gajer, S.G. Kobourov, GRIP: Graph dRawing with Intelligent Placement, J. Graph Alg. Appl. 6 (3) (2002) 203–224.
[23] E.R. Gansner, S.C. North, Improved force-directed layouts, in: Lecture Notes in Computer Science, Lecture Notes in Computer Science, vol. 1547, Springer-Verlag, Berlin, 1998, pp. 364–373.
[24] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[25] T.F. Gonzalez, Clustering to minimize the maximum intercluster distance, Theoret. Comput. Sci. 38 (2–3) (1985) 293–306.
[26] Hadany, D. Harel, A multi-scale algorithm for drawing graphs nicely, DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science 113 (2001).
[27] D. Harel, Y. Koren, A fast multi-scale method for drawing large graphs, in: Proceedings of the 8th Symposium on Graph Drawing (GD), Lecture Notes in Computer Science, vol. 1984, Springer-Verlag, Berlin, 2000, pp. 183–196.
[28] D.S. Hochbaum, D.B. Shmoys, A unified approach to approximate algorithms for bottleneck problems, J. ACM 33 (3) (1986) 533–550.
[29] T. Kamada, S. Kawai, Automatic display of network structures for human understanding, Technical Report 88-007, Dept. of Inf. Science, University of Tokyo, 1988.
[30] M. Kaufmann, D. Wagner, Drawing Graphs: Methods and Models, Lecture Notes in Computer Science, vol. 2025, Springer-Verlag, Berlin, 2001.
[31] K. Kaugars, J. Reinfelds, A. Brazma, A simple algorithm for drawing large graphs on small screens, in: Proceedings of the 3rd Symposium on Graph Drawing (GD), Lecture Notes in Computer Science, vol. 894, Springer-Verlag, Berlin, 1995, pp. 278–281.
[32] S.G. Kobourov, Visualizing large graphs, PhD Thesis, Johns Hopkins University, 2000.
[33] K. Misue, K. Sugiyama, An overview of diagram based idea organizer: D-ABDUCTOR, Research Report IIAS-RR-93-3E, Fujitsu Laboratories Ltd., 1993.
[34] B. Monien, F. Ramme, H. Salmen, A parallel simulated annealing algorithm for generating 3D layouts of undirected graphs, in: Proceedings of the 3rd Symposium on Graph Drawing (GD), Lecture Notes in Computer Science, vol. 1027, Springer-Verlag, Berlin, 1996, pp. 396–408.
[35] D.I. Ostry, Some three-dimensional graph drawing algorithms, Master's Thesis, University of Newcastle, Newcastle, Australia, 1996.
[36] A. Papakostas, I.G. Tollis, Incremental orthogonal graph drawing in three dimensions, in: Proceedings of the 5th Symposium on Graph Drawing (GD), Lecture Notes in Computer Science, vol. 1353, Springer-Verlag, Berlin, 1997, pp. 52–63.
[37] A. Quigley, P. Eades, FADE: graph drawing, clustering, and visual abstraction, in: Proceedings of the 8th Symposium on Graph Drawing (GD), Lecture Notes in Computer Science, vol. 1984, Springer-Verlag, Berlin, 2000, pp. 197–210.
[38] A.J. Quigley, Large scale relational information visualization, clustering, and abstraction, PhD Thesis, University of Newcastle, Australia, 2001.
[39] N. Quinn, M. Breur, A force directed component placement procedure for printed circuit boards, IEEE Trans. Circuits and Systems CAS-26 (6) (1979) 377–388.
[40] M. Sarkar, M.H. Brown, Graphical fisheye views, Comm. ACM 37 (12) (1994) 73–84.
[41] K. Sugiyama, K. Misue, Visualization of structural information: Automatic drawing of compound digraphs, IEEE Trans. Systems Man Cybernet. 21 (4) (1991) 876–892.

[42] D. Tunkelang, JIGGLE: Java interactive general graph layout environment, in: Proceedings of the 6th Symposium on Graph Drawing (GD), Lecture Notes in Computer Science, vol. 1547, Springer-Verlag, Berlin, 1998, pp. 413–422.
[43] C. Walshaw, A multilevel algorithm for force-directed graph drawing, in: Proceedings of the 8th Symposium on Graph Drawing (GD), Lecture Notes in Computer Science, vol. 1984, Springer-Verlag, Berlin, 2000, pp. 171–182.
[44] C. Ware, Information Visualization, Morgan Kaufmann, 2000.
[45] C. Ware, D. Hui, G. Franck, Visualizing object oriented software in three dimensions, in: CASCON 1993 Proceedings, 1993.
[46] C.D. Wickens, Engineering Psychology and Human Performance, Harper Collins, New York, 1991.
[47] D.R. Wood, Multi-dimensional orthogonal graph drawing with small boxes, in: Proceedings of the 7th Symposium on Graph Drawing (GD), Lecture Notes in Computer Science, vol. 1731, Springer-Verlag, Berlin, 1999, pp. 311–322.