Journal of Graph Algorithms and Applications
http://www.cs.brown.edu/publications/jgaa/
vol. 2, no. 1, pp. 1–24 (1998)

Algorithms for Cluster Busting in Anchored Graph Drawing

Kelly A. Lyons
Centre for Advanced Studies
IBM Toronto Laboratory
http://www.cas.ibm.ca/
klyons@ca.ibm.com

Henk Meijer *David Rappaport*
Department of Computing and Information Science
Queen's University
http://www.qucis.queensu.ca/
henk@qucis.queensu.ca daver@qucis.queensu.ca

Abstract

Given a graph G and a drawing or layout of G, it is sometimes desirable to alter or adjust the layout. The challenging aspect of designing *layout adjustment algorithms* is to maintain a user's mental picture of the original layout. We present a new approach to layout adjustment called *cluster busting in anchored graph drawing*. We then give two algorithms as examples of this approach.

The goals of cluster busting in anchored graph drawing are to more evenly distribute the nodes of the graph in a drawing window while maintaining the user's mental picture of the original drawing. We present simple and efficient iterative heuristics to accomplish these goals. We formally define some measures of distribution and similarity and give empirical results based on these measures to quantify our methods. The theoretical analysis of our heuristics presents a formidable challenge, thus justifying our empirical analysis.

Communicated by G. Di Battista: submitted April 1996; revised March 1998.

Research supported in part by an IBM Toronto Laboratory Centre for Advanced Studies PhD Fellowship and by an NSERC of Canada grant.

1 Introduction

A graph $G = (V, E)$ is a set of n nodes $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and a set of edges $E = \{e_{ij} = \{v_i, v_j\} \mid \text{there is an edge between } v_i \in V \text{ and } v_j \in V\}$. The edges can be *directed*, in which case $e_{ij} \neq e_{ji}$ or *undirected*, in which case $e_{ij} = e_{ji}$. The problem of drawing a graph given its combinatorial description G is termed *layout creation* in [25]. Layout creation algorithms try to position the nodes and edges of G such that certain optimization criteria are satisfied.

In the case where graphs with an existing layout must be redrawn, the layouts can have clusters of nodes that are close together, making the labels of the nodes and the interactions between the nodes hard to read and understand. The reapplication of a layout creation algorithm is not useful for layout adjustment, because a new layout that is drastically different from the existing one destroys the user's mental picture of the layout [4, 11, 25]. Given a combinatorial description of a graph G along with an existing drawing D of G, the goal of layout adjustment algorithms is to produce a new drawing D' of G such that the layout can be easily read and such that the user's mental picture of D is not destroyed.

Our results have been applied to the CORDS project at the Centre for Advanced Studies (CAS) at the IBM Software Solutions Toronto Laboratory [3, 13, 14]. The CORDS research project (which ended in 1995) studied the design, development, and management of distributed applications. One of the underlying services provided by the CORDS architecture was the support for visualization.

The following layout adjustment problems were identified as part of the visualization component in the CORDS project. The programming of distributed systems with a vast number of processors and communications links requires a method for visualizing the network as well as the systems running on the network [13]. In particular, there is a requirement to draw graphs of networks in which the nodes represent geographic locations such as cities in a country or buildings on a campus. The geographic locations of the nodes provide meaning to a user who is looking at the layout; therefore, it is important to retain this location information when drawing the graph. Other types of graphs that must be displayed by a visualization system are those that have been created by a user using a graph editor [2]. In this case, the user has put nodes in positions that either have meaning initially or gain importance after the graph has been viewed. After the user has edited the layout, the layout can become hard to read and understand.

In 1993, members of the CORDS team built a prototype to validate the architecture [1], and early versions of the layout adjustment algorithms presented in this paper were used to draw graphs of networks and graphs that were initially drawn manually.

In [10, 11, 25], an approach to layout adjustment called *preserving the mental map* is presented in which some measure of similarity between the node positions in D and the node positions in D' is preserved while trying to separate overlapping nodes.

In [11], several mathematical models of a user's mental map are presented. One such model, called the *cluster* model, uses the notion of the *shape* of a set of points. The idea of using proximity graphs to define the shape of a set of points is used in pattern recognition [31, 32, 33] and suggested in [11] as a good measure of a user's mental map. Proximity graphs of a set of points include the convex hull, the minimum spanning tree, and the sphere of influence graph. The mental map is preserved between two layouts if the proximity graphs defined on the node positions of the two layouts are the same. In [25], these concepts are further explored, algorithms are presented, and some experiences with visualization systems are given.

In this paper, we weaken the requirement that the measure of similarity be preserved. Instead, we try to improve the distribution of the nodes in the new layout according to some measures of distribution (a process we call *cluster busting*) while simultaneously trying to minimize the difference between the two layouts according to some measures of difference (a process we call *anchored graph drawing*). We present two heuristic algorithms for layout adjustment that are based on this approach. Cluster busting in anchored graph drawing was introduced in [21].

A drawing of a graph is denoted by $D = (G, S, W)$, where $G = (V, E)$ is a graph, $S = \{p_0, p_1, \ldots, p_{n-1}\}$ is a set of points, and W is a *window* that contains S. The window is an isothetic rectangle in the plane such that $bl = (x_{bl}, y_{bl})$ is the bottom left corner of W and $tr = (x_{tr}, y_{tr})$ is the top right corner. A drawing $D' = (G, S', W')$ denots a transformation of D where $S' = \{p'_0, p'_1, \ldots, p'_{n-1}\}$ and W' contains S'.

The goal of cluster busting in anchored graph drawing is to produce a new drawing D' of an existing drawing D such that the following criteria are satisfied:

CB1 $W = W'$

CB2 The nodes in D' should be more evenly distributed inside W than the nodes in D,

CB3 The general shape of D' should be the same as D.

These criteria are the same as those presented in [10]. Note that criteria **CB1** and **CB2** do not take the edges of the graph into consideration. It is unclear from the description of criterion **CB3** whether or not the *shape* of the drawing includes the edges of the graph. Our algorithms ignore the edges of the graph when determining the new node positions. The definition of "shape" presented in [10] also excludes the edges of the graph.

In Section 2 we present related work. In Section 3, we present measures of distribution and similarity. Two cluster busting in anchored graph drawing algorithms are presented in Section 4. In Section 5 we present results, and we conclude with future areas of research in Section 6.

2 Related Work

A comprehensive survey of graph drawing algorithms is presented in [8]. Compared to the number of layout creation algorithms available, little work has been done in designing dynamic graph drawing algorithms or layout adjustment algorithms [8]. An early result in dynamic graph drawing is an algorithm for drawing trees that change [27]. More recently, polylogarithmic algorithms that update drawings of trees, special types of digraphs, and general planar graphs are presented in [5].

In [4], user-defined constraints (in the form of linear equations of two variables) are added to an existing layout algorithm to keep drawings stable. Examples of constraints that a user might want to specify include absolute positioning (for example, if nodes are placed on horizontal levels, a node may be constrained to a specific level or a specific position within a level), relative positioning (node p_i is to the left of node p_j), or clusters (grouping nodes together in a cluster that can be further constrained by size and/or position). Constraints are defined on an x, y coordinate system: The fact that node p_i should be vertically above node p_j translates to two equations, $x_i = x_j$, and $y_i < y_j$. A constraint manager maintains a list of all constraints, keeps the set of constraints consistent, and provides functions to add, delete, and query the status of constraints. It is shown how user-defined constraints can be added to many existing layout algorithms.

In [24], *incremental graph layout* allows graphs to be updated without disturbing the existing nodes and edges in the layout. Nodes and edges are added one at a time such that one node cannot overlap another and an edge cannot cross a node. In these layouts, every node is a rectangle and every edge is a sequence of horizontal and vertical segments.

In [10], algorithms are presented that preserve the mental map of the drawing according to the *orthogonal ordering* model of a user's mental map; that is, for each pair of node positions p_i and p_j in D with corresponding positions p'_i and p'_j, respectively, in D':

- $x'_i < x'_j$ if and only if $x_i < x_j$, and
- $y'_i < y'_j$ if and only if $y_i < y_j$

where $p_i = (x_i, y_i)$. Preserving the orthogonal ordering of the nodes while keeping the nodes inside the original-sized display window may not allow enough movement to distribute the nodes so that it can be determined if there are edges between them [22].

As with several other successful methods for drawing general undirected graphs, our algorithms are heuristic; therefore, we present quantitative measures of the layouts that provide evidence that the algorithms satisfy the criteria in practise. In [23, 26, 29, 34], quantitative measures are also used to evaluate layouts produced by heuristic algorithms. For example, a heuristic algorithm that tries to produce drawings with uniform edge lengths, evenly distributed nodes, and minimal edge crossings is presented in [34]. The algorithm finds an initial placement of each node by minimizing a cost function made up of three components representing each of the three criteria. A quantitative measure is defined for each of the criteria, and layouts produced by the algorithm are compared with layouts produced by the algorithms in [6] and [17] based on these measures.

In the following section we define quantitative measurements of layouts for evaluating the cluster busting in anchored graph drawing algorithms presented in Section 4.

3 Measuring Similarity and Distribution

It is straight forward to measure if a drawing is inside a given drawing window; however, quantifying even distribution and similarity are much more difficult. In this section we present one model of distribution and two models of similarity, and define measurements of layouts based on these models. In [22], additional models of distribution and similarity are presented.

3.1 Measuring Distribution

Criterion **CB2** states that the nodes in D' should be more evenly distributed inside W than the nodes in D. In order to judge if the nodes in one drawing are more evenly distributed in W than the nodes in another drawing, we need a method (or methods) for measuring the distribution of points in a given bounded region. We must not only consider the distribution of the nodes relative to each other but also the distance between each node and the boundary of W. We have determined through observation that if the ideal distance between two nodes is d_I, then the ideal distance between a node and the boundary of W is $0.5d_I$.

We call a method for measuring the distribution of points a *model* $\mathcal{E}$ of distribution. Given a model $\mathcal{E}$ of distribution, let the *distribution measure* of the nodes in a drawing $D = (G, S, W)$ according to the model $\mathcal{E}$ be: $\Xi_{\mathcal{E}}(S)$, where $\Xi_{\mathcal{E}}(S) \geq 0.0$ increases as S becomes more evenly distributed under the model $\mathcal{E}$.

An intuitive way of quantifying the distribution of a set of points is to say that a set of points is evenly distributed in a given bounded region if the minimum distance between any two points in the set is maximized; that is, if the distance between a closest pair of points is maximized. We call this model the CP model of distribution and define the measure of distribution according to the CP model as:

$$\Xi_{CP}(S) = \min\{\min_{i \neq j}\{dist(p_i, p_j)\}, \min_{w \in \{tr, bl\}}\{2|x_i - x_w|, 2|y_i - y_w|\}\}$$

where tr and bl are the top right and bottom left corners of W, respectively. It follows that the nodes in a drawing D are more evenly distributed than those in D' under the CP model if $\Xi_{CP}(S) > \Xi_{CP}(S')$.

The cost of computing $\Xi_{CP}(S)$ for a drawing D with n nodes is $O(n)$ time given the Voronoi diagram of the nodes, which can be computed in $O(n \log n)$ time [28].

3.2 Measuring Similarity

Criterion **CB3** states that the general shape of D' should be the same as D. In [11] and [25], several mathematical models of the user's mental map of a drawing D are defined, and D' is said to have the same shape as D if D' and D are the same according to one of the models. Since this condition is relaxed in cluster busting in anchored graph drawing, we must have a way of measuring difference or similarity between D and D' according to a given model. In this section, we present measures of difference between two sets of points based on two models of a user's mental map. Each of the models described in this section quantifies similarity between two sets of node positions and does not consider edges of the input graph. For a given mental map model M, we define $\delta_M(S, S')$ as the number of changes in S' from S under the model M, and let $UB_M(n)$ be an *upper bound* on the number of changes in a drawing with n nodes under the model M. The upper bound is set to the maximum value when the maximum is known; otherwise, the lowest known upper bound is used. The measure of difference according to the model M is given by

$$\mathcal{D}_M(S, S') = \frac{\delta_M(S, S')}{UB_M(n)}$$

In this way, we ensure that $0.0 \leq \mathcal{D}_M(S, S') \leq 1.0$. If $\mathcal{D}_M(S, S') = 0.0$, then there is no difference between the drawings D and D' according to the model M and the transformation from D to D' preserves the mental map under M. We say two drawings D and D' are *not very different* or are *similar* under a model M if $\mathcal{D}_M(S, S')$ is small, and the two drawings are as different as they can be according to measure M if $\mathcal{D}_M(S, S') = 1.0$.

It is important to note that the values of $\mathcal{D}_M(S, S')$ are normalized between 0.0 and 1.0 in order to compare values of $\mathcal{D}_M(S, S')$ for different-sized drawings produced by different layout programs under the same model M. It is inappropriate to compare $\mathcal{D}_{M_1}(S, S')$ and $\mathcal{D}_{M_2}(S, S')$ on the same drawing for different models M_1 and M_2.

The similarity models presented in this section have been chosen based on ideas and results borrowed from the fields of psychology and pattern recognition, and because of their geometric nature, efficiency of computation, and ease of implementation. According to Gregson [19], under the *normative* definition of similarity a declaration is made that a certain operation is being used to measure similarity between two objects but no presupposition is made about how accurate that operation is as an actual judge of similarity.

3.2.1 λ-Matrix (λM) Model

In [18], Goodman and Pollack introduce a process called *geometric sorting*, which defines an ordering on n points in $d > 1$ dimensions. Their idea of sorting generalizes from the one-dimensional case where a set of n numbers is given. Consider these numbers as n points on a horizontal line L. Two sets of points S and S' on a line are in the same order (have the same *order type*) if for every pair of points, $p_i < p_j$ if and only if $p'_i < p'_j$.

According to Goodman and Pollack, two sets of points S and S' in the plane have the same order type (are in the same order) if for every triple of points, (p_i, p_j, p_k) are oriented counterclockwise if and only if (p'_i, p'_j, p'_k) are oriented counterclockwise [18]. Their method of determining if two sets of points have the same order type without having to compare every triple by defining the λ-matrix of a set of points is a follows: Given a set of points S, let $\lambda(p_i, p_j)$ be the number of points to the left of the directed line from p_i to p_j. We define $\lambda(p_i, p_i)$ to equal n. The $n \times n$ matrix containing $\lambda(p_i, p_j)$, for $i = 0, 1, 2, \ldots, n-1$ and $j = 0, 1, 2, \ldots, n-1$ is called the λ-*matrix* of S. The set $\Lambda(p_i, p_j)$ is the set of points to the left of the directed line from p_i to p_j. It is shown in [18] that the sets $\Lambda(p_i, p_j)$ can be determined given the sets $\lambda(p_i, p_j)$, and an algorithm is presented for computing the λ-matrix of a set of n points in $O(n^2 \log n)$ time. In [12], an algorithm is presented that computes the λ-matrix of a set of n points in the plane in $O(n^2)$ time. Two sets of points have the same order type if their λ-matricies are the same.

Several possible applications of this technique are presented in [18], including pattern recognition in which new images can be compared to a given image by comparing their λ-matrices. We use λ-matrices to compare two drawings D and D' of n nodes by computing the sum of the differences in entries in the λ-matrices of S and S':

$$\delta_{\lambda M}(S, S') = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} |\lambda(p_i, p_j) - \lambda(p'_i, p'_j)|$$

In [22], it is shown that the upper bound on $\delta_{\lambda M}(S, S')$ for n nodes is:

$$UB_{\lambda M}(n) = n \left\lfloor \frac{(n-1)^2}{2} \right\rfloor$$

therefore,

$$\mathcal{D}_{\lambda M}(S, S') = \frac{\delta_{\lambda M}(S, S')}{UB_{\lambda M}(n)}$$

The upper bound is achieved when the set of points are all on the convex hull and the ordering around the convex hull is reversed [22].

3.2.2 Distances Moved (DM) Model

A natural measure of difference between two sets of points is the distance that each point has moved. The value of $\delta_{DM}(S, S')$ is computed by summing the distance between p_i and p'_i for $i = 0, 1, 2, \ldots, n-1$:

$$\delta_{DM}(S, S') = \sum_{i=0}^{n-1} dist(p_i, p'_i)$$

The time required to compute $\delta_{DM}(S, S')$ is $O(n)$ since there are n distances. Each distance is at most $\sqrt{2}$ since the sets S and S' are within the unit square; therefore, we define the upper bound on $\delta_{DM}(S, S')$ for n nodes as:

$$UB_{DM}(n) = \sqrt{2}n$$

and let

$$\mathcal{D}_{DM}(S, S') = \frac{\delta_{DM}(S, S')}{UB_{DM}(n)}$$

The upper bound is achieved when all the points move from some corner of W to a diagonal corner.

4 The Layout Algorithms

Both the algorithms presented in this paper are iterative, and at each iteration heuristics are used to determine where to move the nodes. We restrict the movement of the nodes to inside the drawing window, which guarantees that criterion **CB1** is satisfied. In Section 5 we evaluate and compare the algorithms according to criteria **CB2** and **CB3**.

Let $S = S^0$ be the input positions of the nodes, and let $S^t = \{p_0^t, p_1^t, \ldots, p_{n-1}^t\}$ be the positions of the nodes after iteration t. The output positions of the nodes are given by the set S'. After each iteration t, tests can be performed to measure the distribution of the points in S^t and the differences between S and S^t. The algorithms iterate until a specified distribution or similarity measure exceeds a given threshold. If the thresholds are not reached, the algorithms iterate $I_{\max}$ times. When the algorithm returns, the user is presented with the resulting layout, and is given the option of changing the thresholds and continuing or accepting the layout the way it is. The similarity and distribution models to be used, the value $I_{\max}$, and the threshold values can be given as part of the input (for experienced users) or defaults can be used.

4.1 The Voronoi Diagram Cluster Buster (VDCB) Algorithm

The VDCB algorithm is based on the well-studied Voronoi diagram. Given a set $S = \{p_0, p_1, \ldots, p_{n-1}\}$ of n points, the Voronoi diagram of S, $VD(S)$ is a partition of the plane into n convex (not necessarily bounded) regions such that the region or polygon $Vor(p_i)$ associated with the point p_i is the set of points that are closer[1] to p_i than to any other point in S [28, 35]. An implementation by Kenny Wong [36] of Fortune's sweep-line algorithm [16] is used by the implementations of our algorithms.

The *Clipped Voronoi Diagram* of a set S of n points, $CVD(S)$, is $VD(S)$ clipped within W and can be computed in $O(n \log n)$ time. The idea of using the Voronoi diagram of the set S to constrain the movement of the nodes came about while trying to determine how to satisfy criterion **CB3**. Restricting the nodes to move within their Voronoi regions guarantees that p'_i is closer to p_i than to any $p_j \in S$, $j \neq i$. However, in small Voronoi regions that are close together, the allowable movement is too restricted to more evenly distribute the nodes. To allow the nodes in these small regions to become more evenly distributed, the process is iterated. At each iteration, the nodes move to a point that is inside their Voronoi region, and the Voronoi diagram of the new layout is computed. We can no longer guarantee that p'_i is closer to p_i than to any other $p_j \in S$, $j \neq i$.

To more evenly distribute the nodes, the nodes should move away from Voronoi edges they are close to and move closer to the Voronoi edges they are further from; therefore, it was decided to move each node to the *centroid* of the Voronoi region. Given a node p_i, the centroid of $Vor(p_i)$ is defined as:

$$cd_i = \frac{\left(\int_{Vor(p_i)} x, \int_{Vor(p_i)} y\right)}{Area(Vor(p_i))}$$

The n centroids can be computed in $O(n)$ time [22].

The VDCB algorithm is presented in Figure 1. The VDCB algorithm runs in $O(t * (n \log n + T(n)))$ time, where $t \leq I_{\max}$ is the number of iterations and $T(n)$ is the time taken to perform the measurements.

4.2 The GeoForce Algorithm

The Geographic Force or GeoForce algorithm is based on the idea of using forces and springs to model interactions between nodes in a drawing of a graph [7, 9, 15, 17, 20, 29, 30]. The basic idea in each of the spring-based or force-based algorithms is the same: Nodes with no edge between them are repelled from each other by a force that depends on the distance between them and nodes connected by an edge are attracted to each other by a force that depends on the distance between them. The initial layout is usually created arbitrarily. At each iteration of the algorithm, forces are computed on the nodes, and one or all of the nodes are moved to a position according to the acting forces.

[1] The distance between two points is the Euclidean distance.

```
algorithm VDCB(D)

initialize: stop_iterating ← FALSE; t ← 0; S^t ← S
while t < I_max and not stop_iterating
    Compute CVD(S^t)
    for each i
        find cd_i^t
        p_i^{t+1} ← cd_i^t
    t ← t + 1
    for each distribution measure ℰ
        if Ξ_ℰ(S^t) ≥ the defined threshold for ℰ then
            stop_iterating ← TRUE
            S' ← S^t
    for each similarity measure M
        if 𝒟_M(S^t) ≥ the defined threshold for M then
            stop_iterating ← TRUE
            S' ← S^{t−1}
output: D'
```

Figure 1: The VDCB algorithm.

The GeoForce algorithm came about by adapting the idea of the force-based algorithms with the hopes of achieving their success in practise. Criterion **CB3** states that the general shape of the drawing should be similar to the shape of the original drawing; therefore, attractive forces are applied between each node and its position in the original layout. Since the second goal of cluster busting in anchored graph drawing is to more evenly distribute the nodes in the drawing window, repelling forces are applied between every pair of nodes and between each node and the sides of the drawing window.

The forces applied, constants used, and techniques for dealing with the boundary of W have been chosen by implementing different methods, trying a few values in combination with other values, and observing the effect on sample layouts. There are $n+3$ forces repelling each node and one force attracting each node. Therefore, we take the average repelling force and subtract the attractive force. The total force applied to p_i at iteration t is given by:

$$f_i^t = \frac{F_S(p_i^t) + F_R(p_i^t)}{n+3} - f_A(dist(p_i^t - p_i))$$

where $F_S(p_i^t)$ is the repelling force between p_i^t and each of the sides of W, $F_R(p_i^t)$ is the repelling force between p_i^t and p_j^t, $j = 0, 1, \ldots, n-1$, $j \neq i$, and f_A is the attractive force between p_i^t and its original position p_i.

$F_S(p_i^t)$ and $F_R(p_i^t)$ are computed as:

$$F_S(p_i^t) = \sum_{w \in \{bl,tr\}} (f_S(|x_i^t - x_w|) + f_S(|y_i^t - y_w|))$$

and

$$F_R(p_i^t) = \sum_{j \neq i} f_R(dist(p_i^t, p_j^t))$$

The force functions, $f_R(d)$, $f_A(d)$, and $f_S(d)$ are computed as follows. The repelling force should move the nodes to some ideal distance away from one another. Fruchterman and Reingold define the ideal distance between nodes as $d_I = \sqrt{area/n}$, where *area* is the area of the drawing window [17]. We have found that bounding the amount of force applied when $d = 0.0$ and applying no force when $d > d_I$ results in a smooth behaviour. We let $f_R(d)$ be the linear function passing through $(0, C_1)$ and $(d_I, 0)$, where C_1 is a constant representing the maximum force applied. This function is easy to compute, and in practise gives good results for $C_1 = 100.0$ and $d_I = \sqrt{area/n}$.

The attractive force is applied to pull each node towards its original position. We found that the following function gives good results: $f_A(d) = C_2 d$, where $C_2 = 1.0$ is a constant.

Each side of W exerts a repelling force on each of the nodes. We let $f_S(d)$ be a linear function passing through $(0, M_F)$ and $(d_I, 0)$, where M_F is the maximum force exerted. We found that good results were obtained with $M_F = 10.0 * n$.

As Eades does in [9] and Fruchterman and Reingold do in [17], the acting forces are computed for all nodes in the given layout and then the nodes are moved according to the forces, rather than dealing with the nodes one at a time as in [7] and [20].

The GeoForce algorithm differs from other force-directed algorithms because we limit the movement of a node p_i within a region that takes into consideration its proximity to the other nodes. The nodes are only allowed to move within a specified percentage of the distance to their Voronoi edges. Let f_i^t be the total force acting on p_i^t at iteration t such that $p_i^t + f_i^t$ is the point that p_i^t would move to according to the force f_i^t. Extend the ray from p_i^t to $p_i^t + f_i^t$ if necessary until it intersects an edge of $Vor(p_i^t)$. Let $\mathcal{I}_i$ be that intersection point. We set the step-size for the node p_i^t to be $s_i^t = C_4 * dist(p_i^t, \mathcal{I}_i)$ where C_4 is a constant. We then set either $p_i^{t+1} = p_i^t + f_i^t$ or $p_i^{t+1} = p_i^t + \frac{s_i^t * f_i^t}{|f_i^t|}$, whichever is closer to p_i^t. By experimentation, we found that $C_4 = 0.25$ works well.

The GeoForce algorithm operates like a "smart" VDCB algorithm: The nodes are moved to a location inside their Voronoi regions, but the heuristic used to choose that location is based on forces acting on the nodes rather than just blindly choosing the centroid of the regions. The GeoForce algorithm is presented in Figure 2. It runs in $O(t * (n^2 + T(n)))$ time where $t \leq I_{\max}$ is the number of iterations and $T(n)$ is the time taken to perform the measurements.

5 Results

In this section, we evaluate and compare the VDCB and GeoForce algorithms. To do this, we execute each algorithm on a variety of random input layouts and measure the layouts after a differing number of iterations. We generate layouts with $\mathcal{K}$ *clusters* of k points each inside W such that $n = \mathcal{K}k$. The region W can be evenly divided into $\mathcal{K}$ square regions with area $\frac{1}{\mathcal{K}}$. We want to evaluate our layout algorithms on layouts with clusters of nodes such that the interactions between the nodes are hard to read; that is, we want each region that contains a cluster of nodes to be small. Therefore, we let each cluster occupy a square region with $O(\frac{1}{\mathcal{K}^2})$ area. We first compute $\mathcal{K}$ *cluster centres* by generating $\mathcal{K}$ random points inside W: $\{c_0, c_1, \ldots, c_{\mathcal{K}-1}\}$. Let s_i be the square with centre c_i and area $\frac{C}{\mathcal{K}^2}$ where C is a constant. The squares are clipped within W, and k random points are generated uniformly inside each clipped s_i. The result is a set of $\mathcal{K}$ clusters in the unit square. When $\mathcal{K} = n$, we let $p_i = c_i$, $i = 0, 1, \ldots \mathcal{K} - 1$.

Executing the algorithms on a variety of random layouts gives a measure of the goodness of the algorithms with respect to random layouts. For each type of initial layout, we generated 1000 random layouts of that type and took the *average* value for each of the measurements. Let the average value of $\Xi_{\mathcal{E}}(S)$ be $\overline{\Xi}_{\mathcal{E}}(S)$ for the distribution model $\mathcal{E}$, and let the average value of $\mathcal{D}_M(S, S')$ be $\overline{\mathcal{D}}_M(S, S')$ for the similarity model M. The measurements were performed after

```
algorithm GeoForce(D)

initialize: stop_iterating ← FALSE; t ← 0; S^t ← S
while t < I_max and not stop_iterating
    for each i
        Compute the force f_i^t acting on p_i^t
    Compute CVD(S^t)
    for each i
        if dist(p_i^t, p_i^t + f_i^t) < dist(p_i^t, p_i^t + (s_i^t * f_i^t)/|f_i^t|) then
            p_i^{t+1} ← p_i^t + f_i^t
        else
            p_i^{t+1} ← (s_i^t * f_i^t)/|f_i^t|
    t ← t + 1
    for each distribution measure ℰ
        if Ξ_ℰ(S^t) ≥ the defined threshold for ℰ then
            stop_iterating ← TRUE
            S' ← S^t
    for each similarity measure M
        if 𝒟_M(S^t) ≥ the defined threshold for M then
            stop_iterating ← TRUE
            S' ← S^{t-1}
output: D'
```

Figure 2: The GeoForce algorithm.

$t = 1, 2, \ldots, 10, 20, \ldots, 100$ iterations of each of the algorithms.

Table 1 shows values of $\overline{\Xi}_{CP}(S)$ for different types of layouts and for both layout algorithms.

n	$\mathcal{K}$	$n/\mathcal{K}$	$\overline{\Xi}_{CP}(S)$	$\overline{\Xi}_{CP}(S^1)$ GeoForce	$\overline{\Xi}_{CP}(S^1)$ VDCB	$\overline{\Xi}_{CP}(S^{10})$ GeoForce	$\overline{\Xi}_{CP}(S^{10})$ VDCB	$\overline{\Xi}_{CP}(S^{100})$ GeoForce	$\overline{\Xi}_{CP}(S^{100})$ VDCB
25	25	1	0.0297158	0.0594681	0.0867467	0.1045789	0.1274640	0.1188087	0.1702646
	1	25	0.0124560	0.0286982	0.0428854	0.0785006	0.0901596	0.1096014	0.1662676
	5	5	0.0069397	0.0215777	0.0375028	0.0873186	0.1107191	0.1153616	0.1678225
50	50	1	0.0142302	0.0324772	0.0548724	0.0652549	0.0883792	0.0857881	0.1214667
	1	50	0.0061505	0.0150611	0.0258729	0.0425497	0.0529996	0.0620407	0.1141157
	2	25	0.0047297	0.0109252	0.0199791	0.0392802	0.0513587	0.0644586	0.1156817
	5	10	0.0032214	0.0078346	0.0153701	0.0403736	0.0643745	0.0709301	0.1180011
	10	5	0.0024422	0.0077468	0.0180950	0.0442022	0.0732051	0.0768871	0.1195598
	25	2	0.0020711	0.0154545	0.0412309	0.0578902	0.0841695	0.0821875	0.1206015
100	100	1	0.0071465	0.0192338	0.0351218	0.0397486	0.0614420	0.0583786	0.0853836
	1	100	0.0029802	0.0083215	0.0159633	0.0242627	0.0336719	0.0330132	0.0666534
	2	50	0.0022414	0.0056704	0.0119749	0.0201109	0.0277523	0.0344893	0.0716336
	4	25	0.0017113	0.0039890	0.0091445	0.0170900	0.0293285	0.0377601	0.0756969
	10	10	0.0011238	0.0030085	0.0069926	0.0163230	0.0405262	0.0436906	0.0798743
	25	4	0.0008172	0.0028873	0.0124229	0.0195724	0.0514290	0.0502511	0.0829412
	50	2	0.0007292	0.0070734	0.0244130	0.0326964	0.0573149	0.0544592	0.0840577

Table 1: $\overline{\Xi}_{CP}(S^1)$, $\overline{\Xi}_{CP}(S^1 0)$, and $\overline{\Xi}_{CP}(S^1 00)$ for several initial layouts and both layout algorithms.

Tables 2 and 3 give values of $\overline{\mathcal{D}}_{\lambda M}(S, S')$ and $\overline{\mathcal{D}}_{DM}(S, S')$ for different n and for both layout algorithms. According to the similarity model M, the average difference between two random layouts is $\overline{\mathcal{D}}_M(S, P)$.

The experiments show that, on average, the GeoForce and VDCB algorithms more evenly distribute the nodes in a drawing according to the CP model of distribution. In all cases, $\overline{\Xi}_{CP}(S^1) > \overline{\Xi}_{CP}(S)$, and $\overline{\Xi}_{CP}(S^{t+1}) > \overline{\Xi}_{CP}(S^t)$ for all types of input for both layout algorithms.

n	$\overline{\mathcal{D}}_{\lambda M}(S,P)$	Worst $\overline{\mathcal{D}}_{\lambda M}(S,S^1)$ GeoForce	Worst $\overline{\mathcal{D}}_{\lambda M}(S,S^1)$ VDCB
25	0.5645142	0.0696986	0.1397622
50	0.5571432	0.0583581	0.1188550
100	0.5525959	0.0439622	0.0966927

Table 2: $\overline{\mathcal{D}}_{\lambda M}(S,P)$ and $\overline{\mathcal{D}}_{\lambda M}(S,S^1)$ after one iteration of the VDCB and GeoForce algorithms.

n	$\overline{\mathcal{D}}_{DM}(S,P)$	Worst $\overline{\mathcal{D}}_{DM}(S,S^1)$ GeoForce	Worst $\overline{\mathcal{D}}_{DM}(S,S^1)$ VDCB
25	0.3671019	0.0275942	0.0790956
50	0.3721420	0.0183423	0.0701698
100	0.3690213	0.0125748	0.0419749

Table 3: $\overline{\mathcal{D}}_{DM}(S,P)$ and $\overline{\mathcal{D}}_{DM}(S,S^1)$ after one iteration of the VDCB and GeoForce algorithms.

Layouts with roughly the same number of clusters as nodes in each cluster are the most different after several iterations according to the λM model. However, layouts that start out the most evenly distributed according to the CP model change the least according to the λM measure after several iterations of the layout algorithms.

According to the DM model of similarity, layouts with one or two clusters are most different after several iterations, because as the nodes become spread out they move further from where they started than in layouts with more clusters. As with the λM model, layouts that start out the most evenly distributed according to the CP model change the least according to the DM model after several iterations of the layout algorithms.

The VDCB algorithm more evenly distributes the nodes after fewer iterations than the GeoForce algorithm; therefore, we compare the difference between S and S' for layouts S' with the same distribution level. Let S'_A be the layout S after adjustment by the algorithm $A \in \{VDCB, GeoForce\}$. We say that the algorithm A_1 *performs better* than the algorithm A_2 on specific types of layouts if the difference levels are smaller between S and S'_{A_1} than between S and S'_{A_2} when the distribution levels of S'_{A_1} and S'_{A_2} are roughly the same.

The type of initial layout has a significant effect on which layout algorithm performs better. On average, the GeoForce algorithm performs better for layouts with a small number of clusters, and the VDCB algorithm performs better for layouts with a large number of clusters. In the GeoForce algorithm the nodes move less far at each iteration than in the VDCB algorithm for all types of layouts, because of the forces acting on the nodes. In layouts with a small number of big clusters, small Voronoi regions constrain the movement of the nodes in the VDCB algorithm as well; therefore, the number of iterations necessary for the VDCB algorithm to reach a given level of distribution is greater for these types of layouts. For layouts with a large number of small clusters, the VDCB algorithm distributes the nodes evenly after very few iterations where the GeoForce algorithm requires a greater number of iterations to achieve the same level of distribution. During the extra iterations needed, the repelling forces act on the nodes, causing the layouts to become more different from the original layout than those produced by the VDCB algorithm.

For layouts with one cluster uniformly distributed in W initially, one to three iterations of either algorithm is sufficient to distribute the nodes such that their interactions are readable. After one or two iterations, the VDCB algorithm distributes the nodes more evenly than the GeoForce algorithm but the VDCB algorithm moves the nodes further than necessary. The layouts are more similar to the original layout according to each of the measures with the GeoForce algorithm. Figure 3 shows such a layout with $n = 25$ nodes and the layout after one iteration of the GeoForce algorithm. Figure 4 shows the same layout after one iteration of the VDCB algorithm.

Figure 5 shows a layout with $\mathcal{K} = 25$ clusters of size 4 each and the layout after 20 iterations of the GeoForce algorithm. Figure 6 shows the same layout

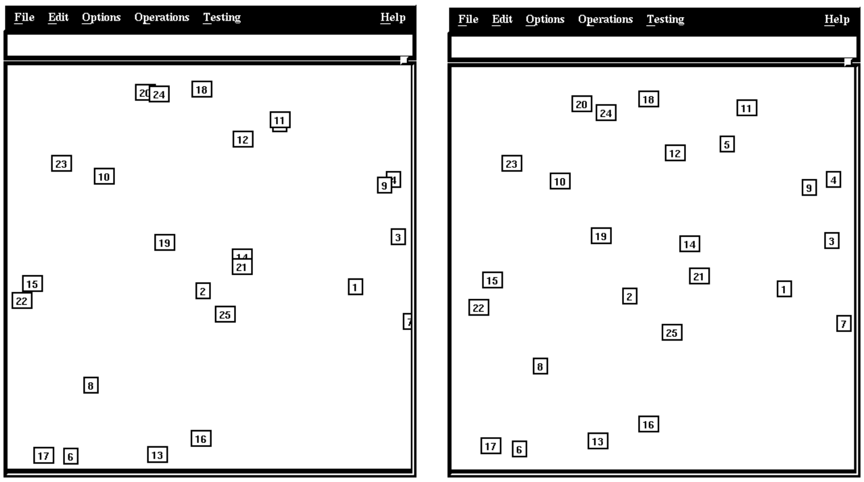

Figure 3: A layout with $\mathcal{K} = 25$ clusters of size 1 each, and that layout after one iteration of the GeoForce algorithm.

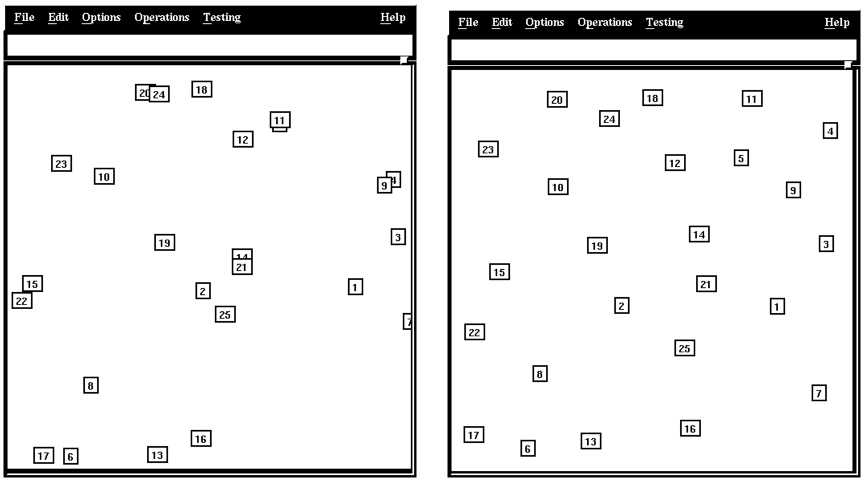

Figure 4: A layout with $\mathcal{K} = 25$ clusters of size 1 each, and that layout after one iteration of the VDCB algorithm.

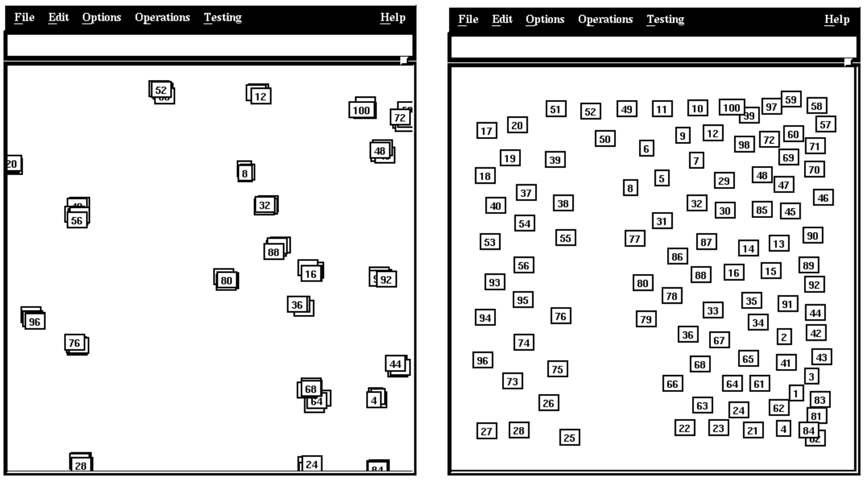

Figure 5: A layout with $\mathcal{K} = 25$ clusters of size 4 each, and that layout after 20 iterations of the GeoForce algorithm.

after 11 iterations of the VDCB algorithm. The layout produced by the VDCB algorithm is more evenly distributed according to the CP model, and is more similar to the original layout according to the similarity measures presented in this paper. Human observers may judge the GeoForce layout to be more similar to its original layout than the VDCB layout. This is partly because the notion of clustered nodes seems to be completely removed in the VDCB layout. The notion of clustering is an important model of a user's mental map [11, 25]; however, the similarity measures presented in this paper do not adequately capture the notion of clustering. In [22], the number of changes in the Delaunay triangulation of the node positions is used as a measure of similarity that better represents the clusters. The layout produced by the VDCB algorithm in Figure 6 was less similar than that produced by the GeoForce algorithm according to the number of changes in the Delaunay triangulation.

Although the GeoForce algorithm was designed to perform cluster busting in anchored graph drawing in a "smarter" way than the VDCB algorithm, we have seen that it does not always do so. In fact, we would recommend using the VDCB algorithm over the GeoForce algorithm in most cases. The possibility remains that a different choice of force functions and/or constants in the GeoForce algorithm might result in an algorithm that consistently performs better than the VDCB algorithm. This possibility is discussed further in the next section.

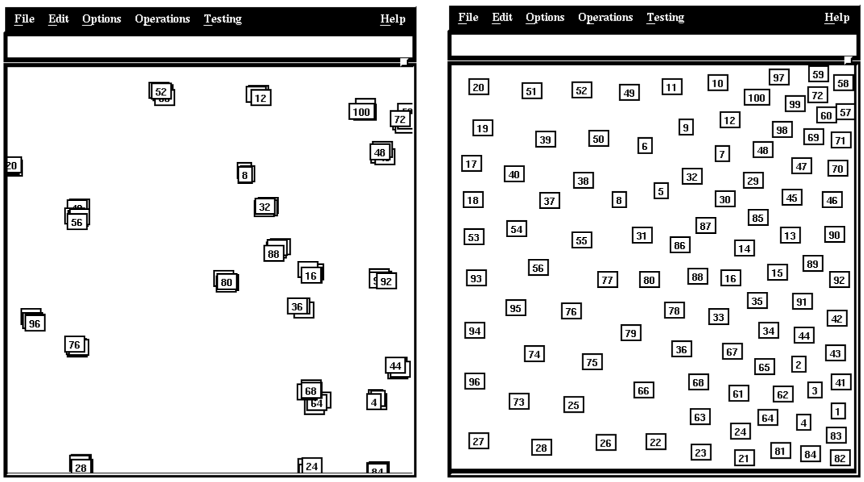

Figure 6: A layout with $\mathcal{K} = 25$ clusters of size 4 each, and that layout after 11 iterations of the VDCB algorithm.

6 Conclusions and Future Work

In this paper, we identified a new approach to drawing graphs with an existing layout that we call cluster busting in anchored graph drawing. We presented two heuristic algorithms for solving this problem. We provided measurements for evaluating layouts produced by the algorithms based on the stated goals. These measurements were used to evaluate and compare the algorithms.

We found that using heuristics is a good approach to drawing graphs in general and for cluster busting in anchored graph drawing in particular. Using quantitative measurements is useful for understanding the behaviour of heuristic algorithms. The measurements allow the algorithms to be tested on a larger number of layouts than if evaluations are based solely on visually judging example layouts.

There are several areas for further research. It remains a difficult problem to use the quantitative measurements to determine stopping conditions, because of the large number of factors involved. Possible enhancements are to allow more user input during the layout process or to use animation. The user watches the layout change at each iteration, and stops the algorithm manually when the resulting layout is satisfactory.

We support the conclusion of Messinger *et al.* in [23] that more research is needed into subjective issues that users find important for "good" drawings of graphs. This is particularly true in the case when the aesthetic factor is something as subjective as similarity.

Different force functions and constants might result in a GeoForce algorithm that consistently performs better than the VDCB algorithm. The forces and constants in the current GeoForce algorithm were determined by trying a few values and viewing sample layouts produced by an algorithm using those values. Subjective judgements were made as to which choices gave better layouts by one or two people observing the layouts. It would be interesting to use the quantitative measures to judge several layouts after each choice of forces and constants. This way many more layouts could be judged on different combinations of the forces and constants.

An important area for future work in layout adjustment algorithms is to take the edges of the graphs into consideration. This is a difficult problem that was also not addressed in algorithms for preserving the mental map [10]. There are a couple of ways that edges can be incorporated into layout adjustment algorithms. First, similarity measures can include changes in the edges in the drawing. Second, layout adjustment algorithms can optimize edge criteria such as keeping the number of edge crossings small in the new layout while retaining similarity in the node positions.

A model of the mental map that takes the positions of the edges in a layout into consideration is presented in [25]. Figure 7(a) taken from [25] shows a drawing of a graph of $n = 6$ nodes and $m = 7$ edges. The edges and nodes in the drawing divide the plane into regions called *faces*. The outside face is bounded by edges and nodes listed in clockwise order: $f_1 = AbBcCgFdeDfEfDa$. Three other faces are given by $f_2 = AdeDa$, $f_3 = CgFde$, and $f_4 = AbBcCed$. A *dual graph* is defined such that there is a node in the dual graph for each face defined above, and there is an edge between two nodes in the dual graph whose faces share a boundary. Figure 7(b) shows the dual graph of the layout in Figure 7(a). Two layouts have the same topology if they have the same dual graph. An adjustment algorithm preserves the topology of a drawing if the new drawing has the same topology as the original drawing. The difference between two topologies (such as the number of edge changes in the dual graph) could be used as a measure of difference between two drawings that includes the edges of the graph. This measure could be used to evaluate our algorithms and other cluster busting in anchored graph drawing algorithms according to the topology model of a user's mental map.

Since both algorithms are iterative, it would be interesting if we could prove that the algorithms converge. In [22], it is shown that in the VDCB algorithm, $n = st$ nodes at vertices of an isothetic grid of size s by t do converge. A discussion of the convergence of the VDCB algorithm for the case of n points in general is also presented. It is an open problem to prove that the VDCB algorithm converges for general input.

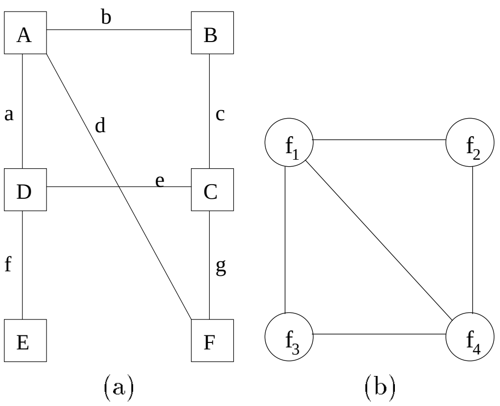

Figure 7: A layout and the dual graph representing its topology.

Acknowledgements

The authors would like to thank Patrick Finnigan, Jacob Slonim, and other members of the CORDS project for their input and ideas as well as the referees whose suggestions simplified the presentation.

References

[1] G. K. Attaluri, D. P. Bradshaw, P. J. Finnigan, N. Hinds, M. Kalantar, K. A. Lyons, A. D. Marshall, J. Pachl, and H. Tran. Operation Jump Start: A CORDS integration prototype using DCE. In *Proceedings of the 1993 CAS Conference Volume II*, pages 621–636, October 1993.

[2] D. W. Bachmann, M. E. Segal, M. M. Srinivasan, and T. J. Teorey. Netmod: a design tool for large-scale heterogeneous campus networks. Technical report, Department of Electrical Engineering and Computer Science and Center for Information Technology Integration, The University of Michigan, Ann Arbor, Michigan, June 1990.

[3] M. A. Bauer, N. Coburn, D. L. Erickson, P. J. Finnigan, J. W. Hong, P.-Å. Larson, J. Pachl, J. Slonim, D. J. Taylor, and T. J. Teorey. A distributed system architecture for a distributed application environment. *IBM Systems Journal*, 33(3):399–425, 1994.

[4] K.-F. Bohringer and F. Newbery Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *Proceedings of the ACM Conference on Computer Human Interaction (CHI)*, pages 43–51, April 1990.

[5] R. F. Cohen, G. Di Battista, R. Tamassia, I. G. Tollis, and P. Bertolazzi. A framework for dynamic graph drawing (extended abstract). In *Proceedings of the Eighth Annual ACM Symposium on Computational Geometry*, pages 261–270, 1992.

[6] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. Technical report, Department of Applied Mathematics and Computer Science, Weizmann Institute of Science, April 1991.

[7] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Journal on Graphics*, 15(4), October 1995.

[8] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.

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

[10] P. Eades and W. Lai. Algorithms for disjoint node images. In *The Fifteenth Australian Computer Science Conference (ACSC-16)*, pages 253–265, January 1992.

[11] P. Eades, W. Lai, K. Misue, and K. Sugiyama. Preserving the mental map of a diagram. Technical Report IIAS-RR-91-16E, International Institute for Advanced Study of Social Information Science, Fujitsu Laboratories Ltd., Aug. 1991.

[12] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15:341–363, 1986.

[13] D. L. Erickson, P. J. Finnigan, G. K. Attaluri, M. A. Bauer, D. Bradshaw, N. Coburn, M. P. Consens, M. Z. Hasan, J. W. Hong, K. A. Lyons, T. P. Martin, G. W. Neufeld, W. Powley, D. Rappaport, D. J. Taylor, T. J. Teorey, and Y. Yemini. CORDS: An update to the prototypes. Technical Report TR-74.120, August 1993.

[14] P. J. Finnigan and K. A. Lyons. Narratives of space and time: Visualization for distributed systems. In *Proceedings of the 1991 CAS Conference*, pages 363–391, October 1991.

[15] C. J. Fisk, D. L. Caskey, and L. E. West. ACCEL: Automated circuit card etching layout. *Proceedings of the IEEE*, 55(11):1971–1982, November 1967.

[16] S. Fortune. A sweepline algorithm for Voronoi digrams. *Algorithmica*, 2:153–174, 1987.

[17] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Softw. – Pract. Exp.*, 21(11):1129–1164, 1991.

[18] J. E. Goodman and R. Pollack. Multidimensional sorting. *SIAM Journal on Computing*, 12(3):484–507, August 1983.

[19] R. A. M. Gregson. *Psychometrics of Similarity*. Academic Press, 1975.

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

[21] K. A. Lyons. Cluster busting in anchored graph drawing. In *Proceedings of the 1992 CAS Conference Volume I*, pages 7–17, November 1992.

[22] K. A. Lyons. *Cluster Busting in Anchored Graph Drawing*. PhD thesis, Queen's University, 1994.

[23] E. B. Messinger, L. A. Rowe, and R. R. Henry. A divide-and-conquer algorithm for the automatic layout of large directed graphs. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-21(1):1–11, January/February 1991.

[24] K. Miriyala, S. W. Hornick, and R. Tamassia. An incremental approach to aesthetic graph layout. In *Proc. Internat. Workshop on Computer-Aided Software Engineering (CASE '93)*, 1993.

[25] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages and Computing*, 6(2):195–209, 1995.

[26] S. Mitrovic and S. Murer. A tool to display hierarchical acyclic dataflow graphs. In N. N. Mirenkov, editor, *Proceedings of the International Conference on Parallel Computing Technologies*, pages 304–315. World Scientific, 1991.

[27] S. Moen. Drawing dynamic trees. *IEEE Software*, 7:21–28, July 1990.

[28] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[29] N. R. Quinn Jr. and M. A. Breuer. A forced directed component placement procedure for printed circuit boards. *IEEE Transactions on Circuits and Systems*, CAS-26(6):377–388, 1979.

[30] L. Steinberg. The backboard wiring problem: A placement algorithm. *SIAM Review*, 3(1):37–50, January 1961.

[31] G. T. Toussaint. Pattern recognition and geometrical complexity. In *Proceedings of the Fifth International IEEE Conference on Pattern Recognition*, pages 1324–1347, December 1980.

[32] G. T. Toussaint. Computational geometric problems in pattern recognition. In J. Kittler, K. S. Fu, and L. F. Pau, editors, *Pattern Recognition Theory and Applications*, pages 73–91. D. Reidel Publishing Company, 1982.

[33] G. T. Toussaint. Computational geometry: Recent results relevant to pattern recognition. In P. A. Devijver and J. Kittler, editors, *Proceedings of the NATO Advanced Study Institute on Pattern Recognition Theory and Applications*, pages 295–305, June 1986.

[34] D. Tunkelang. An aesthetic layout algorithm for undirected graphs. Master's thesis, Massachusetts Institute of Technology, 1992.

[35] G. Voronoi. Nouvelles applications des parametres continus à la theorie des formes quadratiques. Deuxième mémoire: Recherches sur les paralléloèdres primitifs. *Journal fur die Reine und Angewandte Mathematik*, 138:198–287, 1908.

[36] K. Wong and H. A. Müller. An efficient implementation of fortune's plane-sweep algorithm for Voronoi diagrams. Technical report, Department of Computer Science, University of Victoria, Victoria, B.C., October 1991.