Cluster Busting in Anchored Graph Drawing

by

Kelly A. Lyons

A thesis submitted to the
Department of Computing and Information Science
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada
September 1996

Abstract

Graphs are used to model many objects in computer science such as VLSI circuits, networks, and software designs. Using graphs to represent complex structures and relations often improves the presentation of these structures. It is not surprising that there has been a recent demand for algorithms that display graphs on a two-dimensional surface so that the resulting drawing is easy to comprehend. Currently most work in the field of automatic graph drawing addresses the problem of drawing a graph given its combinatorial description. Recently, effort has been directed toward designing algorithms that modify existing layouts. Graphs with an existing layout include graphs of networks in which the nodes have geographic locations and graphs whose existing layouts have been updated by adding and deleting nodes or by moving nodes. These types of layout algorithms are called *layout adjustment* algorithms. The challenge of layout adjustment is to design algorithms that produce layouts that do not destroy the user's mental picture of the existing layout.

We present a new approach to layout adjustment called *cluster busting in anchored graph drawing* and present two algorithms that are based on this approach. Informally, the goals of cluster busting in anchored graph drawing are to distribute more evenly the nodes in the drawing window while keeping the drawing similar to the original drawing. The algorithms we present are iterative and at each iteration a heuristic

is used to decide how to move the nodes. It is a difficult problem to do theoretical analysis on these algorithms; therefore, we provide quantitative measurements of layouts to justify our methods. Two measures of distribution and five measures of similarity are presented that are used to evaluate the algorithms. They are also used to determine stopping conditions for the iterative algorithms. In some cases, we prove how the algorithms behave on certain types of input. In other cases, we provide evidence that the algorithms satisfy the goals of cluster busting in anchored graph drawing on general inputs. The results open several areas for further research.

Co-Authorship

My supervisor, David Rappaport, had an important influence on all aspects of this work. All of the lemmas and theorems were proved by me unless otherwise cited. In particular, Lemma 4.2.5 is the result of joint work with Henk Meijer [57] and Lemmas 6.1.2, 6.1.3, 6.1.4, 6.5.1, Theorem 6.5.2, and the discussion on the convergence of the VDCB algorithm for general input in Section 6.5.2 are the result of joint work with Henk Meijer and David Rappaport [53].

Acknowledgments

I have learned that the length of the acknowledgements section is directly proportional to the length of time it takes one to complete their PhD!

I would like to thank my supervisor, David Rappaport, for his enthusiasm about this subject and for his support.

The members of my examining committee provided valuable feedback and comments that improved the presentation of this dissertation and I am grateful to them for that.

I would like to acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada and the IBM Toronto Laboratory (in the form of NSERC Postgraduate Scholarships, and IBM Student Fellowships, respectively).

I want to thank Henk Meijer for teaching me a lot of things and for his guidance and encouragement.

I am grateful to Jacob Slonim for his support and the many opportunities he has given me.

I want to thank Peter Eades for inviting me to visit Australia when my supervisor was there on sabbatical. I learned a lot during my visit that helped incredibly in my work.

Many students and staff in the Computing and Information Science Department

at Queen's have been great friends and have provided a fun and educational place to study. In particular, the appreciation I have for Irene LaFleche and everything she has done goes beyond words.

Wendy Powley worked hard to develop GLAD which allowed me to test and experiment with the layout algorithms presented in this thesis. I thank her for that and for her great sense of humour.

I would like to thank the people at CAS and the participants of various CASCONs for their insight and comments regarding this work. In particular, I thank Pat Finnigan for identifying the application of layout adjustment initially, and Ian Munro, Eric Mendelsohn, and Ric Holt for many valuable discussions. I also want to thank Frank Eigler for helping me get efficient use of CPUs to run the experiments and for always checking with me before bringing the system down.

I am especially grateful to John Botsford and Cindy Butler for taking on some of my duties when I was working on my thesis and for their unending moral support.

Mary, Kevin, Susan, and Pearl have been very understanding and supportive. They have also made me laugh a lot and I thank them for that. My parents and Sue and Tim have been incredibly loving and supportive for a long time and I am forever thankful and grateful to them.

Last, but most certainly not least, I thank Donny for being unbelievably understanding and for flying to Kingston to surprise me.

This thesis is dedicated to Pappy and Grandma Billie — Even though Grandma Billie didn't live to see it finished, she has been with me through its entirety.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Graphs are used to model many objects in computer science such as VLSI circuits, networks, and large software designs. Using graphs to represent complex structures and relations often simplifies the presentation of these structures. It is not surprising that there has been a recent demand for algorithms that display a graph on a two-dimensional surface so that the resulting drawing is easy to read and understand. As a consequence, several automatic graph drawing algorithms and systems have recently emerged. The field of graph drawing is also called automatic graph layout and the terms "drawing" and "layout" are used interchangeably. We distinguish between algorithms whose goal it is to produce readable drawings of graphs and those whose goal it is to design physical structures such as in VLSI design. Even though the two types of algorithms can sometimes have similar goals, we limit the following discussion to the issue of drawing graphs for the purposes of visualizing objects.

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 $e_{ij} = e_{ji}$. Currently most

work in the field of automatic graph drawing addresses the problem of producing a drawing of a graph given its combinatorial description [14]. The problem of drawing a graph given its combinatorial description is termed *layout creation* in [21]. This problem is difficult because a precise definition of what constitutes a good drawing is subjective and depends on the user and the application. There are, however, certain generally accepted objective criteria considered to be important in a good drawing. These include minimizing the number of edge crossings, keeping edge lengths uniform, and displaying symmetries in the graph. Given the combinatorial description of a graph G, layout creation algorithms try to position the nodes and edges of G such that some or all of these optimization criteria are satisfied.

Recently, effort has been directed toward designing algorithms that redraw graphs with an existing layout. Graphs with an existing layout include graphs of networks in which the nodes have geographic locations and graphs whose existing layouts have been updated by adding and deleting nodes or by moving nodes. In either case, 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. Drawing graphs with an existing layout has applications in visualizing distributed systems. Programming 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 [29]. The networks and systems are represented as graphs and, as part of the visualization system, these graphs must be laid out or drawn automatically on the screen. 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 user has been viewing the graph. After using and editing the graph, the graph can become hard to read and understand. Algorithms that redraw a graph as it undergoes changes are called *dynamic* graph drawing algorithms [5, 9, 59]. In [21], algorithms that redraw graphs with an existing layout are called *layout adjustment* algorithms. To distinguish between these types of algorithms and layout creation algorithms, the latter are called *static* graph drawing algorithms. In [5], layouts that do not change drastically when updates are made to the underlying graph are called *stable* layouts. The reapplication of a static graph drawing algorithm is not useful in a dynamic setting nor for layout adjustment because a new layout that is drastically different from the existing layout destroys the user's mental picture of the layout [5, 20, 21]. 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.

In [19, 20, 21], an approach to layout adjustment called *preserving the mental map* is presented. The idea is to preserve some measure of similarity between the nodes in the original drawing and the nodes in the new drawing while trying to separate overlapping nodes. In [19], algorithms are presented that preserve the *orthogonal ordering* of the nodes; that is, for each pair of node positions p_i and p_j in the original

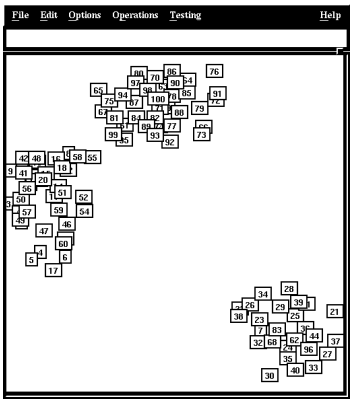

Figure 1.1: A layout in which nodes cannot move very much if their orthogonal ordering is preserved.

drawing with corresponding positions p'_i and p'_j, respectively, in the new drawing:

- $x'_i < x'_j$ 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. Nodes that are close together in the x-direction might be far apart in the y-direction. Figure 1.1 shows a drawing of a graph in which the nodes cannot move very much if their orthogonal ordering is preserved. The nodes in the leftmost cluster cannot move up nor right past the nodes in the middle cluster, and cannot move down past the nodes in the rightmost cluster.

In [21], preserving *proximity relations* between node positions is also suggested as a method for maintaining similarity between two layouts. A *proximity graph* is defined on a set of node positions such that there is an edge between node positions

p_i and p_j if p_i and p_j are "close" according to some definition of close. Several kinds of proximity graphs have been defined [82, 84, 85, 86, 87] including the Euclidean minimum spanning tree, the Delaunay triangulation, and the sphere of influence graph. These proximity graphs are described in Chapter 4. No nontrivial transformation which preserves any of these proximity graphs is currently known [21].

In this dissertation, we present a different approach to the problem of layout adjustment. We weaken the requirement that properties such as orthogonal ordering and proximity relations 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 call this approach to layout adjustment *cluster busting in anchored graph drawing* and present two heuristic algorithms for layout adjustment that are based on this approach. Cluster busting in anchored graph drawing was introduced in [52].

The goals of layout creation and layout adjustment algorithms are specified by identifying objective criteria that layouts produced by the algorithms should satisfy. Various methods are used to show that the layouts meet the given criteria. In many cases it can be proved that the layouts satisfy the criteria. Sometimes this is only true for certain classes of graph. Since producing drawings that satisfy many of the optimization criteria is NP-hard, the most successful algorithms in practise are heuristic in nature. Often very little can be proved about the resulting layouts. In this case, example layouts are presented along with an explanation of experiences to illustrate that the criteria are satisfied most of the time. In some cases, quantitative

measures are used to evaluate layouts produced by algorithms and to compare the layouts with those produced by other algorithms [58, 60, 88].

In this dissertation we define objective criteria for cluster busting in anchored graph drawing. As in [58, 60, 88], we use quantitative measures of the resulting layouts to evaluate our graph drawing algorithms. We show that our algorithms provide reasonable solutions to the problem of cluster busting in anchored graph drawing and that quantitative measurements are helpful for evaluating heuristic layout algorithms.

The rest of this dissertation is organized as follows. In Chapter 2, we review the field of automatic graph drawing and show how cluster busting in anchored graph drawing relates to other work being done. In Chapter 3, we present definitions and terminology used throughout the dissertation. In Chapter 4, we present quantitative measures of distribution and difference. We justify our choice of measures and describe the algorithms used to compute them. In Chapter 5, we present two cluster busting in anchored graph drawing algorithms and discuss issues of time complexity and the heuristics used. In Chapter 6, we evaluate and compare the algorithms using the measures of distribution and difference presented in Chapter 4. In some cases, we prove how the algorithms behave on certain types of input. In other cases, we provide evidence that the algorithms satisfy the goals of cluster busting in anchored graph drawing on general inputs. Finally, in Chapter 7, we summarize the significance of our results and describe areas for further research.

The contributions of this thesis build on the work of [21, 19, 20] by designing algorithms that solve the problem of drawing graphs with an existing layout. There are several applications of this problem and, in particular, our results have been applied to a project at the Centre for Advanced Studies at the IBM Software Solutions

Toronto Laboratory [29, 31]. We have taken a different approach to layout adjustment than in previous work and have designed two algorithms based on this approach. We prove some theoretical results regarding the behaviour of the algorithms under restricted classes of input. We identify quantitative measures of distribution and difference and use these to evaluate layouts produced by the algorithms. As a result, several interesting questions regarding point sets are uncovered. We demonstrate the behaviour of the algorithms on general input by executing these measurements on layouts produced by implementations of the algorithms. We illustrate the benefit of using heuristic algorithms and evaluation measures for drawing graphs.

Chapter 2

Literature Review

Graph drawing algorithms can be classified according to several criteria: The class of graph the algorithm draws, the *graphic standard* used (for example, straight-line edges versus edges with bends), the aesthetics, or optimization criteria being considered, the constraints imposed on the drawing, and the computational complexity of the algorithm [79]. We also classify algorithms according to the method by which the resulting layouts are evaluated.

In this chapter we present examples of previous work in automatic graph drawing. A comprehensive survey of graph drawing algorithms is presented in [14]. We describe the goals of layout algorithms with respect to readability issues. We present existing methods of evaluating whether layouts produced by a layout algorithm satisfy the goals of the algorithm. We then describe cluster busting in anchored graph drawing and show how our results relate to previous work.

2.1 Automatic Graph Drawing

In a drawing, nodes can be represented as circles or boxes and edges as straight-line segments, polygonal chains, or curved segments between nodes [14, 79]. The representation of nodes and edges in the drawing is called the graphic standard of the drawing. *Grid* drawings are ones in which nodes and bends in edges are placed at the vertices of a rectangular grid. In *orthogonal grid* drawings, nodes are placed at vertices of a grid and edges follow the vertical and horizontal lines of the grid. *Straight-line* drawings are ones in which edges are drawn as straight line segments between nodes.

The goals of a layout algorithm can depend on the class of graphs being drawn. Directed and undirected graphs were defined in Chapter 1. A directed graph is also called a *digraph*. The *source node* of a directed edge is the node from which the edge emanates and the *target node* of a directed edge is the node at which the edge terminates. Digraphs without any cycles are called *directed acyclic graphs*, or *DAGs*. *Free trees* are connected undirected acyclic graphs. When one node of a tree is designated as the root, the tree are called *rooted*. DAGs are often drawn such that the edges all point in the same direction to illustrate flow; that is, such that for each edge the target node is above the source node. These types of drawings are called *upward*. In *hierarchical* drawings of digraphs or *hierarchies* the node set is partitioned into subsets such that each subset belongs to a level. *Planar* drawings of planar graphs are ones in which no edges in the drawing cross. Other conventions for drawing graphs have recently emerged [14]. Drawings of rooted trees with horizontal and vertical edges are called *h-v* drawings or *tip-over* drawings [10, 22], and tree drawings in which children are drawn as boxes inside parent boxes are called *inclusion* drawings

[22]. In *compound graphs*, edges are drawn as both polygonal line segments between nodes and inclusion relations [14].

2.1.1 Readability Issues

In applications where a drawing or layout of a graph G is created to be viewed and understood by a user, a main goal of the layout algorithm is to draw G so it conveys meaning to the user and is "readable". A poorly drawn graph can be misleading and cause confusion to a user. Several algorithms have been developed that produce drawings of graphs that are to be viewed and understood [14]. These algorithms attempt to produce drawings that satisfy certain readability criteria.

A difficulty in producing readable and aesthetically pleasing drawings of graphs is being able to define what properties make a graph readable and easy to understand. In [58], Messinger *et. al.* conclude that more research is needed in understanding human interpretations of graph layouts and what properties are identified by humans as aesthetically pleasing in graph layouts. The existing work in automatic graph layout is based upon a set of generally accepted objective criteria:

A1 avoid edge crossings [3, 15, 14, 25, 50]

A2 avoid bends in the edges [3, 15, 14, 80]

A3 keep the total length of the edges small [3, 7, 50]

A4 keep the maximum edge length small [7, 80]

A5 keep the area occupied by the drawing small [3, 7, 10, 15, 80, 50]

A6 display symmetries in the drawing [15, 51]

A7 distribute the nodes evenly [14, 25, 88]

A8 keep edge lengths uniform [14]

For the specific problem of upward drawings of rooted trees, the following properties are also considered important:

A9 nodes on the same level in the tree should be placed along horizontal lines [70, 91]

A10 there should be a minimum separation between two nodes on the same level [14]

A11 parent nodes should be centred above their children [70, 91]

A12 the width of the drawing should be as small as possible [90]

Displaying the notion of flow in directed graphs is important. In this case, another criterion for pleasing drawings of digraphs is:

A13 target nodes should be above source nodes [24, 25]

and in the case of hierarchical drawings

A14 nodes should be placed on horizontal lines in layers [24, 58, 71, 77]

When drawing a directed graph with cycles, a typical strategy is to identify a minimum set of edges that must be reversed so that the graph is acyclic and then use drawing algorithms for DAGs. Finding this set called the *minimum feedback arc set* is NP-hard [38]; however, heuristics can be used and algorithms for drawing DAGs can be incorporated for drawing directed graphs with cycles [24]. If illustrating flow is not important, algorithms for drawing general undirected graphs are used to draw

directed graphs with cycles [14]. Less study has been done on the problem of drawing general undirected graphs that do not have easily identifiable properties [14].

Achieving layouts that satisfy many of the criteria listed above is computationally intractable:

- minimizing the number of edge crossings in a drawing of a graph is NP-hard [23]
- minimizing the total length of the edges in planar orthogonal grid drawings is NP-hard [76]
- minimizing the maximum edge length in planar orthogonal grid drawings is NP-hard [4]
- minimizing the area occupied by a planar orthogonal grid drawing is NP-hard [76]
- drawing a graph so that all edge lengths are the same is NP-hard [48].

Because of the intractability of satisfying many of the optimization criteria, a natural approach to achieving readable layouts is to employ heuristics.

2.2 Measuring Heuristic Algorithms

When heuristics are used to draw graphs, there is often no guarantee that the given criteria are achieved in the resulting layouts. In some cases, it can be proved that the layouts cannot achieve worse than a certain value times the optimum. For example, it can be shown that, for certain values of n, the algorithm in [70] produces drawings

of rooted trees that are $\Theta(n)$ times wider than the best possible [78]. Upward planar straight-line grid drawings of DAGs can occupy an area that is exponential in the number of nodes in the graph [16]; however, an algorithm that allows bends in the edges produces layouts with no more than $2n-5$ total bends that occupy $O(n^2)$ area. An algorithm for drawing general undirected graphs is presented in [51]. A metric for symmetry is defined and it is proven that the algorithm draws graphs such that symmetries are displayed. A few layouts of small example graphs are included to illustrate that the algorithm works well.

The merit of some heuristic algorithms can be observed by the number of times it, and modifications of it, have been used in practical graph drawing applications. Sugiyama *et. al.*, use a heuristic approach to reduce edge crossings in hierarchical layouts of DAGs [77]. Nodes are placed on horizontal levels and a heuristic method called the *barycentric* method is used to reduce the number of edge crossings between levels. The order of nodes on each level is modified in turn starting with the top level and working down (the DOWN-UP procedure) or starting with the bottom level and working up (the UP-DOWN procedure) and the process is iterated for a fixed number of iterations or until the order does not change. The barycentric method is used to permute the nodes on each level. A measure of the average position of a node's neighbours on the next level is called the *barycenter*. The nodes on each level are ordered by their barycenter and the process is iterated. The success of the heuristics used can be seen by the number of variations on this method that have been used in drawing systems for applications [3, 8, 37, 58, 60, 64, 67, 71].

Other heuristic algorithms are evaluated by mere statements that the layouts seem to achieve the specified criteria in practise [36]. These statements are often

accompanied by examples of a few layouts produced by the algorithm [12, 26, 35, 49]. The algorithm presented by Eades in [26] uses a heuristic to display symmetries in general undirected graphs and keep edge lengths fairly uniform. It uses *spring-embedding* which is also known as *force-directed placement*. The graph is represented as a physical model of springs. The nodes are modeled as rings and edges represent springs between pairs of rings. Given an initial layout of the nodes, the forces of the springs on the rings cause the nodes to move until they reach a minimal energy state. The success of this method is seen by its popularity in the literature [12, 35, 43, 49]. In all of these cases, pictures of layouts are provided and statements are made but no proofs are given that the algorithms produce layouts that satisfy the specified aesthetic criteria and no quantitative measurements are made on the layouts. In [49], Kamada and Kawai state that providing proof that their force-directed algorithm produces layouts that satisfy the criteria is a challenging problem. A more detailed discussion of some of the force-directed placement algorithms listed here is presented in Chapter 5.

Quantitative measures have been used for evaluating layouts produced by heuristic algorithms [58, 60, 68, 88]. In [68], a force-directed placement algorithm is presented for placing components on a circuit board such that the sum of the wire lengths is small. The sum of the lengths of the edges in a minimum spanning tree of the graph components is used as the quantitative measure of the layouts and layouts produced by their system are compared against known optimal layouts. The layouts are also compared with layouts produced by other published methods.

An algorithm for producing hierarchical drawings of directed graphs by recursively drawing subgraphs then joining the drawings of the subgraphs together is presented

in [58]. The authors compare layouts using four different measures: execution time, number of crossings, total edge length, and the number of levels drawn. They compare their results with those of another algorithm presented in [71] on fourteen specific input graphs according to the different measures.

A graph layout algorithm for producing grid drawings of DAGs is presented in [60]. The DAGs are data flow graphs of programs from a parallel processing research project. The authors state that the quality of a drawing is difficult to measure in quantitative units; however, they measure the sum of the edge lengths in the layouts to compare each phase of their algorithm on four different data flow graphs.

A heuristic algorithm that tries to produce drawings with uniform edge lengths, evenly distributed nodes, and minimal edge crossings is presented in [88]. 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 [11] and [35] based on these measures.

2.2.1 Dynamic Graph Drawing and Layout Adjustment

Compared to the number of static graph drawing algorithms available, little work has been done in designing dynamic graph drawing algorithms and layout adjustment algorithms [14]. An early result in dynamic graph drawing is an algorithm for drawing trees that change [61]. More recently, algorithms that update drawings of trees, special types of digraphs, and general planar graphs in polylogarithmic time are presented in [9]. The authors prove that the algorithms produce drawings that satisfy certain properties. In [5], user specified constraints (in the form of linear equations of two

variables) are added to an existing layout algorithm to keep drawings stable. Pictures of updated layouts redrawn with the specified constraints are presented and the author states that in practise the constraints provide good layouts. An incremental graph layout algorithm that constructs a layout by adding nodes and edges one at a time is presented in [59]. The algorithm is heuristic and the authors state that it does well in practise. Pictures are provided of layouts produced by the algorithm.

In [19], two algorithms are presented that preserve the orthogonal ordering of the nodes in the graph. Nodes are pushed apart if the boxes modeling the nodes overlap. It is shown that there is no overlap of the nodes in the resulting layout and that the orthogonal ordering of the nodes is preserved.

2.3 Cluster Busting in Anchored Graph Drawing

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 The drawing D' should be in the same size drawing window as D,

CB2 The nodes in D' should be more evenly distributed inside the window 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 [19]. Note that criteria **CB1** and **CB2** do not take the edges of the graph into consideration. In is unclear from the description of criterion **CB3** whether or not the *shape* of the drawing includes the edges of the graph. In Chapter 4, we define measures of the shape that do not

include the edges. Furthermore, our algorithms ignore the edges of the graph when determining the new node positions. The definition of shape presented in [19] also excludes the edges of the graph. In [21] a *topology* measure of the user's mental map is defined that takes the edges of the graph into consideration; however, no layout adjustment algorithms are presented that preserve this measure. Further discussion on this matter is presented in Chapter 7.

As with several other methods for drawing general undirected graphs, our algorithms are heuristic. We cannot prove that the algorithms satisfy the criteria of cluster busting in anchored graph drawing all of the time; however, we present quantitative measures of the layouts which provide evidence that the algorithms satisfy the criteria in practise.

Chapter 3

Preliminaries

In this chapter, we present definitions and terminology that will be used throughout the dissertation. Definitions that are necessary for the understanding of concepts in a particular section of the dissertation are defined where they are used. The Glossary contains a list of the symbols along with their meaning. The Index contains a list of terms used and the page number on which they are defined. We give a brief outline of the results in this dissertation to help put the concepts and definitions presented in this chapter into context.

In this dissertation, we present two algorithms for cluster busting in anchored graph drawing. We also describe seven measurements that are made on layouts produced by the algorithms. The layout algorithms and the algorithms that do the measurements have been implemented. In order to evaluate the layout algorithms, we execute the implementations on sample input layouts and perform each of the measurements on the resulting layouts. To do this on a large number of input layouts, we generate random layouts. In this chapter, we define the format of the input and output for the layout algorithms. Several geometric tools are presented that are used

in the layout algorithms and the algorithms that do the measurements. Finally, we describe how the random layouts are generated.

3.1 Layout Algorithm Input and Output

A drawing $D = (G, S, W)$ is a graph $G = (V, E)$ along with a set of points $S = \{p_0, p_1, \ldots, p_{n-1}\}$ and a *window* W. There is a one-to-one mapping from the set of nodes V onto the set of points S such that the node v_i is at the point $p_i = (x_i, y_i)$ in the drawing D. The window is an *isothetic* rectangle in the plane; that is, the sides of W are parallel to the coordinate axes. The set S is *contained* in W; that is, each point of S is in the interior of W.

Nodes in a drawing can be displayed using a variety of shapes including rectangles, squares, and circles with or without text inside them. In our layout algorithms, the nodes are represented as points. These points can be considered as reference points such as the centres of circles or the bottom left hand corners of rectangles. Further discussion on this matter is presented in Chapter 7.

A drawing $D' = (G, S', W')$ is a transformation of D where $S' = \{p'_0, p'_1, \ldots, p'_{n-1}\}$ is a set of points $p'_i = (x'_i, y'_i)$ in the plane such that there is a one-to-one mapping from V onto the set S', and W' contains S'. The set S' is possibly different from the set S and W' is possibly different from W. When we talk about the *nodes in a drawing* D, we are referring to the mapping from V onto S. When we refer to the *nodes in a graph* G, we mean the set V.

The layout algorithms presented in Chapter 5 take as input a drawing $D = (G, S, W)$ of a graph G with $n \geq 3$ nodes such that no two nodes occupy the same position in the plane, and produce a new drawing $D' = (G, S', W')$ such that $W' = W$.

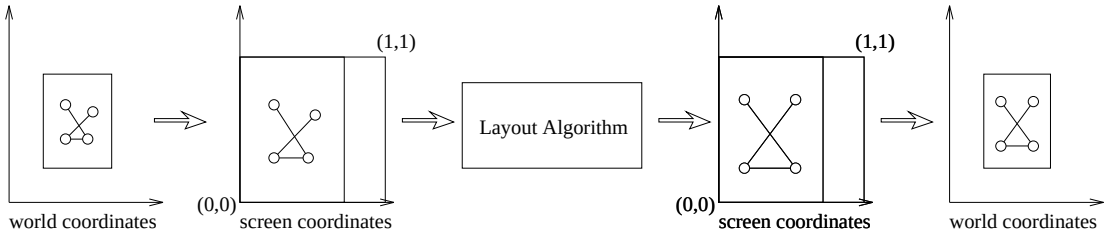

Figure 3.1: The layout system.

The window W has four sides which can be determined by two points: The bottom left corner and the top right corner. Let the bottom left corner be $bl = (x_{bl}, y_{bl})$ and let the top right corner be $tr = (x_{tr}, y_{tr})$.

We distinguish between world and screen coordinates. Using terminology from computer graphics, we say the input coordinates of the points and corners of W are given in *world coordinates* [33]. Prior to executing any layout algorithm, the world coordinates are transformed to the *screen coordinates* of our system. The screen coordinates of our system are between $(0, 0)$ and $(1, 1)$. After execution of a layout algorithm the drawing and window are transformed back to the world coordinates. Figure 3.1 illustrates the layout system described above. For the rest of the discussion we assume the set S and the corners of W are in screen coordinates.

3.2 Model of Computation

The analysis of all algorithms described in this dissertation assumes the *real RAM* model of computation. In the real RAM model each storage location is capable of holding a single real number, operations are executed one at a time, primitive operations are available at unit cost, and analytic functions such as trigonometric

and logarithm functions are executed in unit time [66].

3.3 Geometric Tools Used

Several geometric structures are used by our layout algorithms and the algorithms that do the measurements. In this section we describe these structures and present the worst-case running time of algorithms that compute them.

3.3.1 The Voronoi Diagram (VD)

Given a set $S = \{p_0, p_1, \ldots, p_{n-1}\}$ of n points, the *Voronoi Diagram (VD)* of S, $VD(S)$, is a partition of the plane into n convex regions such that the region associated with the point p_i is the set of points that are closer to p_i than to any other point in S. The distance between two points is the Euclidean distance[1]. More formally, define $H(p_i, p_j)$ as the half-plane containing p_i and bounded by the perpendicular bisector of p_i and p_j. The intersection of $n-1$ half-planes gives the *Voronoi polygon*, or *Voronoi region* of p_i: $Vor(p_i) = \cap_{i \neq j} H(p_i, p_j)$. Each polygon $Vor(p_i)$ is convex and not necessarily bounded with at most $n-1$ sides. The partition defined by these polygons is $VD(S)$. The vertices of the Voronoi polygons are called *Voronoi vertices*, and the edges of the polygons are called *Voronoi edges* [66, 89]. There are at most $3n-6$ Voronoi edges and at most $2n-5$ Voronoi vertices in a Voronoi diagram of n points [66]. Let the set of Voronoi vertices of S be $VV(S)$ and the set of Voronoi edges of S be $VE(S)$; $|VV(S)| = \Theta(n)$ and $|VE(S)| = \Theta(n)$.

The Voronoi diagram of n points can be computed in $O(n \log n)$ time[2] and the

[1] The Euclidean distance between two points p_i and p_j is given by $dist(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

[2] All logarithms are to the base 2 unless otherwise stated.

lower bound for computing the Voronoi diagram of n points is $\Omega(n \log n)$ [66]. An algorithm by Fortune computes the Voronoi diagram of S using the *sweepline technique* [34]. A geometric transformation of the Voronoi diagram is computed by sweeping a horizontal line upwards. When the line intersects a point p_i, it encounters the transformed Voronoi region of p_i for the first time. As the line is swept upwards, the transformed Voronoi regions that it intersects are maintained. An implementation by Kenny Wong of this algorithm is used by the implementations of our algorithms [94].

3.3.2 The Clipped Voronoi Diagram (CVD)

The *Clipped Voronoi diagram (CVD)* of a set S of n points, $CVD(S)$, is $VD(S)$ clipped within W. Denote the boundary of W by $bd(W)$, and let the set of four corners of W be $C(W)$. The set of vertices of $CVD(S)$ is defined as:

$$CVV(S) = (VV(S) \cap W) \cup C(W) \cup (VE(S) \cap bd(W)).$$

The set of edges of $CVD(S)$, $CVE(S)$, is the set of edges in $VE(S) \cap W$ plus the segments of $bd(W)$ between the points defined by $VE(S) \cap bd(W)$ and $C(W)$. Observe that $|CVV(S)| = \Theta(n)$ and $|CVE(S)| = \Theta(n)$ and the sets $CVV(S)$ and $CVE(S)$ can be computed in $O(n)$ time; therefore, the Clipped Voronoi Diagram of n points can be computed in $O(n \log n)$ time.

3.3.3 The Delaunay Triangulation (DT)

The Voronoi diagram of the set S is a subdivision of the plane into n regions. The *straight line dual* of $VD(S)$ is the graph whose node set is S and whose edge set is found by adding a straight line segment between every pair of points in S whose

Voronoi polygons share an edge. The straight line dual of the Voronoi diagram is a triangulation of S called the *Delaunay triangulation (DT)* [66]. The Delaunay triangulation of S, $DT(S)$, can be computed in $O(n \log n)$ time. If $VD(S)$ is available, $DT(S)$ can be computed in $O(n)$ time. The lower bound for computing the Delaunay triangulation of n points is $\Omega(n \log n)$ [66].

3.3.4 The Convex Hull (CH)

The *convex hull (CH)* of a set S of n points, $CH(S)$, is the smallest convex polygon $\mathcal{P}$ such that every point in S is on the boundary of $\mathcal{P}$ or in its interior. The convex hull of n points can be computed in $O(n \log n)$ time and the lower bound for computing the convex hull of n points is $\Omega(n \log n)$ [66].

3.3.5 The Euclidean Minimum Spanning Tree (EMST)

A *Euclidean minimum spanning tree (EMST)* of a set S of n points $EMST(S)$ is a tree of minimum total edge length whose nodes are the points in S [66]. An EMST of a set S of n points can be computed in $O(n)$ time given the Delaunay triangulation of S. Since $EMST(S)$ is a tree, it has $n-1$ edges. The lower bound for computing an EMST of n points is $\Omega(n \log n)$. Note that $EMST(S) \subset DT(S)$ [66].

3.3.6 The Closest Pair of Points

Given a set S of n points, a closest pair of points in S is a pair of points p_i and p_j, $i \neq j$, such that $dist(p_i, p_j)$ is minimized. A closest pair of points in S is joined by an edge in $EMST(S)$; therefore, a closest pair of points in S can be found by computing

$EMST(S)$ in $O(n \log n)$ time and finding a minimum length edge in the $\Theta(n)$ edges in $EMST(S)$.

3.4 Generating Random Node Placements

In order to evaluate and compare the layout algorithms presented in this dissertation, we generate random layouts and execute the layout algorithms on these random layouts. We let W be the unit square and generate a set S of n random points inside W. The screen and world coordinates of the random layouts are the same.

We evaluate and compare the layout algorithms on a variety of possible inputs by generating 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 $p_i = c_i$, $i = 0, 1, \ldots \mathcal{K} - 1$.

We evaluate our layout algorithms on layouts of size $n = 25$, $n = 50$, and $n = 100$. For $n = 25$, we use layouts with $\mathcal{K} = 1$, $\mathcal{K} = 5$, and $\mathcal{K} = 25$ clusters, for $n = 50$, we use layouts with $\mathcal{K} = 1$, $\mathcal{K} = 2$, $\mathcal{K} = 5$, $\mathcal{K} = 10$, $\mathcal{K} = 25$, and $\mathcal{K} = 50$ clusters, and for $n = 100$, we use layouts with $\mathcal{K} = 1$, $\mathcal{K} = 2$, $\mathcal{K} = 4$, $\mathcal{K} = 10$, $\mathcal{K} = 25$, $\mathcal{K} = 50$, and

$\mathcal{K} = 100$ clusters. Figure 3.2 shows example random layouts[3] for $n = 50$ and each sized cluster.

In some cases, two random layouts are required for comparison. The number of nodes must be the same but the type of clustering should be randomly chosen. To generate a random layout of n nodes with a random number of clusters, a random number $\mathcal{K}$ is generated between 1 and n and $\mathcal{K}+1$ clusters are produced as described above such that $\lfloor \frac{n}{\mathcal{K}} \rfloor$ nodes are in $\mathcal{K}$ of the clusters and $n \bmod \mathcal{K}$ nodes are in the $(\mathcal{K}+1)^{st}$ cluster.

[3] The layouts displayed in this dissertation were produced using GLAD [69].

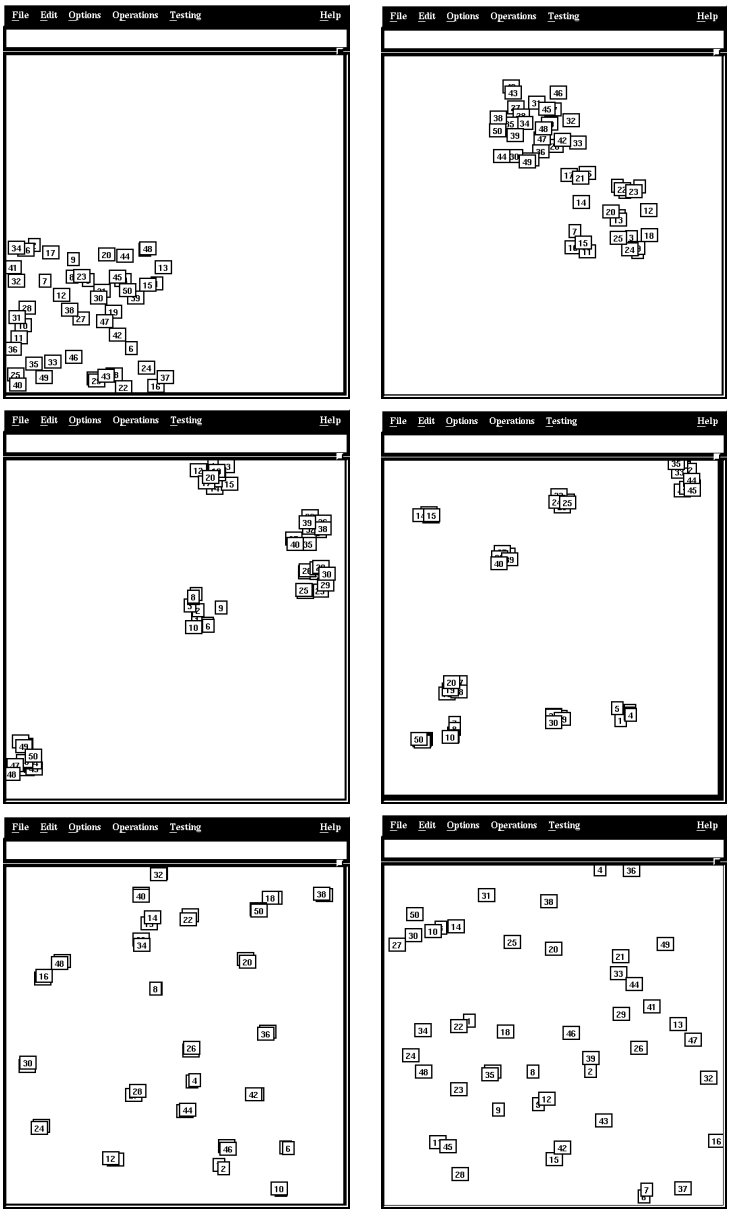

Figure 3.2: Random layouts for $n = 50$ and $\mathcal{K} = 1, 2, 5, 10, 25, 50$.

Chapter 4

Measuring Similarity and Distribution

Informally, the goal of cluster busting in anchored graph drawing algorithms is to distribute more evenly the nodes while keeping the layout similar to the original layout and inside the same size drawing window. In Chapter 2, we defined the following criteria that an adjusted drawing D' of G should satisfy:

CB1 $W' = W$,

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

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

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 chapter we present models of distribution and similarity and define measurements of layouts based on these models. These measurements are used to evaluate the layouts

produced by our algorithms. The algorithms we describe in Chapter 5 are iterative and we also use these measurements to evaluate layouts after each iteration of the algorithms to determine when to stop.

4.1 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 $bd(W)$. We have determined through observation that if the ideal distance between two nodes is d_I then the ideal distance between a node and $bd(W)$ is $0.5d_I$. Two placements of four points inside W are shown in Figure 4.1. We feel the points in Figure 4.1(a) are more evenly distributed inside W than the points in Figure 4.1(b). On the left, the distance between the points and the sides of W is roughly half the distance between adjacent pairs of points. On the right, the distance between the points and the sides of W is about the same as the distance between adjacent pairs of points.

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

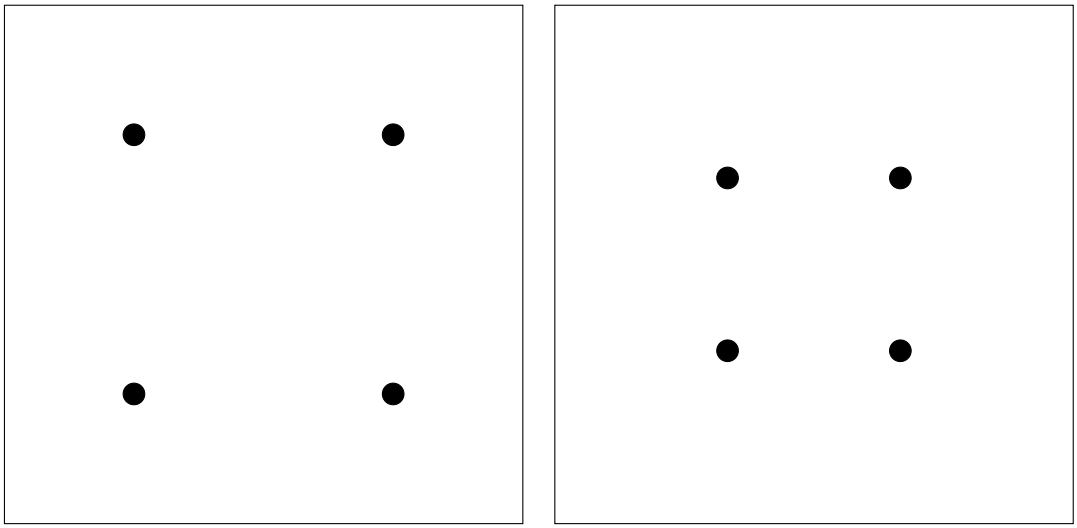

Figure 4.1: Two ways of drawing four points in a square.

$\mathcal{E}$.

We present two models of the distribution: the *Closest Pair (CP)* model and the *Force Minimization (FM)* model. The measure of distribution according to the CP model is the distance between a closest pair of points in S. The measure of distribution according to the FM model is used in [88] to quantify when a set of nodes is evenly distributed. This measure is based on the model of repelling forces between all pairs of nodes.

4.1.1 Closest Pair (CP) Model of Distribution

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. We include the distance between any

point and $bd(W)$. The distance between a node p_i and each side of W is represented as the difference in x and y between p_i and the points bl and tr; for example, the distance from p_i to the left side of W is given by $|x_i - x_{bl}|$. The measure of distribution according to the CP model is defined as:

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

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 problem of maximizing the minimum distance between any pair of n points in a given region W is the same as *packing* n circles with equal radius r inside W such that the circles do not overlap and r is a maximum [72, 81]. If a packing of n equal sized circles in a square can be found such that the radius of the circles is maximized, then the locations of the centres of the circles give a layout of n points such that the minimum distance between any two is maximized. There are several results for packing equal sized circles in the unit square [13, 39, 72, 73]. In [73], a packing is given for $n = 8$ circles such that the circle diameter is $2r = \sqrt{2 - \sqrt{3}} \simeq 0.518$, and in [72], packings are given for $n = 2, \ldots, 9$, and for several $n \leq 340$. Some of these results have been improved in [13]. As far as we know the problem of packing n equal sized circles in a unit square such that their radius is maximized is open for arbitrary n [92].

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 [66].

4.1.2 Force Minimization (FM) Model

An algorithm is presented in [88] for drawing general graphs. Quantitative measurements are performed on layouts produced by the algorithm. One of the measurements is used to judge node distribution. The value computed represents the sum of the node repulsion forces used in the force-directed placement algorithms by Davidson and Harel [12] and Fruchterman and Reingold [35]: The repelling force computed between a pair of nodes p_i and p_j, $i \neq j$, is proportional to:

$$f(p_i, p_j) = \frac{1}{(dist(p_i, p_j))^2}$$

and in [88], the distribution of the nodes is measured by computing the sum of $f(p_i, p_j)$ for every pair of nodes p_i and p_j, $i \neq j$. The idea behind this measure is that when the nodes are evenly distributed, the system of forces will be close to a global minimum; therefore, when the sum of the acting forces is minimized, the nodes are evenly distributed.

We define a model of distribution based on this idea called the Force Minimization (FM) model. Since the distance between each node and the sides of W is included in this model, we define the following function:

$$F(S) = \sum_{i=0}^{n-2} \left(\sum_{j=i+1}^{n-1} \frac{1}{(dist(p_i, p_j))^2} \right) + \sum_{i=0}^{n-1} \sum_{w \in \{tr, bl\}} \left(\frac{1}{(2|x_i - x_w|)^2} + \frac{1}{(2|y_i - y_w|)^2} \right)$$

Note that $F(S)$ *decreases* as the nodes become more evenly distributed according to the FM model. Since we define $\Xi_{\mathcal{E}}(S)$ to increase as the nodes become more evenly distributed under the model $\mathcal{E}$, we let

$$\Xi_{FM}(S) = (F(S))^{-1}.$$

It follows that if $\Xi_{FM}(S) > \Xi_{FM}(S')$, the nodes in S are more evenly distributed according to FM model than those in S'. Note that since $F(S) \neq 0.0$, $\Xi_{FM}(S)$

is defined for all S. Note also that all of the distances are greater than 0 because no two nodes are at the same point in the initial layout and no node is on $bd(W)$. Furthermore, our layout algorithms do not move any two nodes to the same point and do not move any node onto $bd(W)$.

Since all the node positions are inside W, $dist(p,q) \leq \sqrt{2}$ for any two nodes p and q, and $F(S)$ has a very large numeric value while $\Xi_{FM}(S)$ has a very small numeric value. Therefore, we scale the drawing by multiplying each node position by 100.0 prior to computing $\Xi_{FM}(S)$. The cost of computing $\Xi_{FM}(S)$ for a drawing D with n nodes is $O(n^2)$.

4.2 Measuring Similarity

Criterion **CB3** states that the general shape of D' should be the same as D. In [20] and [21], 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 exactly 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 five 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, let the difference measure between two sets of points S and S' according to the model M be:

$$\mathcal{D}_M(S, S'),$$

where

$$0.0 \leq \mathcal{D}_M(S, S') \leq 1.0.$$

If $\mathcal{D}_M(S, S') = 0.0$, then there is no difference between the drawings D and D' according to the model M. In this case, 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 $\mathcal{D}_M(S, S') = 1.0$.

We make no claims that the models presented in this section are actual judges of similarity according to how humans would judge similarity between two drawings. We use measures based on the models as a means of evaluating and comparing our layout algorithms as well as for determining when to stop iterating our algorithms. Several fields of study both outside and inside computing science use judgements of similarity for classification, identification, and comparison. The 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, the above definition of similarity is termed *normative* similarity [42]. 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 [42].

Psychologists are interested in measuring how humans perceive similarity between objects. Goldmeier looks at how humans perceive similarity by performing experiments in which subjects are given a model object and other objects, and asked to

evaluate how similar the objects are to the model object [40]. Based on these experiments, he concludes that angular and distance relationships within a figure are crucial in judging similarity, and features such as symmetry and grouping of parts are also important.

Most of the mental map models presented in this section are based on features that Goldmeier concludes are used when observing one figure to be similar to another. A measure based on the *All Distances (AD)* model computes differences in distance relationships between every pair of node positions in the two drawings. Symmetries in the node positions of the two drawings are compared using a measure based on the λ-*Matrix (λM)* model. The λ-matrix of a set of points is defined in Section 4.2.2.

As suggested in [20], preserving some notion of clusters of the nodes in a drawing D can be done by maintaining the relationships in a proximity graph of the nodes in D. The use of this type of mental map model is supported by the conclusion of Goldmeier that the grouping of parts is an important feature in determining similarity. The idea of using proximity graphs to define the "shape" of a set of points is also used in pattern recognition [82, 84, 85]. A measure based on the *Delaunay Triangulation Edges (DE)* model computes changes in the Delaunay triangulations of the node positions. We also present a measure that computes the sum of the distances that every node in D has moved to produce D'. We base this measure on the *Distances Moved (DM)* model. Finally, we define a measure based on the *Orthogonal Ordering (OO)* model.

Each of the measures are described in more detail in the following sections. For each measure, we describe the algorithm we use for computing the measure and its complexity, and discuss some properties of the measure. 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 measure of difference according to the model M is given by

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

In this way, we ensure that $0.0 \leq \mathcal{D}_M(S, S') \leq 1.0$. 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.

4.2.1 All Distances (AD) Model

Given two drawings D and D' of a graph G with n nodes, the number of changes under the AD model is computed by finding the sum of the differences in distance between every pair of nodes in the two drawings:

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

The time to compute $\delta_{AD}(S, S')$ is $O(n^2)$ for D and D' with n nodes since there are $\Theta(n^2)$ pairs of nodes.

There are $\frac{n(n-1)}{2}$ terms in the summation and each term has a value that is at most $\sqrt{2}$ because S and S' are within the unit square; therefore, we define the upper bound on $\delta_{AD}(S, S')$ for n nodes as:

$$UB_{AD}(n) = \frac{n(n-1)}{2}\sqrt{2}$$

and let:

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

Note that $\delta_{AD}(S, S') < UB_{AD}(n)$ for $n > 2$, but when $n = 2$ and p_0 and p_1 are at opposite corners of the unit square in D, and $p'_0 = p'_1$ in D', $\mathcal{D}_{AD}(S, S') = 1.0$. The only situation in which $\mathcal{D}_{AD}(S, S') = 0.0$ is when D' is a translation, rotation, and/or reflection of D.

4.2.2 λ-Matrix (λM) Model

In [41], Goodman and P

and S', it can be determined from their λ-matrices if they have the same order type.

Several possible applications of this technique are presented in [41]. It is suggested that in pattern recognition new images can be compared to a given image by comparing their λ-matrices. If the new image has preserved the relative orientations of points (such as a view of an object from a slightly different perspective than the standard view), its λ-matrix will be the same as that of the original image. If the new image has undergone some local changes (such as in hand-written character analysis), its λ-matrix will not be the same as that of the original image but will *correlate highly* with it. As far as we know, the application of λ-matrices to pattern recognition has not been explored further [65].

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':

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

An upper bound on $\delta_{\lambda M}(S, S')$ can be determined by observing that the elements in each row of a λ-matrix depend on the elements in the other rows of the λ-matrix; however, if the maximum difference for a single row independent of the other rows is achievable in each row for two configurations S and S' then $\delta_{\lambda M}(S, S')$ is a maximum. Let $\lambda_{\mathbf{i}} = (a_0 a_1 \ldots a_{n-2})$ be the sequence of elements of the i^{th} row of the λ-matrix of S not including $\lambda(p_i, p_i)$, and let $\lambda'_{\mathbf{i}} = (a'_0 a'_1 \ldots a'_{n-2})$ be the sequence of elements of the i^{th} row of the λ-matrix of S' not including $\lambda(p'_i, p'_i)$. Note that $\lambda_{\mathbf{i}}$ and $\lambda'_{\mathbf{i}}$ are sequences of $n-1$ integers such that $0 \leq a_j, a'_j \leq n-2$, $j = 0, 1, \ldots, n-2$.

For now, assume that no three nodes are collinear. This assumption will be relaxed later. The following lemma characterizes those rows of a λ-matrix that contain the elements 0 and $n-2$. Consider $\lambda_{\mathbf{i}}$ and a similar argument holds for $\lambda'_{\mathbf{i}}$.

Lemma 4.2.1 *The node p_i is on the convex hull of S if and only if there are elements $n-2 \in \lambda_{\mathbf{i}}$ and $0 \in \lambda_{\mathbf{i}}$.*

Proof: Let p_i be on the convex hull of S and, without loss of generality, let its two neighbours be p_{i-1} and p_{i+1} in the clockwise and counterclockwise directions, respectively. Since the line segments $\overline{p_{i-1}p_i}$ and $\overline{p_ip_{i+1}}$ are convex hull edges, the lines through them are supporting lines of S and $\lambda(p_i, p_{i-1}) = 0$ and $\lambda(p_i, p_{i+1}) = n-2$.

If there are two nodes p_{i-1} and p_{i+1} such that $\lambda(p_i, p_{i-1}) = 0$ and $\lambda(p_i, p_{i+1}) = n-2$, then the lines through p_i and p_{i-1} and through p_i and p_{i+1} are supporting lines of S; therefore, the line segments $\overline{p_{i-1}p_i}$ and $\overline{p_ip_{i+1}}$ must be edges of the convex hull of S, and p_i is on the convex hull of S. □

Let the sum of the differences in the elements in row i be $\Sigma_i = \sum_{j=0}^{n-2} |a_j - a'_j|$.

Lemma 4.2.2 *If Σ_i is a maximum then p_i is on the convex hull of S and p_i is not collinear with two or more other nodes.*

Proof: Assume that Σ_i is a maximum and that p_i is not on the convex hull of S. By Lemma 4.2.1, if p_i is not on the convex hull of S then $0 \notin \lambda_{\mathbf{i}}$ and $n-2 \notin \lambda_{\mathbf{i}}$. If $0 \notin \lambda_{\mathbf{i}}$ and $n-2 \notin \lambda_{\mathbf{i}}$ then there is an $a_k \in \lambda_{\mathbf{i}}$ such that $a_k = a_l$ for some other $a_l \in \lambda_{\mathbf{i}}$, $0 < a_k, a_l < n-2$. Consider $a'_k \in \lambda'_{\mathbf{i}}$ and $a'_l \in \lambda'_{\mathbf{i}}$. For any value of a'_k and a'_l in the range $[0, n-2]$, the sum $|a'_k - a_k| + |a'_l - a_l|$ will increase by either setting a_k or a_l to 0 or $n-2$.

Assume that Σ_i is a maximum with p_i on the convex hull of S and p_{i-1} and p_{i+1} collinear with p_i. Then $a_{i+1} = n-3$ and $a_{i-1} = 0$. For any value of a'_{i+1} and a'_{i-1}, the sum $|a_{i+1} - a'_{i+1}| + |a_{i-1} - a'_{i-1}|$ increases by either setting $a_{i+1} = n-2$ or by setting $a_{i+1} = 0$ and $a_{i-1} = n-2$. □

Therefore, if Σ_i is a maximum, then p_i is on the convex hull of S and p_i is not collinear with two or more other nodes in S. A symmetric argument can be given to show that if Σ_i is a maximum then p'_i is on the convex hull of S' and p'_i is not collinear with two or more other nodes in S'. If Σ_i is maximized for each row i of the λ-matrices of S and S' then every node in S and S' must be a convex hull node and no three nodes can be collinear.

Lemma 4.2.3 *If every node in S is on the convex hull of S and no three nodes are collinear then $\lambda_{\mathbf{i}}$ contains the elements $0, 1, \ldots, n-2$ for $i = 0, 1, \ldots n-1$.*

Proof: Let p_{i-1} and p_{i+1} be the clockwise and counterclockwise neighbours of p_i on the convex hull of S, respectively. The line from p_i to p_{i+1} has $n-2$ nodes to its left. As this line is rotated in a counterclockwise direction from p_i, it encounters each node of S and the count of nodes to its left decreases by exactly one until it reaches p_{i-1} when the count is 0. □

Without loss of generality, assume that $a_j = j$ in $\lambda_{\mathbf{i}}$ for $j = 0, 1, \ldots, n-2$. We can represent the ordering of the elements in $\lambda'_{\mathbf{i}}$ by the following permutation:

$$\Pi_{\lambda'_{\mathbf{i}}} = \begin{pmatrix} 0 & 1 & \ldots & n-2 \\ a'_0 & a'_1 & \ldots & a'_{n-2} \end{pmatrix}$$

All permutations for which $\Sigma_i = \sum_{j=0}^{n-2} |a'_j - j|$ is a maximum are defined in Lemma 4.2.5 in Section 4.2.5. The permutation

$$\Pi_{\lambda'_{\mathbf{i}}} = \begin{pmatrix} 0 & 1 & \ldots & n-2 \\ n-2 & n-3 & \ldots & 0 \end{pmatrix}$$

satisfies the conditions in Lemma 4.2.5 and occurs for each row i when every node in S and S' is a convex hull node and the ordering of the nodes on the convex hull of S' is the opposite ordering of those on the convex hull of S.

Lemma 4.2.4 *Given two sets of points S and S' with $n \geq 3$ nodes,*

$$\delta_{\lambda M}(S, S') \leq n \left\lfloor \frac{(n-1)^2}{2} \right\rfloor$$

Proof: The previous argument shows that the difference between entries in each of row i in the λ-matrix of S and the λ-matrix of S' is a maximum when every node in S and S' is a convex hull node and the ordering of the nodes on the convex hull of S' is the opposite ordering of those on the convex hull of S; that is, when $a'_j = n-2-a_j$, $j = 0, 1, \ldots, n-2$ for each row. By Lemma 4.2.5 in Section 4.2.5 the value of $\sum_{j=0}^{n-2} |j - (n-2-j)| = \lfloor \frac{(n-2)^2}{2} \rfloor$. If $\sum_{j=0}^{n-1} |\lambda(p_i, p_j) - \lambda(p'_i, p'_j)|$ is a maximum for each row i then $\delta_{\lambda M}(S, S') = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} |\lambda(p_i, p_j) - \lambda(p'_i, p'_j)|$ is a maximum and $\delta_{\lambda M}(S, S') \leq n \left\lfloor \frac{(n-1)^2}{2} \right\rfloor$. □

Therefore, we define the upper bound on $\delta_{\lambda M}(S, S')$ for n nodes as:

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

and let:

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

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. Two sets of points S and P such that $\mathcal{D}_{\lambda M}(e$

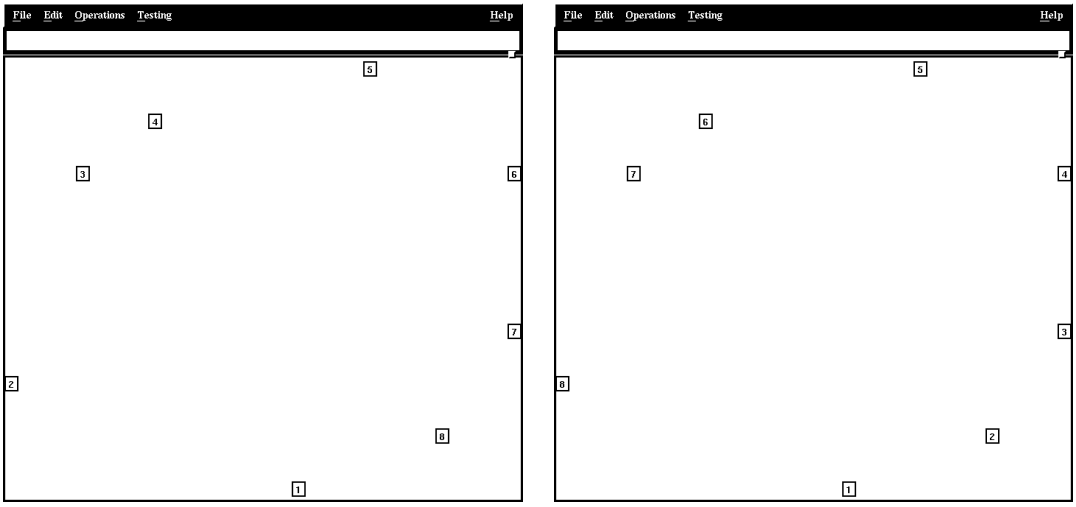

Figure 4.2: Two sets of points S and P such that $\mathcal{D}_{\lambda M}(S, P) = 1.0$.

in [82, 84, 85, 86, 87], including the Euclidean minimum spanning tree (EMST), the Delaunay triangulation (DT), the *relative neighbourhood graph (RNG)* [83], the *sphere of influence graph (SIG)* [87], and the *α-hull* [27]. The *lune* defined by two points p_i and p_j is found by intersecting the circle centred at p_i with radius $r = dist(p_i, p_j)$ with the circle centred at p_j with radius r. In the relative neighbourhood graph of a set S of points, $RNG(S)$, there is an edge between two points p_i and p_j in S if and only if no other point of S lies in the lune defined by p_i and p_j. Let C_i be the circle centred at p_i with radius equal to the distance from p_i to its nearest neighbour in S. In the sphere of influence graph of a set S of points, $SIG(S)$, there is an edge between two points p_i and p_j in S if and only if C_i and C_j intersect. The *α-hull* of a set of points is a generalization of the convex hull of a set of points [27]. Let α be a negative real number. Then the α-hull of a set of points is the intersection of all closed complements of discs with radius $-1/\alpha$ that contain all of the points of S.

Given a proximity graph $PG(S)$ of n points let $P_{PG}(p_i, p_j) = 1$ if there is an edge

between p_i and p_j in $PG(S)$, and let $P_{PG}(p_i, p_j) = 0$ otherwise for $i, j = 0, 1, \ldots, n-1$, $i \neq j$. A difference measure based on the *proximity graph edges (PE)* model is computed by finding the sum of the differences in $P_{PG}(p_i, p_j)$ and $P_{PG}(p'_i, p'_j)$:

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

The time to compute $\delta_{PE}(S, S')$ is

$$\max\{O(T_{PG}(n)), O(S_{PG} \log S_{PG})\}$$

where $T_{PG}(n)$ is the time to compute the given proximity graph on n nodes, and $S_{PG}(n)$ is the maximum number of edges in the given proximity graph on n nodes. First, the proximity graphs of S and S' are computed in in $O(T_{PG}(n))$ producing two lists of edges E_{PG} and E'_{PG} of length $O(S_{PG}(n))$ each: $E_{PG} = \{\{p_i, p_j\} | i < j$ and p_i and p_j are joined by an edge in $PG(S)\}$, and $E'_{PG} = \{\{p'_i, p'_j\} | i < j$ and p'_i and p'_j are joined by an edge in $PG(S')\}$. These two lists are sorted in $O(S_{PG}(n) \log S_{PG}(n))$ time such that $(p_i, p_j) < (p_k, p_l)$ iff $p_i \leq p_k$ and $p_j < p_l$, and compared in $O(S_{PG}(n))$ time.

Delaunay Triangulation Edge (DE) Model

We have chosen the Delaunay triangulation as the proximity graph to use as a model of the user's mental map. Recall that $T_{DT}(n) = O(n \log n)$, $S_{DT}(n) = O(n)$; therefore, $\delta_{DT}(S, S')$ can be computed in $O(n \log n)$ time. There are at most $3n - 6$ edges in the Delaunay triangulation of n points [66] and if $DT(S)$ and $DT(S')$ have no edges in common, then there are at most $2(3n - 6)$ changes in $\delta_{DE}(S, S')$; therefore, we let the upper bound on $\delta_{DE}(S, S')$ with n nodes be:

$$UB_{DE}(n) = 6n - 12$$

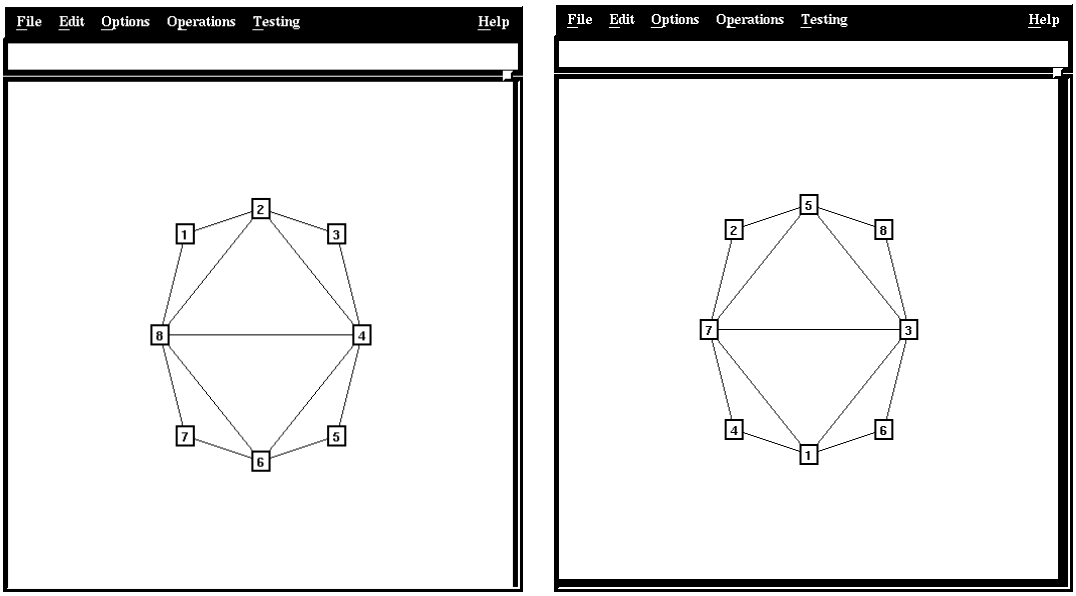

Figure 4.3: Two sets of points that have no Delaunay edges in common.

and let:

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

The upper bound is achieved when exactly three nodes are on the convex hull. Figure 4.3 shows two sets of points that have no Delaunay edges in common. The value of $\mathcal{D}_{DE}(S, P)$ for these two layouts S and P is not 1.0. In fact, it is 0.7222222 because the number of Delaunay edges is 13 instead of the maximum possible of 18 when there are three nodes on the convex hull. In [17], Dillencourt shows that any inner triangulation of a polygon is *realizable* as a Delaunay triangulation. A triangulation T is realizable as a DT if there exists a DT that is combinatorially equivalent to T. In [17], it is also shown that *spined triangles* in which a chain of triangles all share a common edge are realizable as DTs. Figure 4.4 shows an example of a spined triangle. Even though there are $3n - 6$ edges in this triangulation, there is no way to create a

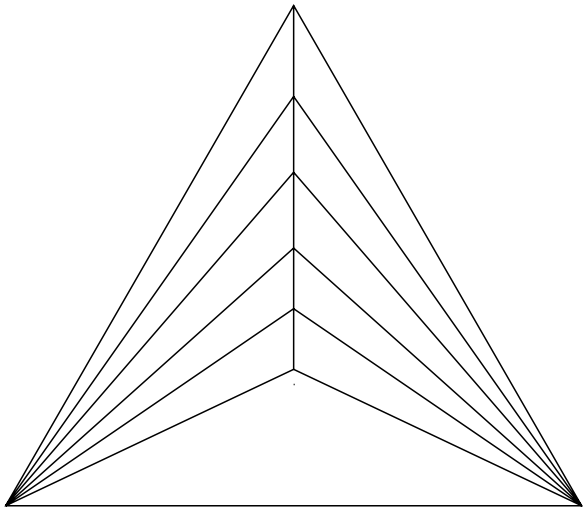

Figure 4.4: An example of a spined triangle.

triangulation with the same number of nodes and such that no edges are in common since the two bottom nodes are connected to all other nodes. It is an open problem to characterize two Delaunay triangulations with $3n - 6$ edges that have no edges in common.

4.2.4 Distances Moved (DM) Model

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 of the points move from some corner of W to a diagonal corner. The only time that $\mathcal{D}_{DM}(S, S') = 0.0$ is when $S' = S$.

4.2.5 Orthogonal Ordering (OO) Model

The difference measure based on the OO model computes how different two sets of points are in their x- and y-orderings. Let $R_x(p_i)$ be the *rank* of p_i in the x-direction and let $R_y(p_i)$ be the rank of p_i in the y-direction. The rank of an element in sorted order is the number of elements preceding it plus one. The value of $\delta_{OO}(S, S')$ is computed by summing the difference in rank of p_i and p'_i in the x- and y-directions:

$$\delta_{OO}(S, S') = \sum_{i=0}^{n-1} (|R_x(p_i) - R_x(p'_i)| + |R_y(p_i) - R_y(p'_i)|)$$

The time to compute $\delta_{OO}(S, S')$ is $O(n \log n)$ since there are n nodes, sorting can be done in $O(n \log n)$ time, and the difference in rank can be computed in $O(n)$ time.

We now discuss the upper bound on $\delta_{OO}(S, S')$. Without loss of generality, we limit the following discussion to the x-ordering of the points. Let the x-values of n points be given in non-decreasing order. If $x_i = x_j$ we impose an ordering on them such that x_j follows x_i in the ordering if and only if $i < j$. Without loss of generality, let $R_x(p_i) = i$ and $R_x(p'_i) = a_i$, $i = 0, 1, \ldots, n-1$. Therefore,

$$\Pi_{S,S'} = \begin{pmatrix} 0 & 1 & \ldots & n-1 \\ a_0 & a_1 & \ldots & a_{n-1} \end{pmatrix}$$

is the permutation of the indices that takes the x-ordering of the points in S to the x-ordering of the points in S'. Let the sum of the differences in the ranks in

the x-direction be: $\delta_x(S, S') = \sum_{i=0}^{n-1} |a_i - i|$. The following lemma characterizes the permutations for which $\delta_x(S, S')$ is maximized. Lemma 4.2.5 is the result of joint work with Henk Meijer [57].

Lemma 4.2.5 *Let* $I = \{0, 1, \ldots, \lfloor \frac{n}{2} \rfloor - 1\}$, *and let* $J = \{\lceil \frac{n}{2} \rceil, \ldots, n-1\}$. *If* n *is odd, let* $K = \{\frac{n-1}{2}\}$. *The maximum value of* $\delta_x(S, S')$ *is reached in permutations with* $a_i \in J$ *for* $i \in I$ *and* $a_j \in I$ *for* $j \in J$. *If* n *is odd then the maximum value of* $\delta_x(S, S')$ *is achieved by fixing* $a_k \in K$ *for* $k = \frac{n-1}{2}$ *or by swapping* a_k *with any other element in one of the permutations just defined.*

Proof: We first prove the case when n is even and then prove the case when n is odd.

Case 1 (n even): If there exists an $a_i \in I$, $i \in I$ then there is an $a_j \in J$, $j \in J$. Swapping a_i and a_j results in $\delta_x(S, S')$ increasing since $\max\{a_i, i\} < \min\{a_j, j\}$ which means that there are four cases to consider:

1. $a_i \leq i < a_j \leq j$
2. $i \leq a_i < a_j \leq j$
3. $a_i \leq i < j \leq a_j$
4. $i \leq a_i < j \leq a_j$

In all cases, it can be shown that $|a_i - i| + |a_j - j| < |a_i - j| + |a_j - i|$.

It is easy to see that any permutation of $a_i \in J$, $i \in I$ or $a_j \in I$, $j \in J$ gives the same value of $\delta_x(S, S')$ since swapping two elements $a_i, a_{i+1} \in J$ results in a_i becoming 1 position closer to i and a_{i+1} becoming 1 position further from $i+1$, leaving $\delta_x(S, S')$ unchanged.

Case 2 (n odd): By the argument above, if $a_{\frac{n-1}{2}} = \frac{n-1}{2}$ the maximum value of $\delta_x(S, S')$ (with $a_{\frac{n-1}{2}} = \frac{n-1}{2}$) occurs when $a_i \in J$ for $i \in I$ and $a_j \in I$ for $j \in J$. Given such a permutation, it is easy to see that the value of $\delta_x(S, S')$ remains unchanged if $a_{\frac{n-1}{2}}$ is swapped with another element a_l in the permutation since either $|a_{\frac{n-1}{2}} - \frac{n-1}{2}|$ increases by $\frac{n-1}{2}$ and $|a_l - l|$ decreases by the same amount or vice versa. □

By Lemma 4.2.5, we can find a permutation for which $\delta_x(S, S')$ (similarly, $\delta_y(S, S')$) is maximized. Let the permutation of the indices of S' sorted by x-coordinate be:

$$\Pi_{S,S'} = \begin{pmatrix} 0 & 1 & \dots & n-1 \\ n-1 & n-2 & \dots & 0 \end{pmatrix}$$

This permutation satisfies the criteria given in Lemma 4.2.5. The sum of the differences in the ranks for this permutation is given by:

$$2 \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - 1} (n - 1 - 2i) = \left\lfloor \frac{n^2}{2} \right\rfloor$$

Therefore, we define the upper bound on $\delta_{OO}(S, S')$ for n nodes as:

$$UB_{OO}(n) = 2 \left\lfloor \frac{n^2}{2} \right\rfloor$$

and let

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

The upper bound is achieved when the points in S are permuted in the manner described in Lemma 4.2.5 in both the x- and y-directions. There are $((\frac{n}{2})!)^4$ orderings of n points for which $\mathcal{D}_{OO}(S, P) = 1.0$ when n is even, and $(n\lfloor \frac{n}{2} \rfloor !)^4$ orderings of n points for which $\mathcal{D}_{OO}(S, P) = 1.0$ when n is odd. Figure 4.5 shows two sets of points S and P such that $\mathcal{D}_{OO}(S, P) = 1.0$.

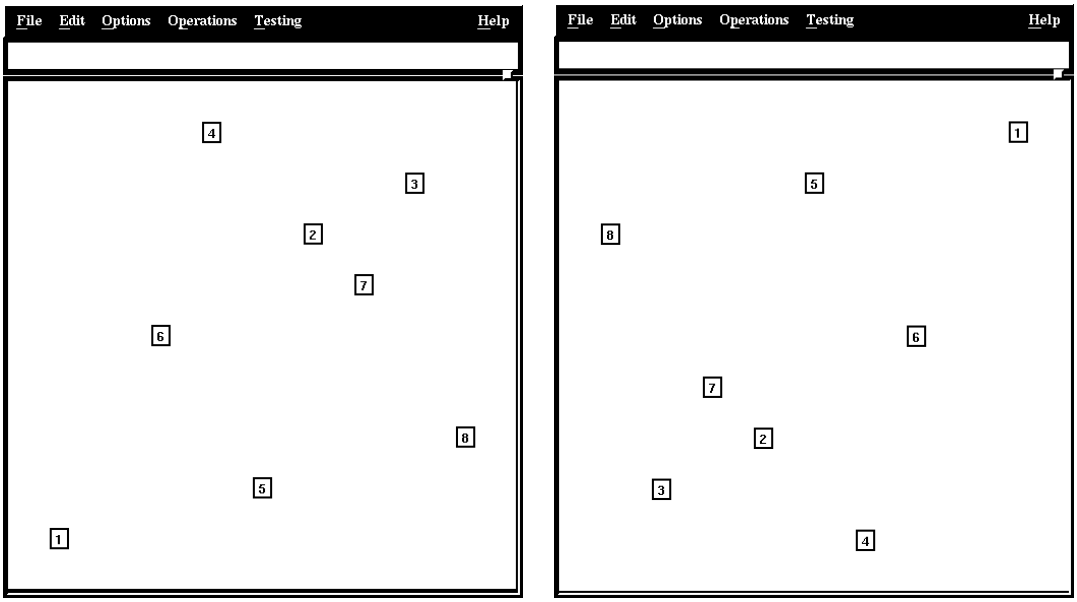

Figure 4.5: Two sets of points S and P such that $\mathcal{D}_{OO}(S, P) = 1.0$.

4.3 Conclusion

In this chapter we presented two measurements of distribution and five measurements of similarity. These measurements are used to evaluate our layout algorithms and to determine stopping conditions of our algorithms. A discussion of their use for these purposes is presented in Chapter 6. Tables 4.1 and 4.2 summarize the results presented in this chapter.

Measure ($\mathcal{E}$)	$\Xi_{\mathcal{E}}(S)$	Time
CP	$\min\{\min_{i\neq j}\{dist(p_i,p_j)\}, \min_{w\in\{tr,bl\}}\{2\lvert x_i-x_w\rvert, 2\lvert y_i-y_w\rvert\}\}$	$O(n\log n)$
FM	$(F(S))^{-1}$, where $F(S)=\sum_{i=0}^{n-1}\left(\sum_{j=i+1}^{n-1}\frac{1}{(dist(p_i,p_j))^2}+\sum_{w\in\{tr,bl\}}\left(\frac{1}{(2\lvert x_i-x_w\rvert)^2}+\frac{1}{(2\lvert y_i-y_w\rvert)^2}\right)\right)$	$O(n^2)$

Table 4.1: Overview of the Distribution Measures.

Measure (M)	$\delta_M(S,S')$	$UB_M(n)$	Time
AD	$\sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1}\lvert dist(p_i,p_j)-dist(p'_i,p'_j)\rvert$	$\frac{n(n-1)}{2}\sqrt{2}$	$O(n^2)$
λM	$\sum_{i=0}^{n-1}\sum_{i=0}^{n-1}\lvert\lambda(p_i,p_j)-\lambda(p'_i,p'_j)\rvert$	$n\lfloor\frac{(n-1)^2}{2}\rfloor$	$O(n^2)$
DE	$\sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1}\lvert P_{DE}(p_i,p_j)-P_{DE}(p'_i,p'_j)\rvert$	$2(3n-6)$	$O(n\log n)$
DM	$\sum_{i=0}^{n-1} dist(p_i,p'_i)$	$\sqrt{2}n$	$O(n)$
OO	$\sum_{i=0}^{n-1}(\lvert R_x(p_i)-R_x(p'_i)\rvert+\lvert R_y(p_i)-R_y(p'_i)\rvert)$	$2\lfloor\frac{n^2}{2}\rfloor$	$O(n\log n)$

Table 4.2: Overview of the Similarity Measures.

Chapter 5

The Layout Algorithms

In this chapter, two layout algorithms for cluster busting in anchored graph drawing are presented. Both of the algorithms 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 statisfied. However, we cannot prove that the heuristics produce layouts such that criteria **CB2** and **CB3** are satisfied in all cases. In Chapter 6 we evaluate and compare the algorithms according to these two criteria.

The algorithms take as input a drawing $D = (G, S, W)$ of a graph G, and produce as output a new drawing $D' = (G, S', W)$ of G. 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 difference between S and S^t. These measurements are used to determine when the algorithms should stop iterating. If no measurements are made on the layouts, the algorithms iterate a fixed number of times. Let $\Gamma \subseteq \{CP, FM\}$ be a

subset of the distribution models defined in Section 4.1. For each model $\mathcal{E}$, let $\mathcal{T}_{\mathcal{E}}$ be a predetermined threshold value of the distribution measure according to the model $\mathcal{E}$. If $\Xi_{\mathcal{E}}(S^t) \geq \mathcal{T}_{\mathcal{E}}$ for any $\mathcal{E} \in \Gamma$, then the algorithm stops iterating and $S' = S^t$. Let $\Upsilon \subseteq \{AD, \lambda M, DE, DM, OO\}$ be a subset of the similarity models defined in Section 4.2. For each model M, let $\mathcal{T}_M$ be a predetermined threshold value of the difference measure according to the model M. If $\mathcal{D}_M(S, S^t) > \mathcal{T}_M$ for any $M \in \Upsilon$, then the algorithm stops iterating and $S' = S^{t-1}$. At this point, 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.

Since we cannot prove that the algorithms converge (see Section 6.5), we cannot guarantee that the threshold values will be reached; therefore, a maximum number of iterations $I_{\max}$ is also used when determining whether to stop iterating. If $t = I_{\max}$ then the algorithm stops iterating and $S' = S^t$. If $\Gamma = \emptyset$ and $\Upsilon = \emptyset$ then no measurements are made and the algorithm simply iterates $I_{\max}$ times.

The subsets Γ and Υ and the values $I_{\max}$, $\mathcal{T}_{\mathcal{E}}$ for each $\mathcal{E}$, and $\mathcal{T}_M$ for each M can be given as part of the input (for experienced users) or defaults can be used. Possible default values have been determined through experiments and a discussion of the issues and choices is presented in Section 6.4.

5.1 The Voronoi Diagram Cluster Buster (VDCB)

In the VDCB algorithm, nodes are moved to a point within their Voronoi regions in the current layout. 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 criteria **CB3**. Restricting the nodes to move within their Voronoi regions, guarantees that

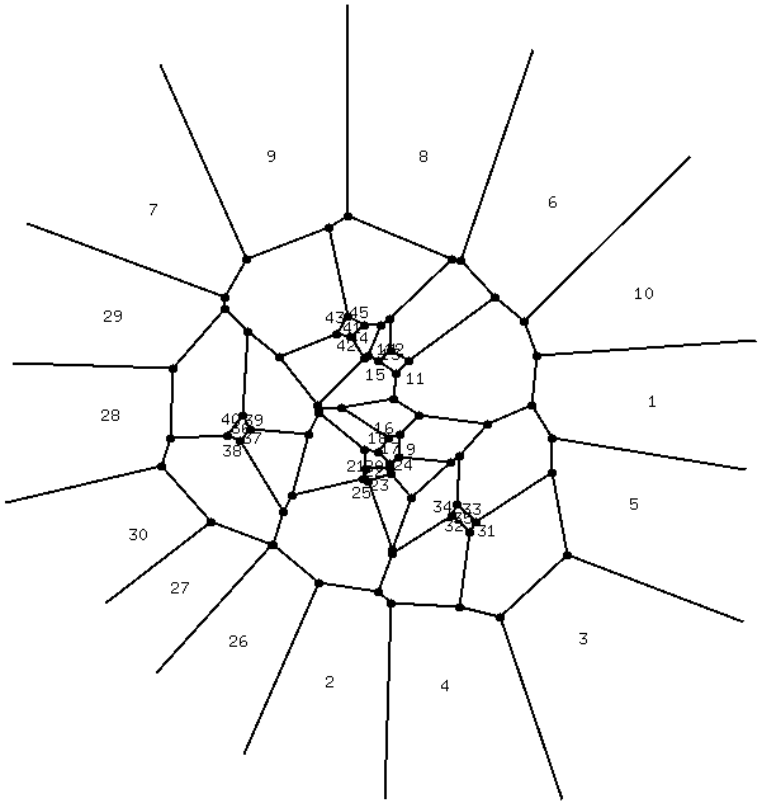

Figure 5.1: Some of the Voronoi regions do not allow very much movement.

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. Figure 5.1 shows an example of a layout of nodes in which the Voronoi regions of some of the nodes are small[1]. Nodes such as those numbered 17, 20, and 22 are close together and cannot move very far if they are forced to stay within their Voronoi regions. 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

[1]This Voronoi diagram was computed and displayed using Mathematica [93].

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 $p_j \in S$, $j \neq i$.

Since some of the Voronoi regions are unbounded, the Clipped Voronoi diagram is used to ensure that each Voronoi region is a closed convex polygon, and any point that is inside its Voronoi region in $CVD(S)$ is inside W. In the following discussion, when we refer to the Voronoi diagram, Voronoi vertices, or Voronoi regions, we are referring to the Clipped Voronoi diagram.

The first question to resolve is where inside $Vor(p_i)$ to move p_i. If a node p_i is close to an edge of $Vor(p_i)$ then it is close to the node that shares that edge, and if p_i is far from an edge of $Vor(p_i)$ then it is far from the node that shares that edge. 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 a *central location* in its Voronoi region. Two choices for a central location are the *centre of mass* of the Voronoi vertices and the *centroid* of the Voronoi region. Given a node p_i such that $Vor(p_i)$ has the following Voronoi vertices: $\{v_1 = (x_{v_1}, y_{v_1}), v_2 = (x_{v_2}, y_{v_2}), \ldots, v_k = (x_{v_k}, y_{v_k})\}$, the centre of mass of these vertices is defined as:

$$cm_i = \left(\frac{x_{v_1} + x_{v_2} + \ldots + x_{v_k}}{k}, \frac{y_{v_1} + y_{v_2} + \ldots + y_{v_k}}{k} \right)$$

Since each Voronoi region is convex, the centre of mass of its vertices lies inside the region. The centroid of $Vor(p_i)$ is defined as:

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

and can be computed by taking the sum of the centroids of k triangles defined by $\{\triangle p_i v_1 v_2, \triangle p_i v_2 v_3, \ldots, \triangle p_i v_{k-1} v_k\}$ each multiplied by its area and all divided by the

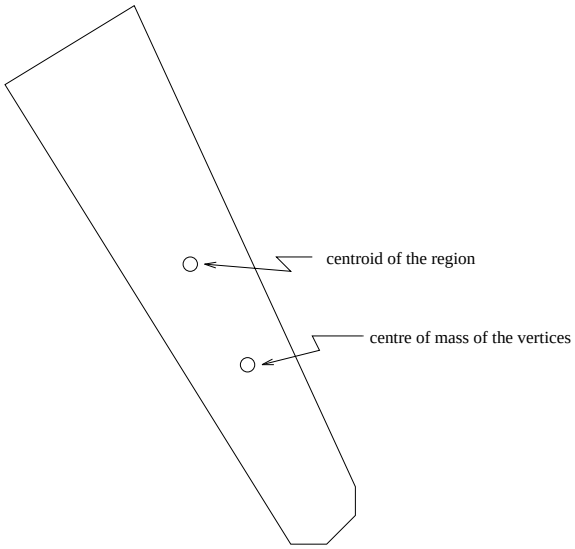

Figure 5.2: A Voronoi region with several vertices close together.

area of $Vor(p_i)$.

If $Vor(p_i)$ has two or more vertices that are close together, then using the centre of mass of the vertices causes p_i to be moved closer to those vertices even when there is more open space in $Vor(p_i)$ in the other direction. Figure 5.2 shows an example of such a Voronoi region. Moving the node to the centroid of its Voronoi region in this situation moves the node further into the open space which is intuitively a more central location. For this reason, the central location in $Vor(p_i)$ has been chosen as the centroid of $Vor(p_i)$.

In summary, the VDCB algorithm is iterative and heuristic. The heuristic used at each iteration is to move each node to the centroid of its Voronoi region in the current Voronoi diagram of the layout of the nodes. The algorithm takes a drawing $D = (G, S, W)$ as input. Possibly empty sets Γ and Υ of distribution and similarity models along with threshold values and a maximum number of iterations $I_{\max}$ are

```
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 E ∈ Γ
        if Ξ_E(S^t) ≥ T_E then
            stop_iterating ← TRUE
            S' ← S^t
    for each M ∈ ϒ
        if D_M(S^t) ≥ T_M then
            stop_iterating ← TRUE
            S' ← S^{t-1}
output: D'
```

Figure 5.3: The VDCB algorithm.

available at the start of the algorithm. If these values are not given as part of the input, default values are used. The VDCB algorithm is presented in Figure 5.3.

The centroids of the Voronoi regions can be computed in $O(n)$ time since there are $\Theta(n)$ edges in the Clipped Voronoi diagram and each Voronoi edge is used in the computation of two centroids. Therefore, 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.

5.2 The GeoForce Algorithm

The Geographic Force algorithm or GeoForce algorithm is based on the idea of using forces and springs to model interactions between nodes in a drawing of a graph. Variations of this idea have been presented in [12, 26, 32, 35, 49, 68, 75]. The basic idea in each of these 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 the two nodes and nodes connected by an edge are attracted to each other by a force that depends on the distance between them. In [49], the Euclidean distance between node positions along with the length of the shortest path between nodes in the graph are used to calculate the repelling force. In most cases, the initial layout is computed randomly and the algorithms are iterative. At each iteration, forces are computed on the nodes and one or all of the nodes are moved to a position according to the acting forces. Several techniques are used to decide when to stop iterating including simulated annealing [12], choosing a fixed number of iterations [26, 35], and stopping when a state with minimal energy in the springs is reached [49].

The above algorithms vary in several aspects: The functions describing the forces, the constants used in the force functions, the method for determining when to stop iterating, the technique for dealing with the sides of the drawing window, whether all or one node is moved at each iteration, and the *step-size*, or the amount of movement allowed at each iteration. Issues affecting these decisions include the time complexity of the algorithm and the quality of the resulting layout. In some cases, quality is sacrificed for time.

The GeoForce algorithm came about by adapting the idea of the force-based algorithms with the hopes of achieving the same success in practise that the force-based algorithms have. 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 and experimenting. The basic GeoForce algorithm that was first implemented has undergone several modifications. We have found that slight changes in any of these choices can have a big effect on the quality of the resulting layout. Consequently, a description of the evolution of the GeoForce algorithm is interesting as well as necessary to the understanding of its final version.

The issues we address in the next three sections are: How we ensure that the nodes stay within W, what forces we decided to apply, and the step-size. For each issue, we describe approaches used in the most recent force-based algorithms, describe our experiences, and explain how our final decisions were made. Further discussions on possible future directions is postponed until Chapter 7.

5.2.1 Keeping the Nodes Inside W

We first describe the way in which we decided to deal with the drawing window. Initially, we started by using the same approach as Eades in [26]: The forces are

allowed to act as in a real system and possibly push the nodes outside the window. After the forces are applied, and the nodes are moved, the drawing is rescaled to fit inside the window before iterating. We found two problems with this approach. First, after several iterations, the configuration tends to take the shape of a circle; that is, the nodes do not move into the corners of W. Secondly, the choice of where to place the nodes inside W after rescaling is arbitrary. We decided it would be better to include the window as part of the algorithm as is done in the VDCB algorithm.

To do this, we experimented with a different approach suggested by Davidson and Harel in [12]. They add a term to their energy function that is proportional to the sum of one over the distance squared from each node to each window boundary. The nodes are kept from becoming too close to the boundary and from congregating around the centre of the drawing by adjusting the constants on the force exerted on the nodes from $bd(W)$ and the repelling force between every pair of nodes. We tried their approach with several constants and found that we could not guarantee that all nodes would stay inside W. An additional property of the algorithm by Davidson and Harel may explain why their approach worked in their algorithm and not in ours. They move only one node p_i at a time and limit its movement to a point on the boundary of a circle centred at p_i whose radius decreases at each iteration. Their algorithm moves one node a small amount at each iteration which slows it down considerably. Since we are trying to achieve fast layouts, the amount by which the nodes move at each iteration is necessarily larger than in their algorithm. The effect is that if a node gets close to $bd(W)$ or starts out close to $bd(W)$ the acting force is very large pushing the node a long way from the boundary. In some cases, the node is pushed past the opposite window boundary.

We tried stopping a node at a window boundary if the forces acting on it pushed it outside that window boundary as suggested by Fruchterman and Reingold in [35]. We experimented with different constants and forces but found that we could not keep the nodes from congregating along the boundary of W. Consequently, a modification of the approach by Davidson and Harel combined with the idea from Fruchterman and Reingold was adopted. We decided to limit the movement of a node p_i so that it cannot move outside the window and also to keep p_i within a region that takes into consideration its proximity to the other nodes. The region chosen is the Voronoi region in $CVD(S)$. The nodes are only allowed to move within a specified percentage of the distance to their Voronoi edges. This specified percentage constitutes the step-size and is discussed in Section 5.2.3. We continue to use repelling forces between every pair of nodes and between each node and the sides of W, and attractive forces between every node and its position in the original layout. As Eades does in [26] and Fruchterman and Reingold do in [35], 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 [12] and [49].

These ideas along with a careful choice of forces result in a cluster busting algorithm that uses forces and 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 cost of this more intelligent algorithm is an $O(n^2)$ computation time because of the application of the node repelling forces. The two algorithms are compared in detail in Chapter 6.

5.2.2 The Forces

Each of the forces is given as functions of the distance between the node position p_i and the object (side of W, original position, or position of another node) exerting the force on p_i. Let d be the distance between p_i and the object exerting the force. Let $f_A(d)$ be the function that describes the attractive force, let $f_R(d)$ be the function that describes the repelling force between nodes, and let $f_S(d)$ be the function that describes the repelling force between a node and $bd(W)$. We describe each of the three forces in turn but obviously the choice of one depends on the choices of the others.

We start with the repelling force. The reason for applying repelling forces is to keep the nodes away from each other. Fruchterman and Reingold define the ideal distance between nodes as

$$d_I = \sqrt{\frac{area}{n}}$$

where *area* is the area of the drawing window [35]. They assign attractive and repulsive forces that apply a net force of 0.0 when $d = d_I$. Eades [26] and Davidson and Harel [12] use a repelling force that is proportional to $\frac{1}{d^2}$. The repelling force used by Fruchterman and Reingold is proportional to $\frac{1}{d}$ [35]. These force functions are made more or less steep by multiplying by constants. In any case, as d gets close to 0.0, the force approaches infinity. When the force approaches infinity for small d, two nodes that are very close together are pushed very far apart by this strong force. We have found that bounding the amount of force applied for $d = 0.0$ results in a smoother behaviour. Furthermore, when the distance is larger than the ideal distance, no force should be applied at all. We considered using a force proportional to $\frac{1}{d^2}$ or $\frac{1}{d}$ shifted such that when the distance is 0.0, the force is a constant, but found that we got

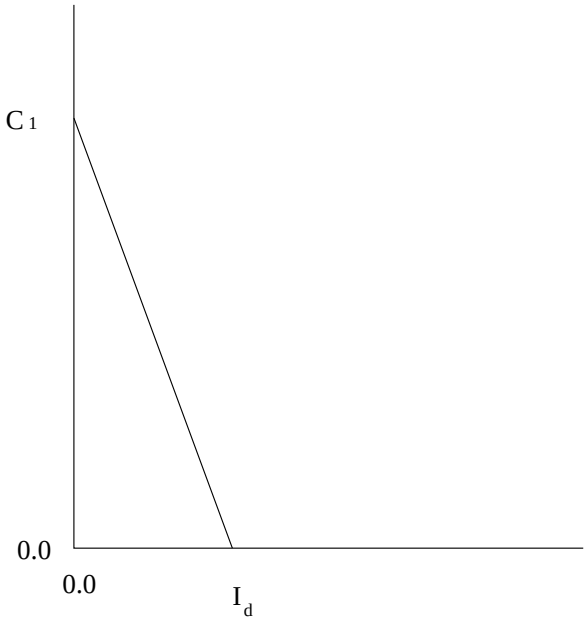

Figure 5.4: The repelling force.

similar results by letting $f_R(d)$ be a straight line segment from $(0, C_1)$ to $(d_I, 0)$, where C_1 is a constant representing the maximum force applied. The function $f_R(d)$ is shown in Figure 5.4. This function is easy to compute and in practise gives good results for $C_1 = 100.0$ and $d_I = \sqrt{area/n}$. The choice of $C_1 = 100.0$ was decided after trying a few values in combination with the constants used in the other force functions and observing the effect on a few sample layouts.

The attractive force is applied to pull each node towards its original position. In force directed graph layout algorithms, an attractive force is applied between each pair of nodes that share an edge in the graph. In [26], Eades uses an attractive force that is proportional to the natural logarithm of the distance between the nodes. The attractive force used by Fruchterman and Reingold in [35] and by Davidson and Harel in [12] is proportional to d^2. We tried using forces proportional to both of these

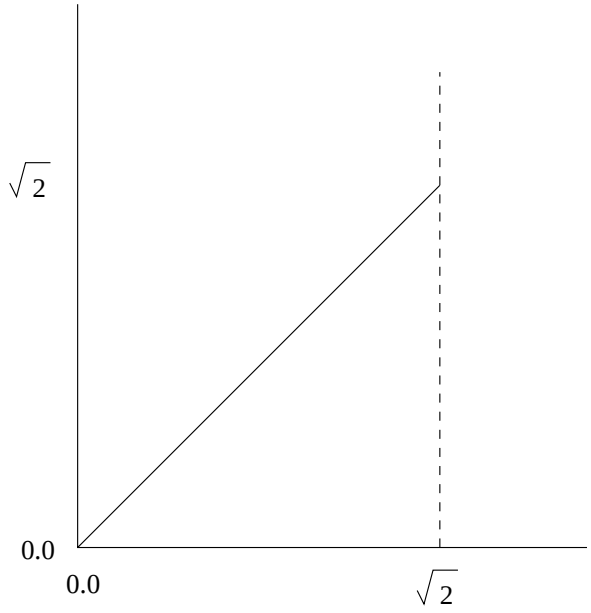

Figure 5.5: The attractive force.

functions but found that the following function gives good results: $f_A(d) = C_2 d$ where C_2 is a constant. Figure 5.5 shows the function $f_A(d)$. We found that $C_2 = 1.0$ works well by trying a few forces and constants in combination with constants in the other force functions and observing the effect on some sample layouts.

Each side of W exerts a repelling force on each of the nodes. In [12] and [35], the side repelling force is the same as the node repelling force except for constants. Following this idea, we let $f_S(d)$ be a straight line segment from $(0, M_F)$ to $(d_I, 0)$, where M_F is the maximum force exerted. We found that setting the ideal distance to $\sqrt{area/n}$ results in a better distribution of the nodes according to our measures than setting the ideal distance to $\frac{1}{2}\sqrt{area/n}$ even though in Section 4.1 we noted that the ideal distance between a node and $bd(W)$ is roughly half the ideal distance between nodes.

We found that good results were obtained with $M_F = 10.0 * n$. Since there are $n-1$ repelling forces from other nodes acting on a node but only four repelling forces acting on a node from the sides of W, making M_F proportional to n helps balance the forces. A maximum force of slightly less than this causes a node that is very close to the boundary of W to be forced onto the boundary. Once the node is on the boundary, it will not likely move off because the net force exerted on it wants to move it further outside W. Since a node is not allowed to move outside its Voronoi region, the node stays on the boundary of W. A force slightly stronger than this one keeps the nodes too far from the boundary of W.

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 as a function of d is given by:

$$f(d) = \frac{f_S(d) + f_R(d)}{n+3} - f_A(d)$$

5.2.3 Step-Size

Finally, we discuss the choice of step-size for the GeoForce algorithm. The step-size is computed as a percentage of the distance from p_i^t to the boundary of its Voronoi region in the direction of the force vector acting on p_i^t. 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 $p_i^{t+1} = p_i^t + f_i^t$ or $p_i^{t+1} = p_i^t + \frac{s_i^t * f_i^t}{|f_i^t|}$ which ever is closer to p_i^t.

We experimented with several values of C_4. If C_4 is too large, big changes in the

node positions occur at each iteration which results in a node coming very close to the window boundary then jumping far away and then moving close to the boundary again after a few more iterations. Obviously, we want a process that will tend to stabilize. With values of C_4 that are too small, the process takes several iterations to stabilize. We found that a step-size of 0.25 works well. The GeoForce algorithm is presented in Figure 5.6.

The cost of the GeoForce algorithm is dominated by the fact that the repelling forces are computed between every pair of nodes. Therefore, the GeoForce algorithm 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.

```
algorithm GeoForce(D)

initialize: stop_iterating ← FALSE; t ← 0; S^t ← S
while t < I_max and not stop_iterating
    for each i
        initialize: f_i ← 0.0
    for each i
        for each j ≠ i
            f_i ← f_i + f_R(dist(p_i^t, p_j^t))
        for each w ∈ {bl, tr}
            f_i ← f_i + f_S(|x_i^t − x_w|)
            f_i ← f_i + f_S(|y_i^t − y_w|)
        f_i ← f_i / (n+3)
        f_i ← f_i − f_A(dist(p_i^t, p_i))
    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 E ∈ Γ
        if Ξ_E(S^t) ≥ T_E then
            stop_iterating ← TRUE
            S' ← S^t
    for each M ∈ Υ
        if D_M(S^t) ≥ T_M then
            stop_iterating ← TRUE
            S' ← S^{t−1}
output: D'
```

Figure 5.6: The GeoForce algorithm.

Chapter 6

Results

In this chapter we evaluate and compare the layout algorithms presented in Chapter 5 using the measurements presented in Chapter 4. Properties of the layout algorithms and a description of default threshold values for the stopping conditions are also included. Recall the goal of cluster busting in anchored graph drawing: Given a drawing $D = (G, S, W)$ of a graph G, the new drawing $D' = (G, S', W')$ should satisfy the following criteria:

CB1 $W' = W$,

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

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

We use the measurements described in Chapter 4 to evaluate layouts quantitatively according to the second and third criteria. Since the layout algorithms guarantee that $W' = W$, we present no further discussion on criterion **CB1**. An implicit fourth criterion of the layout algorithms is that they be computationally efficient. The

speed and efficiency of the algorithms are judged by their asymptotic worst-case time complexity.

In some cases we are able to prove that the algorithms satisfy the criteria according to some of the measures for specific types of input. In other cases we provide evidence that the algorithms satisfy the criteria by performing experiments on different configurations of initial inputs. In order to perform these experiments, the layout algorithms and algorithms that execute the measurements were implemented[1] on an IBM RS6000 in C.

We have concentrated our efforts on proving results for the VDCB algorithm since the behaviour of the VDCB algorithm is better understood than that of the GeoForce algorithm. Proving that force-directed placement algorithms produce layouts that satisfy specific criteria is a challenge [12, 26, 35, 49]. One might be able to prove something trivial about layouts produced by the GeoForce algorithm since it is guaranteed that each node stays inside its Voronoi region at each iteration.

We are able to prove results about layouts after one iteration of the VDCB algorithm since layouts after one iteration are easier to describe. Evaluating the layouts after one iteration is also interesting because it is guaranteed that each node is closer to its initial position than to the initial position of any other node. This property can be viewed as a quantitative measure of similarity.

We executed each algorithm on a variety of input layouts and measured the layouts after a differing number of iterations. 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 each model $\mathcal{E}$ of distribution and let the average value of $\mathcal{D}_M(S, S')$ be $\overline{\mathcal{D}}_M(S, S')$ for each model M of similarity. The

[1] The Voronoi diagram code was supplied by Kenny Wong [94].

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

The rest of this chapter is organized as follows. In Section 6.1, we evaluate the layout algorithms according to each of the distribution measures (criterion **CB2**), and in Section 6.2 we evaluate the layout algorithms according to the similarity measures (criterion **CB3**). In Section 6.3, we compare the layout algorithms and the evaluation measures and in Section 6.4 we discuss default threshold values for the measures used to determine stopping conditions for the algorithms. We also present recommendations and conclusions based on these comparisons and the results presented in this chapter. Several more recommendations and conclusions are presented in Chapter 7.

A natural question to ask about the algorithms is whether or not the layouts converge in the limit. That is, whether the layout reaches a point where another iteration of the given algorithm will not change the layout. In Section 6.5 we discuss this question and present convergence results for the VDCB algorithm on specific input configurations.

6.1 Distribution – Satisfying Criterion CB2

Criterion **CB2** states that the nodes in D' should be more evenly distributed in W than those in D. In Chapter 4, we presented two models of distribution: CP and FM. It would be nice if we could show that measures based on these models always increase (that is, the layouts always become more evenly distributed according to these models) when either of the given layout algorithms is executed. We show that this is not always true for general inputs. For specific input we show that the distribution of the nodes increases with the VDCB algorithm and we provide evidence that in

general the nodes become more evenly distributed according to the models for both algorithms.

6.1.1 The CP Model of Distribution

In this section we prove that after one iteration of the VDCB algorithm, the layout does not necessarily become more evenly distributed according to the CP model of distribution. On the other hand, we show that for n nodes on an isothetic line or n nodes at the vertices of an isothetic grid the distribution measure based on the CP model increases. It is an open problem to provide similar results for the GeoForce algorithm. We use results from experiments to show that in general the nodes become more evenly distributed in layouts produced by the VDCB algorithm and the GeoForce algorithm according to the CP model of distribution.

The VDCB Algorithm

The following example shows that one iteration of the VDCB algorithm does not always more evenly distribute the nodes according to the CP model of distribution.

Lemma 6.1.1 *Given the node positions S, let S^1 be the node positions after one iteration of the VDCB algorithm. In some cases,*

$$\Xi_{CP}(S^1) < \Xi_{CP}(S)$$

Proof: For ease of presentation, we assume the node positions and W have been uniformly scaled such that $\Xi_{CP}(S) = 2$. Let $p_1 = (0, 1)$ and $p_2 = (0, -1)$ be Voronoi neighbours and note that $dist(p_1, p_2) = 2$. Let $Vor(p_1)$ be an equilateral triangle whose bottom side is horizontal and on the x-axis and such that each side has length

$2\sqrt{3}$. The mid-point of the bottom side is at the point $(0, 0)$. The node p_1 has two additional Voronoi neighbours besides p_2. Let these neighbours be: $p_3 = (-\sqrt{3}, 2)$ and $p_4 = (\sqrt{3}, 2)$. Note that the distance between p_1 and each of its Voronoi neighbours is 2.

Consider the equilateral triangle obtained by reflecting $Vor(p_1)$ about its bottom side. Let $Vor(p_2)$ be the quadrilateral defined by cutting this triangle with the line $y = -2$. Therefore, p_2 has three additional Voronoi neighbours besides p_1. Let these neighbours be: $p_5 = (0, -3)$, $p_6 = (-\sqrt{3}, -2)$, and $p_7 = (\sqrt{3}, -2)$. Each of these Voronoi neighbours is at a distance of 2 from p_2. It can be shown that the distance between each of the Voronoi neighbours $\{p_3, p_4, p_5, p_6, p_7\}$ is no less than 2. Additional nodes can be placed outside the convex hull of $\{p_1, p_2, \ldots, p_7\}$ such that each internode distance is no less than 2. The window W can be drawn such that all nodes are no closer than the distance 1 to a side of W. Thus, $\Xi_{CP}(S) = 2$ and p_1 and p_2 are a closest pair of nodes. Figure 6.1 illustrates the configuration of the seven nodes described above.

It is straightforward to verify that p_1 is at the centroid of $Vor(p_1)$ such that $p_1^1 = p_1$. It is also easy to see that the centroid of $Vor(p_2)$ is at the point $(0, -\frac{5}{6})$. Therefore, $dist(p_1^1, p_1^2) = \frac{11}{6} < dist(p_1, p_2) = 2$ and $\Xi_{CP}(S^1) < \Xi_{CP}(S)$. $\square$

Although in general the distribution of the nodes according to the CP model may decrease after one iteration of the VDCB algorithm, for the special case of n nodes on an isothetic line in W, the CP measure does not decrease after one iteration of the VDCB algorithm and, in fact, eventually increases. This property of the VDCB algorithm was previously published in [53].

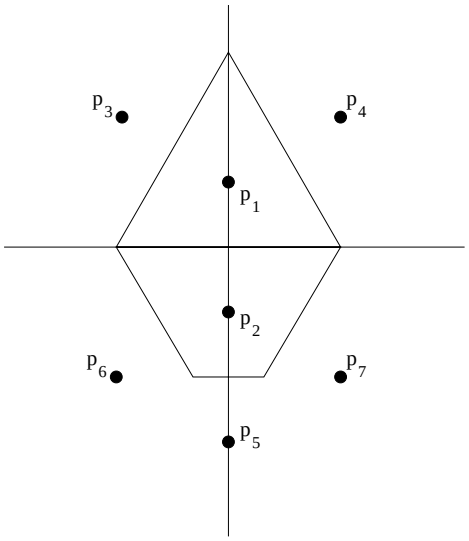

Figure 6.1: Configuration for which $\Xi_{CP}(S^1) < \Xi_{CP}(S)$.

Let the set S of n nodes be positioned on an isothetic line, l in W. Assume, without loss of generality, that l is horizontal, and that $x_{bl} \leq x_0 \leq x_1 \leq \cdots \leq x_{n-1} \leq x_{tr}$. Define the vector $\mathbf{\Delta} = [d_0, d_1, \ldots, d_n]$ as a sequence of horizontal distances where the distance between x_{bl} and p_0 is d_0, the distance between p_{n-1} and x_{tr} is d_n, and the distance between p_{i-1} and p_i is $2d_i$ for $i = 1, 2, \ldots n-2$. The value of $\Xi_{CP}(S) = 2\min_{0 \leq i \leq n}\{d_i\}$. Thus, if we show that the value $\frac{\Xi_{CP}(S)}{2} = \min_{0 \leq i \leq n}\{d_i\}$ does not decrease after an iteration of the VDCB, we have shown that $\Xi_{CP}(S)$ does not decrease after an iteration of the VDCB algorithm.

Let $pb(p, q)$ be the perpendicular bisector between nodes p and q. For every pair of adjacent nodes, p_i, p_{i+1}, $i = 0, \ldots, n-2$, $pb(p_i, p_{i+1})$ is a vertical line through the midpoint of the line segment between p_i and p_{i+1}. The Voronoi region $Vor(p_i)$ of the node p_i in $CVD(S)$ is an isothetic rectangle with the bottom left corner at the point $(\frac{x_{i-1}+x_i}{2}, y_{bl})$, and the top right corner at the point $(\frac{x_i+x_{i+1}}{2}, y_{tr})$. The region $Vor(p_0)$

Voronoi region. The width of $Vor(p_i)$ is $d_i^t + d_{i+1}^t$ for $i = 0, 1, \ldots, n-1$, therefore:

$$2d_i^{t+1} = \frac{d_{i-1}^t + d_i^t}{2} + \frac{d_i^t + d_{i+1}^t}{2}$$

$$\Longrightarrow d_i^{t+1} = \frac{1}{4}d_{i-1}^t + \frac{1}{2}d_i^t + \frac{1}{4}d_{i+1}^t$$

The width of $Vor(p_0^t)$ is $d_0^t + d_1^t$ and:

$$d_0^{t+1} = \frac{d_0^t + d_1^t}{2}$$

$$\Longrightarrow d_0^{t+1} = \frac{1}{2}d_0^t + \frac{1}{2}d_1^t$$

Similarly, the width of $Vor(p_{n-1}^t)$ is $d_{n-1}^t + d_n^t$, and:

$$d_n^{t+1} = \frac{d_{n-1}^t + d_n^t}{2}$$

$$\Longrightarrow d_n^{t+1} = \frac{1}{2}d_{n-1}^t + \frac{1}{2}d_n^t$$

□

We now show that if the nodes are equally spaced on l, the VDCB algorithm does not move any of the nodes.

Lemma 6.1.3 **(a)** *If the positions of the nodes do not change at iteration $t+1$ of the VDCB, then*

$$d_0^t = d_1^t = \cdots = d_n^t.$$

(b) *If*

$$d_0^t = d_1^t = \cdots = d_n^t,$$

then the positions of the nodes do not change at iteration $t+1$ of the VDCB.

Proof: (a) If the positions of the nodes do not change at iteration $t+1$ of the VDCB, then all of the nodes are at the centroid of their Voronoi regions at time t. Consider the values of d_i^t, $i = 0, 1, \ldots, n$. If p_0^t is at the centroid of its Voronoi region, then:

$$d_0^t = d_1^t.$$

If p_{n-1}^t is at the centroid of its Voronoi region, then:

$$d_{n-1}^t = d_n^t.$$

If p_i^t, $1 \leq i \leq n-2$, is at the centroid of its Voronoi region then:

$$dist(pb(p_{i-1}, p_i), p_i) = dist(p_i, pb(p_i, p_{i+1})).$$

We know that:

$$\begin{aligned} dist(p_{i-1}, pb(p_{i-1}, p_i)) &= dist(pb(p_{i-1}, p_i), p_i) \\ &= d_i^t \end{aligned}$$

and

$$\begin{aligned} dist(p_i, pb(p_i, p_{i+1})) &= dist(pb(p_i, p_{i+1}), p_{i+1}) \\ &= d_{i+1}^t \end{aligned}$$

by the definition of perpendicular bisectors. Therefore, $d_i^t = d_{i+1}^t$ for $i = 1, 2, \ldots n-2$, and since $d_0^t = d_1^t$ and $d_{n-1}^t = d_n^t$,

$$d_0^t = d_1^t = \cdots = d_n^t$$

if the positions of the nodes do not change at iteration $t+1$.

(b) Let $d_0^t = d_1^t = \cdots = d_n^t$ at iteration t of the VDCB. By Lemma 6.1.2 for $i = 1, 2, \ldots, n-1$,

$$d_i^{t+1} = \frac{1}{4}d_{i-1}^t + \frac{1}{2}d_i^t + \frac{1}{4}d_{i+1}^t$$

for $i = 0$,

$$d_0^{t+1} = \frac{1}{2}d_0^t + \frac{1}{2}d_1^t$$

and for $i = n$,

$$d_n^{t+1} = \frac{1}{2}d_{n-1}^t + \frac{1}{2}d_n^t.$$

Therefore, $d_i^{t+1} = d_i^t$ for $i = 0, 1, \ldots, n$ and the positions of the nodes do not change at iteration $t+1$ of the VDCB. □

We are now ready to present the proof that $\Xi_{CP}(S^{t+1}) \geq \Xi_{CP}(S^t)$ for n nodes on an isothetic line.

Lemma 6.1.4 *Consider* $\boldsymbol{\Delta}^t = [d_0^t d_1^t \ldots d_n^t]$ *at iteration* t *of the VDCB and let* $\Gamma = [d_j^t d_{j+1}^t \ldots d_{j+k-1}^t]$ *be a longest sequence of distances in* $\boldsymbol{\Delta}^t$ *such that*

$$d_j^t = d_{j+1}^t = \cdots = d_{j+k-1}^t = \frac{\Xi_{CP}(S^t)}{2},$$

then

(a) *If* $k = n+1$ *then all the distances are equal and:*

$\frac{\Xi_{CP}(S^t)}{2} = \frac{\Xi_{CP}(S^{t+1})}{2}$

(b) *If* $k < n+1$ *then not all of the distances are equal and:*

(i) *if* $j = 0$ *or* $j + k - 1 = n$ *then the sequence is at an end of* $\boldsymbol{\Delta}^t$ *and:*

after k *iterations of the VDCB,* $d_i^{t+k} > \frac{\Xi_{CP}(S^t)}{2}$, *for* $i = 0, 1, \ldots, n$.

(ii) *otherwise,*

after $\lceil \frac{k}{2} \rceil$ *iterations of the VDCB,* $d_i^{t+\lceil \frac{k}{2} \rceil} > \frac{\Xi_{CP}(S^t)}{2}$, *for* $i = 0, 1, \ldots, n$.

Proof: **(a)** If $k = n+1$ then $d_0^t = d_1^t = \cdots = d_n^t = \frac{\Xi_{CP}(S^t)}{2}$ and subsequent iterations of the VDCB will not move any node by Lemma 6.1.3.

(b) If $k < n+1$, then there are two situations:

(i) if $j = 0$ or $j + k - 1 = n$ then either $d_{j+k}^t > \frac{\Xi_{CP}(S^t)}{2}$ or $d_{j-1}^t > \frac{\Xi_{CP}(S^t)}{2}$, respectively, but not both. If $j = 0$ then $d_{j+k}^t > \frac{\Xi_{CP}(S^t)}{2}$, and we know by Lemma 6.1.2 that

if $j + k = n$ then

$$d_{j+k-1}^{t+1} = \frac{1}{2} d_{j+k-1}^t + \frac{1}{2} d_{j+k}^t$$

otherwise

$$d_{j+k-1}^{t+1} = \frac{1}{4} d_{j+k-2}^t + \frac{1}{2} d_{j+k-1}^t + \frac{1}{4} d_{j+k}^t$$

In either case, $d_{j+k-1}^{t+1} > d_{j+k-1}^t = \frac{\Xi_{CP}(S^t)}{2}$.

If $j + k - 1 = n$ then $d_{j-1}^t > \frac{\Xi_{CP}(S^t)}{2}$, and we know by Lemma 6.1.2 that

if $j - 1 = 0$ then

$$d_j^{t+1} = \frac{1}{2} d_{j-1}^t + \frac{1}{2} d_j^t$$

otherwise

$$d_j^{t+1} = \frac{1}{4} d_{j-1}^t + \frac{1}{2} d_j^t + \frac{1}{4} d_{j+1}^t$$

In either case, $d_j^{t+1} > d_j^t = \frac{\Xi_{CP}(S^t)}{2}$.

Therefore, if $j = 0$ or $j + k - 1 = n$, then after each iteration of the VDCB, the length of Γ strictly decreases by exactly 1 and after k iterations of the VDCB, $d_i^{t+k} > \frac{\Xi_{CP}(S^t)}{2}$, $i = 0, 1, \ldots, n$.

(ii) If $j > 0$ and $j + k - 1 < n$ then $d_{j+k}^t > \frac{\Xi_{CP}(S^t)}{2}$ and $d_{j-1}^t > \frac{\Xi_{CP}(S^t)}{2}$. From (i), after

one iteration of the VDCB, $d_j^{t+1} > \frac{\Xi_{CP}(S^t)}{2}$ and $d_{j+k-1}^{t+1} > \frac{\Xi_{CP}(S^t)}{2}$; therefore, after $\lceil \frac{k}{2} \rceil$ iterations of the VDCB, $d_i^{t+\lceil \frac{k}{2} \rceil} > \frac{\Xi_{CP}(S^t)}{2}$, $i = 0, 1, \ldots, n$. $\square$

In Lemma 6.1.4 we showed that for n nodes on an isothetic line, $\Xi_{CP}(S^{t+1}) \geq \Xi_{CP}(S^t)$. This result can also be used to show that $\Xi_{CP}(S^{t+1}) \geq \Xi_{CP}(S^t)$ for $n = pq$ nodes at the vertices of a (not necessarily evenly spaced) isothetic grid. The result applies to the nodes on each horizontal and vertical line in the grid. At each iteration, all nodes with the same x-coordinate (or y-coordinate) are guaranteed to be moved such that they have the same x-coordinate (or y-coordinate). These configurations are not *stable* configurations; that is, if one node is slightly off a horizontal or vertical line, the VDCB algorithm causes the rest of the nodes to be moved off the lines.

In Practise

We are able to provide evidence that the distribution of the nodes according to the CP model increases for both algorithms on most input graphs by presenting experimental results. Values of $\overline{\Xi}_{CP}(S)$ and $\overline{\Xi}_{CP}(S^1)$ for both layout algorithms are presented in Table 6.1. The configurations for each n are listed in order from the largest $\overline{\Xi}_{CP}(S)$ to the smallest $\overline{\Xi}_{CP}(S)$. Note that, for each type of initial layout, one iteration of the VDCB algorithm more evenly distributes the layouts according to the CP model than one iteration of the GeoForce algorithm. We compare the layout algorithms in Section 6.3 and defer our discussion on this observation until then. Several other observations and conclusions can be drawn from the results in this table.

In all cases, $\overline{\Xi}_{CP}(S^1) > \overline{\Xi}_{CP}(S)$. For the GeoForce algorithm, $\overline{\Xi}_{CP}(S^1) \geq 2\overline{\Xi}_{CP}(S)$ and for the VDCB algorithm $\overline{\Xi}_{CP}(S^1) \geq 3\overline{\Xi}_{CP}(S)$. Layouts that are the least evenly distributed according to the CP model initially are the second most evenly distributed

n	$\mathcal{K}$	$n/\mathcal{K}$	$\overline{\Xi}_{CP}(S)$	$\overline{\Xi}_{CP}(S^1)$ GeoForce	$\overline{\Xi}_{CP}(S^1)$ VDCB
25	25	1	0.0297158	0.0594681	0.0867467
	1	25	0.0124560	0.0286982	0.0428854
	5	5	0.0069397	0.0215777	0.0375028
50	50	1	0.0142302	0.0324772	0.0548724
	1	50	0.0061505	0.0150611	0.0258729
	2	25	0.0047297	0.0109252	0.0199791
	5	10	0.0032214	0.0078346	0.0153701
	10	5	0.0024422	0.0077468	0.0180950
	25	2	0.0020711	0.0154545	0.0ffiqffiffiQffiBTffiΘΘΘcm1ΘΘΘcmffiΘ

of the layouts after one iteration of either algorithm. These layouts have two nodes in each cluster and a closest pair of nodes is most likely a pair within the same cluster. For these layouts one iteration is sufficient to evenly distribute the nodes because they do not suffer from the problem described in Chapter 5 in which nodes with small Voronoi regions cannot move very far after one iteration. One iteration of the VDCB algorithm has a greater effect on layouts with four clusters of size 25 each and five clusters of size 10 each for the same reason. This effect is not evident in the GeoForce algorithm because the forces acting on the nodes and the step-size limit how far the nodes can move at each iteration.

We counted the number of layouts out of 1000 for which $\Xi_{CP}(S^1) < \Xi_{CP}(S)$ and found that the only cases occurred in layouts with one cluster or with n clusters. These initial layouts are the most evenly distributed according to the CP model of all the layouts tested. At most fourteen layouts out of 1000 resulted in $\Xi_{CP}(S^1) < \Xi_{CP}(S)$.

Our experiments also showed that $\overline{\Xi}_{CP}(S^{t+1}) > \overline{\Xi}_{CP}(S^t)$ for all types of input for both layout algorithms. We computed $\overline{\Xi}_{CP}(P)$ for P pseudo-random in the uniform distribution in W and determined the number of iterations t of each algorithm required for $\overline{\Xi}_{CP}(S^t)$ to exceed this value for any input layout S. Let the $\overline{\Xi}_{CP}(P)$ for n nodes pseudo-random in the uniform distribution in W be $U_{CP}(n)$: $U_{CP}(25) = 0.0297158$, $U_{CP}(50) = 0.0142302$, $U_{CP}(100) = 0.0071465$. We found that $\overline{\Xi}_{CP}(S^t)$ for each type of initial layout is greater than $U_{CP}(n)$ for $n = 25, 50, 100$ after $t = 1$ iteration of the VDCB algorithm, and after at most $t = 5$ iterations of the GeoForce algorithm.

The evidence shows that, on average, the GeoForce and VDCB algorithms more evenly distribute the nodes in a drawing according to the CP model of distribution.

The algorithms have a greater effect on layouts that are not evenly distributed initially, and in practise the VDCB and GeoForce algorithms more evenly distribute the nodes according to the CP model of distribution after a small number of iterations (< 20). From this evidence we conclude that the VDCB and GeoForce algorithms satisfy criterion **CB2** according to the CP model of distribution.

6.1.2 The FM Model of Distribution

We are able to provide evidence that the distribution of the nodes increases according to the FM model for both algorithms on most input graphs by presenting experimental results. The values of $\overline{\Xi}_{FM}(S)$ and $\overline{\Xi}_{FM}(S^1)$ are presented in Table 6.2. As with the CP model, one iteration of the VDCB algorithm produces layouts that are more evenly distributed according to the FM model than those after one iteration of the GeoForce algorithm. We defer the discussion on this until Section 6.3. The configurations for each n are listed in order from the largest $\overline{\Xi}_{FM}(S)$ to the smallest $\overline{\Xi}_{FM}(S)$. Similar observations and conclusions can be drawn from this table as were presented in Section 6.1.1 for the CP model of distribution.

In all cases, $\overline{\Xi}_{FM}(S^1) < \overline{\Xi}_{FM}(S)$. After one iteration of the GeoForce algorithm, $\overline{\Xi}_{FM}(S^1) > 2\overline{\Xi}_{FM}(S)$ and after one iteration of the VDCB algorithm $\overline{\Xi}_{FM}(S^1) > 4\overline{\Xi}_{FM}(S)$. As we observed when measuring distribution using the CP model, the algorithms have a greater effect on layouts that are the less evenly distributed initially.

As with the CP model, the average distribution values according to the FM model show that the algorithms have a stronger effect on layouts with small sized clusters. In fact, this effect is more apparent when using the FM model than the CP model because $\Xi_{FM}(S)$ takes all distances into account. Layouts with many small clusters

n	$\mathcal{K}$	$n/\mathcal{K}$	$\overline{\Xi}_{FM}(S)$	$\overline{\Xi}_{FM}(S^1)$ GeoForce	$\overline{\Xi}_{FM}(S^1)$ VDCB
25	25	1	0.6162528	2.0379908	2.7045424
	1	25	0.2410819	0.6260715	0.9486961
	5	5	0.1012481	0.5462194	1.2305337
50	50	1	0.1384782	0.4620808	0.6106493
	1	50	0.0486722	0.1242102	0.1750963
	2	25	0.0305724	0.0830936	0.1327854
	5	10	0.0179013	0.0681594	0.1483973
	10	5	0.0123025	0.0839520	0.2532130
	25	2	0.0112638	0.1946997	0.4902170
100	100	1	0.0306211	0.1077898	0.1374717
	1	100	0.0105053	0.0256376	0.0345418
	2	50	0.0062303	0.0159106	0.0236832
	4	25	0.0039211	0.0107779	0.0194776
	10	10	0.0020561	0.0089870	0.0239982
	25	4	0.0013230	0.0131635	0.0676458
	50	2	0.0012583	0.0392903	0.1126117

Table 6.2: $\overline{\Xi}_{FM}(S^1)$ for several initial layouts and both layout algorithms.

consist of small and large distances. After one iteration of the VDCB or GeoForce algorithm, the small distances increase and the large distances decrease.

We counted the number of layouts out of 1000 for which $\Xi_{FM}(S^1) < \Xi_{FM}(S)$ and found that after one iteration of the GeoForce algorithm as many as thirty-seven layouts out of 1000 resulted in $\Xi_{FM}(S^1) < \Xi_{FM}(S)$. In each of these cases nodes in clusters that are close to $bd(W)$ are pushed closer to $bd(W)$ after one iteration of the GeoForce algorithm because of the forces acting on them from the other nodes in the cluster. After subsequent iterations, the forces repelling the nodes from $bd(W)$ are stronger and the forces from the other nodes in the cluster are less strong.

Our experiments also show that $\overline{\Xi}_{FM}(S^{t+1}) > \overline{\Xi}_{FM}(S^t)$ for all types of input for both layout algorithms. We computed $\overline{\Xi}_{FM}(P)$ for P pseudo-random in the uniform distribution in W and determined the number of iterations t required for $\overline{\Xi}_{FM}(S^t)$ to exceed this value. Let the average value of $\overline{\Xi}_{FM}(P)$ for n nodes pseudo-random in the uniform distribution in W be $U_{FM}(n)$: $U_{FM}(25) = 0.6162528$, $U_{FM}(50) = 0.1384782$, $U_{FM}(100) = 0.0306211$. We found that $\overline{\Xi}_{FM}(S^t)$ for each different type of initial layout is greater than $U_{FM}(n)$ for $n = 25, 50, 100$ after $t = 2$ iterations of the VDCB algorithm, and after at most $t = 5$ iterations of the GeoForce algorithm.

The experiments using the FM model of distribution give similar results as those using the CP model of distribution. From the evidence presented we conclude that the VDCB and GeoForce algorithms satisfy criterion **CB2** according to the FM model of distribution.

6.1.3 Summary – Distribution

In this section, we have shown that one iteration of the VDCB algorithm does not necessarily more evenly distribute the nodes according to the CP model of distribution for general input. We have proved that for n nodes on an isothetic line or at the vertices of an isothetic grid, the distribution measure according to the CP model does not decrease after one iteration of the VDCB algorithm and eventually increases. We have also provided evidence that on average both the VDCB algorithm and the GeoForce algorithm more evenly distribute the nodes according to both measures. We conclude that in practise the layout algorithms produce layouts that satisfy criterion **CB2**.

6.2 Similarity Measures – Satisfying Criterion CB3

Criterion **CB3** states that the shape of the drawing D' should be the same as D. In Chapter 4, we presented five measures of similarity. In this section, we provide evidence that the algorithms satisfy criterion **CB3** by showing that on average the difference between S and S' is much smaller than the average difference between two random layouts of nodes according to each of the similarity models. We also discuss how the amount of difference between S and S' varies with the number and size of the clusters in S.

6.2.1 The AD Model of Difference

In practise, $\overline{\mathcal{D}}_{AD}(S, S^1) < \overline{\mathcal{D}}_{AD}(S, P)$ for two random layouts S and P but the amount by which it is less depends on the starting configuration of the nodes and n. Table

n	$\overline{\mathcal{D}}_{AD}(S,P)$	Worst $\overline{\mathcal{D}}_{AD}(S,S^1)$ GeoForce	Worst $\overline{\mathcal{D}}_{AD}(S,S^1)$ VDCB
25	0.2093309	0.0299834	0.0864216
50	0.2082873	0.0205307	0.0594193
100	0.2069523	0.0137859	0.0425756

Table 6.3: $\overline{\mathcal{D}}_{AD}(S,P)$ and $\overline{\mathcal{D}}_{AD}(S,S^1)$ after one iteration of the VDCB and GeoForce algorithms.

6.3 shows $\overline{\mathcal{D}}_{AD}(S,P)$ and the largest $\overline{\mathcal{D}}_{AD}(S,S^1)$ over all initial inputs tested. Note $\overline{\mathcal{D}}_{AD}(S,S^1)$ for either algorithm decreases as n increases. As n increases, the number of small Voronoi regions increases and the amount the nodes can move after each iteration is less.

For the GeoForce algorithm, $\overline{\mathcal{D}}_{AD}(S,S^{100}) < \overline{\mathcal{D}}_{AD}(S,P)$ for all input configurations. For the VDCB algorithm, $\overline{\mathcal{D}}_{AD}(S,S^{100}) < \overline{\mathcal{D}}_{AD}(S,P)$ for all input configurations when $n = 100$. When $n = 25$, $\overline{\mathcal{D}}_{AD}(S,S^{20}) < \overline{\mathcal{D}}_{AD}(S,P)$ and when $n = 50$, $\overline{\mathcal{D}}_{AD}(S,S^{60}) < \overline{\mathcal{D}}_{AD}(S,P)$.

The amount of difference between S and S' according to the AD model depends on the number of clusters in the input layout. Layouts with one or two clusters are most different after several iterations because as the nodes become spread out they have moved a greater distance from where they started. Layouts that are evenly distributed initially according to the CP and FM models change the least according to the AD measure after several iterations of both layout algorithms. If a layout is already fairly evenly distributed, it should not be changed very much by the 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 6.4: $\overline{\mathcal{D}}_{\lambda M}(S,P)$ and $\overline{\mathcal{D}}_{\lambda M}(S,S^1)$ after one iteration of the VDCB and GeoForce algorithms.

6.2.2 The λM Model of Difference

Table 6.4 shows $\overline{\mathcal{D}}_{\lambda M}(S,P)$ and the largest $\overline{\mathcal{D}}_{\lambda M}(S,S^1)$ over all initial inputs tested. As with the AD model, $\overline{\mathcal{D}}_{\lambda M}(S,S^1)$ decreases as n increases. For all input configurations $\overline{\mathcal{D}}_{\lambda M}(S,S^{100}) < \overline{\mathcal{D}}_{\lambda M}(S,P)$ for both the GeoForce algorithm and the VDCB algorithm.

The amount of difference between S and S' according to the λM measure depends on the clustering of the input layouts. The λM model differs from the AD model in that layouts with roughly the same number of clusters as nodes in each cluster are the most different after several iterations.

As with the AD model, layouts that start out the most evenly distributed according to the CP and FM measures change the least according to the λM measure after several iterations of layout algorithms.

6.2.3 The DE Model of Difference

Table 6.5 shows $\overline{\mathcal{D}}_{DE}(S,P)$ and the largest $\overline{\mathcal{D}}_{DE}(S,S^1)$ over all initial inputs tested. The value of $\overline{\mathcal{D}}_{DE}(S,P)$ for two random layouts S and P increases with n because the number of possible ways of choosing $3n-6$ edges out of $n(n-1)/2$ total possible edges

n	$\overline{\mathcal{D}}_{DE}(S,P)$	Worst $\overline{\mathcal{D}}_{DE}(S,S^1)$ GeoForce	Worst $\overline{\mathcal{D}}_{DE}(S,S^1)$ VDCB
25	0.6869710	0.0901812	0.1518913
50	0.8031806	0.1128993	0.1705347
100	0.8729796	0.1308044	0.1804337

Table 6.5: $\overline{\mathcal{D}}_{DE}(S,P)$ and $\overline{\mathcal{D}}_{DE}(S,S^1)$ after one iteration of the VDCB and GeoForce algorithms.

increases as n increases. Therefore, unlike the other similarity measures, $\overline{\mathcal{D}}_{DE}(S,S^1)$ increases as n increases; however, after 100 iterations of either algorithm $\overline{\mathcal{D}}_{DE}(S,S^{100})$ does not increase as n increases. This is because the same effect is being experienced as with the other measures: As n increases, the number of small Voronoi regions increases and the amount the nodes can move after each iteration is less.

For all input configurations $\overline{\mathcal{D}}_{DE}(S,S^{100}) < \overline{\mathcal{D}}_{DE}(S,P)$ for both the GeoForce and VDCB algorithms. The amount of difference between S and S' according to the DE model depends on the clustering of the input layouts and, like the λM model, layouts with roughly the same number of clusters as nodes in each cluster are the most different after several iterations. Layouts that are the most evenly distributed according to the CP and FM models initially change the least according to the DE model after several iterations of layout algorithms.

6.2.4 The DM Model of Difference

Table 6.6 shows $\overline{\mathcal{D}}_{DM}(S,P)$ and the largest $\overline{\mathcal{D}}_{DM}(S,S^1)$ over all initial inputs tested. As with the other measures, $\overline{\mathcal{D}}_{DM}(S,S^1)$ for either algorithm decreases as n increases.

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

For all input configurations $\overline{\mathcal{D}}_{DM}(S,S^{100}) < \overline{\mathcal{D}}_{DM}(S,P)$ for both the GeoForce algorithm and the VDCB algorithm. The amount of difference between S and S' according to the DM model depends on the clustering of the input layouts. As with the AD model, 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. Layouts that start out the most evenly distributed according to the CP and FM models changes the least according to the DM model after several iterations of the layout algorithms.

6.2.5 The OO Model of Difference

Table 6.3 shows $\overline{\mathcal{D}}_{OO}(S,P)$ and the largest $\overline{\mathcal{D}}_{OO}(S,S^1)$ over all initial inputs tested. As with all models but the DE model, $\overline{\mathcal{D}}_{OO}(S,S^1)$ for either algorithm decreases as n increases. For all input configurations $\overline{\mathcal{D}}_{OO}(S,S^{100}) < \overline{\mathcal{D}}_{OO}(S,P)$ for both the
S
algorithms.

The amount of difference between S and S' according to the OODifls

n	$\overline{\mathcal{D}}_{OO}(S,P)$	Worst $\overline{\mathcal{D}}_{OO}(S,S^1)$ GeoForce	Worst $\overline{\mathcal{D}}_{OO}(S,S^1)$ VDCB
25	0.6688814	0.0602756	0.1405930
50	0.6686408	0.0450624	0.1072064
100	0.6663388	0.0318940	0.0810908

Table 6.7: $\overline{\mathcal{D}}_{OO}(S,P)$ and $\overline{\mathcal{D}}_{OO}(S,S^1)$ after one iteration of the VDCB and GeoForce algorithms.

iterations. Layouts that start out the most evenly distributed according to the CP and FM models change the least according to the OO model after several iterations of layout algorithms.

6.2.6 Summary – Similarity Measures

In this section we provided evidence that the layout algorithms satisfy criterion **CB3**. We showed that $\overline{\mathcal{D}}_M(S,S') < \overline{\mathcal{D}}_M(S,P)$ after several iterations of both algorithms for each of the similarity models M. For all but the DE model, the average difference decreases as n increases because as n increases, the number of small Voronoi regions increases and the amount the nodes can move after each iteration decreases. For measurements based on models of distance, the AD and DM models, the greatest difference is observed for layouts with one or two clusters. In this case, as the nodes become spread out, they move a greater distance from where they started than in layouts with more clusters. For all other similarity measures, the greatest difference is observed for medium sized clusters. The least difference occurs for layouts that are the most evenly distributed initially.

6.3 Comparing the Algorithms

In this section we compare the GeoForce and the VDCB algorithms with respect to to speed and criteria **CB2** and **CB3** presented in Chapter 4. In Section 5.2.1, we described the GeoForce algorithm as a cluster busting algorithm that behaves like a smart VDCB algorithm by using forces. While designing the GeoForce algorithm and determining the forces, we concluded that the GeoForce algorithm did a better job of cluster busting in anchored graph drawing than the VDCB algorithm. This judgement was based on examining layouts produced by the two algorithms. After examining the test data we have found that the GeoForce algorithm does not always do a better job of cluster busting in anchored graph drawing according to the evaluations described in Chapter 4. In this section we discuss the results of running the tests and present some example layouts produced by the two layout algorithms.

The VDCB algorithm is an $O(t * n \log n)$ time algorithm and the GeoForce algorithm is an $O(t * n^2)$ time algorithm where t is the number of times the algorithm is iterated. In practise, the VDCB algorithm is faster than the GeoForce algorithm for the same sized layouts. If speed of execution is a major consideration, for large n, it makes sense to choose the VDCB algorithm over the GeoForce algorithm on this basis alone. For smaller n and for situations in which speed is not as crucial, there are several other issues to consider when deciding which algorithm to use including the type of initial layout (highly clustered or not), the number of iterations, and the measurements used.

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.

We found the results were not necessarily consistent across the different similarity measures. For certain types of layout the VDCB algorithm performs better according to all models except the AD model. For all types of layout, the VDCB algorithm does a better job according to the DE and DM models of similarity. This observation was surprising since we initially thought the DE model would evaluate similarity according to how clusters in the layout were retained. Our visual evaluations of layouts seemed to show that the GeoForce algorithm kept clusters together better than the VDCB algorithm. Our experiments seemed to show that either the DE model does not measure cluster similarity as we originally thought or on average the GeoForce algorithm does a poorer job of keeping clusters of nodes together than the VDCB algorithm. Upon further consideration, it appears that the AD model provides a better measure of how well clusters are retained than the DE model. In Figure 6.2, we show a drawing D of $n = 50$ nodes with $\mathcal{K} = 2$ clusters and the drawing D' after 20 iterations of the GeoForce algorithm. Figure 6.3 shows the same drawing after 10 iterations of the VDCB algorithm. The layout produced by the VDCB algorithm is more evenly distributed according to both measures of distribution and is more similar to the original layout according to all similarity measures except the AD measure.

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

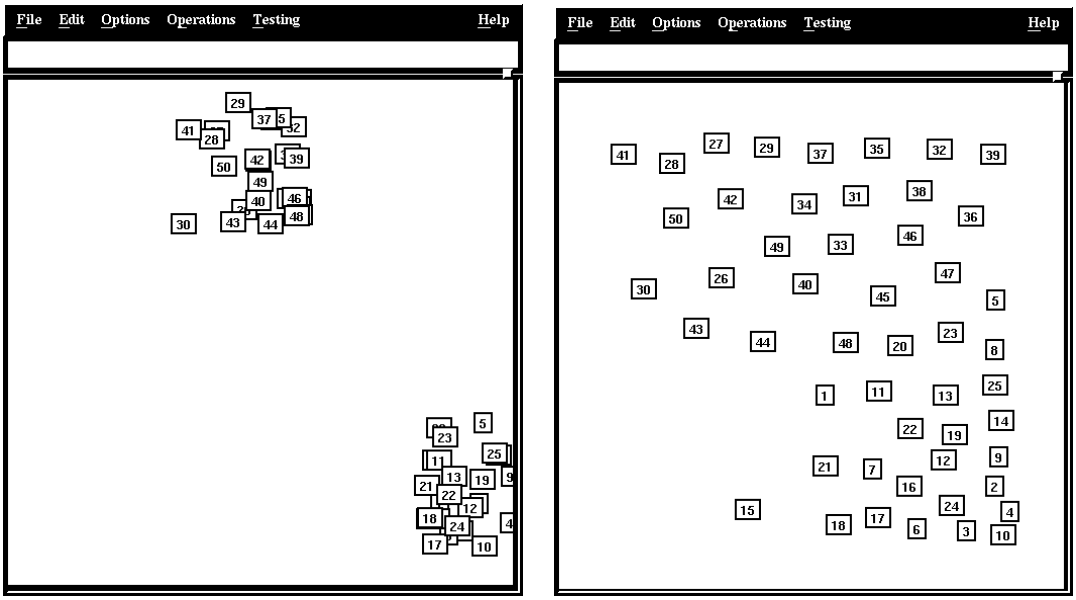

Figure 6.2: A layout with $\mathcal{K} = 2$ clusters of size 25 each and that layout after 20 iterations of the GeoForce algorithm.

large number of clusters. For layouts that are pseudo-random in the uniform distribution 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. From this observation, it appears that the GeoForce algorithm performs better for layouts that are pseudo-random in the uniform distribution in W. Figure 6.4 shows such a layout with $n = 25$ nodes. and the layout after one iteration of the GeoForce algorithm. Figure 6.5 shows the same layout after one iteration of the VDCB algorithm.

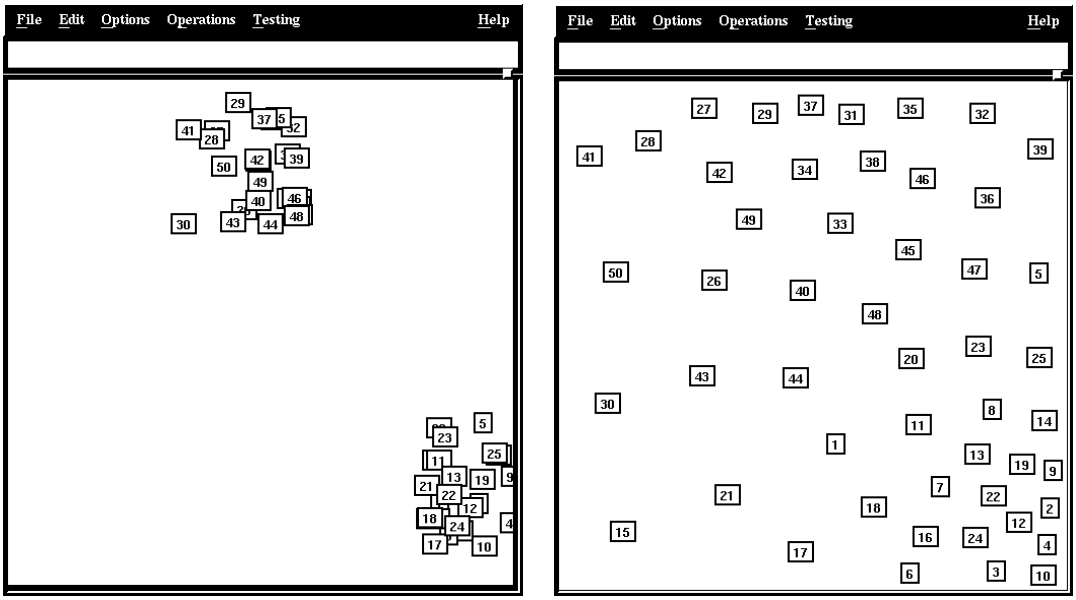

Figure 6.3: A layout with $\mathcal{K} = 2$ clusters of size 25 each and that layout after 10 iterations of the VDCB algorithm.

On average it takes more iterations for layouts produced by the GeoForce algorithm to have the same distribution level as layouts produced by the VDCB algorithm. For layouts with a small number of clusters it takes the GeoForce algorithm anywhere from 2 to 5 times as many iterations as the VDCB algorithm to produce layouts with the same distribution level. For layouts with a larger number of clusters it takes the GeoForce algorithm anywhere from 3 to 15 times as many iterations as the VDCB algorithm to produce layouts with the same distribution level. 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

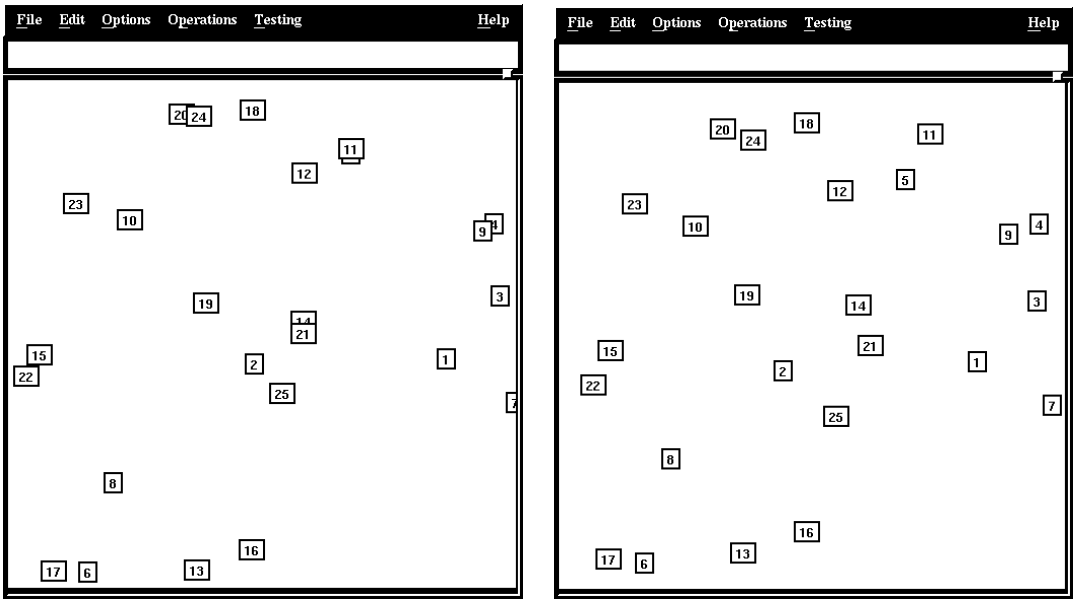

Figure 6.4: A layout with $\mathcal{K} = 25$ clusters of size 1 each and that layout after 1 iteration of the GeoForce algorithm.

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.

Figure 6.6 shows a layout with $\mathcal{K} = 25$ clusters of size 4 each and the layout after 20 iterations of the GeoForce algorithm. Figure 6.7 shows the same layout after 11 iterations of the VDCB algorithm. The layout produced by the VDCB algorithm is more evenly distributed according to the distribution measures and is more similar to the original layout according to all of the similarity measures expect for the DE

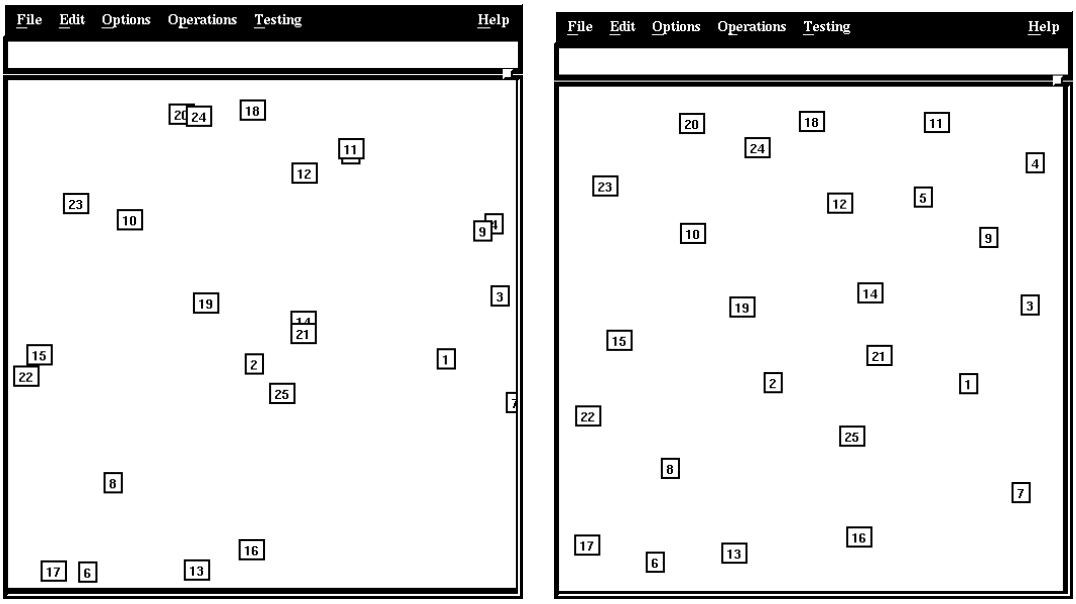

Figure 6.5: A layout with $\mathcal{K} = 25$ clusters of size 1 each and that layout after 1 iteration of the VDCB algorithm.

measure; therefore, the VDCB algorithm performs better on this layout than the GeoForce algorithm according to the measures.

We have observed that for initial layouts with a small number of large clusters the GeoForce algorithm performs better than the VDCB algorithm. This observation poses a problem for automatically choosing which layout algorithm to use given an initial layout. It appears that the choice of algorithm depends on the number of clusters. Unfortunately, the number of clusters cannot be easily determined by computing the distribution levels of the initial layouts. For example, as can be seen in Tables 6.1 and 6.2 it is difficult to distinguish between layouts with $\mathcal{K} = 5$ clusters of size 10 each and layouts with $\mathcal{K} = 10$ clusters of size 5 each on the basis of the distribution measures. Determining the number of clusters in a layout of points is a

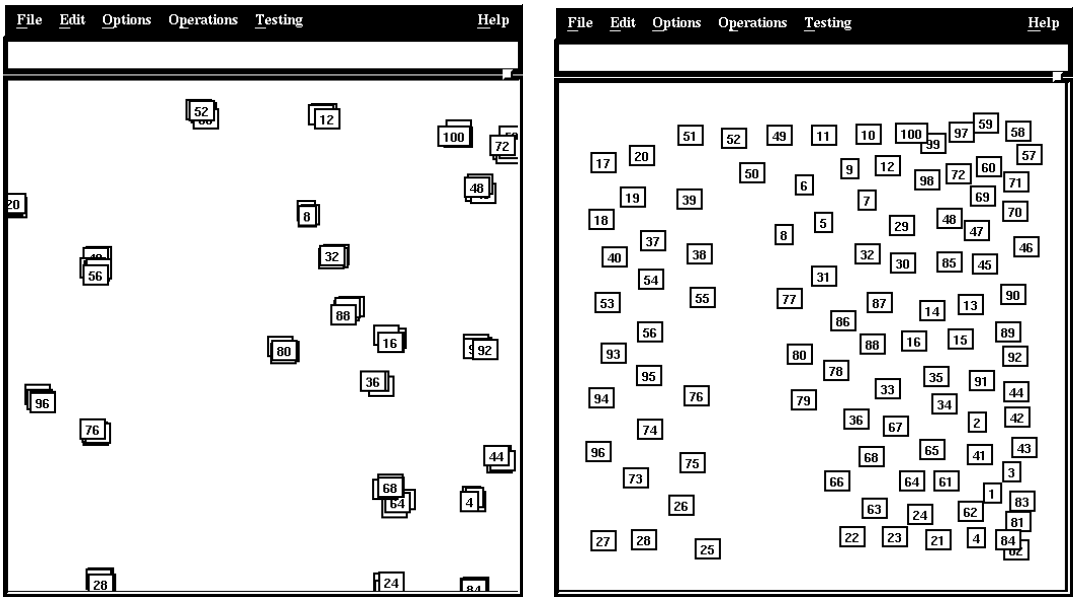

Figure 6.6: A layout with $\mathcal{K} = 25$ clusters of size 4 each and that layout after 20 iterations of the GeoForce algorithm.

difficult problem [6, 30, 55, 56].

One possibility is that the number of clusters can be provided as part of the input. In the absence of knowledge about the number of clusters in the input graph, it is recommended that the VDCB algorithm be used. It is faster and takes few iterations to produce evenly distributed layouts. The experiments show that for layouts with a small number of clusters on which the GeoForce algorithm might have performed better, the VDCB algorithm still performs reasonably well. Since the VDCB algorithm is fast, it can be applied first and if the user is unhappy with the layout, the GeoForce algorithm can then be used.

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 in general it does not always do so. In fact, we would recommend using the

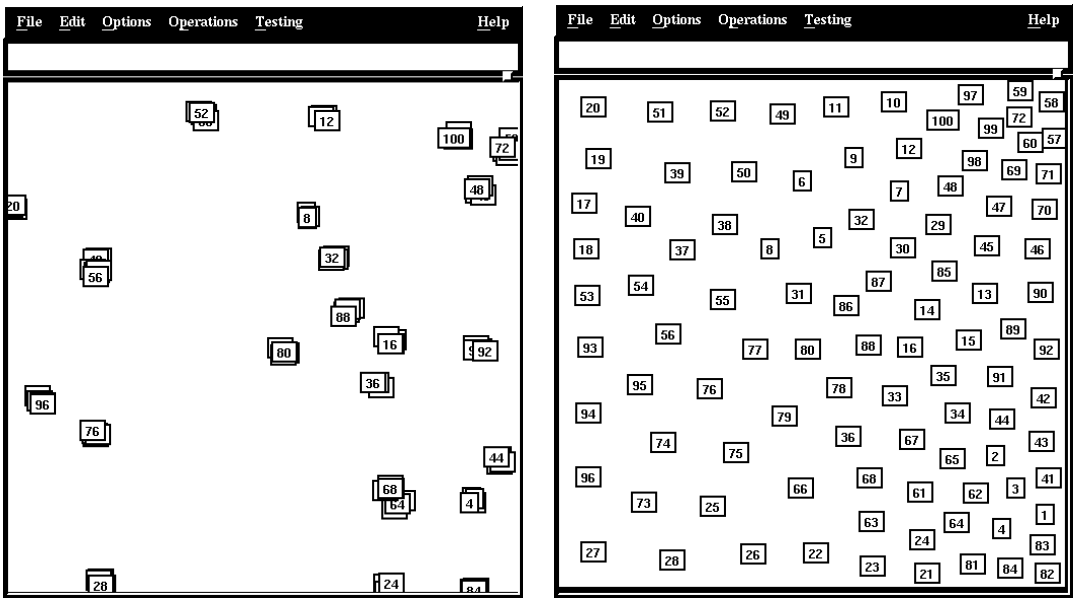

Figure 6.7: A layout with $\mathcal{K} = 25$ clusters of size 4 each and that layout after 11 iterations of the VDCB algorithm.

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. We discuss this further in Chapter 7.

6.4 Determining the Default Threshold Values

In this section we discuss default threshold values for the distribution and similarity measures, and the default number of iterations. We also discuss which tests should be used in different circumstances. Determining the default threshold values is a difficult problem because deciding what levels are appropriate is subjective. Furthermore, the distribution measures, the DE similarity measure, and the number of iterations

needed to reach a given level of distribution depend on n. To try to come up with default threshold values we looked at layouts produced by the algorithms and visually determined which ones were spread out enough according to our opinion as well as using the results of the experiments.

6.4.1 Thresholds for the Distribution Measures

Determining threshold values for the distribution measures requires identifying an ideal level of distribution for a layout with n nodes. In layouts with this ideal distribution level the interaction between the nodes should be easily read. In Section 6.3 we concluded that between one and three iterations of the GeoForce algorithm on layouts that are pseudo-random in the uniform distribution in W produces reasonably good layouts. We found that one iteration for $n = 25$, two iterations for $n = 50$, and three iterations for $n = 100$ are sufficient. One way to determine the default threshold value for layouts of size $n = 25$ is to use the average distribution levels after one iteration of the GeoForce algorithm on layouts that are pseudo-random in the uniform distribution in W. This value could be computed for each value of n and stored in a table that could be read by the layout algorithms. Alternatively, the values for several n could be plotted and a function based on the plot could be determined. This approach would save the space required by the table and the problem of having to consider all possible values of n. A plot of appropriate default distribution levels according to the CP and FM measures is presented in Figure 6.8. Obviously, values are required for several more n but from these plots it appears that as n doubles, the default CP measure decreases by slightly more than $\frac{2}{3}$ and the default FM measure decreases by some function of $\frac{1}{10}$.

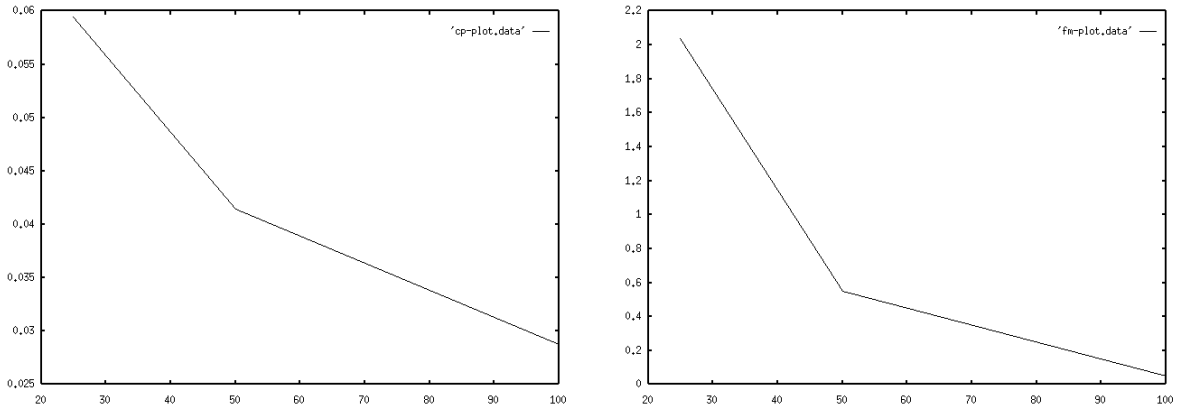

Figure 6.8: The plots of n versus default CP values on the left and n versus default FM values on the right.

6.4.2 Thresholds for the Difference Measures

Unlike the distribution measures, the similarity measures do not directly depend on n because they are normalized. The one exception is the DE measure. Determining if two layouts are similar is subjective and depends on the application. Incorporating user input in determining the default values for the difference measures would be helpful. If we choose default difference levels that are too small, the layout algorithms will stop before the layouts become distributed enough to be readable. On the other hand, if we choose distribution levels that are too high (say at the average level of difference between two random layouts), the algorithms will always stop because a threshold level of distribution in the layouts is reached or the maximum number of iterations is reached. We define a *tolerance factor* $0 \leq \mathcal{F} \leq 1$ that is a fraction of the average difference between two random layouts. Setting $\mathcal{F} = 1$ is the same as setting $\mathcal{T}_M = \overline{\mathcal{D}}_M(S, P)$ for two random layouts S and P according to the model M. Setting $\mathcal{F} = 0$ insists that the layout does not change according to the given model. The

tolerance factor depends on the user and the applications. We prefer to work with a tolerance factor of $\mathcal{F} = \frac{1}{4}$; therefore, the default threshold for a difference model M is $\mathcal{T}_M = \frac{1}{4}\overline{\mathcal{D}}_M(S, P)$. A method such as those described in 6.4.1 would have to be used to determine threshold values of the DE measure for various values of n.

6.4.3 Determining the Default Number of Iterations

The number of iterations needed to produce layouts that satisfy the goals of cluster busting in anchored graph drawing depends on the number of nodes in the layout, the number of clusters in the starting layout, and the layout algorithm. In Section 6.3, we argue that the number of clusters cannot be easily determined by computing the distribution levels of the initial layouts. For this reason, we consider only the number of no[illegible]

6.4.4 Deciding Which Measures to Use

There are several issues to consider when deciding which measurements to use. The VDCB takes $O(n \log n)$ time for each iteration and it may not make sense to measure the layout with an $O(n^2)$ time algorithm. For this reason, measurements based on the FM, AD, and λM models should not be used to determine stopping conditions of the VDCB algorithm. Any of the measurements can be used to determine when to stop iterating the GeoForce algorithm. While comparing and evaluating the algorithms, we found that in most cases the CP and FM measures correlate. For this reason, it makes sense to use the CP measure to determine stopping conditions when speed is a consideration.

Determining when two layouts are similar is subjective and depends on the application. The choice of which difference measure should be used to determine stopping conditions depends on the user and the application. User input for this choice is essential.

6.5 Convergence

Since both algorithms are iterative, it would be interesting if we could prove that the algorithms converge. In Section 6.1.1 we showed that the distribution of n nodes on an isothetic line or at the vertices of an isothetic grid eventually increases according to the CP measure of distribution. This result does not imply that the layout converges with $d_0 = d_1 = \cdots = d_n$. In this section we show that with the VDCB algorithm, n nodes on an isothetic line or at the vertices of an isothetic grid converge to the situation where $d_0 = d_1 = \cdots = d_n$. We also present a discussion of the convergence

of the VDCB algorithm for the case of n points in general and compare the VDCB algorithm with two other algorithms.

We ran experiments to determine how many iterations of the implementations of the VDCB and GeoForce algorithms are needed for the resulting layout to "converge"; that is, how many iterations it takes before the maximum difference between any x- or y-value differs from the previous iteration by less than a certain value. The value we chose is 0.000001. The GeoForce algorithm did not stabilize after over 20000 iterations for the layouts tested. A more careful choice of force functions and/or constants might produce a GeoForce algorithm that converges in practise.

6.5.1 The VDCB Algorithm in Practise

We executed the VDCB algorithm on 1000 randomly generated layouts with a varying number of clusters. After each iteration of the VDCB algorithm we computed the change in x- and y-values for each node. When the maximum change was less than 0.000001, we declared the layout to be converged and reported the number of iterations it took. Table 6.8 shows the results of running the above experiments. For each type of initial layout tested, the minimum, maximum, and average number of iterations needed to converge are given. All of the layouts tested converged according to the above definition. Not surprisingly, as n increases, the average number of iterations necessary for convergence increases. Layouts that are more evenly distributed according to the CP and FM models of distribution take fewer iterations to converge on average.

For each of the converged layouts S', we computed $\overline{\Xi}_{\mathcal{E}}(S')$ for each of the distribution models and $\overline{\mathcal{D}}_M(S, S')$ for each of the similarity models. For a given n, regardless

n	$\mathcal{K}$	$n/\mathcal{K}$	Iterations		
			Min	Max	Average
25	1	25	145	1585	348.847
	5	5	120	1428	326.826
	25	1	129	1149	311.055
50	1	50	248	2587	599.760
	2	25	229	1967	573.484
	5	10	226	2413	573.832
	10	5	211	3025	560.128
	25	2	219	2134	548.641
	50	1	222	2498	514.911
100	1	100	458	3251	1076.681
	2	50	465	3679	1044.003
	4	25	475	3355	1016.804
	10	10	405	3635	987.254
	25	4	402	3133	918.872
	50	2	340	2657	888.499
	100	1	353	3710	863.582

Table 6.8: VDCB convergence data for different initial layouts.

of the distribution of the initial layouts, the distribution levels of the resulting layouts are the same. The average distribution and difference levels for the converged layouts are very close to the average levels after $t = 100$ iterations of the VDCB algorithm. On average, after 100 iterations of the VDCB algorithm the layouts do not change very much according to each of the measurements even though a greater number of iterations is required for the layouts to converge according to the above definition of convergence.

6.5.2 The VDCB Algorithm in Theory

A natural question to ask about the VDCB algorithm is if it converges theoretically. That is, after continued iterations of the VDCB, do the nodes end up in positions that are centroids of their Voronoi regions? Two algorithms similar to the VDCB are presented in [46] and [54] and the question of convergence was posed about the algorithm in [54]. In [46], a geographical optimization problem is investigated and an algorithm similar to the VDCB is presented. In this section, we show how these results relate to the question of convergence of the VDCB and show that the VDCB converges for n nodes on an isothetic line in W. The results in this section were previously published in [53].

Convergence for n Nodes on an Isothetic Line

Previously, we showed that the VDCB algorithm causes the distances d_i to eventually increase for n nodes on an isothetic line or at the vertices of an isothetic grid. This does not imply convergence of the VDCB algorithm to the situation that the distances are equal. We now consider the question of convergence of the VDCB for the case

of n nodes on an isothetic line l in W. Assume, without loss of generality, that l is horizontal. Before stating the results, we present a brief reminder of properties of eigenvalues and eigenvectors.

Given a square matrix $\mathbf{A}$, a non-zero vector $\mathbf{x}$ is an *eigenvector* of $\mathbf{A}$ if there is a scalar λ such that $\mathbf{Ax} = \lambda\mathbf{x}$; that is, if $\mathbf{Ax}$ is a scalar multiple of $\mathbf{x}$. We say that λ is an *eigenvalue* of $\mathbf{A}$ and $\mathbf{x}$ is an eigenvector of A *corresponding* to the eigenvalue λ. The eigenvalues of an $n \times n$ matrix $\mathbf{A}$ are the same as those of the transpose of $\mathbf{A}$, $\mathbf{A}^T$. Let $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_r}$ be the eigenvectors of $\mathbf{A}$ and let $\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_r}$ be the eigenvectors of $\mathbf{A}^T$, $r \leq n$. The vector $\mathbf{y}_i$ associated with the eigenvalue λ_i is called the *left* eigenvector of $\mathbf{A}$ because the equation:

$$\mathbf{A}^T\mathbf{y}_i = \lambda_i\mathbf{y}_i$$

can be written as:

$$\mathbf{y}_i^T\mathbf{A} = \lambda_i\mathbf{y}_i^T.$$

The eigenvector $\mathbf{x}_i$ of $\mathbf{A}$ associated with the eigenvalue λ_i such that:

$$\mathbf{Ax}_i = \lambda_i\mathbf{x}_i$$

is called the *right* eigenvector of $\mathbf{A}$.

As in Section 6.1.1, let $\mathbf{\Delta}^t = [d_0^t d_1^t \ldots d_n^t]$ be the vector of distances between adjacent nodes on l at iteration t. Lemma 6.1.2 of Section 6.1.1 shows how the distances in $\mathbf{\Delta}^{t+1}$ can be expressed as a linear function of the distances in $\mathbf{\Delta}^t$. This means we can express the transformation from $\mathbf{\Delta}^t$ to $\mathbf{\Delta}^{t+1}$ as the multiplication of an $(n+1) \times (n+1)$ matrix M times a vector $\mathbf{\Delta}^t = [d_0^t d_1^t \ldots d_n^t]$ of length $n+1$:

$$\mathbf{\Delta}^{t+1} = \mathbf{M}\mathbf{\Delta}^t$$

$$\begin{bmatrix} d_0^{t+1} \\ d_1^{t+1} \\ \vdots \\ d_n^{t+1} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & \cdots & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & \cdots & 0 \\ & & & \ddots & & & \\ 0 & \cdots & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & \cdots & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & \cdots & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} d_0^t \\ d_1^t \\ \vdots \\ d_n^t \end{bmatrix}$$

Figure 6.9: $\mathbf{\Delta}^{t+1} = \mathbf{M}\mathbf{\Delta}^t$

as shown in Figure 6.9. Therefore, given a starting vector $\mathbf{\Delta}^0 = [d_0^0 d_1^0 \ldots d_n^0]$, the vector of distances after t iterations of the VDCB is given by

$$\mathbf{\Delta}^t = \mathbf{M}^t \mathbf{\Delta}^0$$

To prove that the VDCB algorithm converges for n nodes on a horizontal line, we apply results for *primitive* matrices presented in [74] to the matrix $\mathbf{M}$. A square non-negative matrix $\mathbf{A}$ is said to be primitive if there exists an integer k such that each element of the matrix $\mathbf{A}^k$, the matrix A multiplied by itself k times, is greater than $\mathbf{0}$ [74]. It is easy to see that the the matrix $\mathbf{M}$ is a primitive matrix by observing that for $k = n$, each element of $\mathbf{M}^k$ is greater than $\mathbf{0}$.

From the *Perron–Frobenius Theorem for primitive matrices* presented in [74], if $\mathbf{A}$ is an $(n+1) \times (n+1)$ non-negative primitive matrix then there exists an eigenvalue λ_1 of $\mathbf{A}$ such that:

1. λ_1 is real and $\lambda_1 > 0$,

2. strictly positive left and right eigenvectors can be associated with λ_1,

3. $\lambda_1 > |\lambda_i|$ for any eigenvalue λ_i, $i \neq 1$ of $\mathbf{A}$,

4. the eigenvectors associated with λ_1 are unique to constant multiples

Let $\lambda_1, \lambda_2, \ldots, \lambda_r$ be the distinct eigenvalues of $\mathbf{A}$ where $r \leq n+1$ such that $\lambda_1 > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_r|$, and let m_i be the *multiplicity* of λ_i where $m_1 + m_2 + m_3 + \cdots + m_r = n+1$. The multiplicity m_i of an eigenvalue λ_i is equal to the dimension of the space of eigenvectors associated with λ_i. More details can be found in a linear algebra book such as [44].

The following result about primitive matrices is presented in [74]:

Theorem 6.5.1 (From [74]) *For any* primitive *matrix* $\mathbf{A}$ *and* $\lambda_1, \lambda_2, \ldots, \lambda_r$ *defined as above:*
If $\lambda_2 \neq 0$ *then as* $t \to \infty$,

$$\mathbf{A}^t = \lambda_1^t \mathbf{x}_1 \mathbf{y}_1^T + O(t^s |\lambda_2|^t)$$

element-wise, where $s = m_2 - 1$, *and* $\mathbf{x}_1$ *and* $\mathbf{y}_1$ *are positive right and left eigenvectors of* λ_1, *respectively, such that* $\mathbf{y}_1^T \mathbf{x}_1 = 1$.

We first show that $\lambda_1 = 1$ and that $0 \leq \lambda_i < 1$ for $i = 2, 3, \ldots r$. Consider the following equation:

$$\mathbf{M}\mathbf{x} = \lambda \mathbf{x} \tag{6.1}$$

where $\mathbf{x} = [\chi_0 \chi_1 \ldots \chi_n]$ is an eigenvector vector of distances corresponding to the eigenvalue λ. See Figure 6.10.

Lemma 6.5.1 *If* $\xi = \{\lambda_1, \lambda_2, \ldots, \lambda_r\}$, $r \leq n+1$ *is the set of distinct eigenvalues of* $\mathbf{M}$, *such that*

$$\lambda_1 > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_r|,$$

then $\lambda_1 = 1$.

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & \cdots & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & \cdots & 0 \\ & & & \ddots & & & \\ 0 & \cdots & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & \cdots & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & \cdots & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_n \end{bmatrix} = \begin{bmatrix} \lambda\chi_0 \\ \lambda\chi_1 \\ \vdots \\ \lambda\chi_n \end{bmatrix}$$

Figure 6.10: The matrix $\mathbf{M}$ and an eigenvalue λ: $\mathbf{Mx} = \lambda\mathbf{x}$

Proof: We show that there is no eigenvalue λ that is greater than 1 and that there is no eigenvalue λ that is less that 0.

Assume that there is a $\lambda > 1$. If this is true, we establish that $\chi_0 < \chi_1 < \cdots < \chi_n$. From equation (6.1), we have

$$\frac{1}{2}\chi_0 + \frac{1}{2}\chi_1 = \lambda\chi_0.$$

$$\Longrightarrow (\lambda - \frac{1}{2})\chi_0 = \frac{1}{2}\chi_1$$

If $\lambda > 1$, then $\lambda - \frac{1}{2} > \frac{1}{2}$ and $\chi_0 < \chi_1$.

By induction, we can show that $\chi_i < \chi_{i+1}$ for $i = 1, 2, \ldots, n-1$. We establish the base case by showing that $\chi_1 < \chi_2$. From equation (6.1), we have

$$\frac{1}{4}\chi_0 + \frac{1}{2}\chi_1 + \frac{1}{4}\chi_2 = \lambda\chi_1$$

$$\Longrightarrow (\lambda - \frac{1}{2})\chi_1 = \frac{1}{4}\chi_0 + \frac{1}{4}\chi_2$$

If $\lambda > 1$, then $\lambda - \frac{1}{2} > \frac{1}{2}$ and either $\chi_1 < \chi_0$ or $\chi_1 < \chi_2$ or both. Since we showed that $\chi_0 < \chi_1$, then $\chi_1 < \chi_2$. Assume that $\chi_k < \chi_{k+1}$ if $\lambda > 1$, and consider χ_{k+1}. From equation (6.1),

$$\frac{1}{4}\chi_k + \frac{1}{2}\chi_{k+1} + \frac{1}{4}\chi_{k+2} = \lambda\chi_{k+1}$$

$$\Longrightarrow (\lambda - \frac{1}{2})\chi_{k+1} = \frac{1}{4}\chi_k + \frac{1}{4}\chi_{k+2}$$

If $\lambda > 1$, then $\lambda - \frac{1}{2} > \frac{1}{2}$ and either $\chi_{k+1} < \chi_k$ or $\chi_{k+1} < \chi_{k+2}$ or both. By the assumption, $\chi_k < \chi_{k+1}$, therefore, $\chi_{k+1} < \chi_{k+2}$, and $\chi_i < \chi_{i+1}$ for $i = 1, 2, \ldots, n-1$. We have shown that if there is a $\lambda > 1$ then $\chi_0 < \chi_1 < \cdots < \chi_n$.

We now show that if there is a $\lambda > 1$ then $\chi_n < \chi_{n-1}$ which results in a contradiction. If $\lambda > 1$ then from equation (6.1), we have

$$\frac{1}{2}\chi_{n-1} + \frac{1}{2}\chi_n = \lambda\chi_n$$

$$\Longrightarrow (\lambda - \frac{1}{2})\chi_n = \frac{1}{2}\chi_{n-1}$$

If $\lambda > 1$, then $\lambda - \frac{1}{2} > \frac{1}{2}$ and $\chi_n < \chi_{n-1}$.

We have shown that there is no eigenvalue λ such that $\lambda > 1$. A similar argument can be given to show that there is no eigenvalue λ such that $\lambda < 0$ by showing that if there is a $\lambda < 0$ then $|\chi_0| < |\chi_1| < \cdots < |\chi_n|$ and establishing a contradiction by showing that if there is a $\lambda < 0$ then $|\chi_n| < |\chi_{n-1}|$.

It is easy to see that $\lambda_1 = 1$ is an eigenvalue of $\mathbf{M}$ by observing that $\mathbf{M}\mathbf{x}_1 = \mathbf{x}_1$ for $\mathbf{x}_1 = [\chi_0\chi_1 \ldots \chi_n]$ such that $\chi_0 = \chi_1 = \cdots = \chi_n$. □

We use the previous lemma to show that for n nodes on an isothetic line in W, the VDCB algorithm converges to the situation where $d_0 = d_1 = \cdots = d_n$.

Theorem 6.5.2 *For n nodes on an isothetic line in W the VDCB algorithm converges to the situation where $d_0 = d_1 = \cdots = d_n$.*

Proof: Since the matrix $\mathbf{M}$ is primitive, Theorem 6.5.1 applies to $\mathbf{M}$ and, as $t \to \infty$,

$$\mathbf{M}^t \to \lambda_1^t \mathbf{x}_1 \mathbf{y}_1^T + O(t^s|\lambda_2|^t).$$

As $t \to \infty$, $O(t^s|\lambda_2|^t) \to 0$ because $0 \leq |\lambda_2| < 1$ by Lemma 6.5.1, and if we write $|\lambda_2| = 1/c$ for some integer c, then it is easy to see that

$$\lim_{t\to\infty} \frac{t^s}{c^t} = 0$$

Since $\lambda_1 = 1$, as $t \to \infty$,

$$\mathbf{M}^t \to \mathbf{x}_1 \mathbf{y}_1^T.$$

It is easy to verify that

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

and

$$\mathbf{y}_1^T = \begin{bmatrix} \frac{1}{2n} & \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} & \frac{1}{2n} \end{bmatrix}$$

are right and left eigenvectors of $\mathbf{M}$ associated with the eigenvalue $\lambda_1 = 1$ such that $\mathbf{y}_1^T \mathbf{x}_1 = 1$.

Given n nodes on an isothetic line in W,

$$\lim_{t\to\infty} \mathbf{M}^t = \begin{bmatrix} \frac{1}{2n} & \frac{1}{n} & \cdots & \frac{1}{n} & \frac{1}{2n} \\ \frac{1}{2n} & \frac{1}{n} & \cdots & \frac{1}{n} & \frac{1}{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{1}{2n} & \frac{1}{n} & \cdots & \frac{1}{n} & \frac{1}{2n} \end{bmatrix}$$

Therefore, given a starting vector of distances $\mathbf{\Delta}^0 = [d_0^0 d_1^0 \cdots d_n^0]$:

$$\lim_{t\to\infty} \mathbf{M}^t \mathbf{\Delta}^0 = \begin{bmatrix} \frac{1}{2n}d_0^0 + \frac{1}{n}d_1^0 + \cdots + \frac{1}{n}d_{n-1}^0 + \frac{1}{2n}d_n^0 \\ \frac{1}{2n}d_0^0 + \frac{1}{n}d_1^0 + \cdots + \frac{1}{n}d_{n-1}^0 + \frac{1}{2n}d_n^0 \\ \vdots \\ \frac{1}{2n}d_0^0 + \frac{1}{n}d_1^0 + \cdots + \frac{1}{n}d_{n-1}^0 + \frac{1}{2n}d_n^0 \end{bmatrix}$$

and the distances converge to the situation where

$$\begin{aligned} d_0^t &= d_1^t = \cdots = d_n^t \\ &= \frac{d_0 + 2d_1 + \cdots + 2d_{n-1} + d_n}{2n} \end{aligned}$$

as the number of iterations of the VDCB algorithm approaches infinity.

As in Lemma 6.1.4, this result can be used to show that for n nodes at the vertices of a (not necessarily evenly spaced) isothetic grid, the VDCB algorithm converges to the situation where the horizontal distances are equal and the vertical distances are equal.

Convergence of n Nodes in the Plane

In this section we discuss the convergence of three algorithms: the *k-means algorithm* as it is presented by MacQueen in [54], a geographical optimization algorithm we call the *IMO algorithm* presented by Iri, Murota, and Ohya in [46], and the VDCB algorithm. Each algorithm takes as input a set of points in a closed convex region. In the k-means algorithm, k points are chosen randomly initially and new points are computed as the *means* or *averages* of a group of points. In the IMO algorithm the n input points are called *facilities*. In order to carry out the comparison of these two algorithms with the VDCB, we consider each algorithm to take as input the set $S = \{p_0, p_1, \ldots, p_{n-1}\}$ of n nodes in an isothetic rectangle W. We let each node $p_i \in S$ be at the initial position p_i^0 such that at iteration t of an algorithm, it is at position p_i^t, $i = 0, 1, \ldots, n-1$.

In [54], MacQueen proves convergence of the k-means algorithm. Given the set S of n nodes placed randomly in W such that p_i has starting weight $w_i = 1$, $i = 0, 1, \ldots, n-1$, the k-means algorithm consists in iterating the following: Compute

the Voronoi diagram of S clipped within W, choose a point z at random in W and let its nearest neighbour in S be p_j. Move p_j such that:

$$p_j^{t+1} \leftarrow \frac{w_j p_j^t + z}{w_j + 1}$$

and adjust the weight of p_j:

$$w_j \leftarrow w_j + 1.$$

Following the terminology in [54], we let the centroid of the Voronoi region $Vor(p_i)$ of the node p_i be u_i, and we say the position of a node p_i is *unbiased* if $p_i = u_i$.

The IMO algorithm which is similar to the VDCB and the k-means algorithm is presented in [46] where a geographical optimization problem is considered. Given the set S of n nodes placed randomly in W, variants on an algorithm that attempt to minimize the following function are described:

$$F(p_0, p_1, \ldots, p_{n-1}) = \frac{1}{2} \sum_{i=0}^{n-1} \int_{Vor(p_i)} f(|z - p_i|^2) dz.$$

We let the function $f(|z-p_i|^2) = |z-p_i|^2$. The algorithm variants have the following form: Given initial (random) positions of the nodes, at each iteration t, a search direction δ^t and distance α^t are computed for each node p_i, $i = 0, 1, \ldots, n-1$ and p_i is moved such that:

$$p_i^{t+1} \leftarrow p_i^t + \alpha^t \delta^t.$$

This process is iterated until some stopping condition is met. The algorithm can vary according to the way in which the δ_i's and α_i's are chosen and according to which criteria are used for the stopping condition.

Let u_i^t be the centroid of $Vor(p_i)$ at iteration t. Recall that at each iteration of the VDCB algorithm, p_i is moved such that:

$$p_i^{t+1} \leftarrow u_i^t$$

for $i = 0, 1, \ldots, n-1$.

We can modify the VDCB algorithm and a variant of the IMO algorithm to look more like the k-means algorithm. In the VDCB and the IMO algorithms, a new position is computed for each of the n nodes and all nodes are moved to their new positions at each iteration of the algorithm. In the k-means algorithm, at each iteration of the algorithm, one node is moved to a new position. The first modification we make to the VDCB and the IMO algorithm is to move only one node at each iteration; the question is which node to move. In the k-means algorithm, a point z is chosen at random in W and the node to which it is closest is moved. Therefore, in the VDCB and the IMO algorithms, nodes are chosen with probability depending on the size of their Voronoi regions: nodes with a large area Voronoi region are chosen with higher probability than nodes with a small area Voronoi region.

Let p_j be the node to be moved at the current iteration. In the k-means algorithm, p_j is moved in the direction of the randomly chosen point z where the nearest neighbour of z in S is p_j and the *expected* value of z is u_j. The distance p_j is moved is $\frac{1}{w_j^t+1}$ times the distance between z and p_j. That is, given the position p_j^t at iteration t,

$$p_j^{t+1} \leftarrow p_j^t + \frac{1}{w_j^t + 1}(z - p_j^t)$$

where the expected value of z is u_j^t and w_j^t is the weight of p_j^t at iteration t. In a variant of the IMO algorithm, p_j is moved in the direction of u_j by some distance. That is,

$$p_j^{t+1} \leftarrow p_j^t + \alpha(u_j^t - p_j^t)$$

where α is a computed distance. In the VDCB, p_j is moved to u_j. That is,

$$p_j^{t+1} \leftarrow p_j^t + (u_j^t - p_j^t)$$

Therefore, in the IMO algorithm, we let $\alpha = \frac{1}{w_j^t+1}$ and we further modify the VDCB such that at iteration $t+1$, the node p_j is moved $\frac{1}{w_j^t+1}$ times the distance between u_j^t and p_j^t.

MacQueen shows that as the number of iterations of the k-means algorithm, $t \rightarrow \infty$, the random variable $W(p_0^t, p_1^t, \ldots, p_{n-1}^t)$ converges where

$$W(p_0^t, p_1^t, \ldots, p_{n-1}^t) = \sum_{i=0}^{n-1} \int_{Vor(p_i)} |z - p_i^t|^2 dz.$$

It is shown that $W(p_0^t, p_1^t, \ldots, p_{n-1}^t)$ converges to the value:

$$V(q_0, q_1, \ldots, q_{n-1}) = \sum_{i=0}^{n-1} \int_{Vor(p_i)} |z - q_i|^2 dz$$

for some set $Q = (q_0, q_1, \ldots, q_{n-1})$ of points in W that are distinct and unbiased. Note that it is not shown that the set S^t converges to a given configuration. From MacQueen's convergence proof, it is shown that:

$$lim_{t \rightarrow \infty} \sum_{i=0}^{n-1} dist(p_i^t, u_i^t) = 0$$

The convergence proof in [54] for the k-means algorithm supports the possibility that the modifications described above for the VDCB algorithm and the IMO algorithm also converge. It is an open problem to prove that the VDCB algorithm and the IMO algorithm converge. Another interesting problem is to determine configurations of unbiased nodes. For example, if $n = pq$, the nodes are unbiased if they are at vertices of an evenly spaced isothetic grid of size $p \times q$ in R such that the horizontal grid lines are evenly spaced and the vertical grid lines are evenly spaced. As noted earlier, this configuration is unbiased but not stable.

Chapter 7

Summary and Conclusions

In this dissertation, 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 and proved some properties about the algorithms under restricted classes of input. We have implemented the layout algorithms and executed them on several types of input layouts. We provide measurements for evaluating layouts produced by the algorithms based on the stated goals. These measurements were implemented and used to evaluate and compare the algorithms as well as to determine when to stop iterating the algorithms. The results in this dissertation open several areas for future work.

7.1 General Conclusions

Several general conclusions can be drawn from this work. As evident from our results and other results described in Chapter 2, 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.

It remains a difficult problem to use the quantitative measurements to determine stopping conditions because of the large number of factors involved. The choice of default distribution and difference thresholds is fairly subjective. A possible enhancement is to allow more user input during the layout process. In early sessions with the layout system, the user spends time adjusting the threshold values and choosing which measurements are used. These values and preferred measurements are stored in the user's personal preference file which can be used later to provide stopping conditions that are tailored to fit the needs of the user. Another possible improvement is to use animation. The user watches the layout change at each iteration and stops the algorithm manually when the resulting layout is satisfactory. Animation can also be used as a tool for cluster busting in anchored graph drawing and in other forms of layout adjustment. If a user watches the transformation from the initial layout to the resulting layout, the user may be able to tolerate a larger difference between the two layouts than if they see the jump from the initial layout to the resulting layout in one step [63].

7.2 Areas for Future Research

There are several areas for future research. In any system that uses quantitative measurements to evaluate layouts, mathematical definitions of the aesthetic criteria

must be determined. In some cases as with the number of edges that cross, this task is straightforward. In the case of keeping the drawing similar to the original drawing, this task is much more difficult. The job of the mathematician or graph drawing algorithm designer is to translate the aesthetic criteria to mathematical measurements. It is the job of the psychologist and sociologist as well as the computer human factors expert to determine what measures constitute similarity. We support the conclusion of Messinger *et. al.* in [58] 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.

In this dissertation, the number of changes in the Delaunay triangulation of the node set is used as a measure of similarity between two layouts. The number of edge changes in several other proximity graphs such as the Sphere of Influence Graph [87] and the α-shape of a set of points [27] could also be used as similarity measures. It would be interesting to determine how the layout algorithms perform according to these measures of similarity and how the results correlate with those for the Delaunay edge changes.

Through our experience with people when demonstrating the layout algorithms discussed in this dissertation, it became apparent that different aspects of the layout are more important to some people than to others. There needs to be a tighter coupling between the graph drawing algorithms and the graph editor/displayer that is used. It would be useful to have a mechanism for users to select specific nodes that should remain fixed in the drawing. Alternatively, it would be beneficial to allow the user to select only one part of the layout to be adjusted. Most of the drawing may be readable and only a portion that has been updated needs to be redrawn. Both

of these features could easily be incorporated into the layout algorithms presented. The selected nodes that are fixed could be input as marked to the layout algorithm. During execution, their Voronoi regions would be computed and the other nodes would not be allowed to move inside these regions. In the GeoForce algorithm these nodes could exert a force on the other nodes but the other nodes would not exert a force on them.

To adjust only a portion of the layout, the user can use a mouse or similar pointing device to draw a box B around a subset $D_U \subseteq D$ of the drawing such that $U \subseteq S$. Since the algorithms take a drawing inside a window as input, they can be executed on the drawing D_U inside B. The resulting drawing is the same as the original drawing with D_U replaced by D'_U. Ultimately, it would be interesting to automatically decide which regions in the layout need to be adjusted by computing clusters in the layout and measuring the distribution of the clusters. This problem is difficult because of the difficulty in computing clusters in a set of points [6, 30, 55, 56]. If the number of clusters $\mathcal{K}$ is known, an algorithm in [1] finds $\mathcal{K}$ clusters such that the minimum distance between any pair of clusters is maximized.

As noted in Section 6.3, sometimes one iteration of the VDCB algorithm moves the nodes farther than they need to be in order for the layout to be readable. As in the GeoForce algorithm a step-size could be used in the VDCB algorithm. At each iteration, the nodes move some fraction of the distance to the centroid of their Voronoi regions. More iterations are required to reach the same level of distribution of the nodes. It would be interesting to run experiments to compare the VDCB algorithm with different step-sizes and the results in Chapter 6.

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 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.

Further work could also be done to try to improve the efficiency of the GeoForce algorithm. In [35], a *grid-variant* of the typical force-directed placement algorithm is discussed. In this algorithm, the drawing window is divided into a grid. For each node, repelling forces are computed between it and only those nodes in grids neighbouring its own grid. It was found that this variant produced layouts that were nearly equivalent to layouts produced by applying the repelling forces to all nodes but on average was faster. This method or a similar method could be used to improve the speed of the GeoForce algorithm in practise.

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 [19]. 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.

In [5], a layout adjustment algorithm is presented that takes the edges of the graph

into consideration. It is shown how layout constraints can be integrated into the hierarchical layout presented in [77]. After using the barycentric method to determine the ordering of the nodes on each level such that there are few edge crossings, constraints are defined that correspond to the relative ordering of the nodes on each level. When the layout is updated the constraints are applied to the nodes in regions that did not change; however, the constraints are weakened in the regions that changed. In this way, the algorithm takes the edges of the graph into consideration during layout adjustment.

A model of the mental map that takes the positions of the edges in a layout into consideration is presented in [21]. Figure 7.1(a) taken from [21] 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 7.1(b) shows the dual graph of the layout in Figure 7.1(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. There are currently no known layout adjustment algorithms that preserve the topology of a drawing [18]. 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

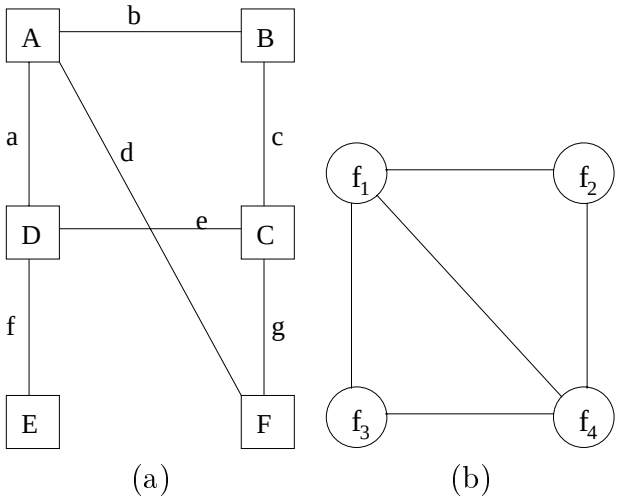

Figure 7.1: A layout and the dual graph representing its topology.

algorithms according to the topology model of a user's mental map.

In our layout algorithms the nodes are represented as points rather than closed regions such as a rectangle. To display a graph, the points are used as reference points such as the bottom left corner of a rectangle or the centre of a circle. The difficulty with this interpretation is that even if the reference point is inside W there is no guarantee that the box or circle is entirely contained in W. For example, if the reference point is the bottom left corner of a box then the entire box will be to the right and above the left and bottom sides of W but part of the box may be outside the right or top sides of W. One way to deal with this in our layout algorithms is add the maximum height of a node box to the top side of W and the maximum width of a node box to the right side of W. Another more complicated solution is to determine cluster busting in anchored graph drawing algorithms that take the size of the nodes into consideration such as the ForceScan algorithm presented in [19].

7.3 Open Problems

Several open problems have been identified by this research and the problem of layout adjustment.

- In Chapter 1, we show that preserving the orthogonal ordering of the nodes may not allow enough movement to distribute the nodes so that the interactions between them are readable and the nodes stay inside W. It is an open problem to show whether preserving the mental map according to the other similarity measures such as proximity relations allows enough movement to distribute the nodes so that the interactions between them are readable. There has been related work in the area of computing the maximum distance that any point can move such that the combinatorial structure of the EMST or Delaunay triangulation of the points is preserved. Given the EMST of a set S of n points, the *sensitivity* measure of S is the maximum distance by which any point can move without altering the combinatorial structure of the $EMST(S)$ [62]. An algorithm in [62] is presented for computing the sensitivity measure of a set. Hurtado and Ramos-Alonso define the sensitivity measure of a set S as the *tolerance* of $EMST(S)$ and have been able to compute the tolerance of a number of other structures including the Delaunay triangulation [45].

- Prove that the difference between a layout produced by the VDCB algorithm and the original layout can never achieve the theoretical upper bound on the number of changes according to the λM and OO measures of difference.

- Characterize sets of points S and P for which $\mathcal{D}_{DE}(S, P) = 1.0$.

- Find the maximum amount that a layout can change after one iteration of the VDCB algorithm according to each of the similarity measures.
- Prove that the VDCB algorithm converges for general input.
- Identify configurations of unbiased points; that is, sets of points that are at the centroids of their Voronoi regions. Determine which of these configurations are stable.
- In [17], it is shown that any inner triangulation of a polygon is realizable as a Delaunay triangulation. In [47], trees that can be realized as SIGs are characterized. Preserving the mental map can be seen as a type of realization problem. We can ask questions such as: Given a triangulation T of a set of node positions S, can T be realized as a triangulation T' such that the minimum edge length in T' is maximized?

Bibliography

[1] T. Asano, B. Bhattacharya, M. Keil, and F. Yao. Clustering algorithms based on minimum and maximum spanning trees. In *Proceedings of the Fourth Annual ACM Symposium on Computational Geometry*, pages 252–257, 1988.

[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] C. Batini, E. Nardelli, and R. Tamassia. A layout algorithm for data flow diagrams. *IEEE Transactions on Software Engineering*, SE-12(4):538–546, April 1986.

[4] S. N. Bhatt and S. S. Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Information Processing Letters*, 25:263–267, 1987.

[5] 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.

[6] E. Boros and P. L. Hammer. On clustering problems with connected optima in euclidean spaces. *Discrete Mathematics*, 75:81–89, 1989.

[7] F. J. Brandenburg. On the complexity of optimal drawings of graphs. In M. Nagl, editor, *Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science No. 441*, pages 166–180. Springer-Verlag, 1989.

[8] M.-J. Carpano. Automatic display of hierarchized graphs for computer-aided decision analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-10(11):705–715, November 1980.

[9] 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.

[10] P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. Technical report, Dipartmento di Informatica e Sistemistica, Università Degli Studi di Roma, La Sapienza, November 1991.

[11] 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.

[12] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. submitted for publication, July 1993.

[13] C. de Groot, R. Peikert, and D. Würtz. The optimal packing of ten equal circles in a square. Technical Report 90-12, IPS Research Report, IPS (Interdisciplinary Project Center for Supercomputing), ETH-Zentrum, CH-8092 Zurich, August 1990.

[14] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: An annotated bibliography. Draft, 1993.

[15] G. Di Battista, R. Tamassia, and I. G. Tollis. Area requirement and symmetry display in drawing graphs (extended abstract). In *Proceedings of the Fifth Annual ACM Symposium on Computational Geometry*, pages 51–60, 1989.

[16] G. Di Battista, R. Tamassia, and I. G. Tollis. Area requirement and symmetry display of planar upward drawings. *Discrete and Computational Geometry*, 7:381–401, 1992.

[17] M. B. Dillencourt. Realizability of delaunay triangulations. *Information Processing Letters*, 33:283–287, 1989/1990.

[18] P. Eades. Personal Correspondence, 1994.

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

[20] 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., Numazu

office: 140 Miyamoto, Numazu-shi, Shizuokay 410-03, Japan, Tokyo office: 17-25, Shinkamata 1-chome, Ohta-ku, Tokyo 144, Japan, August 1991.

[21] P. Eades, W. Lai, K. Misue, and K. Sugiyama. Layout adjustment and the mental map. Manuscript, March 1994.

[22] P. Eades, T. Lin, and X. Lin. Two tree drawing conventions. To appear in: International Journal of Computational Geometry and Applications, 1994.

[23] P. Eades, B. McKay, and N. Wormald. On an edge crossing problem. In *Proceedings of the 9th Australian Computer Science Conference*, pages 327–334. Australian National University, 1986.

[24] P. Eades and K. Sugiyama. How to draw a directed graph. *Journal of Information Processing*, 13(4), 1990.

[25] P. Eades and L. Xuemin. How to draw a directed graph. In *IEEE Workshop on Visual Languages*, pages 13–17, 1989.

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

[27] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, IT-29(4), July 1983.

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

[29] 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.

[30] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the Twentieth Annual ACM Symposium on the Theory of Computing*, pages 434–444, May 1988.

[31] 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.

[32] 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.

[33] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics Principles and Practice, Second Edition.* Addison-Wesley Publishing Company, 1990.

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

[35] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana-Champaign, Illinois, June 1990.

[36] E. R. Gansner, E. Koutsofios, S. C. North, and K. P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, March 1993.

[37] E. R. Gansner, S. C. North, and K. P. Vo. DAG – a program that draws directed graphs. *Software Practice and Experience*, 18(11):1047–1062, November 1988.

[38] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[39] M. Goldberg. The packing of equal circles in a square. *Mathematics Magazine*, 43:24–31, 1970.

[40] E. Goldmeier. *Similarity in Visually Perceived Forms*. International Universities Press, Inc., 1972. Psychological Issues, Vol. VIII, No. 1, Monograph 29.

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

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

[43] M. Himsolt. GraphEd: An interactive graph editor. In *Symposium on Theoretical Aspects of Computer Science (STACS) 1989, Lecture Notes in Computer Science, No. 349*, pages 532–533, 1989.

[44] Kenneth Hoffman and Ray Kunze. *Linear Algebra*. Prentice Hall, 1971.

[45] F. Hurtado and P. Ramos-Alonso. Personal Correspondence, 1993.

[46] M. Iri, K. Murota, and T. Ohya. A fast Voronoi-diagram algorithm with applications to geographical optimization. In *Proceedings of the Eleventh IFIP*

Conference on System Modelling and Optimization, Lecture Notes in Control and Information Sciences No. 59, pages 273–288, July 1983.

[47] M. S. Jacobson, M. J. Lipman, and F. R. McMorris. Trees that are sphere-of-infleunce graphs. Manuscript.

[48] D. S. Johnson. The NP-completeness column: An ongoing guide. *The Journal of Algorithms*, 3:89–99, 1982.

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

[50] C. Kosak, J. Marks, and S. Shieber. Automating the layout of network diagrams with specified visual organization. Technical Report TR-22-91, Harvard University Center for Research in Computing Technology, Aiken Computation Laboratory, 33 Oxford St., Cambridge, Mass., 02138, April 1993. To appear in IEEE Transactions on Systems, Man, and Cybernetics.

[51] R. J. Lipton, S. C. North, and J. S. Sandberg. A method for drawing graphs. In *Proceedings of the First Annual ACM Symposium on Computational Geometry*, pages 153–160, 1985.

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

[53] K. A. Lyons, H. Meijer, and D. Rappaport. Properties of the Voronoi diagram cluster buster. In *Proceedings of the 1993 CAS Conference Volume II*, pages 1148–1163, October 1993.

[54] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[55] D. W. Matula and R. S. Sokal. Properties of gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical Analysis*, 12(3):205–222, July 1980.

[56] N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, February 1984.

[57] H. Meijer. Personal Correspondence, 1994.

[58] 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.

[59] K. Miriyala and R. Tamassia. An incremental approach to aesthetic graph layout in CASE tools. Manuscript, 1993.

[60] 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.

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

[62] C. Monma and S. Suri. Transitions in geometric minimum spanning trees. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, pages 239–249, June 1991.

[63] F. Newbery Paulisch. Personal Correspondence, 1993.

[64] F. Newbery Paulisch and W. F. Tichy. EDGE: An extendible graph editor. *Software Practice and Experience*, 20(S1):63–88, June 1990.

[65] R. Pollack. Personal Correspondence, 1993.

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

[67] L. B. Protskoj, P. G. Sorenson, J. P. Tremblay, and D. A. Schaefer. Towards the automatic generation of software diagrams. *IEEE Transactions on Software Engineering*, 17(1):10–21, January 1991.

[68] 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.

[69] D. Rappaport and W. Powley. GLAD: Graph Layout Algorithm Demonstrator. Developed at Queen's University and IBM Software Solutions Toronto Laboratory as part of the CORDS project, 1994.

[70] E. Reingold and J. Tilford. Tidier drawing of trees. *IEEE Transactions on Software Engineering*, SE-7(2):223–228, 1981.

[71] L. A. Rowe, M. Davis, E. Messinger, C. Meyer, C. Spirakis, and A. Tuan. A browser for directed graphs. *Software Practice and Experience*, 17(1):61–76, January 1987.

[72] J. Schaer. The densest packing of 9 circles in a square. *Canadian Mathematical Bulletin*, 8(3):273–277, April 1965.

[73] J. Schaer and A. Meir. On a geometric extremum problem. *Canadian Mathematical Bulletin*, 8(1):21–27, February 1965.

[74] E. Seneta. *Non-negative Matrices and Markov Chains*. Springer-Verlag, 1981.

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

[76] J. A. Storer. On minimal node-cost planar embeddings. *Networks*, 14:181–212, 1984.

[77] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(2):109–125, 1981.

[78] K. J. Supowit and E. M. Reingold. The complexity of drawing trees nicely. *Acta Informatica*, 18:377–392, 1983.

[79] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):61–79, January/February 1988.

[80] R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Transactions on Circuits and Systems*, 36(9):1231–1235, September 1989.

[81] G. Fejes Tóth. New results in the theory of packing and covering. In P. Gruber and J. M. Wills, editors, *Convexity and its Applications*, pages 318–359. Birkhäuser, 1983.

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

[83] G. T. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980.

[84] 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.

[85] 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.

[86] G. T. Toussaint, editor. *Computational Morphology*. North-Holland, 1988.

[87] G. T. Toussaint. A graph-theoretical primal-sketch. In G. T. Toussaint, editor, *Computational Morphology*, pages 229–260. Elsevier Science Publishers B. V. (North-Holland), 1988.

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

[89] 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.

[90] J. Q. Walker II. A node-positioning algorithm for general trees. *Software Practice and Experience*, 20(7):685–705, July 1990.

[91] C. Wetherell and A. Shannon. Tidy drawing of trees. *IEEE Transactions on Software Engineering*, SE-5(5):514–520, 1979.

[92] J. Wilker. Personal Correspondence, November 1993.

[93] Stephen Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley Publishing Company, Inc., 1991.

[94] 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.

Glossary

AD	The All Distances model of similarity
$bd(W)$	The boundary of W
$bl = (x_{bl}, y_{bl})$	The bottom left corner of W
$\mathcal{C}$	A set of circles
c_i	A cluster centre
cd_i	The centroid of the region $Vor(p_i)$
cm_i	The centre of mass of the Voronoi vertices of $Vor(p_i)$
$C(W)$	The set of four corners of W
CB1	Cluster busting criteria 1: $W = W'$
CB2	Cluster busting criteria 2: The nodes in D' should be more evenly distributed inside W than the nodes in D
CB3	Cluster busting criteria 3: The general shape of D' should be the same as D
CP	The closest pair model of distribution
CH	A convex hull

$CH(S)$	The convex hull of a set S of points
CVD	A Clipped Voronoi diagram
$CVD(S)$	The Clipped Voronoi diagram of a set S of points
$CVE(S)$	The set of edges of $CVD(S)$
$CVV(S)$	The set of vertices in $CVD(S)$
$D = (G, S, W)$	An original drawing of a graph
$D' = (G, S', W')$	A new drawing of a graph
d_I	The ideal distance
DE	The Delaunay edge changes model of similarity
$dist(p, q)$	The Euclidean distance between points p and q
DM	The distances moved model of similarity
DT	A Delaunay triangulation
$DT(S)$	The Delaunay triangulation of a set S of points
$\mathcal{D}_M(S, P)$	A measure of the similarity between S and P according to the model M
$\overline{\mathcal{D}}_M(S, P)$	A measure of the similarity between S and P averaged over 1000 layouts S according to the model M
$\delta_M(S, P)$	The number of changes in P from S under the model M
$\mathbf{\Delta}$	A sequence of horizontal distances
$\mathcal{E}$	A model of distribution
E	A set of edges in a graph
EMST	A Euclidean minimum spanning tree

$EMST(S)$	A Euclidean minimum spanning tree of a set S of points
$f_A(d)$	The attractive force as a function of the distance d
$f_R(d)$	The repelling force as a function of the distance d
$f_S(d)$	The repelling force between a point and a side of W as a function of d
FM	The force minimization model of distribution
$\mathcal{F}$	The tolerance factor of difference
G	A graph
Γ	The set of models of distribution: $\Gamma = \{CP, FM\}$
$H(p_i, p_j)$	The half-plane containing p_i and bounded by the perpendicular bisector of p_i and p_j
$I_{\max}$	The default maximum number of iterations
$\mathcal{K}$	The number of clusters in a set of points
k	The size of each cluster
$\Lambda(p_i, p_j)$	The set of points lying to the left of the directed line $\vec{l}(p_i p_j)$
$\lambda(p_i, p_j)$	The number of points in $\Lambda(p_i, p_j)$
λM	The λ-matrix model of similarity
M	A model of similarity

n	The number of nodes in a graph
OO	The orthogonal ordering model of similarity
P	A set of node positions
$p_i = (x_i, y_i)$	A node position which is a point in the plane
$\overline{p_i p_j}$	The line segment with endpoints p_i and p_j
$\overrightarrow{p_i p_j}$	The ray directed from p_i to p_j
$P_{PG}(p_i, p_j)$	1 if there is an edge between p_i and p_j in PG and 0 otherwise
$pb(p_i, p_j)$	The perpendicular bisector of p_i and p_j
PG	A proximity graph
$PG(S)$	A proximity graph defined on a set S of points
$R_x(p_i)$	The rank of p_i in the x direction
$R_y(p_i)$	The rank of p_i in the y direction
RAM	Random-Access Machine
s_i	A square region
$\mathcal{S}$	A region in the plane
S	A set of node positions
S_{PE}	The maximum number of edges in a proximity graph PE of a set of n points

t	The number of iterations
$\mathcal{T}_{\mathcal{E}}$	A threshold value of distribution measure according to the model $\mathcal{E}$
T_{PE}	Time to compute the proximity graph PE of a set of n points
$tr = (x_{tr}, y_{tr})$	The top right corner of W
$UB_M(n)$	The upper bound on the $\delta_M(S, P)$ for S and P with n points
Υ	The set of models of similarity: $\Upsilon = \{AD, \lambda M, DE, DM, OO\}$
V	A set of nodes in a graph
VD	A Voronoi diagram
$VD(S)$	The Voronoi diagram of a set S of points
VDCB	The Voronoi diagram cluster buster algorithm
$VE(S)$	The set of edges of $VD(S)$
$Vor(p_i)$	The Voronoi region of p_i
$VV(S)$	The set of vertices in $VD(S)$
W	The drawing window
$\Xi_{\mathcal{E}}(S)$	A measure of the distribution of S according to the model $\mathcal{E}$
$\overline{\Xi}_{\mathcal{E}}(S)$	A measure of the distribution of S averaged over 1000 layouts S according to the model $\mathcal{E}$

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Vita

Name:	**Kelly A. Lyons**
Place and year of birth:	Marathon, Ontario, 1963
Education:	Queen's University, 1981–1985 B.Sc. (Honours, Computing and Information Science) 1985 Queen's University, 1987–1989 M.Sc. (Computing and Information Science) 1990 Queen's University, 1989–1994
Experience:	Development Analyst, IBM Canada Ltd. Laboratory, Toronto, Ontario, 1985–1987 Teaching Assistant, Department of Computing and Information Science, Queen's University, 1987-1992 Instructor, Department of Computing and Information Science, Queen's University, 1991
Awards:	Queen's Honours Matriculation Scholarship 1981-1983 James H. Rattray Scholarship 1984-1985 NSERC Postgraduate Scholarship I & II 1987-1989 NSERC Postgraduate Scholarship III & IV 1989-1991 Best Student Paper at CASCON'92, IBM Centre for Advanced Studies Conference IBM Fellowship 1992 and 1993

Publications:

Kelly A. Lyons,
An Efficient Algorithm for Identifying Objects Using Robot Probes, MSc Thesis, Department of Computing and Information Science, Queen's University, Kingston, ON, 1989

Hazel Everett, Kelly A. Lyons, Bruce Reed and Diane Souvaine,
Illuminating Squares on a Transversal, Proceedings of the Third Annual Canadian Conference on Computational Geometry, Vancouver, British Columbia, August 5-10, 1991, 118-121.

Patrick J. Finnigan and Kelly A. Lyons,
Narratives of Space and Time: Visualization for Distributed Applications, Proceedings of the 1991 CAS Conference, IBM Canada Ltd. Laboratory Centre for Advanced Studies, Toronto, Ontario, Canada, October, 1991, 363-391.

Selim G. Akl and Kelly A. Lyons,
Parallel Computational Geometry, Prentice Hall, 1992.

Kelly A. Lyons,
Cluster Busting in Anchored Graph Drawing,
Proceedings of the 1992 CAS Conference Volume I,
IBM Canada Ltd. Laboratory Centre for Advanced Studies,
Toronto, Ontario, Canada, November, 1992, 7-17.

Gopi K. Attaluri, Dexter Bradshaw, Patrick J. Finnigan,
Nigel Hinds, Michael Kalantar, Kelly A. Lyons,
Andrew D. Marshall, Jan Pachl, and Hong Tran,
Operation Jump Start: A CORDS Integration Prototype using DCE, Proceedings of the 1993 CAS Conference Volume II,
IBM Canada Ltd. Laboratory Centre for Advanced Studies,
Toronto, Ontario, Canada, October, 1993, 621-636.

Kelly A. Lyons, Henk Meijer, and David Rappaport,
Properties of the Voronoi Diagram Cluster Buster,
Proceedings of the 1993 CAS Conference Volume II,
IBM Canada Ltd. Laboratory Centre for Advanced Studies,
Toronto, Ontario, Canada, October, 1993, 1148-1163.

Kelly A. Lyons and David Rappaport,
An Efficient Algorithm for Identifying Objects Using Robot Probes, Accepted for publication in
The Visual Computer, 1994.