An Algorithmic Framework for Visualising and Exploring Multidimensional Data

By

Greg Ross

Submitted In Partial Fulfilment of the
Requirements for the Degree of
Doctor of Philosophy

University of Glasgow
Glasgow, Scotland
June 2006

Abstract

To help understand multidimensional data, information visualisation techniques are often applied to take advantage of human visual perception in exposing latent structure. A popular means of presenting such data is via two-dimensional scatterplots where the inter-point proximities reflect some notion of *similarity* between the entities represented. This can result in potentially interesting structure becoming almost immediately apparent.

Traditional algorithms for carrying out this dimension reduction tend to have different strengths and weaknesses in terms of run times and layout quality. However, it has been found that the combination of algorithms can produce hybrid variants that exhibit significantly lower run times while maintaining accurate depictions of high-dimensional structure.

The author's initial contribution in the creation of such algorithms led to the design and implementation of a software system (HIVE) for the development and investigation of new hybrid variants and the subsequent analysis of the data they transform. This development was motivated by the fact that there are potentially many hybrid algorithmic combinations to explore and therefore an environment that is conducive to their development, analysis and use is beneficial not only in exploring the data they transform but also in exploring the growing number of visualisation tools that these algorithms beget.

This thesis describes three areas of the author's contribution to the field of information visualisation. Firstly, work on hybrid algorithms for dimension reduction is presented and their analysis shows their effectiveness. Secondly, the development of a framework for the creation of tailored hybrid algorithms is illustrated. Thirdly, a system embodying the framework, providing an environment conducive to the development, evaluation and use of the algorithms is described. Case studies are provided to demonstrate how the author and others have used and found value in the system across areas as diverse as environmental science, social science and investigative psychology, where multidimensional data are in abundance.

Table of Contents

1. Introduction 1
1.1 Introduction and background 1
1.2 Motivation 2
1.3 Research aims 4
1.3.1 Thesis statement 4
1.3.2 Key research questions 4
1.3.3 Approach 5
1.4 Thesis structure 5
2. Information Visualisation 7
2.1 Scientific visualisation 7
2.2 Information visualisation 9
2.3 Abstraction 10
2.3.1 Gestalt principles 11
2.3.2 Visual structures 14
2.3.3 Data types 14
2.4 Dimensionality 15
2.4.1 1-dimensional visualisation 16
2.4.2 2-dimensional visualisation 17
2.4.3 3-dimensional visualisation 18
2.4.4 4+ -dimensional visualisation 20
2.5 Interactivity 22
2.5.1 Affordance and appropriation 23
2.5.2 Time 23
2.5.3 Interaction mechanisms 24
2.6 Conclusions 28
3. Clustering Algorithms 29
3.1 Hierarchical 30
3.1.1 Agglomerative single-link clustering 31
3.1.2 Scatter/Gather: An application of hierarchical clustering 32
3.2 Partitional 33
3.2.1 K-means 33
3.2.2 Bisecting K-means 34
3.2.3 NNS with K-means: An application of a partitioning clustering algorithm 35
3.3 Density-based 37
3.4 Graph-theoretic 40

3.4.1 The minimal spanning tree as a basis for clustering 41
3.4.2 Other graph-theoretic clustering algorithms 42
3.5 Grid-based 44
3.6 Model-based 46
3.7 Conclusions 47
4. Dimension reduction 49
4.1 Projection techniques 50
4.1.1 Principal Component Analysis (PCA) 51
4.1.2 Singular Value Decomposition (SVD) 54
4.1.3 Projection Pursuit 55
4.1.4 Random Projection (RP) 56
4.1.5 FastMap 58
4.2 Kohonen's Self-Organising Feature Map 61
4.2.1 Batch-mode SOM 62
4.3 Multidimensional Scaling 64
4.3.1 Torgerson's classical metric MDS 66
4.3.2 Non-metric MDS 69
4.3.3 MDS for feature selection 73
4.4 Force-directed placement 76
4.5 Conclusions 79
5. Hybrid clustering and layout algorithms 82
5.1 Hybrid algorithms for clustering 83
5.2 Hybrid algorithms for dimension reduction 85
5.3 A novel hybrid algorithm for dimension reduction 86
5.3.1 Distance metric 89
5.3.2 Brodbeck and Girardin's Interpolation algorithm 90
5.3.3 An improved interpolation technique 93
5.3.4 Evaluation of the full layout algorithm 96
5.3.5 A hybrid variant based upon K-means 97
5.4 Fast non-metric multidimensional scaling 99
5.4.1 Evaluation of fast NMDS 103
5.5 Voronoi-based clustering algorithm 106
5.5.1 Preattentive cluster identification 106
5.5.2 A novel clustering algorithm 107
5.6 Conclusions 119
6. Visualisation environments 120
6.1 Data-flow model 121
6.1.1 Visual programming 122

6.1.2 Data-flow architecture 123
6.1.3 Some examples 126
6.1.4 Relevance of data-flow based scientific visualisation to the HIVE framework. 130
6.2 Information visualisation environments 132
6.2.1 Multiple views for information visualisation 133
6.2.2 Information workspaces 138
6.2.3 Data, information and knowledge 140
6.2.4 Relevance of the information workspace concept to the HIVE framework 142
6.3 Visualisation system design theory 144
6.3.1 Ecological interface design (EID) 144
6.3.2 Cognitive dimensions of notations 146
6.3.3 Dimensions of expression 149
6.3.4 Abstraction 150
6.3.5 Relevance to the HIVE framework 152
6.4 Conclusions 153
7. The hybrid information visualisation environment (HIVE) 156
7.1 Multiple-view co-ordination 157
7.2 Combinatorial hybrid approach 158
7.2.1 3-stage hybrid approach 160
7.3 Adaptability to different variable types and heterogeneous data 163
7.4 Implementation of HIVE 163
7.4.1 System architecture 164
7.4.2 Graph manager 165
7.4.3 Visual modules 165
7.4.4 Ports 166
7.4.5 Linking and the composition model 166
7.4.6 Hybrid algorithm generation 168
7.5 Examples 169
7.5.1 Comparison of spring model layouts 170
7.5.2 Exploration of a real data set 171
7.5.3 Using MDS for feature selection 174
7.6 Design review 176
7.6.1 Data-flow model 176
7.6.2 Information visualisation environments 177
7.6.3 Visualisation system design theory 178
7.6.4 HIVE features 179
7.7 Conclusions 179
8. Algorithmic profiling 181
8.1 Multiple runs module 181

8.2 Stress and clock modules 182
8.3 Shepard plot 183
8.4 Coordination of profiling modules in HIVE 184
8.5 Case studies 186
8.5.1 Batch job of executions for algorithm evaluation 186
8.5.2 Exploratory analysis of synthetic data 188
8.5.3 Exploratory analysis of real data 189
8.6 Conclusions 193
9. Text-mining in HIVE 195
9.1 Vector representation of documents 195
9.2 Where is particular literature within a HIVE layout? 200
9.3 Conclusions 204
10.HIVE user engagement 206
10.1 Questionnaire 207
10.2 Examples of HIVE's use 210
10.2.1 Development and evaluation of a new FDP algorithm 210
10.2.2 Steerable dimension reduction 212
10.2.3 A bioinformatics chain description tool 213
10.2.4 HIVE for psychological profiling 214
10.3 Conclusions 218
11.Conclusions 220
11.1 Summary 220
11.2 Ongoing work with CIP 222
11.3 Aggregation of flow networks 223
11.4 Automatic routing of sub-layouts 223
11.5 Usability studies 225
11.6 Contribution and thesis statement 225
11.6.1 Novel algorithms 226
11.7 Research questions 227
11.8 Reflection and design implications 228
Appendix A: List of HIVE modules 232
Appendix B: Algorithmic 'cookbook' 234
Appendix C: HIVE user questionnaire 235
Bibliography 238

List of Figures

Figure 2.1 A screenshot from the IBM DX scientific visualisation system [AT95a] depicting an unsteady flow simulation over a space shuttle launch vehicle. 8

Figure 2.2 Initial survey results obtained by HMS Scott showing images of the coastline of Sumatra where the earthquake that resulted in the Indian Ocean tsunamis occurred. It is hoped that these visualisations will help scientists understand the cause of such natural phenomena and help predict them in the future..... 8

Figure 2.3 A Honeycomb© [Hon04] view of results returned by the Google internet search engine. The visualisation is based upon Johnson and Shneiderman's *treemap* [JS91]: a technique designed to utilize space efficiently in the display of hierarchical information structures..... 9

Figure 2.4 The *link map* visualisation in $^{\text{nickleby}}$KIT®. Each node represents an issue raised in respect to the author's research. Nodes that are deemed as being closely related are linked. The layout was produced by a force-directed placement algorithm for graph-drawing (see Section 4.4). 10

Figure 2.5 A scatterplot has the potential to make groups of points appear as individual perceptual units (clusters). For example, the author would assume that in making reference to 'A' in the figure, the reader's attention would be drawn to the upper cluster as a *whole* and not the single point nearest to the label. 12

Figure 2.6 An example of the Gestalt principle of good continuation. Both of the above images represent sunshine intensity over an extended period of time, however, the spiral visualisation [WAM01] more clearly shows the day/night periods. 13

Figure 2.7 The familiarity rule. 13

Figure 2.8 An example of reduced representation. The figure depicts two source code modules, each of which is on a folding axis. 17

Figure 2.9 The 'perspective wall' distorts a 2-d layout so that the focus at the centre of the screen is most legible while the remainder of the layout is peripheral. The user can scroll potentially interesting parts of the layout to the fore and still be afforded the context of neighbouring regions..... 18

Figure 2.10 Cat-a-Cone [HK97] arranges each level of a hierarchical categorisation scheme in a 3-d view to utilise space efficiently. This technique, like the 'perspective wall', uses perspective distortion to clarify the focus (the node closest to the viewer) while maintaining the context of the adjacent nodes..... 19

Figure 2.11 A screenshot from Spire [Wis99] – a tool based upon Wise's themescape [WTP*95]. A document corpus is represented via a landscape metaphor in which the themes that run through the collection are mapped to visual attributes. 20
Figure 2.12 An InfoCrystal [Spo93] representing three search criteria or inputs, A, B and C and all possible Boolean queries in normal conjunctive form. The interior icons can be embellished to show the results of submitting the respective queries to a document collection. In this example, these inputs define a 3-d search space, however, Spoerri has demonstrated the application of InfoCrystals to more than three inputs. 21
Figure 2.13 An example of the output of a SOM, depicting the *concept areas* relating to electronics. 22
Figure 2.14 North et al.'s Visible Human Explorer interface. The overview of the human body is tightly coupled with axial detail view. The user can sweep a horizontal line across the overview to dynamically update the detailed cross-section view. 25
Figure 2.15 Double-ended sliders to the left and bottom of the plot in filmfinder allow the user to zoom and pan along the two axes, essentially filtering the view. 27
Figure 2.16 A double-ended slider with a histogram, showing a range selection. 27
Figure 3.1 A dendrogram, which when cut at different levels, will produce different clusters. 30
Figure 3.2 The single-link clustering algorithm. 31
Figure 3.3 The K-means algorithm. 34
Figure 3.4 A screen shot of the K-means NNS experimental program. Green points represent randomly distributed data, the blue points represent K-means centroids, and the red points indicate 'queries' while the black points are the approximate nearest neighbours to the (red) query points. 36
Figure 3.5 A case where the K-means NNS is only approximate. 36
Figure 3.6 Clusters of different densities. When the clustering is via a global density parameter, only clusters [A, D and E] or [A, B and C] will be found. 38
Figure 3.7 A Voronoi tessellation of a 2-d point pattern consisting of two clusters. Notice how the polygons of points inside the clusters have smaller areas than those towards the outside. Duyckaerts and Godefroy use this property to automatically find clusters. In the above example an area threshold has been set and polygons within it are shaded – each cluster is distinguished by shading with a different colour. This image was generated by HIVE [RC03a, RC03b]. 39
Figure 3.8 Points in a 2-d data space (left) and their minimal spanning tree (right). 40
Figure 3.9 The MST above shows that intra-cluster distances are shorter than inter-cluster distances. Deletion of edges *A* and *B* would result in three separate connected sub-graphs representing the clusters. 42
Figure 3.10 Partitioning of a 2-d data space into regular cells. The resulting structure, called a map [HK98] or grid structure [Sch96], can intuitively be generalised to the multi-dimensional case. 45

Figure 4.1 Projection of 3-dimensional points onto a plane. 51
Figure 4.2 PCA projections of a 3-dimensional swiss roll-shaped data set. The image on the left is the projection of the whole set. The image on the right is a PCA projection of the highlighted cross-section of the image on the left. These projections were generated by the author's HIVE software. 53
Figure 4.3 Since the first principal component is the direction of greatest variance in the data, the outliers shown in blue on the left, dominate the regression. This results in the projection onto the principal component (right) where possibly significant structure, such as the two clusters shown, has been lost. 54
Figure 4.4 A PCA projection of a 3-d cube (left). A random projection of the cube distorts mutual similarities (right). Both of the above projections were produced by the HIVE software [RC03a, RC03b]. 57
Figure 4.5 Object O_i is projected using the Cosine Law onto the line passing through the pivot objects O_a and O_b. 58
Figure 4.6 Objects O_i and O_j are projected onto the hyperplane H, perpendicular to the line through O_a and O_b. 60
Figure 4.7 A screen shot from the author's batch-SOM implementation. 63
Figure 4.8 The coloured points bounding the figure, ranging from red to violet show the configuration obtained by Shepard's MDS algorithm when run on the colour-similarity data. It is clear that this closely follows the familiar colour circle (centre). The original figure [She62] has been rotated and flipped in this reproduction for ease of comparison. 65
Figure 4.9 The point configuration on the left was recovered from the original 2-d data after they were transformed into proximities. The Shepard plot on the right shows the shape of the function used in the transformation. These images were produced in HIVE. 71
Figure 4.10 An SSA layout of variables representing the MOs of a set of arson offences. 74
Figure 4.11 An SSA layout of the variables of a scientific data set is used to select a subset (highlighted in yellow) for subsequent dimension reduction (two left-hand frames). The projection of the full data set is shown on the right-hand frame. 76
Figure 4.12 An illustration of Eades' concept of the spring model. The image on the left shows steel rings held in random positions causing the connecting springs to be stretched or compressed. The image on the right depicts the system in a state of minimal energy after the rings have been *let go* resulting in the springs reverting to their rest lengths. For clarity, only springs connecting adjacent rings are shown. 77
Figure 4.13 The same data set as in Figure 4.9 is fed into Chalmers' spring model and the layout is shown on the left. The corresponding Shepard plot is shown on the right. 79
Figure 4.14 A taxonomy of dimension reduction techniques. 80
Figure 5.1 The sub-quadratic ($O(N\sqrt{N})$) algorithm for non-linear dimension reduction. 88
Figure 5.2 Chalmers' spring model is initialised by randomly positioning the $\sqrt{N}$ sample in 2-d. 88

Figure 5.3 The spring model interatively produces an accurate layout of the sample. 88
Figure 5.4 The remaining (N - $\sqrt{N}$) items are interpolated onto the sample layout. The spring model can then be run for a constant number of iterations to refine the layout. In this case that is not necessary. 88
Figure 5.5 Brodbeck and Girardin's interpolation routine [BG98]. 91
Figure 5.6 Brodbeck and Girardin's interpolation often provides a sub-optimal layout, even for 2-d data. 92
Figure 5.7 For a 2-d layout of 2-d data, the recovered distances should exactly match the original distances. This one-to-one relation should manifest itself as a 45 degree slope in the Shepard plot, however, in this case it can be seen that the interpolation routine has resulted in deviation from this slope. 92
Figure 5.8 An outline of the new interpolation algorithm. 93
Figure 5.9 The placement of item i begins with finding its parent point P in the initial layout. A circle of radius r (proportional to the high-dimensional distance between i and the item represented by P) is centred on P. Quadrant and binary search over the circle's circumference finds the position i_0 that minimises the summed discrepancy between the high- and low-dimensional distances to the subset of layout points. This position is then refined by iteratively adding an aggregate force vector, moving the item to its final position i_1. 94
Figure 5.10 The new interpolation routine has produced a much more accurate layout of the 2-d data. 95
Figure 5.11 A comparison of Brodbeck and Girardin's original routine with the new interpolation algorithm. These results were obtained by Morrison [Mor04] using the author's HIVE software. The results for each algorithm were averaged over 10 runs for 2-d data sets ranging from 1000 to 10,000 items. 95
Figure 5.12 Results obtained by Morrison [Mor04] showing a comparison of Chalmers' spring model with the new algorithm. 96
Figure 5.13 The image on the left is a layout of a random data sample. The image on the right is a layout of items that are closest to the K-means centroids of the data. The original data used are 2-d and consist of 7239 items. Both layouts contain 85 ($\sqrt{7239}$) points from the data set. 98
Figure 5.14 A comparison of the K-means based algorithm with the sample+interpolation and Chalmers' algorithms. 98
Figure 5.15 An outline of the fast NMDS algorithm. 100
Figure 5.16 The image on the left shows the layout of a 3-d cube obtained by the first stage of the fast NMDS algorithm. This shows how monotonicity is only locally preserved. The image on the right shows a layout obtained by Chalmers' spring model on the same data. Here the overall structure is better preserved but local regions remain rough. 102

Figure 5.17 Stress is plotted after every 5 iterations for Shepard's NMDS and the fast NMDS algorithms. 103
Figure 5.18 Run times for Shepard's algorithm and the fast NMDS algorithm. 104
Figure 5.19 The image on the left shows a layout of proximity data obtained by the fast NMDS algorithm. The layout on the right was produced by the novel hybrid algorithm described in Section 5.3. 105
Figure 5.20 The Shepard plot on the left shows that the fast NMDS algorithm has recovered the function relating proximities to real Euclidean distances. The plot on the right shows that the hybrid spring model was not able to recover this function with the same degree of accuracy. 106
Figure 5.21 The winged-edge data structure. This maintains a compromise between a compact representation of the Voronoi geometry and fast retrieval of vertex, edge and polygon incidence relations. Edges are used to keep track of the geometry and are represented by arrays of start and end vertices, polygon faces, predecessors and successors. 109
Figure 5.22 The top-left graph is the Voronoi diagram, which can be used for density-based clustering. The top-right graph is the dual of the Voronoi diagram, the Delaunay tessellation, whose sub-graphs include the minimum spanning tree (shown here in red) and therefore is suitable for graph-theoretic clustering applications. 110
Figure 5.23 Data set used for clustering (a). Perimeter threshold reduced gradually (b) - (f). 112
Figure 5.24 Pseudocode for the first stage of the clustering algorithm. 114
Figure 5.25 Pseudocode for the second stage of the clustering algorithm. 115
Figure 5.26 Clustering results of the first version of the clustering algorithm on benchmark data sets. Points of the same colour are deemed to be in the same cluster. 116
Figure 5.27 Clustering results of the final version of the clustering algorithm on benchmark data sets. The modified algorithm correctly identifies most clusters. 118
Figure 6.1 An example of a data-flow architecture. Blocks represent functional modules and the arrows represent the flow of data. 124
Figure 6.2 An outline of the hybrid algorithmic architecture. 131
Figure 6.3 A conceptual view of the "process of knowing". 140
Figure 6.4 Linked meaning triangles. 141
Figure 6.5 The Java model-view-controller architecture. 152
Figure 7.1 Two screen-shots of the HIVE interface. The image on the left illustrates interconnected components that import, transform and render multidimensional data. The algorithmic components collectively represent the $O(N\sqrt{N})$ hybrid algorithm of Section 5.3. Thick lines that link modules represent data-flows while thin ones, connecting scatterplots and other visualisations, represent the connections between interlinked interactive views. The image on the right shows the same scatterplots enlarged and supplemented with a fisheye table component (bottom-right) and

histograms (bottom-left). The data consist of 5000 points sampled from a 3-d 'S' shaped distribution. 157
Figure 7.2 Data input to components in a hybrid algorithmic architecture can be categorised by the ranges of dimensionality and cardinality they are best suited for – high, medium or low. Each component transforms the data, effectively moving across the 3x3 grid. The hybrid spring model in Section 5.3 produces a low-dimensional layout of a large high-dimensional data set i.e. a move from (*H*, *H*) to (*L*, *H*) that involves several steps shown as dotted lines in the figure: sampling, which reduces *N*, then a spring model of the sample, which reduces *D*, and then interpolation, which increases *N*. 159
Figure 7.3 The proposed model of the hybrid approach for scalability and adaptability. 161
Figure 7.4 The system architecture of the HIVE framework. 164
Figure 7.5 When HIVE is in link mode, all Swing components are hidden while port representations are rendered. 166
Figure 7.6 The top part of the image shows a link from a data source to a sample module. The bottom half of the image shows a link after it has been selected and bent by the user. 168
Figure 7.7 A network of the three types of components: data source (2-d geometric data), algorithm (Chalmers' spring model) and visualisation (fisheye table and scatterplot). ... 170
Figure 7.8 An example demonstrating the non-deterministic nature of the spring model. The expanded view of the bottom spring model component shows controls for changing parameters such as freeness, velocity and damping as well as controls for setting convergence criteria. 171
Figure 7.9 The leftmost scatterplot shows the output of neural PCA. The middle scatterplot shows the data after interpolation around the K-means centroids while the right scatterplot illustrates the output of the final spring model component. The highlighted cluster is a small subset of erroneous PAR measurements. These clusters are much clearer in the hybrid algorithm's plots than with PCA. The histogram shows the PAR distribution at a depth of 10 metres. The outlying peak (far-left) has been selected and this highlights the clusters in the scatterplots. 172
Figure 7.10 Dashed arrows represent the HIVE-generated hybrid algorithm spanning the space from (*M*, *M*) to (*L*, *M*) via K-means, Chalmers' spring model and Interpolation (clockwise). The solid arrow represents the manually instantiated PCA module. 174
Figure 7.11 Port connections for using MDS for feature selection and subsequent analysis. 175
Figure 7.12 MDS for feature selection in action. 175
Figure 8.1 As a spring model runs, stress is measured and charted against each iteration. The plateau, after around 26 iterations, shows that the algorithm has fallen into a local minimum while the layout (of the 2-d data) shows that more iterations are required to break out of the minimum configuration. 183
Figure 8.2 The Shepard plot module. 184

Figure 8.3 The top two images show 2-d layouts using linear PCA (left) and spring model (right). Below each layout is the associated Shepard plot. The PCA layout has no pair of objects at a greater distance from each other than in high-dimensional space. This is confirmed by the fact that no objects appear below the diagonal in the PCA's Shepard plot..... 185

Figure 8.4 The yellow modules represent different stages of a hybrid algorithm. The Multiple Runs module coordinates a sequence of executions, loading data and parameters into each component. Charts plot run time against data set size at various stages of the algorithm. The bottom left chart shows run times under three separate sets of parameters for stage three. Having connected the various components and provided instructions to the MR module, the algorithm executions and chart plotting may proceed unsupervised. 187

Figure 8.5 PCA layout and Shepard plot working together interactively to help build user understanding of a data set. The PCA layout (a). A selection is made in the Shepard plot of points corresponding to distances in the layout that may benefit from further analysis (highlighted region) (b). The selection in the Shepard diagram is also highlighted in the PCA layout (c). A re-projection of the selected points and their immediate region confirms their misrepresentation in the original layout (d). 189

Figure 8.6 Various algorithmic and visualisation components working together in a coordinated environment. A PCA layout is made and the associated Shepard diagram is used to detect a local area that might be better represented if considered separately. A Voronoi tessellation component is used to cluster the data, and extract the cluster containing the previously identified local area (the yellow objects in the central component). This cluster is processed with a spring model (FDP) routine, which uncovers two sub-clusters that the author had not previously been able to identify..... 191

Figure 8.7 . A close-up view of the Voronoi module in Figure 8.6. 192

Figure 8.8 The left image shows the spring model layout of a selected Voronoi cluster within a PCA layout. On the right is the original PCA layout. Selecting the C-shaped sub-cluster on the left highlights the corresponding objects in the PCA layout, helping one understand the overlap or separation of sub-clusters in the PCA. 192

Figure 8.9 Shepard diagrams based upon the PCA layout of the full data set (left) and the spring model layout of the selected cluster (right). Red lines are drawn at 45 degrees to help detect the extent to which points deviate from this diagonal..... 193

Figure 9.1 Initial experimental layouts. Cosine measure using normalised term frequency (tf) (a). Euclidean distance using tf (b). Cosine measure using tf-idf (c) and Euclidean distance using tf-idf (d). 196

Figure 9.2 Layouts of the abstracts data set using the same measures as in Figure 9.1 but with the empty documents removed..... 198

Figure 9.3 The points that are highlighted in red represent documents that contain the phrase "multidimensional scaling". No clustering seems to occur and upon reproducing the layouts, the points appear in different positions with respect to each other and the other points. 198
Figure 9.4 Histograms showing the distribution of sample Euclidean distances and cosine measures of dissimilarity. The histograms show that the samples do indeed cluster tightly around a high positive value. 199
Figure 9.5 The histogram shows that the distribution of dissimilarities is now spread out more. The layout accordingly shows potentially interesting structure in the data. 200
Figure 9.6 A screenshot from HIVE of a text-mining application (right) - the top-left visual module is the text collection. The text-viewer component is next to the scatterplot layout and two search modules appear in the top-right of the figure. For clarity, the layout is reproduced in the image to the left of the screenshot. 201
Figure 9.7 The layout from Figure 9.6 is labelled to indicate general themes running through the documents in different regions. Some HIVE papers are highlighted in the data mining and clustering region at the upper-left side of the layout. 203
Figure 9.8 A layout of terms in HIVE. This can be used for feature selection as in Section 7.5.3. 204
Figure 10.1 The first 16 questions coded and laid out by a spring model in HIVE. The responses of the selected users in the scatterplot are highlighted in the table. The stress in this configuration was measured as 0.087 indicating a good fit. 208
Figure 10.2 Morrison's hybrid algorithm in HIVE. The large component at the top-left is the multiple runs module. Morrison used this in conjunction with clock and stress modules to evaluate the algorithms. 211
Figure 10.3 The image on the left shows a layout obtained by Williams and Munzner using the novel layout algorithm of Section 5.3. The image on the right shows an overview of the same data set using MDSteer. Black boxes represent bins that have no unplaced points while red boxes represent bins in which there are points still to be placed. 213
Figure 10.4 A chain of visual modules representing bioinformatics applications in Darroch's Chain Description Tool – a customised version of HIVE. 214
Figure 11.1 A layout of a data set is clustered (bottom-left module). Each cluster feeds into a *cluster-picker* (top-centre module) which can route each cluster to another process for subsequent analysis. In this case spring models are used to layout each of the two largest clusters independently. 224

List of Tables

Table 2.1 Gestalt laws of grouping. 11
Table 2.2 Variable types. 14
Table 2.3 Categories of interaction speeds..... 23
Table 3.1 The main steps involved in cluster analysis. 29
Table 7.1 Ordinal categories of cardinality and dimensionality..... 160
Table 7.2 Linking rules for HIVE's composition model..... 167

Contributing Publications

1. G. Ross, A. J. Morrison and M. Chalmers. Visualisation techniques for users and designers of layout algorithms. *In Proceedings of the 9th International Conference on Information Visualisation (IV05)*, pages 579–586, IEEE Computer Society Press, 2005.

2. G. Ross, A. J. Morrison and M. Chalmers. Coordinating views for data visualisation and algorithmic profiling. *In Proceedings of the IEEE International Conference on Coordinated and Multiple Views in Exploratory Visualization 2004*, pages 3–14, IEEE Computer Society Press, 2004.

3. G. Ross and M. Chalmers. A visual workspace for constructing hybrid MDS algorithms and coordinating multiple views. *Information Visualization*, 2(4):247-257, Palgrave Macmillan, 2003.

4. G. Ross and M. Chalmers. A visual workspace for hybrid multidimensional scaling algorithms. *In Proceedings of the IEEE Symposium on Information Visualization*, pages 91–96, IEEE Computer Society Press, 2003.

5. A. J. Morrison, G. Ross and M. Chalmers. Fast multidimensional scaling through sampling, springs and interpolation. *Information Visualization* 2(1):68–77, Palgrave Macmillan, 2003.

6. A. J. Morrison, G. Ross and M. Chalmers. A hybrid layout algorithm for sub-quadratic multidimensional scaling. *In Proceedings of the IEEE Symposium on Information Visualization*, pages 152–160, IEEE Computer Society Press, 2002.

Acknowledgements

Thank you to Laura, and my family, for patience and support during the course of my research, and Alistair Morrison, my friend and colleague, for good laughs and good advice. I am extremely grateful to Matthew Chalmers for his encouragement and supervision.

I am also grateful to Nickleby HFE Ltd., especially Ron McLeod, for funding my work. I would also like to thank Iadh Ounis for guidance and advice, and my officemates, Agathe Girard and John Williamson, for valuable discussions and entertaining banter.

Declaration of Originality

The material presented in this thesis is the result of my own research carried out at the Department of Computing Science at the University of Glasgow working under the supervision of Dr. Matthew Chalmers and Dr. Iadh Ounis, except where explicitly stated otherwise. All other referenced material has been given full acknowledgement in the text.

1. Introduction

1.1 Introduction and background

To gain actionable knowledge from the ever-increasing sea of data facing analysts, data must be represented in a way that any pertinent information contained is made available as quickly as possible. It is well known that of all the senses, the human visual system has by far the greatest bandwidth for communicating information to the brain and it is for this reason that data are often represented graphically [CMS99].

A major challenge, however, is in graphically depicting abstract data. Abstract data are those observations or measurements that have no direct physical derivation and therefore do not immediately lend themselves to the spatial mappings required for visual rendering. This is compounded by complex data sets consisting of many items, each consisting of many variables. In the endeavour to make sense of these multivariate data by thinking of them in spatial terms, they are referred to as *multidimensional* data.

When data are presented graphically on a spatial substrate, interesting features such as patterns, trends and Gestalt forms might be revealed. A very popular means of achieving this is to plot the data as points against the axes of a two-dimensional scatterplot. If the data dimensionality is relatively low, for example four, then scatterplot points can be rendered as glyphs, whose positions denote two dimensions, and whose visual properties encode the remaining dimensions in retinal variables such as shape, size or colour [CMS99]. However, when the data dimensionality is too high to directly map to position and other visual structures, the data must be transformed in such a way that they are represented by a lower number of derived dimensions that retain as much of the original information as possible. This is known as the challenge of *dimension reduction* or *multidimensional scaling (MDS)*.

Many researchers have developed dimension reduction algorithms, sometimes referred to as *layout* algorithms, each with different benefits and drawbacks. Some algorithms can be very effective at reducing dimensionality whilst preserving the high dimensional relationships, but be too inefficient to scale up to data sets with high cardinality. On the other hand, some algorithms might be fast but be unable to accurately capture the original information, thus interesting patterns can elude the analyst. To address this, some researchers have investigated

the diligent combination of algorithms to minimise the individual weaknesses while making the most of their strong points.

The challenges presented by this hybrid algorithmic approach include determining which algorithmic components should be combined and in what order, as well as how to assess their performance. The advantages of the combination of algorithms, and the challenges they present, motivate the author in pursuing this rich avenue of research.

The main objective of the research is to develop a framework for creating clustering and layout algorithms, and to develop and evaluate a platform in which they can be used to explore multidimensional data. The framework has been embodied in a software system called HIVE (Hybrid Information Visualisation Environment) that has a novel hybrid algorithmic architecture at its core. This architecture enables algorithmic and visualisation components to be combined so that they complement each other in producing effective hybrid visualisation applications.

It is the intention of this thesis to provide a proof of concept for the HIVE framework via an account of advances in HIVE and observations of its use by the author and other information visualisation practitioners.

1.2 Motivation

Dimension reduction can be achieved either by linear projection algorithms such as Principal Component Analysis (PCA), or by non-linear techniques such as those in the family of Force-Directed Placement (FDP) algorithms. Both techniques consider data as a set of vectors where each element is a value for a particular variable or attribute. This allows each datum to be regarded as a point in a high-dimensional space. Linear dimension reduction techniques tend to seek a projection of these high-dimensional points on a plane, maximising variance or some other projection index. While this process can be fast and therefore more readily applied to large data sets, it is achieved via a linear combination of the variables and therefore potentially interesting non-linear structure in the data can evade detection.

Non-linear techniques, on the other hand, have more freedom to find a low-dimensional representation of the data in which complex relationships in the data are preserved. However, non-linear techniques tend to exhibit high computational complexity and therefore are not so applicable to large data sets. This is a frustrating drawback because it is commonly the case that it is harder to find interesting patterns as data sets grow in size, while the applicability of the non-linear algorithms that have more potential in finding such structure diminishes because of their complexity.

This dilemma is the prime motivator of the author's research and has prompted work into the development of faster algorithms for non-linear dimension reduction. It has been found that the hybrid combination of algorithmic components can produce models exhibiting significant reduction in computational complexity while preserving latent high-dimensional structure within data. These algorithms provide an opportunity to gain more insight into the exploration of larger data sets.

Non-linear techniques lend themselves to intuitive heuristic improvements [CT98], because they are often modelled upon simple physical systems such as the spring model [Ead84], and this helps when creating novel hybrid solutions. However, there are many hybrid algorithms and potential improvements to explore and this, in itself, presents a large problem space. The algorithms, the views of the data they produce and the interaction mechanisms that supplement their utility are shown to be useful tools in the exploration of multidimensional data. However, the number of tools available is growing as fast as the data and therefore visualisation techniques can be called upon to help the designer build them and the user decide how and when to use them.

To address this, a development environment is required in which dimension reduction algorithms can, on the one hand, be quickly prototyped and profiled, and on the other hand, be used to explore data. This dual role of an environment, both for the development and use of dimension reduction algorithms, is due to the premise that the best way of developing and evaluating visualisations is not only by the analysis of the computational aspects such as time and space complexity but also through their use in anger on exploratory tasks.

This thesis shows how the algorithmic development environment created by the author has been used to build effective hybrid algorithms and how they have been used to gain insight into abstract multidimensional data. More generally, the work explores the way that making a visualisation that is customised to one's data and interests, and which takes advantage of a palette of algorithmic components, can be a complex task – a task that may be aided by modern tools for interaction and visualisation. It would be frustrating and limiting for designers and for users if powerful tools for analysis were themselves difficult to analyse and understand. Therefore it is suggested that the use of visualisation for visualisation – in the form of well-designed interaction with the algorithmic components, processes and parameters of a visualisation system – might afford deeper insight into the visualised information itself.

1.3 Research aims

The aim of the research described in this thesis is to provide a framework in which the hybrid approach to non-linear dimension reduction can be thoroughly investigated. The framework should be implemented in a system in which algorithm designers and analysts can quickly prototype and experiment with potential solutions; it should be conducive to the visual exploration of the data space as well as the solution space. Hybrid algorithms naturally lend themselves to providing multiple views of data as they are transformed and this can be a bonus. It should be possible to assess algorithms with respect to this expressiveness and the opportunity it presents for affording richer interaction with the data they transform.

1.3.1 Thesis statement

An algorithmic development environment can be used to build effective hybrid dimension reduction algorithms and can provide insight into abstract multidimensional data. Building algorithms for information visualisation through the use of visualisation techniques expedites the exploration of the algorithms as well as the data they transform.

1.3.2 Key research questions

Previous work with hybrid algorithms has provided evidence of their efficacy, but in order to validate a general framework and environment for their creation and use, certain questions must be answered:

1. Which algorithmic components should be combined?
2. When should the different types of algorithms be used?
3. As well as facilitating the creation and evaluation of hybrid algorithms, can the system be effective in allowing the exploration of the data they transform?
4. Is visualisation good for creating new visualisations?

The first two questions pertain to an algorithmic ‘cookbook’ for the creation and evaluation of hybrid algorithms for dimension reduction. The last two questions enquire as to their use within HIVE as an environment where algorithm creation is integrated with data exploration. The answers to these questions will determine whether the framework can enhance hypothesis formation, experimentation and analysis – a fundamental cycle in visual information-seeking.

1.3.3 Approach

To answer the first two research questions, the author began experimenting with combinations of algorithmic components to create hybrid dimension reduction and clustering solutions. The palette of components the author used was made up from most of the algorithms described in Chapters 3 and 4. It was found that by matching the complexity of algorithmic components to the complexity of data as they are transformed, effective and efficient hybrid algorithms can emerge. This is documented in Chapter 5 and forms the basis of the hybrid algorithmic framework detailed in Chapter 7, identifying which algorithmic components should be used and when they should be applied.

A software environment was developed to test the framework and the hybrid solutions it helps generate. Chapter 6 details a review of the literature that was carried out as a requirements gathering phase. It is suggested that the algorithms should not only be tested using quantitative measurements such as runtime and stress, but they should also be tested for their effectiveness in allowing users to explore data. The algorithmic development environment and the studies of user engagement described in Chapter 10 answered the third research question: as well as facilitating the creation and evaluation of hybrid algorithms, can the system be effective in allowing the exploration of the data they transform?

The approach taken to answer the fourth research question (is visualisation good for creating new visualisations?) was to build the functionality to allow users to create visualisation applications through visual programming. Also, to profile the underlying algorithms via visual profiling methods such as those described in Chapter 8. Again, observations from user engagement helped answer this question.

1.4 Thesis structure

The rest of this thesis is structured as follows:

Chapter 2: A survey of the information visualisation literature is summarised. The chapter describes the major advances and the most popular interaction techniques employed in information visualisation.

Chapter 3: A detailed exposition of clustering algorithms as a means for reducing data for subsequent visualisation is provided. The six main categories of clustering algorithms are hierarchical, partitional, density-based, graph-based, grid-based and model-based. Examples of each category are described.

Chapter 4: Dimension reduction is another way of performing data reduction, somewhat orthogonal compared to the clustering approach. Again, the emphasis is on visualisation and several important dimension reduction algorithms are described.

Chapter 5: The merits of hybrid combinations of clustering and layout algorithms with respect to run time and output quality are illustrated. Examples from the literature are provided before going on to describe the author's own work in this area. A novel hybrid spring model is shown to outperform what was the fastest non-linear dimension reduction algorithm. A new non-metric MDS algorithm is also discussed. Finally a new hybrid clustering algorithm is illustrated.

Chapter 6: Given the advantages of the hybrid approach to creating clustering and layout algorithms as described in Chapter 5 it is proposed that a custom environment (HIVE) for their creation, evaluation and use would be beneficial. This chapter details a review of the literature regarding the design and development of visualisation environments as a requirements gathering phase and precursor to the implementation of HIVE.

Chapter 7: The design and implementation of HIVE is described. A novel hybrid algorithmic framework, at the heart of HIVE, is proposed to help guide the algorithm designer and semi-automatically create hybrid algorithms. Examples of HIVE's use are given along with a reflection upon the main issues raised in Chapter 6.

Chapter 8: This chapter provides an account of how HIVE's suite of tools has been extended to aid in the evaluation and intervention of hybrid algorithms.

Chapter 9: An example of how HIVE can be applied to the analysis of unstructured text is given. It was surmised that the ability for HIVE to analyse such data would boost its adoption by other researchers with the aim to gain more feedback on its use and areas for improvement.

Chapter 10: Since the early stages of HIVE's development, the author has made the source code freely available. This chapter provides an account of feedback from users. Examples and case studies of its use by others and what the author has learned from this are given. The case studies demonstrate two modes of use: one where new algorithms and visualisations are developed, and one where the software is used solely for data exploration.

Chapter 11: The thesis is concluded with a summary of the work undertaken and the value of its contribution to the field of information visualisation. The author reflects on the design implications of HIVE and potentially fruitful avenues for future research are also described.

2. Information Visualisation

Visualisation is of a holistic nature – it is more than the sum of its parts. It is essentially a cognitive aid that provides inspiration or insight into the previously latent relationships within data. This is called *cognitive amplification* [CMS99]. The subject of this thesis resides in the field of information visualisation and this chapter begins by describing the field's roots in scientific visualisation before discussing some of the facets that comprise the visualisation of abstract data.

2.1 Scientific visualisation

In 1987 the National Science Foundation (NSF) in the United States published a report, *Visualization in scientific computing* [MDB87] that paved the way for the fields of scientific and information visualisation. The emphasis on visualisation was (and still is) dominant because the NSF recognised that visualisation provides scientists with a tool that can transform their myriad data into images that allow people to recognise patterns. It was also realised that when visualising simulations of physical systems, the scientists could *steer* the simulations by changing the parameters used in their calculations and immediately gain visual feedback, whereas previously the calculations would require to be rerun in entirety.

Scientific visualisation occurs when physical data are represented by graphics portraying a physical system, allowing scientists to explore its properties. In this way the visualisation is an external aid supporting the human's mental model of a system; it helps humans perceive its properties and amplifies cognition [CMS99]. If it were not for apt visualisations then humans could easily fall foul of information overload. If someone were to look at a database table consisting of thousands of records, each representing an object (datum), and each with numerous variables (columns), it would be almost impossible to gain an overview of structure in the data. The virtue of visualisation also enhances communication and teaching because much of the information portrayed cannot be easily communicated in print [DBM89].

Application areas for scientific visualisation include molecular modelling, medical imaging, meteorology, astrophysics, flow analysis and seismology. (see Figures 2.1 and 2.2). The common property of the data predominant in such fields is that the variables are inherently spatial and can map directly on to a spatial substrate rendered on a CRT.

A typical implementation for scientific visualisation systems is in the form of a modular data-flow architecture where data are piped through a set of modules, each of which has a specific purpose such as carrying out calculations, rendering or controlling parameter values. This piping of data is akin to the familiar UNIX pipe command for controlling the flow of data in inter-process communication [Hae88]. The data-flow architecture allows the modules to be connected in a network that ultimately shapes the application with respect to its input, transformations and graphical rendering. This notion has been extended to allow users to explicitly build the data-flow network, usually through direct manipulation of representations of the modules at the interface, and effectively build their own applications [AT95a, BBB*93, Hae88, UFK*89].

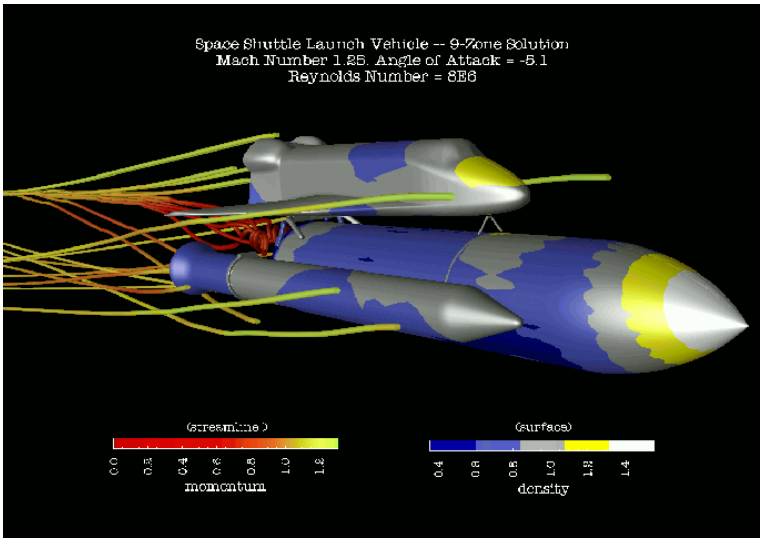


Figure 2.1 A screenshot from the IBM DX scientific visualisation system [AT95a] depicting an unsteady flow simulation over a space shuttle launch vehicle.

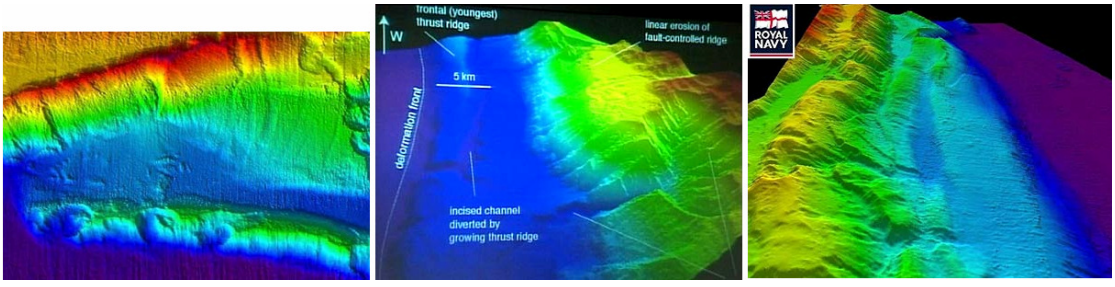


Figure 2.2 Initial survey results obtained by HMS Scott showing images of the coastline of Sumatra where the earthquake that resulted in the Indian Ocean tsunamis occurred. It is hoped that these visualisations will help scientists understand the cause of such natural phenomena and help predict them in the future.

2.2 Information visualisation

According to Card, Mackinlay and Shneiderman, the phrase *Information Visualisation* was first adopted in [RCM89]. In this context *information* refers to non-physically based abstract data and *visualisation* is the use of computers to visually render these data in such a way that humans can interactively explore their structure. Information visualisation is inspired by scientific visualisation but in this case the data to be turned into information are abstract and generally have no straightforward physical derivation. Figure 2.3 provides an example in which the abstract data, in this case search results returned by Google, can be interactively and pictorially summarised according to criteria such as hit rank and web-page size.

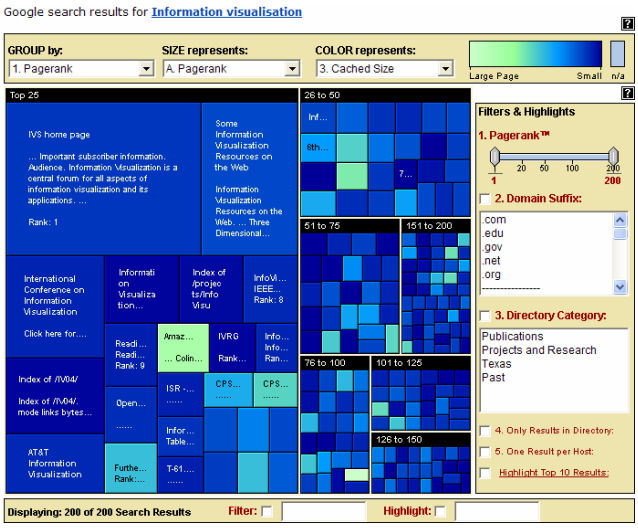


Figure 2.3 A Honeycomb© [Hon04] view of results returned by the Google internet search engine. The visualisation is based upon Johnson and Shneiderman's *treemap* [JS91]: a technique designed to utilize space efficiently in the display of hierarchical information structures.

Information visualisations are holistic and have many facets such as interaction mechanisms, spatial representations and abstraction. The fields of Human Computer Interaction (HCI), cognitive, Gestalt and ecological psychology influence them. The types of data and their volume in terms of data set size and dimensionality are key issues in determining their form, and therefore mathematical algorithms play a major part in both data transformations and visual rendering.

Applications of information visualisation include stock market analysis, project management, risk analysis, information retrieval etc. The literature presents many information visualisation techniques and workspaces borne of diverse architectures. Some examples are Visage [RCK*97], IVEE [AW95], Information Visualizer [CRM91] and snap-together visualisation [NS00a, NS00b, Nor01, NS01]. Figure 2.4 shows an application of visualisation in project management. This tool is part of a commercial issues-tracking package [Nic04] developed by the author for Nickleby HFE Ltd. In this case, the author has used the package to track issues relevant to his PhD research. The visualisation shown here is of a subset of the issues, arranged according to how they are interrelated.

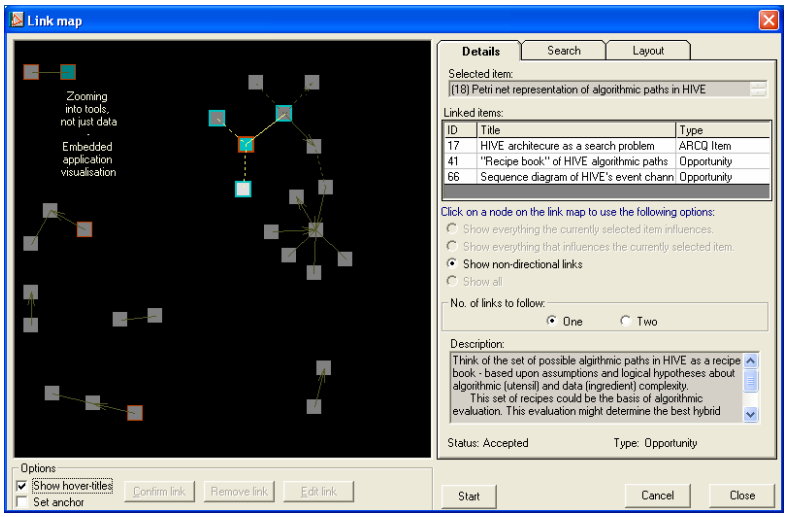


Figure 2.4 The *link map* visualisation in nickleby KIT®. Each node represents an issue raised in respect to the author's research. Nodes that are deemed as being closely related are linked. The layout was produced by a force-directed placement algorithm for graph-drawing (see Section 4.4).

2.3 Abstraction

Abstraction is a conceptual representation of a physical (or non-physical) object. In computer graphics this abstraction is the visual rendering of properties of the object. This presents less of a problem in scientific visualisation because the properties tend to be physical. However, in information visualisation, the properties of objects tend to have no straightforward derivation from physical space and thus the problem of creating visual representations that appeal to the human's perception is harder.

There are four prominent considerations in the process of creating visual representations of abstract data. These are dimensionality, which will be discussed in more

detail in the next section, data types, Gestalt principles and visual structures. Each of these must be taken into account in order to provide an effective mapping of data onto a perceptual visual form, namely a 2- or 3- dimensional spatial substrate.

2.3.1 Gestalt principles

Humans can interpret visual information very quickly. When we look at a picture, whether static or animated, our visual system allows us to perceive patterns and relationships between components of the picture. For example, a group of points on a scatterplot, which are close together, will be perceived to be a cluster and therefore stand out as one perceptual unit. This is illustrated in Figure 2.5. It is with regard to such automatic pattern or grouping detection that Gestalt principles exist.

Gestalt is the German translation of the word *shape* or *form* and is the inspiration for the Gestalt school of psychology that, in the early part of the 20th century, investigated some perceptual grouping properties and devised the *Gestalt laws of grouping* [Rom01]. Table 2.1 provides a categorisation of these laws [CMS99].

Rule	Description
Prägnanz / Figural goodness	Visual perception groups stimuli into a good figure. In this context, good means simple, regular, symmetrical etc.
Familiarity	Groups are more likely to appear if they seem familiar or meaningful.
Similarity	When presented with several stimuli, those that are similar to one another tend to be perceived as a group.
Closure	Contours that are spaced close together tend to be united.
Good continuation	A consecutive straight or curved path of close spacing through a set of objects is perceived as a group.
Proximity	Objects that are close to one another are perceived as a group / cluster.
Common fate	When objects are moving in the same direction they are seen as a group.

Table 2.1 Gestalt laws of grouping.

The discovery of these principles means that they can be exploited to produce visualisations where the human can perceive aggregate structures or patterns to form a visual indexing. This

means that individual objects within a depiction become easier to find and thus in a good visualisation, exhaustive searching is not required. For example, the *spring model* [Ead84] was proposed to produce aesthetically pleasing graph layouts but has been widely used to produce layouts of general data objects. This meant that clusters could be formed and thus aid in analysing the intrinsic relationships within data [Cha96].

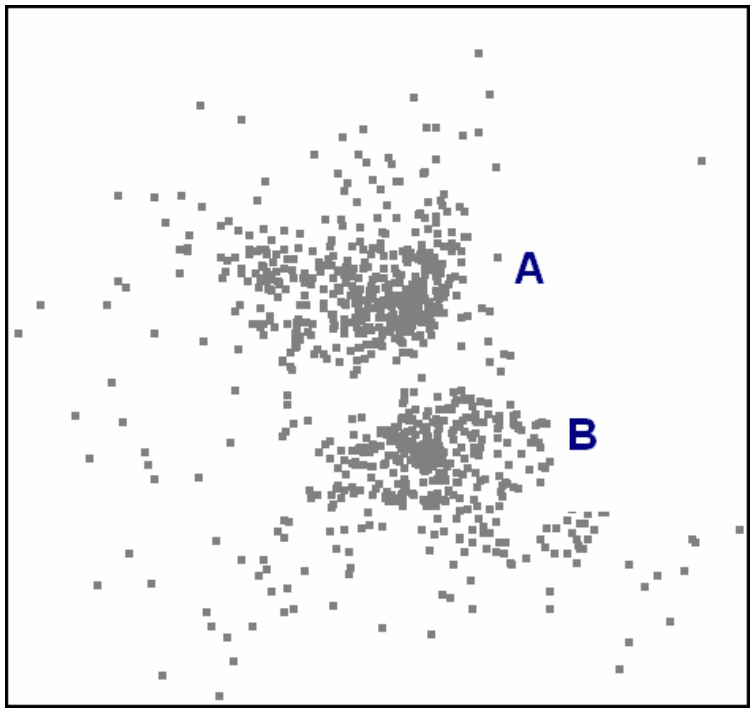


Figure 2.5 A scatterplot has the potential to make groups of points appear as individual perceptual units (clusters). For example, the author would assume that in making reference to 'A' in the figure, the reader's attention would be drawn to the upper cluster as a *whole* and not the single point nearest to the label.

Also, in [WAM01] time series data are mapped onto a spiral in order to make better use of screen real estate and to aid in the detection of cycles. This can be considered as an example of the good continuation rule and is illustrated in Figure 2.6. However, it should be noted that the groups formed within a visual representation are only useful if they reflect actual relations within the data and are not a side-effect of the underlying rendering process.

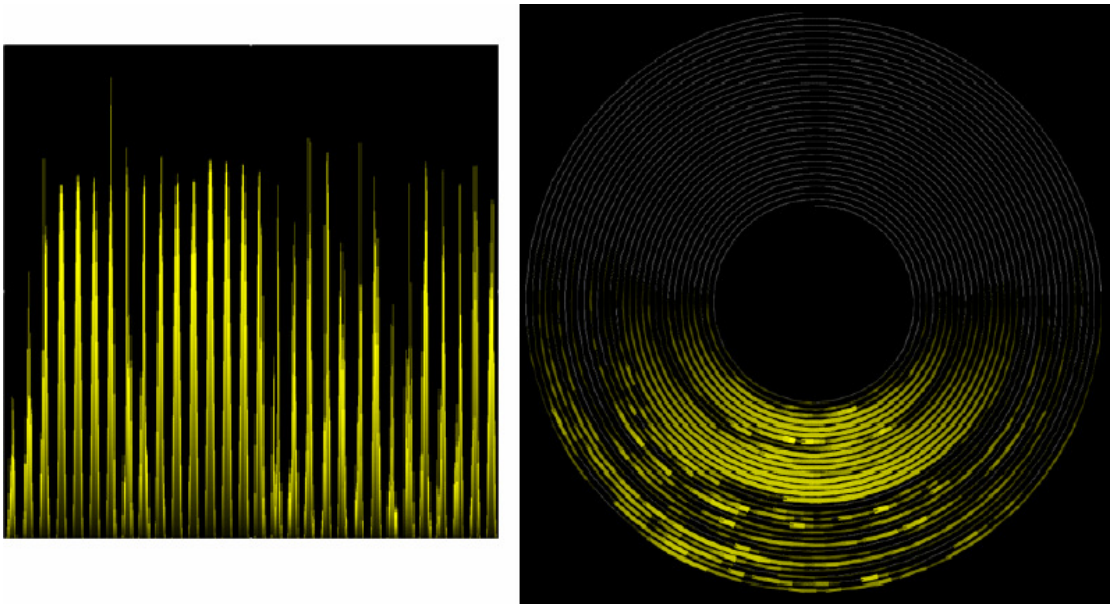


Figure 2.6 An example of the Gestalt principle of good continuation. Both of the above images represent sunshine intensity over an extended period of time, however, the spiral visualisation [WAM01] more clearly shows the day/night periods.

The Gestalt principles of organisation indicate that in a good layout, abstract data can be organised to provide a visualisation that reveals information in the structure and relationships within the data. As a final example on the importance of the Gestalt principles, consider the following figure:

Figure 2.7 The familiarity rule.

The above figure illustrates that at the heart of Gestalt theory is the proposition that in perception the whole is more than the sum of its parts.

2.3.2 Visual structures

The mapping of data as abstract objects to symbols within a space is closely related to the Gestalt principles, specifically the similarity rule described above. However, there is more to visualisation than merely grouping similar objects. In the case of using visualisations for analysis and problem solving, individual entities may require comparison. For example, in a frequency domain graph, at what frequencies are the highest magnitudes exhibited? This brings to bear the need to distinguish between the types of variable considered and the spatial substrate in which they are represented. The main categories for variable types are as follows:

Category	Description
Nominal	can only be = or != to other values
Ordinal	can obey <, ≤, > and ≥ relations
Quantitative	continuous values allowing mathematical axioms of division, multiplication, subtraction and addition

Table 2.2 Variable types.

In [CM97], a list of graphical properties is described and the appropriate mapping of variable types to some of these is demonstrated. The graphical properties are *marks* (including points, lines, areas, surfaces and volumes), position in space, and *retinal properties*, including shape, size, orientation etc. It has been established that certain variable types are better mapped onto specific graphical properties than others, i.e. some properties are more effective encoders of information than others. For example, in [CMS99] it is stated that greyscale is better for encoding and comparing nominal variables than quantitative variables.

The careful use of graphical properties is essential in creating a visualisation that communicates information to the user. There have been a number of models proposed which aim to classify data by the type of visualisation that could best convey information. Several of these are described in [Rob99] where an algebraic method is proposed to describe visualisations in order to guide the visual designer in creating the most effective depiction of abstract data.

2.3.3 Data types

Where the type of variables considered in a visualisation can suggest the most appropriate representative glyphs and symbols, the overall intrinsic structure of data (internal relationships

and dimensionality) can suggest the utilisation of pre-existing visualisation types. As an example, temporal data may lend itself to being presented as a Gantt chart.

In [Shn96], Shneiderman describes a *Task by Data Type Taxonomy* whereby a designer can choose between given examples of visualisations depending upon the type of data to be processed. The seven data types Shneiderman outlines are:

- 1-dimensional
- 2-dimensional
- 3-dimensional
- multi-dimensional
- temporal
- tree
- network

This taxonomy was devised with Shneiderman's *Visual Information seeking mantra* [Shn96] in mind: "*Overview first, zoom and filter, then details-on-demand.*" Shneiderman suggested that each component of the mantra is one of the salient tasks in visual information seeking.

The major point of this section is to show that when a data type is known, there may already be tried and tested techniques for presenting a visualisation and therefore provide a basis for discussion or prevent the designer from 're-inventing the wheel' for new tools.

2.4 Dimensionality

Dimensionality pertains to the number of attributes or variables that are to be considered for every object within a visualisation. For example, a geographical position can be described by two variables: latitude and longitude. Dimensionality is an important consideration in information visualisation because humans can only readily perceive structures within a low number of dimensions. If a set of objects of three dimensions or less is to be visualised, then the dimensions can be mapped directly onto a set of orthogonal axes. Considering the example above, a set of geographical positions may be displayed by mapping longitude to the x-axis and latitude to the y-axis, while maintaining the proportional distance interrelationships between points. However, there are many cases where the entities to be visualised have many dimensions and therefore there is no direct mapping to a 2- or 3-dimensional substrate. As an example, consider the visualisation of a corpus of textual documents where each unique word

contained within the set is regarded as a dimension. In this case the dimensionality of the space in which the objects reside can go into the tens of thousands.

In this section some of the techniques that have been applied to the visualisation of low (≤3) and high (>3) dimensional data will be discussed.

2.4.1 1-dimensional visualisation

A common example of 1-dimensional data is a list. Lists may be composed of any variable types, but in this section strings represented as ordinals will be considered. In [Eic94a] a visualisation tool called SeeSoft is presented where lines of source code are greatly visually compressed into narrow rectangles along a folding axis (see Figure 2.8). The author of [Eic94a] describes this method as *reduced representation* and claims that up to 50,000 lines of code can be displayed within one screen. The beauty of this approach is that the reduced representation holds all of the spatial pattern information within the data set in the same way as the original text, but reduced in size so that an overview is gained that maintains recognisable groupings of the unreduced text. The system also offers interactive features such as a *magic lens* to allow users to magnify and read sections of code. The retinal variable, colour, is also used to map statistical information such as modification requests to lines of code. In this way, the user can scan the overview of the code and automatically process the colour information to detect patterns and areas of interest for deeper examination. It may be argued that one of the contributors to the effectiveness of this visualisation is the familiarity rule of the Gestalt principles. The reduced representation of the source code does not distort the proportional natural layout of the data and therefore sections (groups) of lines may remain recognisable. The SeeSoft tool is also a good example of a visualisation of 1-dimensional data because it exhibits the use of a folding axis. A folding axis is an axis that is designed to use available space more efficiently by folding back on its self at certain points (again, see Figure 2.8). It is a 2-d method for visualising 1-d data and therefore can be considered as a 'dimension expansion' technique. This technique can be applied to the visualisation of data of dimensionality $d > 1$, but it is most effective when variables of only one dimension are to be depicted.

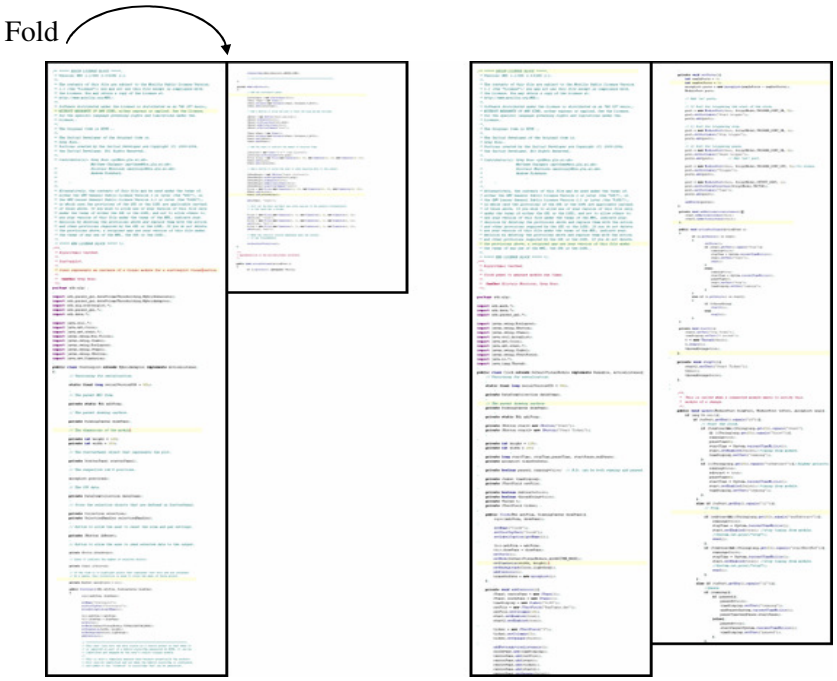


Figure 2.8 An example of reduced representation. The figure depicts two source code modules, each of which is on a folding axis.

2.4.2 2-dimensional visualisation

When dealing with data of two dimensions, the visualisation process is often based upon a simple direct mapping onto two axes. The most common form of a 2-dimensional visualisation is a geographical map where locations are placed according to the longitude and latitude variables.

Data that are comprised of two-variable entities are described as planar. This is because they map directly onto a flat 2-dimensional surface or plane. However, an interesting twist in the display of a 2-dimensional layout was proposed in [MRC91] where a 'perspective wall', shown in Figure 2.9, is described to transform 2-d layouts into a 3-d representation. The basic idea is that 2-d layouts with large aspect ratios can be distorted so that the central part of the layout is entirely visible to the user while the far left and right portions appear to stretch off into the distance. This is a technique inspired by the *bifocal lens* [SA82, ATS82]. This serves the purpose of affording the user detail and overview simultaneously and is closely related to the ideas of Furnas [Fur86]. The perspective wall is also an example of a type of folding axis in the 2-d case, where the two dimensions of the plane are folded in the direction of the third dimension (away from the user).

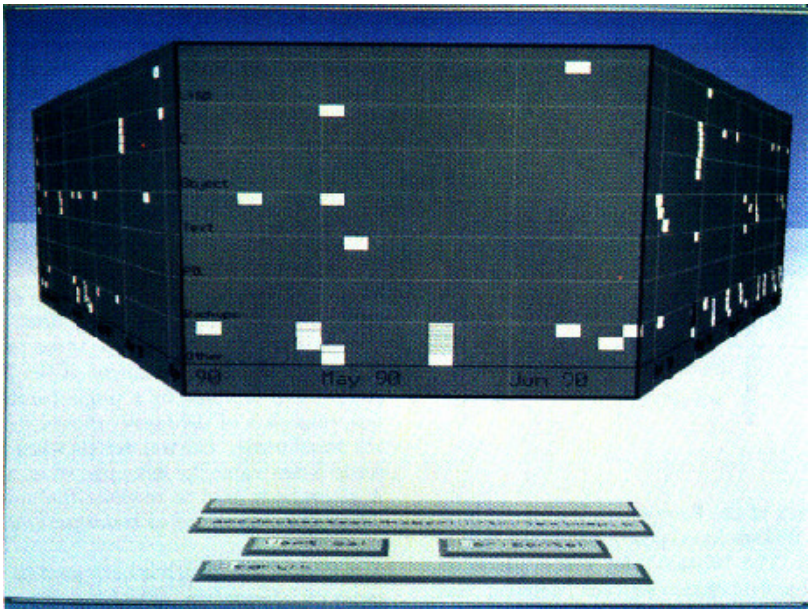


Figure 2.9 The 'perspective wall' distorts a 2-d layout so that the focus at the centre of the screen is most legible while the remainder of the layout is peripheral. The user can scroll potentially interesting parts of the layout to the fore and still be afforded the context of neighbouring regions.

2.4.3 3-dimensional visualisation

3-dimensional visualisation is most prominent in the field of scientific visualisation where collective bodies of 3-d physical data are often the basis of analysis. When a physical object is modelled it is useful to render it in three dimensions to conform to the mental model of the person who is viewing it. Examples include the visualisation of molecular structures and the physiology of the human body.

Although 3-dimensional abstract data can be directly mapped into a 3-dimensional visualisation space, for example a 3-d bar chart could be used to depict a company's profit for different products across various cities, abstract data often gains little from this embellishment. This may be because there is no inherent physical mental model to sustain. On the other hand, there have been attempts to use a 3-dimensional space to navigate complex data structures. In [HK97], a system called Cat-a-Cone consists of a hierarchical *ConeTree* [RMC91] which is displayed in three dimensions to make better use of screen real estate (see Figure 2.10). In [Ren94] a tool named Galaxy of News organises textual information in a 3-d space where similarity between texts is reflected by their proximity to one another. As the user navigates through the space semantic zooming is employed to show or elide text and detail, depending upon the user's position in the space. However, there can be serious disadvantages to rendering abstract data in three dimensions. A problem exhibited by the Cat-a-Cone system

is that the nodes of the tree can become occluded and therefore the amount of information to be gleaned at any one time is reduced. Also, in the Galaxy of News system, the lack of a referential horizon and ground plane can cause the user to be disoriented. In the words of Chalmers [Cha93], *"Our skills in…mental model-making, as honed on our everyday '2.1D' world, become more difficult to employ."* In the context of this quote, Chalmers describes a metaphor of a 2.1-d landscape for representing the distribution of a corpus of documents. This type of visualisation can be called an *information landscape* or *themescape* [WTP*95] and is based upon the premise that the metaphor can provide landmarks and other natural aids to allow the user to build a mental map of the corpus. Figure 2.11 depicts a visualisation based upon Wise's themescape [WTP*95].

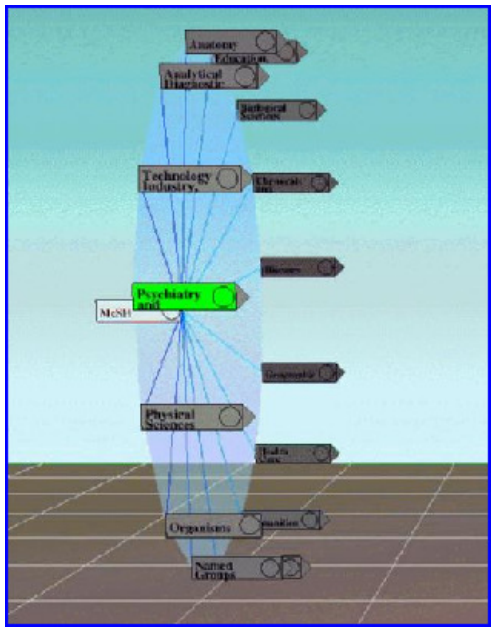


Figure 2.10 Cat-a-Cone [HK97] arranges each level of a hierarchical categorisation scheme in a 3-d view to utilise space efficiently. This technique, like the 'perspective wall', uses perspective distortion to clarify the focus (the node closest to the viewer) while maintaining the context of the adjacent nodes.

Although 3-dimensional visualisations can be impressive, they do, in general, create cumbersome overheads. They require more powerful hardware and require more intensive processing in the visual transformations; navigation is more complex because at least six degrees of freedom of movement may be required and it is more difficult to incorporate textual objects that are often predominant in information visualisation [CMS99].

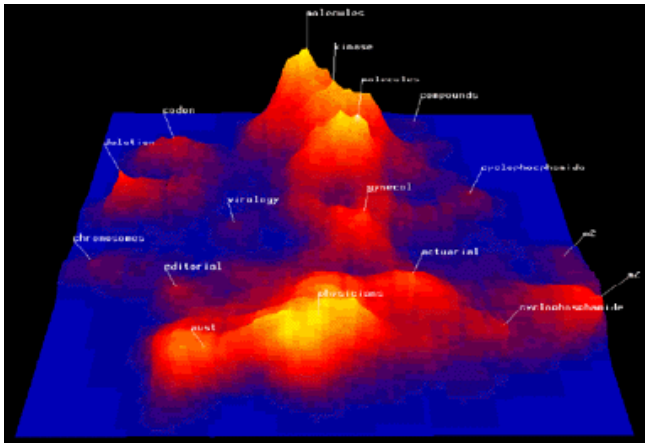


Figure 2.11 A screenshot from Spire [Wis99] – a tool based upon Wise's themescape [WTP*95]. A document corpus is represented via a landscape metaphor in which the themes that run through the collection are mapped to visual attributes.

2.4.4 4+ -dimensional visualisation

In statistical analysis, data sets that are comprised of objects consisting of more than three variables are described as multivariate or hypervariate and are considered as n-attribute items dispersed within an n-dimensional space. Thus, in information visualisation these generally come under the rubric of the multidimensional. There has been a great deal of work concentrating on the visualisation of multidimensional data, spurred on by the fact that there is no possible way of directly mapping multidimensional objects onto a set of visually perceptive axes. However, there are some shortcuts available to multidimensional data at the lower end of the scale. For example, three dimensions of 4-dimensional data may be mapped onto points in a 3-d substrate and the fourth dimension mapped to colour, or shape, but beyond this, more innovative techniques must be derived.

Spoerri [Spo93] proposes a tool called InfoCrystal (Figure 2.12) for querying and visualising results for information retrieval. The idea is to generalise the Venn diagram to discretely display the distribution of objects of more than three dimensions. This system is intuitive because of the familiarity with the Venn diagram; however, as the number of dimensions to be depicted increases the complexity of the graphics soon becomes overwhelming.

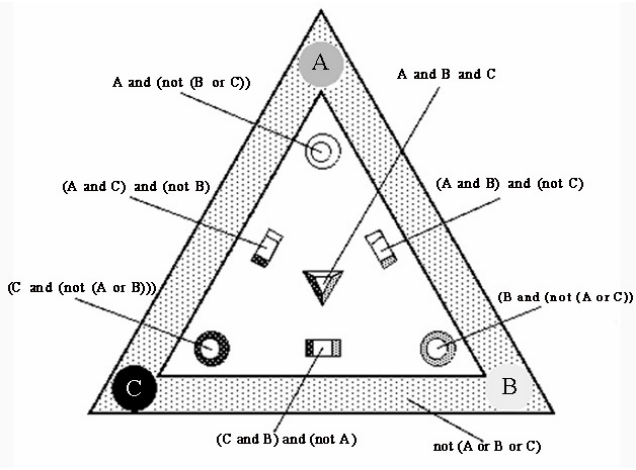


Figure 2.12 An InfoCrystal [Spo93] representing three search criteria or inputs, A, B and C and all possible Boolean queries in normal conjunctive form. The interior icons can be embellished to show the results of submitting the respective queries to a document collection. In this example, these inputs define a 3-d search space, however, Spoerri has demonstrated the application of InfoCrystals to more than three inputs.

Tweedie et al. [TSDS96] present the 'Prosection Matrix'. This idea stems from the statistical technique of representing all possible combinations of pairs of variables for a data set as a matrix of scatterplots. They embellished this technique by adding an interaction technique called *brushing* [BC87] that allows selected points in one scatterplot to be highlighted in others. In this case the brushing entails using sliders (one for each dimension/parameter) to define selected parameter ranges so that points in one scatterplot, depicting the relationship between p1 and p2 for instance, can be highlighted according to the selected range of p3 for example. Hence the name *prosection* was derived from **pro**jection of a **section**. This technique, like that in InfoCrystal, also becomes intractable for visualising data of many dimensions because the number of scatterplots required is equal to $N(N-1)/2$ where N is the number of dimensions.

As the dimensionality of data increases, the plausible techniques for clearly depicting the influence of all of the attributes falls sharply in number and in effectiveness. It is partly for this reason that methods such as Multidimensional Scaling (MDS), Principal Components Analysis and a plethora of clustering algorithms exist. Specifically, in information visualisation, their role is to map the objects from their high-dimensional space to points in two or three dimensions. These techniques will be discussed in more detail in later chapters, but for now, some examples of their application will be given.

Lin et al. [LSM91] take advantage of Kohonen's self-organising feature map (SOM) [KKL*00], to map high-dimensional textual documents onto a discrete 2-d grid. As stated

earlier, a corpus of textual documents has dimensionality roughly equal to the number of unique terms contained within, and therefore it is impossible to directly map the documents as points in this high-dimensional space into two or three dimensions. Lin et al. proposed that the SOM could be used to create *concept areas* in the plane of the SOM which would effectively partition the corpus into classes and thus give insight into the topology of the corpus at a glance (see Figure 2.13).

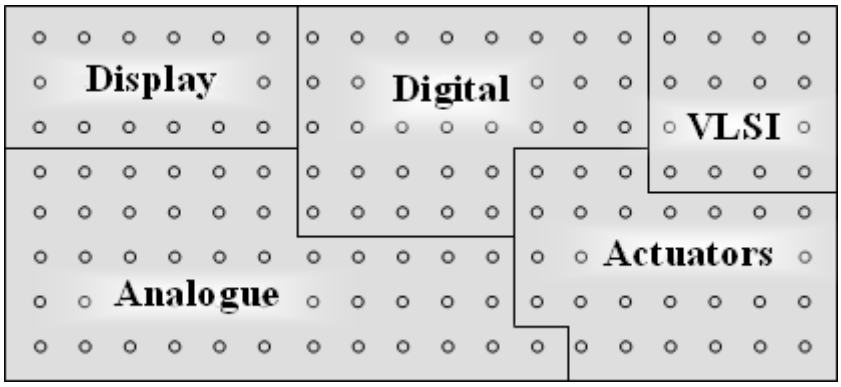


Figure 2.13 An example of the output of a SOM, depicting the *concept areas* relating to electronics.

The only drawback with this approach is that only the topology of the corpus is communicated. Relationships between individual documents cannot be visualised as only the cluster centres are depicted and the discrete grid-like output of the SOM ensures that these are all evenly spaced. The SOM is described in more detail in Section 4.2.

In a paper by Rodden et al. [RBSW01], another discrete visualisation maps images onto a grid to aid in browsing. In this case an unspecified MDS algorithm is used to create a continuous 2-d layout of objects so that similar images are placed close together, and then one of several algorithms proposed by Basalj [Bas00] is utilised to *discretise* the space in order to remove occlusions. This approach may be considered as an alternative heuristic to the SOM.

2.5 Interactivity

It is difficult to communicate the intrinsic and latent relationships within high-dimensional abstract data through a single static representation. For this reason, mechanisms which afford the user interactive control over the representation are required to unlock the information that can only be revealed in dynamic visualisations.

2.5.1 Affordance and appropriation

An important aspect that blends interactive visualisation with the premise of good graphical user interface (GUI) design is the issue of affordance. The user quickly understands the use of a device for a given function or activity. When affordance exists in the design of an interface, whether it is physical or in the digital domain, the resulting system is easier and maybe even pleasant to use [Nor88]. Sometimes affordances can be accidental, in which case the user may appropriate the functions to his or her own ends in different ways to those the interface designer intended or even considered. From the perspective of interface evaluation via observation of use, this can be advantageous in offering insight into the correct way to implement complex functions.

2.5.2 Time

Another important consideration in the design of interactive systems is the speed of interaction. In [CRM91] three categories of interaction speed are described:

Time	Category	Description
0.1 seconds	perceptual processing	Stimuli presented within 0.1s of each other are perceived to be a single stimulus. An example of this is in animations comprised of several stills.
1 second	immediate response	The minimum time in which a user may respond to stimuli.
10 seconds	unit task	Described as the time taken for a simple action, requiring minimal cognition.

Table 2.3 Categories of interaction speeds.

Inspired by the interaction between humans, Robertson et al. [RCM89], proposed an interface architecture called the *cognitive coprocessor* to match the impedance between the user and an automated information agent. Essentially, the response times of the system should match the capabilities and expectations of the user when reacting to stimuli and carrying out elemental tasks.

Shneiderman [Shn83] describes *Direct Manipulation*, which shows that a short response time for visual feedback is very important. Direct manipulation can be described as a metaphor for manipulating graphical objects as if using one's own hands, in order to conform to the user's expectations of what should happen. For example, when a file is dragged over

the recycle bin on a Windows OS desktop, and then let go, the file disappears as if the file has fallen into the bin. Shneiderman also describes the supplanting of textual query languages (in the user interface) such as SQL with direct manipulation in the form of *Dynamic Queries* [Shn94]. Dynamic queries provide immediate feedback during query formulation by updating results as the queries are built.

From the above, it can be considered that direct manipulation mechanisms must react to the user's actions within 0.1 seconds for the perceived continuity of physical motion.

2.5.3 Interaction mechanisms

In a paper by Shneiderman [Shn96], his visual information seeking mantra is described: *"Overview first, zoom and filter, then details-on-demand"*. According to Shneiderman this indicates the basic elements required in an interactive visualisation when seeking information. However, this implies that visual information seeking is a sequential process where a series of views are presented in isolation. The following sub-sections describe interaction mechanisms that have been developed to integrate such views so that overview and detail can be presented simultaneously, and zooming and filtering can be applied within the context of the original view.

2.5.3.1 Overview plus detail

An overview of a visual representation is important to afford the user navigation and pattern detection. As a result, searching can be enhanced. However, both the whole overview and the finer-grained details of local data structures are often required to facilitate analysis and evaluation of smaller portions of data. The overview enables a high-level view to help orient the user while (s)he drills down into the details.

A typical guise of overview plus detail is the zoom function. In Eick's SeeSoft tool [Eic94a], a separate window can be shown over the reduced representation in order to allow the user to read individual lines of code. The advantage of this is that the user may perceive where he or she is within the overview and also gain finer details of that area. This, applied in SeeSoft, is an example of *focus plus context* and mimics the human's visual system where the bandwidth is split between the peripheral view and the higher resolution focus [CMS99]. This allows people to understand something by its context as well as its detail within the context.

Two everyday examples of overview + detail include Windows Explorer – the overview is provided by a treeview while the detail is shown as a set of file and folder icons in a separate pane – and Adobe Acrobat [Ado04] where the thumbnail view gives a (reduced representation) overview of a PDF file, next to the detailed text. North, Shneiderman and

Plaisant [NSP96] contributed a more novel application in the visualisation of a medical digital library. The overview is a longitudinal cut of a human body and the detail view consists of an axial cross-section (see Figure 2.14).

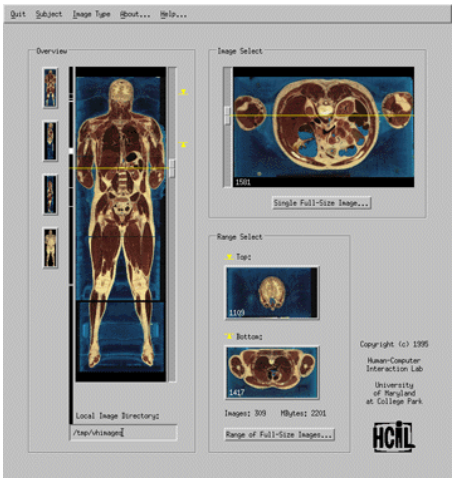


Figure 2.14 North et al.'s Visible Human Explorer interface. The overview of the human body is tightly coupled with axial detail view. The user can sweep a horizontal line across the overview to dynamically update the detailed cross-section view.

The following subsection is inspired by the problem of overview plus detail and describes some of the methods for achieving it.

2.5.3.2 Focus plus context

Focus plus context is related to overview plus detail by the fact that the context is provided by the overview and the focus is on the finer detail. A classic example of the focus plus context method is Furnas's *Generalised Fisheye Views* [Fur86]. In this paper, Furnas defines a *degree of interest* (DOI) function that is used to assign a number reflecting the importance to the user of an object within a visual structure, given his or her current task. This number can then be used to reduce or remove detail from less important areas of the view.

This is an example of a distortion technique similar to the Perspective Wall [MRC91]. Another example of view distortion for focus plus context is the *hyperbolic tree* [LRP95], where it is proposed that by mapping large hierarchical trees onto a hyperbolic plane, the hyperbolic geometry will create a fisheye-like distortion. The part of the plane that is the least distorted (more detailed and larger) is that which is closest to the viewpoint in the centre of the screen, i.e. the focus, whereas other areas are more distorted and shrink the embedded objects as the distance from the focus increases. Direct manipulation is used here to rotate the

hyperbolic plane and thus move the focus, and as a result very large tree hierarchies may be displayed.

Zooming is another well established technique for gaining insight into detailed areas of a view while maintaining the context. There are two predominant types of zoom mechanisms: semantic and logical. Logical zooming lends itself more to our familiar notion of zooming as it describes the perspective notion of objects being larger when closer and smaller as the distance between them and the viewer increases. For this reason, logical zooming can be described as physical or geometric because our psychophysical perspective pertains mainly to changes in object size and colour saturation for this type of zooming. The SeeSoft tool described above makes use of logical zooming.

On the other hand, semantic zooming, which may or may not contain aspects of logical zooming, pertains more to the idea that as the area of focus approaches objects, the level of abstraction is changed – mappings of data attributes to graphical properties change. A user-interface proposed by Bederson and Hollan called Pad++ [BH94] provides semantic zooming. Here, direct manipulation of a focus point is used to provide additional details to objects that appear under the focus. The *Magic Lens* [FS95] is another example of the zooming paradigm and has been demonstrated as a way to supplant textual database querying because multiple lenses (focus points) can be used in conjunction to form Boolean expressions that filter or abstract details of the objects being visualised.

This now leads on to describing filtering within an information space. As stated earlier it is difficult to map high dimensional abstract data into a single static view. As a result MDS techniques have been devised to reduce the dimensionality in order to be able to display high-dimensional objects in a 2- or 3-d point space, usually preserving some *distance* function. Although this provides an overview of the data set's distribution, the contribution of individual attributes can be hard to interpret in these scatterplot-like displays – some variables may be more dominant than others. It is for this reason that filtering comes into its own as a means for drilling down into the data to help find latent relationships. As described, the magic lens is one means but there are many other GUI components that may afford the filtering process. One such component is the slider control, which provides an example of direct manipulation for view transformation. In [CM97] it is stated that using sliders, a user can take into account additional variables without these being mapped to retinal properties.

In essence, a slider is a GUI control that allows the user to define ranges (double-ended sliders) or to select individual values or thresholds. Eick [Eic94b] describes the use of sliders to filter or highlight items within a view as determined by the value or range of values selected by the slider. An example of which can be found in Ahlberg and Shneiderman's *filmfinder*

[AS94a] as shown in Figure 2.15. In filmfinder, the range selected in a double-ended slider maps to a zoom function, while the position of the range, along the slider's scale, maps to a pan function.

Eick then goes on to describe graphical embellishments such as using the space inside the slider to depict the distribution of the data being analysed (see Figure 2.16) and thereby provide clues in information seeking.

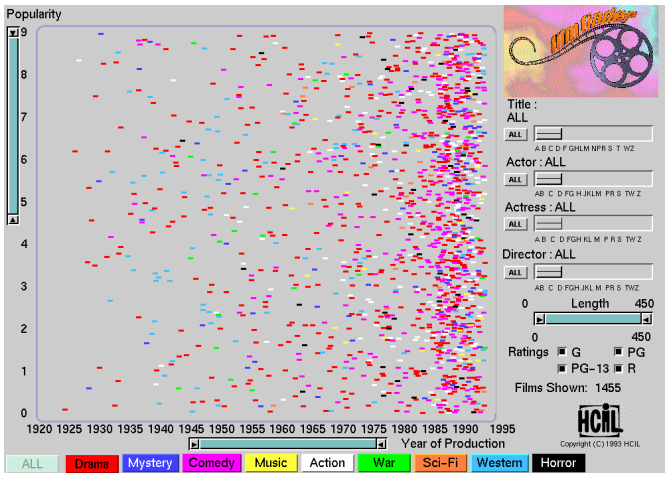


Figure 2.15 Double-ended sliders to the left and bottom of the plot in filmfinder allow the user to zoom and pan along the two axes, essentially filtering the view.

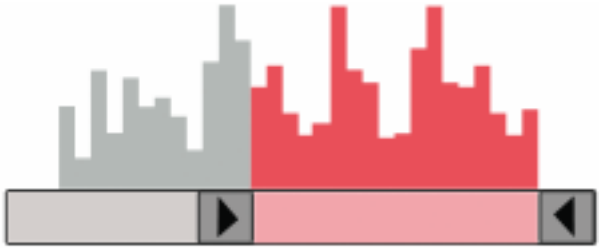


Figure 2.16 A double-ended slider with a histogram, showing a range selection.

Tweedie et al. [TSDS96] also make use of this type of enhanced slider. In their *influence explorer* tool, where histograms are used in conjunction with sliders, the sliders are all interlinked so that when the selection of one slider is changed, the effect can be seen by highlighting sections of the histograms of other sliders.

2.6 Conclusions

In this chapter some of the techniques used in information visualisation, and their motivations, have been described. Information visualisation serves as a key to unlock the black box of abstract data, to reveal the interrelationships and salient properties. It is holistic in nature – it is more than just a collection of glyphs, axes and graphical structures. Through abstraction, visualisation can help the user create mental models of the data and gain insight into their structure. Through interaction, the user is prompted to ask questions and then be provided with the answers. It can afford the user navigation and browsing of abstract data whose elements reside in a bewildering number of dimensions. It is envisaged that in the future many more interesting and novel devices will be devised to aid the user's perception of complex data and their interrelationships.

Visualisation relies upon intuitive reduction of data so that their representation is simplified and therefore information is easier to convey. A very popular means of attaining this is via cluster analysis. By considering data as points in a data space, clustering algorithms can find contiguous groups of closely related points thus reducing the representative size of the data set to the number of clusters. Another approach is to reduce the dimensionality of the data so that it conveys as much of the original information as possible using a small number of derived dimensions. As will be seen in later chapters, clustering and dimension reduction algorithms often go hand-in-hand to create hybrid solutions that provide an efficient and effective basis for visualisation.

This thesis is concerned with the development of a system and framework for building and using such hybrid algorithms. Most of the visualisation techniques described above have been called upon to assist the user in creating algorithms, as well as for interacting with their output. The next chapter discusses the prominent methods for clustering data.

3. Clustering Algorithms

In many disciplines, clustering is used as an exploratory tool for multidimensional data. It is essentially the process of organising points (or patterns) in a multidimensional space into groups based upon similarities. When applied to information retrieval, van Rijsbergen's cluster hypothesis [vRij79] succinctly states that if a document is similar to one that is contained within a known cluster of documents, then it is with high probability that it is also similar to the other documents within that cluster.

Clustering provides a compact representation of data – instead of coping with a large amount of data, the clusters can be regarded as classes or categories and therefore made easier to understand and manipulate, especially when visualised. Jain et al. [JMF99] state that the process of clustering can be broken into the steps described in Table 3.1:

Clustering step	Description
Pattern representation	This includes determining which features (dimensions) of the objects comprising the data set are to be considered in calculating cluster memberships.
Pattern proximity measure	This defines how the similarity between objects is measured. This is normally based upon a distance function, of which the Euclidean distance is most widely used.
Clustering/grouping	This defines how the clusters are created. Methods include hierarchical, partitional, density-based, graph-based and model-based.
Data abstraction	Devising a succinct description of the clusters based upon their members. These representations are known as *cluster digests* when applied in scatter/gather [CKPT92], and as *concept areas* [LSM91] in SOMs.
Assessment of output	Evaluating the validity of the clusters.

Table 3.1 The main steps involved in cluster analysis.

Note that in the above, a final step may be added that leads to adaptation of the algorithm as a result of the assessment of its output. This is especially the case in supervised learning algorithms.

In real data, clusters come in all different shapes, sizes and densities with varying degrees of noise thrown in. It is therefore the goal of clustering algorithms to extract clusters in as many of these situations as possible. However, this is not a trivial task. Some clustering

algorithms cope well with, for example, finding clusters of varying sizes and densities, but not within the presence of noise or for different shaped clusters.

In this chapter, several types of iterative, unsupervised clustering techniques are described, along with their visual representations where appropriate. Hierarchical and partitioning algorithms are described before going on to distinguish between graph-theoretic, density-based, grid-based and model-based approaches. The underlying algorithms generally concentrate on the high-dimensional clustering process but do not often provide a direct mapping to a lower dimensional visualisation. However, it is suggested that these could form a pre-processing step in dimension reduction. The chapter following this will describe algorithms that reduce the dimensionality of the data so that clusters can be directly visualised resulting in a more intuitive representation.

Unsupervised algorithms will be focussed upon because they endeavour to find the natural groupings within abstract data where little is known *a priori* as to the intrinsic classes contained and structure of the data.

3.1 Hierarchical

Hierarchical clustering produces one large cluster (the entire data set) and partitions of sub-clusters. These sub-clusters, in turn, contain clusters, and so on. This can be represented by a structure called a dendrogram (see Figure 3.1).

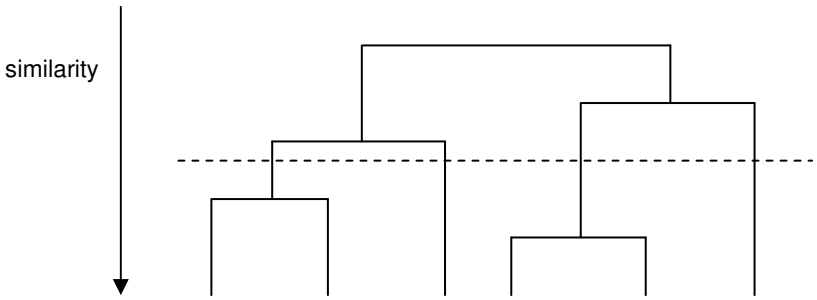


Figure 3.1 A dendrogram, which when cut at different levels, will produce different clusters.

Hierarchical clustering algorithms can be split into two types: *agglomerative* and *divisive*. Hierarchical agglomerative clustering (HAC) starts by considering each individual element in the data as a unit cluster and proceeds by merging elements (clusters) together until the desired number of clusters is reached or all of the elements in the data set have been included in a cluster. With regard to the dendrogram, this method may be seen as creating the

hierarchy from the leaf nodes upwards. Examples of this type of algorithm are *single-link* and *complete-link*.

However, one drawback of HAC is that most of the clusters in the lower levels of the dendrogram are very small and very close together and therefore tend to be a waste of computation. This is especially problematic in HAC because these small clusters are necessarily formed before the larger ones. Manoranjan et al. [MLST03] call this the *90-10 rule* and use it to increase computational efficiency when validating clusters.

The alternative, divisive approach (known as *numeric taxonomy* in the field of *machine learning* [LS97a]), works by initially considering the entire data set as one large cluster. The algorithm then proceeds by partitioning the set until, as above, a criterion for stopping is met. With this approach the *90-10 rule* can be taken into consideration when deciding when to stop the clustering process.

3.1.1 Agglomerative single-link clustering

Single-link clustering – also referred to as *nearest neighbour* clustering – is one of the oldest clustering techniques (see [Sib73] for an implementation). It is an agglomerative hierarchical algorithm in that it starts with every point belonging to its own cluster and progressively merges clusters until one large cluster, containing all of the points, remains. The distance between two clusters is taken as the shortest distance from any member of one cluster to any member of the other. Pseudocode for the single-link algorithm is shown in Figure 3.2.

```
1. given N points to cluster, assign each to its own cluster, i.e.
   initially there are N clusters
2. find the closest pair of clusters and merge them
3. compute the distance/(dis)similarity between the new cluster and
   all other clusters
4. repeat steps 2 and 3 until all items are members of a single
   cluster of size N
```

Figure 3.2 The single-link clustering algorithm.

The output of the algorithm is a nested set of graphs represented by a dendrogram (see Figure 3.1), which when cut at the desired level (of distance/(dis)similarity) yields a clustering of the data. One of the major drawbacks of this approach is that it suffers from a *chaining effect*

where chains of close points form bridges between clusters and therefore erroneously merge them. As a result, the single-link algorithm tends to return elongated clusters [JMF99].

Two variants of this algorithm are *complete-link* and *average-link* techniques. These algorithms are identical apart from the way in which distance between clusters is measured. The complete-link method treats the distance between two clusters as the maximum distance between any member of one cluster and any member of the other. This serves to circumvent the chaining effect incurred by the single-link method and tends to return compact clusters. The average-link method treats the distance between two clusters as the average distance between any member of one cluster and to any member of the other.

While these hierarchical algorithms are simple and intuitive, they have a number of disadvantages. They tend to be quite computationally expensive ($O(N^2)$) and may prove to be infeasible for use with large amounts of high-dimensional data. In information visualisation, views often need to be dynamic and be generated on the fly, and therefore need fast clustering algorithms with respect to the underlying data. Also, dendrograms can be very hard to interpret, especially when the data set is large. Although the user does not have to estimate the number of clusters in advance, deciding which merging/splitting strategy to apply and determining where to split the dendrogram can be difficult.

3.1.2 Scatter/Gather: An application of hierarchical clustering

Cutting et al. [CKPT92] describe a hierarchical clustering technique for information retrieval aimed at overcoming the problem of users not being able to initially and precisely define their search goal. This technique is called *scatter/gather* and is inspired by the way that people use the table of contents of a textbook in order to gain a sense of the structure of the book as a precursor to searching for specific topics. The process works by clustering (scattering) the corpus into a number of clusters and presenting the user with short summaries of each cluster. These are called *cluster digests.* The user can then select (gather) one or more of these clusters and the system performs another clustering stage upon this sub-collection. This process continues, iteratively refining the results returned to the user until individually enumerated documents are presented.

Pirolli et al. [PSHD96] indicate that scatter/gather allows the user to more efficiently browse a textual corpus, as opposed to searching for specific topics. This leads to gaining an insight into the distribution of topics within the corpus, which is a result in itself, thus allowing the user to build a mental model. Armed with this perception of the corpus, the user can then search the corpus and receive more accurate results.

Scatter/gather requires that the clustering algorithms used can be applied on-line. This is due to its interactive nature, requiring speedy responses to users' requests. It is for this reason that algorithms such as *Buckshot* (Section 5.1) are used to obtain clusters in rectangular time complexity.

3.2 Partitional

Unlike hierarchical clustering, partitional clustering forms a single partition in the data set, i.e. no dendrogram is created, and because of this, it is less computationally expensive. Generally, the input to partitional clustering is the data set and the desired number (*k)* of clusters.

The most common partitional routine is an optimisation algorithm. Algorithms of this type, such as K-means [Mac67], aim to minimise a cost function which is associated with each cluster.

3.2.1 K-means

This algorithm is iterative and centroid-based. This means that the algorithm goes through an unknown number of cycles of creating clusters and updating the centroids until it finally converges. The algorithm starts by selecting *k* items from the data set, where *k* is the desired number of clusters to obtain – these are the initial cluster centroids and are often referred to as *seeds*. The next step is to assign each of the remaining elements from the data set to the centroid that is closest (in Euclidean space). Once all of the data items have been assigned to a cluster, the centroids are recomputed by calculating the average of all of the cluster members and the process of clustering begins again. This iterative process continues until no items change cluster membership. At this point it can be said that the algorithm has converged. It should be noted that the data patterns considered must consist of continuous value vectors because their arithmetic mean must be calculable. Also, the cost function that K-means endeavours to minimise is the common *sum-of-squares* criterion, i.e. it should minimise the sum of the squares of the inter-object distances within each cluster.

Pseudocode for the K-means algorithm is provided below:

1. let $L = \{l_1, \ldots, l_k\}$ be a random subset of the data set $P = \{p_1, \ldots, p_n\}$ where $k < N$
2. create k arrays to hold the cluster members
3. for each data point $s \in P$ find the closest centroid k_i in L, according to Euclidean distance, and allocate s to the appropriate array
4. calculate the arithmetic mean of each cluster and let these now be the members of L
5. if the values in L have not changed then return the clusters as the solution, otherwise go to step 3

Figure 3.3 The K-means algorithm.

The attractiveness of the K-means clustering algorithm is that it is easy to implement and its computational complexity is reasonable [JMF99]. Its time complexity is $O(CNk)$ where C is the number of iterations, and its space complexity is $O(N + k)$. Another advantage of K-means is due to its iterative nature. Iterative algorithms tend to allow the addition of new data to the set after convergence is achieved, rather than performing the clustering from scratch. This is because the clusters that are formed can be regarded as categorical classes of the data and therefore the addition of new items simply implies associating them with the closest cluster. However, K-means does have at least two drawbacks. The first is that the clusters that it is trying to find are assumed to lie in a spherical Gaussian distribution [BF98]. The second is that, although it will converge, the algorithm is sensitive to the initial choice of cluster centroids. Both of these points mean that K-means will often converge to a local rather than a global minimum. However, regarding the initialisation problem, Bradley [BF98] has developed a technique using multiple runs of the algorithm and then choosing the best outcome according to a measure of validity. Another solution is the use of *classifier ensembles* [KHDM98] where the outcomes of several clustering runs are regarded as *votes* for the obtained clusters (classes) – the clusters that receive the majority of the votes (appear consistently) are retained. Classifier ensembles will be discussed further in Chapter 5.

3.2.2 Bisecting K-means

A variant of the traditional K-means called *Bisecting* K-means is described by Steinbach et al. [SKK00]. This works in exactly the same way as regular K-means with the exception of the selection of the number of cluster centres. Initially two centroids are chosen and the clusters are produced in the same way, then according to intra-cluster variance or size, one of these

clusters is split into two more clusters. This continues until some convergence criterion is met (for example, the centroid values stop changing).

In [SDB01] it is stated that bisecting K-means is useful because it produces a binary taxonomy, useful in document retrieval. Also, in [SKK00], the algorithm is conjectured to be superior to traditional K-means because the clusters are nearer to being uniform in size and this results in lower entropy.

Bisecting K-means, although based on a partitional algorithm can, however, imply the production of hierarchical clusters. This is due to its ability to produce a binary tree similar to the dendrogram.

3.2.3 NNS with K-means: An application of a partitioning clustering algorithm

Nearest Neighbour Searching describes a way to find the most similar item (nearest neighbour) within a data set to a given query. It was considered that this approach could be useful for speeding up the process in the addition of new data to a pre-configured spring model layout. An informal experiment was set up to use the K-means algorithm for such a purpose. The experiment consisted of a program written in VB 6.0 that plotted 2-dimensional normally distributed random points on a plane. K-means was then used as the clustering stage to a Buckshot algorithm that was run on these data so that $\sqrt{N}$ clusters were formed. The idea then was to simulate the submission of a query (addition of a new point) into this space by plotting points at random on the plane. In order to find the nearest neighbour of each 'query', the query is first compared to each of the K-means centroids to find the closest in Euclidean distance. Once this closest cluster centroid was found, each of the cluster members are then compared to the query and the closest point is the one that is returned as the nearest neighbour. See Figure 3.4 below. This approach is faster than an exhaustive search of the data because only $\sqrt{N} + m$ distance calculations are required, where m is the number of cluster members associated with the closest centroid.

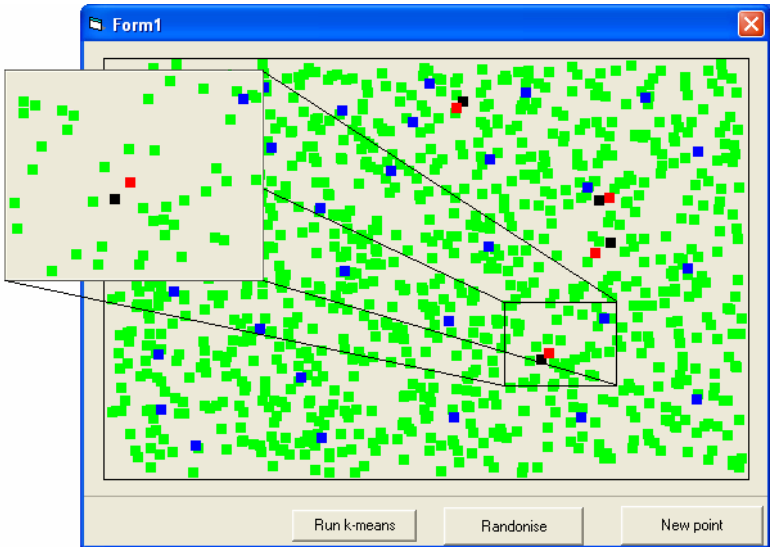


Figure 3.4 A screen shot of the K-means NNS experimental program. Green points represent randomly distributed data, the blue points represent K-means centroids, and the red points indicate 'queries' while the black points are the approximate nearest neighbours to the (red) query points.

As can be seen from the above, the results prove quite promising when applied to this normally distributed random data. However, there are cases envisaged when the algorithm will not return the correct nearest neighbour, consider Figure 3.5 below:

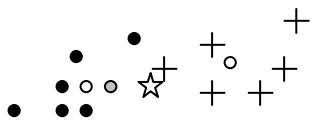


Figure 3.5 A case where the K-means NNS is only approximate.

In this scenario, the two hollow circles are centroids and the black points and crosses are their respective cluster members. If the star is regarded as the query then clearly it is the cross just above it that is its nearest neighbour. However, the algorithm returns the grey point to its left as the closest instead of the cross. This is because the algorithm initially looks for the closest centroid. It is only the set of black points that are considered to contain the closest point because in this case, the cluster centroid for the black points is closer than the other centroid that actually represents the cluster containing the nearest neighbour. This means that this approach to finding a nearest neighbour, given a new point, is only an approximate solution, but it is envisaged that it will still perform well for introducing new points into layouts by initially placing them somewhere within the vicinity of the best position.

3.3 Density-based

Instead of basing a clustering upon the proximity of data points to representative centroids, as is the case in partitioning algorithms, density-based clustering seeks groups of points that are dense, and which are separated by sparse regions. The advantage of this approach is that clusters of various shapes can be retrieved while noise points – those data that do not belong to any cluster in particular, including outliers – can be effectively filtered out [SEKX98].

A well-known density-based technique called DBSCAN was proposed by Ester et al. [EKSX96]. In their proposal, density is associated with a point by determining the number of neighbouring points within a specific radius ε, referred to as an *Eps*-neighbourhood. When a point has a pre-specified minimum number of points *MinPts* within this radius, it is classified as a *core point* and a cluster *C* is created for it and its *Eps*-neighbours. Next, each member of *C* is checked to see if it is also a core point and if it is, it and its *Eps*-neighbours are added to *C*. This process is continued until no more points can be added to *C*. After all possible clusters have been found in this manner, points that are not classified as core points or cluster members are classified as noise points. Points that are not deemed as noise or core points are called *border* points and are considered as members of the cluster associated within ε of the nearest core point.

Two disadvantages of this algorithm are that 1) it relies on the user to specify the *MinPts* and ε inputs and 2) these are global parameters and therefore clusters of different densities cannot be found. For example, when the density threshold is quite low, only the densest clusters will be found while sparser ones will be discarded as noise. On the other hand, if the threshold is lower, dense clusters might be merged (see Figure 3.6). As Estivill-Castro and Lee point out, density is hard to define; the user should not have to guess a global density threshold [CL02].

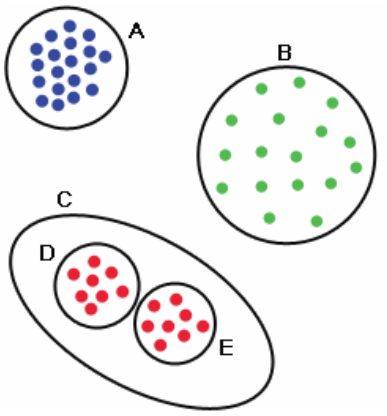


Figure 3.6 Clusters of different densities. When the clustering is via a global density parameter, only clusters [A, D and E] or [A, B and C] will be found.

To overcome the shortcomings described above, Ankerst et al. developed an algorithm called OPTICS [ABKS99]. This method imposes a special ordering upon the data set called the *cluster order* which is based upon the distances between points within an *Eps*-neighbourhood threshold. This ordering captures information equivalent to the output of the DBSCAN algorithm over a range of ε values and therefore can find clusters of different densities simultaneously. Another advantage is that the distances, called *reachability distances*, provide a view of the clustering structure inherent in the data when plotted against this cluster order. The algorithm requires the user to specify values for ε and *MinPts* but the authors provide heuristics to determine these and show that it is much less sensitive than DBSCAN to the values chosen.

While OPTICS provides a solution to finding clusters of disparate densities without relying too heavily on the user to specify abstract parameters, traditional Euclidean density-based techniques still do suffer from one major drawback when clustering high-dimensional data. Ertöz et al. [ESK03] pointed out the fact that as the dimensionality of the data increases, the number of points required to maintain a specific density – that is, a certain number of points per unit volume of space – increases exponentially. This is known as the *curse of dimensionality* [Fri94]. To alleviate this, Ertöz et al. defined an alternative notion of density in their graph-theoretic clustering algorithm, based upon *shared nearest neighbour* graphs. This will be discussed further in the Section 3.4.

Another approach for density-based clustering was proposed by Duyckaerts and Godefroy for studying neural densities in the thalamus and cortex of the human brain [DGH94]. In this application, the authors wished to be able to compare neuronal densities and

subsequently find clusters of neurons[1]. To achieve this, the authors use a Voronoi tessellation of part of the brain surface. The tessellation partitions each neuron into the area that it occupies such that any point within this area is closer to that neuron than any other and the result is that each neuron resides in a convex polygon (Section 5.5 for a description of Voronoi tessellations). In a Voronoi tessellation, clusters produce groups of contiguous polygons with small areas compared to their neighbours [OBSC00]. Duyckaerts and Godefroy take advantage of this property to define a simple clustering algorithm that first finds the smallest polygon, adds its neuron to a cluster and then *grows* the cluster by examining its neighbours, adding them if their polygonal areas are within a predefined threshold. When the cluster cannot be grown any further, the next smallest polygon, that is not part of the cluster, is found and a new cluster is created. This process continues until no more polygons can be added to a cluster (see Figure 3.7). The time complexity of this algorithm is dominated by the computation of the Voronoi tessellation which is $O(N \log N)$, where N is the number of neurons.

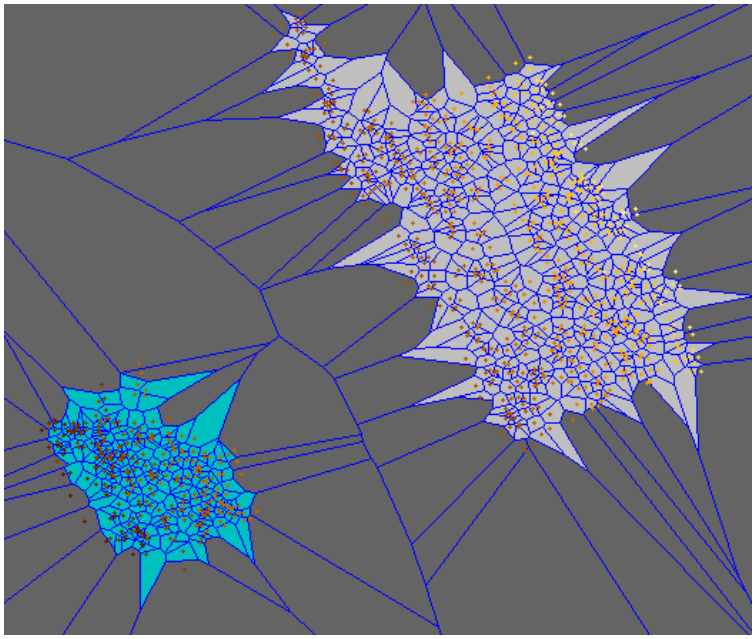


Figure 3.7 A Voronoi tessellation of a 2-d point pattern consisting of two clusters. Notice how the polygons of points inside the clusters have smaller areas than those towards the outside. Duyckaerts and Godefroy use this property to automatically find clusters. In the above example an area threshold has been set and polygons within it are shaded – each cluster is distinguished by shading with a different colour. This image was generated by HIVE [RC03a, RC03b].

[1] Within the context of this chapter, neurons should be taken as analogous with points within a 2-d data space.

The Voronoi tessellation is shown to exhibit other useful properties for analysing spatial density. By sampling mean densities, confidence intervals can be used to make statistical comparisons of density. Also, the coefficient of variance of polygon areas allows one to distinguish between regular, clustered and random point distributions. The main drawback of this approach, however, is that the time complexity of computing a Voronoi tessellation increases exponentially with dimensionality. For this reason, its application to clustering is best suited to 2-d applications.

3.4 Graph-theoretic

Graph theory provides a succinct notation for expressing relationships between points in a data space. A graph is defined by the formula $G = (V, E)$ where V denotes a set of vertices and E denotes a set of edges (arcs connecting the vertices). Given this definition, a data space can be modelled as a graph when points are considered as vertices, and constraints upon their relationships are considered as edges. The graph forms a data structure that encapsulates information about how data are related. As an example, consider the 2-d data space in Figure 3.8. If the notion of a relationship is constrained to any point and its nearest neighbours, then drawing edges between such related pairs of points can yield a graph called the *minimal spanning tree* [Pri57].

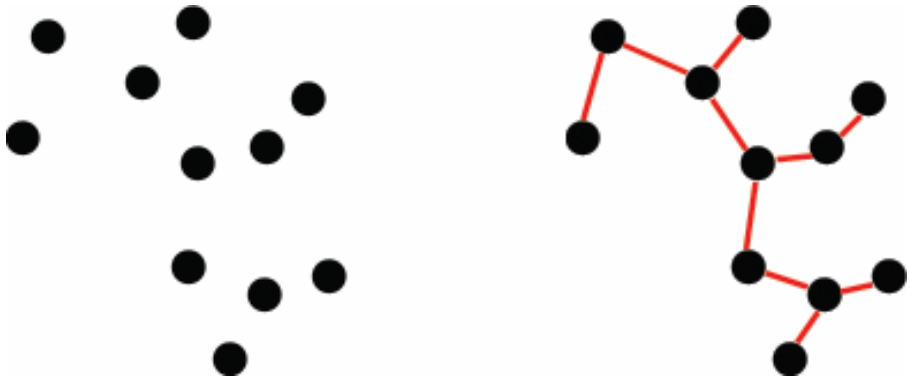


Figure 3.8 Points in a 2-d data space (left) and their minimal spanning tree (right).

There is a diverse range of graph types, each with different properties that can be utilised to model difficult problems. In the field of artificial intelligence, *state space* graphs are used to model the structure of complex systems. The state space graph provides an efficient way of traversing a sequence of state transitions to a desired outcome, such as winning a game of chess. Another problem modelled by graphs is the well known *travelling*

salesman problem. In this case the graph is traversed in search of the shortest *Hamiltonian cycle* – a path where each vertex is visited exactly once and the last visited is the starting vertex [LS97a].

The versatility of graphs in modelling complex problems can also be applied to clustering data. A data set D can be represented by a *complete weighted* graph $G(D)$. Complete, if every point is connected to every other point, and weighted if the lengths of the edges correspond to the distance or dissimilarity between points. By the careful deletion of edges, individual clusters can be separated into sub-graphs – connected components of $G(D)$ – and returned as the clustering solution. Thus, the deletion of edges is pivotal in the graph-theoretic approach to clustering.

Since the number of edges in a complete graph is quadratic with the number of vertices, it is an expensive representation. However, sub-graphs that are less expensive to compute, store and traverse can retain enough information to define clusters and therefore simplify the problem of deciding which edges to remove. Thus the *modus operandi* of many graph-theoretic clustering algorithms consists of two steps. The first is to define a graph that efficiently contains the information necessary to define clusters. The second is to employ a strategy of edge removal that will decompose the graph and reveal clusters. The best-known graph-theoretic algorithm reflects this. It starts with a minimal spanning tree and then removes the longest edges [Zah71].

3.4.1 The minimal spanning tree as a basis for clustering

Let $D = \{d_i\}$ be a data set where each $d_i = \left(a_i^1,...,a_i^k\right)$ is a datum representing the set of attribute values *1* through *k*. A weighted and undirected graph $G(D) = (V, E)$ can be defined where $V = \{d_i \mid d_i \in D\}$ is the set of vertices and $E = \{(d_i, d_j) \mid d_i, d_j \in D \text{ and } d_i \neq d_j\}$ is the set of edges. Since an edge exists between every pair of vertices, *G(D)* is a complete graph. Furthermore, the weight of each edge $(d_i, d_j) \in E$ can be represented by some measure of dissimilarity such as Euclidean distance.

A connected sub-graph containing every vertex of $G(D)$ and no cycles is called a *spanning tree* and the spanning tree with the lowest sum of edge weights is the minimal spanning tree (MST), i.e. the shortest path connecting all vertices without any cycles. In an MST such as that in Figure 3.9 it may be observed that points in a cluster are connected by short edges while longer edges connect points between different clusters [XOD02].

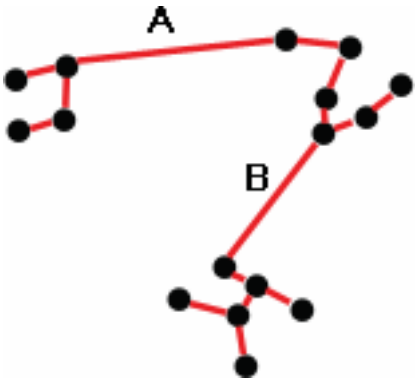


Figure 3.9 The MST above shows that intra-cluster distances are shorter than inter-cluster distances. Deletion of edges *A* and *B* would result in three separate connected sub-graphs representing the clusters.

This observation provides an intuitive basis for clustering. By deleting the longer edges of the MST, the data can be decomposed into clusters.

Zahn developed the most famous graph-theoretic algorithm [Zah71] in which edges that are significantly longer than nearby edges are removed from the MST (see Page [Pag74] for an implementation). Zahn called these edges *inconsistent* and defined them as edges whose length is more than *f* times the average length of nearby edges and more than *s* standard deviations larger than the average.

The advantages of the MST in Zahn's algorithm are that it is not contingent on the order of input and it can detect clusters of different shapes, sizes and densities. However, the construction of the MST dominates the time complexity – taking $O(||E|| \log ||E||)$ time using Kruskal's algorithm – and the user must specify quantities for *f* and *s*.

3.4.2 Other graph-theoretic clustering algorithms

While the MST provides an explicit graph representation for clustering, it is also the basis for single-link hierarchical clustering where clusters are the connected components of the MST [JMF99]. The major difference between Zahn's algorithm and single-link is in the way in which edges are removed to reveal clusters. Zahn's approach is to identify and remove inconsistent edges whereas in single-link, the user cuts the dendrogram at a specific level in the hierarchy.

Another graph commonly used in clustering is the Delaunay graph. This graph is popular because it contains the MST and the relative neighbourhood graph (RNG) as sub-graphs [OBSC00] and therefore contains more information conducive to clustering; it can also be computed in $O(N \log N)$ time for the 2-d case. The Delaunay graph forms a triangulation over the data by defining edges between all pairs of points that are Voronoi neighbours.

Estivill-Castro and Lee [CL02] use this in their AUTOCLUST algorithm to find clusters by searching for their boundaries. They postulate that edge lengths at the boundaries of clusters exhibit more variability because they connect inter- and intra-cluster points. The authors use both local and global neighbourhood relations to identify edges which are statistically significant with respect to this variability.

Eldershaw and Hegland [EH97] also utilise the Delaunay graph. They noted that Zahn's technique of removing edges from the graph that exhibit a length greater than f times the average length of nearby edges, was not applicable because in the Delaunay graph, vertices on cluster borders tend to have more incident edges that span between clusters and this pulls up the average length resulting in all edges being preserved. To overcome this, they consider the choice of an edge-length threshold p as a reduced clustering problem. In the search for p, they note that this is a classification of the edges into two classes; one for short intra-cluster edges ($\leq p$) and one for long inter-cluster edges that should be removed ($>p$). To identify a suitable threshold, a range of values for p are fed into a cost function $T(p)$ that measures how "neatly split" the edges are between these two classes. By plotting $T(p)$, the authors demonstrate that the global minimum can be easily found. Eldershaw and Hegland's approach is interesting because it reduces the M-dimensional clustering problem into a two-cluster 1-D problem. However, the drawback of this approach is that the threshold parameter p is global and therefore clusters of different densities may evade detection. Again, the time complexity of this algorithm is dominated by the triangulation of the Delaunay graph which increases with dimensionality by $O(N^{\lfloor d/2 \rfloor + 1})$ [Aur91].

Up to this point, each of the graph-theoretic approaches described have relied on graph edges being defined by a direct dissimilarity measure between points. However, as Ertöz et al. [ESK03] point out, direct measures of (dis)similarity can be misleading. One example is in measuring the Euclidean distance between texts; Ertöz et al. show that the distance between documents that do not share any common terms can be lower than between documents that do share terms. They also point out that similarity measures such as the Jaccard coefficient and cosine similarity can be "unreliable" when overall similarity between points is low. To counter these shortcomings, rather than using direct measures of (dis)similarity some clustering algorithms define similarity in terms of the number of nearest neighbours shared by points. For example if u and v are two points that are close to each other and also close to a set of points S then their similarity is "confirmed" by their mutual proximity to points in S. When an edge is drawn between every pair of points that have each other in their list of k nearest neighbours this forms the shared nearest neighbour (SNN) graph [JP73]. The weight of an edge in an SNN graph, i.e. the similarity between its end points is defined as follows:

$$similarity(u,v) = size(NN(u) \cap NN(v)) \quad (3.1)$$

$NN(u)$ and $NN(v)$ are the nearest neighbour lists of u and v respectively and similarity is proportional to the size of their intersection. This similarity measure works well in high-dimensions and the SNN graph can be calculated in $O(N \log N)$ time [KHK99].

CHAMELEON [KHK99] is a two-phase clustering algorithm that utilises the SNN graph. In the first phase it creates a sparse SNN graph and partitions it into many dense sub-clusters. Density of a point is taken as the sum of the edge weights incident to the point. In the second phase, an agglomerative hierarchical clustering algorithm is used to successively merge the sub-clusters according to their relative inter-connectivity and relative closeness.

Another approach using the SNN graph was proposed by Ertöz et al. [ESK03]. Here the density based clustering algorithm DBSCAN is extended by building an SNN graph and defining density as the number of points within a radius (*eps*-neighbourhood) determined by SNN similarity. While this algorithm still requires the user to input the *eps*-neighbourhood and *MinPts* parameters (see Section 3.3) it outperforms DBSCAN by being able to detect clusters of different densities. This is because the *eps*-neighbourhood is relative to shared nearest neighbours and not a direct measure of dissimilarity.

In summary, graph-theoretic clustering algorithms generally take extra time in graph construction but the good thing about graphs is that they can succinctly represent the information required to detect clusters of different shapes, sizes and densities when noise is present.

3.5 Grid-based

By dividing a data space into contiguous regions, it is possible to index and summarise points according to the cells in which they reside. While this provides a form of data compression, it also potentially provides fast access to information such as the neighbourhood relationships, which is required for clustering. Grid-based clustering algorithms work upon such a representation to take advantage of the fast information access and the intuitive notion of density as defined by the number of points in a cell.

The grid-based approach is the dual of graph-theoretic clustering; it was stated in Section 3.4 that the *modus operandi* of many graph-theoretic clustering algorithms consists of two steps. The first is to define a graph that efficiently contains the information necessary to

define clusters; the equivalent of this in Grid-based clustering is to partition the data space into regular cells. The second step in graph-theoretic clustering is to employ a strategy of edge removal that will decompose the graph and reveal clusters; in the grid-based method, the equivalent is to employ a strategy for merging cells (See Figure 3.10).

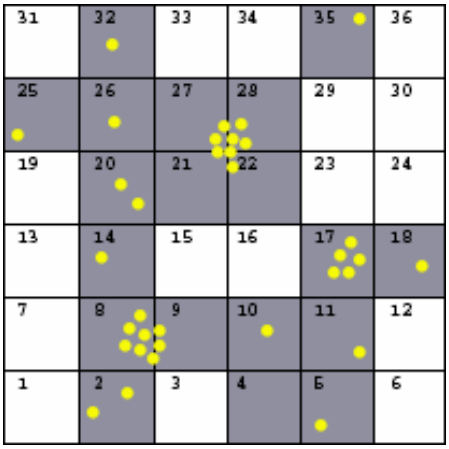


Figure 3.10 Partitioning of a 2-d data space into regular cells. The resulting structure, called a map [HK98] or grid structure [Sch96], can intuitively be generalised to the multi-dimensional case.

Wang et al. developed an algorithm called STING [WYM97] in which the data space is divided into nested rectangular cells that map to a cluster hierarchy. The smallest cells – those most deeply nested – are represented by summary statistics such as mean and standard deviation calculated directly from the data contained. The statistics of higher level cells are computed from their children. By using summary statistics to represent the data, the information required for clustering and subsequent querying can be stored in main memory even if the original data set is too large. The authors do not provide details on the time complexity required to initially partition the space, although it is suspected that it will be around $O(N \log N)$ because of the recursive nature of the cell division, N being the number of data points. However, the authors do show that the time complexity involved in querying the structure is $O(k)$ where k is the number of most deeply nested cells and $k << N$.

Schikuta considers grid-based clustering as a process of organising the value space surrounding points rather than organising the points themselves [Sch96]. Schikuta's approach revolves around the use of a structure similar to a hash table for representing the data. This is referred to as a grid structure and is developed by introducing points, one by one, into the structure. The grid structure initially consists of one large hypercube with an upper bound on the number of points it can contain. Once this upper bound is met, the hypercube is recursively split into two new hypercubes, demarking a division point on the scale for each

dimension, and new data are allocated to a hypercube according to their attribute values. This approach, like STING, also produces a hierarchical clustering result. The algorithm works by finding the densest hypercubes (those hypercubes with the largest ratio of number of points to volume), making them cluster centres and then examining their neighbours, adding them to the clusters in order of increasing density. Each time a hypercube is merged into a cluster, a corresponding level is created in the cluster hierarchy.

Although grid-based clustering can provide an efficient representation of a data set, there are several disadvantages. Since data points are summarised according to the contents of the cell in which they reside, individual points do not have equal importance. Also, the number of cells in which to partition the space grows exponentially with dimensionality. Hinneburg and Keim [HK98] also point out that clusters can potentially be split up over many grid cells resulting in the necessity of the expensive process of remerging them.

3.6 Model-based

Unlike the clustering methods described in the previous sections, clusters can be modelled by probability density functions. Although similar to density-based clustering, this approach instead considers density distribution functions rather than the raw densities. The most basic model-based clustering technique assumes that each cluster follows a Gaussian distribution, and that the data set can therefore be modelled as a mixture of Gaussians. The parameters of the individual Gaussian mixture components, namely the mean, the variance, the mixing weights, and the number of components, can be estimated from the data.

The estimation of these parameters is known as the maximum-likelihood parameter estimation problem. Given a set of points (observations) $X = \{x_1, x_2,\ldots, x_n\}$, that are drawn from a set of unknown distributions $E = \{e_1, e_2,\ldots, e_k\}$, the density at point x_r with respect to distribution e_i is given by the density function $f_i(x_r \mid \theta)$ where θ is the set of unknown parameters. The likelihood of the parameters, given the input point data, is $L(\theta, \tau \mid X)$.

$$L(\theta,\tau \mid X) = \prod_{r=1}^{n}\sum_{i=1}^{k}\tau_r^i f(x_r \mid \theta) \tag{3.2}$$

Where τ_r^i is the probability that point x_r belongs to distribution e_i.

The most common method of maximising the likelihood function is via the Expectation-Maximisation (EM) algorithm [DLR77]. The algorithm has two steps and begins by initially estimating the set of parameters and iteratively rescoring the input data accordingly. The score of a data point represents the likelihood of it belonging to a particular component of the mixture model. In the second step, the algorithm updates the set of parameters to increase their likelihood given the data. These two steps are repeated until the model converges to a local maximum of the likelihood function. Data points that are allocated the same mixture component are deemed to be in the same cluster.

While model-based clustering can outperform single-link and K-means algorithms, it can exhibit high time and space complexity when the mixture parameters are left unconstrained. Another disadvantage is that the user is left to specify the number of clusters [Fas99].

3.7 Conclusions

Clustering techniques can be classified according to six categories: hierarchical, partitional, density-based, graph-theoretic, grid-based and model-based. It is evident these categories are interrelated. For example, bisecting K-means is a partitional algorithm but it can also build a cluster hierarchy; the most common graph-theoretic techniques (single-, average- and complete-link) are also predisposed to produce cluster hierarchies. Also, model-based techniques formally use probability density functions (PDFs) and can therefore be considered as density-based.

An important issue concerning clustering algorithms is in choosing the correct type of algorithm to apply in different situations. For example, if the data are of high cardinality and dimensionality, a partitional algorithm might be appropriate for faster computation; if the data contains clusters of varying shapes, then a graph-based method might be more appropriate. However, the common traits among the different clustering methods and the ability of some methods to do better than others, has naturally led to the exploration of hybrid clustering algorithms. The continuing research into such algorithms is looking promising for creating more general purpose techniques with higher efficiency that are able to support interactive applications. In Chapter 5, a new hybrid clustering algorithm, borrowing from graph- and density-based approaches is described.

In the context of information visualisation, clustering allows for a richer representation of data. Clustering results can be in the form of interactive dendrograms or scatterplots (in the 2-d or 3-d case) where latent structure in the data is made clearly visible and interpretable.

Also, the ability of clustering algorithms to decompose a data set into a smaller number of significant units is also advantageous to dimension reduction as demonstrated in Chapters 4 and 5. More generally, individual clustering algorithms can be considered as computational tools that analysts can select according to their circumstances. If these algorithms are packaged into an easily accessible toolbox then it would allow their flexible application and potentially their combination for efficient hybrid clustering and dimension reduction solutions. In the next chapter, dimension reduction techniques will be described and it is shown that by reducing data dimensionality, latent structure can be made more readily visible.

4. Dimension reduction

As shown in the previous chapter, the cardinality of data, i.e. the number of individual data items, can be reduced to a smaller number of distinct clusters. This reduced representation makes understanding, manipulating and visualising the data easier. However, there exists an orthogonal approach to acquiring a reduced representation of the data and this is achieved by reducing dimensionality.

Consider a data set $X = \{x_1, x_2, \ldots, x_n\}$ consisting of n data items, each represented as a vector of d variables, observations or measurements, $x_i = [a_{1,i}, a_{2,i}, \ldots, a_{d,i}]^T$. It is common to consider d as dimensionality thus providing a set of axes that define the data space. One of the most effective and scalable ways to graphically present an overview of data is via a scatterplot because it can plot all of the data against two or three axes when $d \leq 3$. As demonstrated in Section 2.3.1, point patterns in scatterplots such as clusters can visibly stand out as individual perceptual units. The scatterplot brings out Gestalt qualities and therefore clustering is carried out automatically by the human visual system. However, when $d > 3$, data cannot be directly depicted as points on a scatterplot unless the number of remaining dimensions is sufficiently small enough to be encoded into retinal variables such as shape, colour, size and orientation of glyphs [CMS99]. A common approach in exploratory data analysis is to produce a matrix of 2-d scatterplots from all possible and unique pairs of dimensions [BC87] but the drawback here is that the number of scatterplots is equal to $d(d - 1)/2$ and therefore can quickly become overwhelming.

In such cases it is desirable to reduce the dimensionality and this can be achieved by *feature selection* or *feature extraction*. These are terms borrowed from the nomenclature of cluster analysis [JMF99] but they are directly relevant to the process of dimension reduction. When a scientist is making observations of an experiment, deciding which measurements to record is a manual form of feature selection, however, when the number of possible variables is large and their interrelationships are unclear, this task becomes more complicated. Feature selection is the process of filtering out dimensions (features) that are deemed redundant or irrelevant. If two dimensions are highly correlated then one might be redundant; they might both refer to a single higher level observation and therefore one dimension could be used in their place. In the context of text mining where each unique word can be considered as a dimension, redundancy is dealt with by replacing content-bearing words by their word stems or synonyms. For example the terms *visualise* and *visualisation* might be replaced by *visual*.

The text mining analogy also provides a good example of filtering out irrelevant dimensions. Stop words (articles and connectives) are treated as irrelevant noise – they are deemed not to bear any content – and are therefore removed [Sal71]. Another method of feature selection, illustrated in Section 4.3.3, is based upon clustering dimensions according to the correlation coefficients of their observations and visualising the results. In this case, the user can see which dimensions are potentially redundant and remove them from the analysis.

Feature selection is usually carried out as a pre-processing stage to reduce dimensionality and therefore make data more manageable for time and space intensive processes such as those involved in text mining and information retrieval. Feature extraction, on the other hand, is a process that often follows feature selection, and involves transforming a set of dimensions into a smaller set of derived dimensions. For example, Principal Component Analysis (see Section 4.1.1) is a type of feature extraction where the resulting dimensions are derived from a linear combination of the original dimensions. In Section 4.3.3, it is shown that dimension reduction tasks commonly encountered in the field of investigative psychology consist of a cyclic process of feature selection and extraction. Here, the two processes feed into each other as the analyst forms, tests and refines hypotheses on the relationships between variables and between the data items composed of the variables.

Dimension reduction is primarily concerned with finding a smaller data space (or *low-dimensional embedding*) that contains as much information as possible from the original space. Regarding the author's research, it is desirable to reduce the number of dimensions to two and therefore accommodate visualisation of data via scatterplots so that latent structure is pronounced. This chapter describes some of the most prominent techniques in dimension reduction. Most of the techniques are accompanied with examples of their salient properties as observed from the author's experiments.

4.1 Projection techniques

One way of reducing dimensionality is to geometrically project data into a lower dimensional space. A simple example is shown in Figure 4.1 where points in a 3-dimensional Euclidean space have been projected onto a plane that is parallel to the z-x plane. Projection techniques typically produce an embedding space that is derived from a linear combination of the original dimensions. One such example is PCA where the projection plane is effectively rotated to a position where the projected points retain as much variance as possible. PCA and other projection techniques will be discussed in the following subsections.

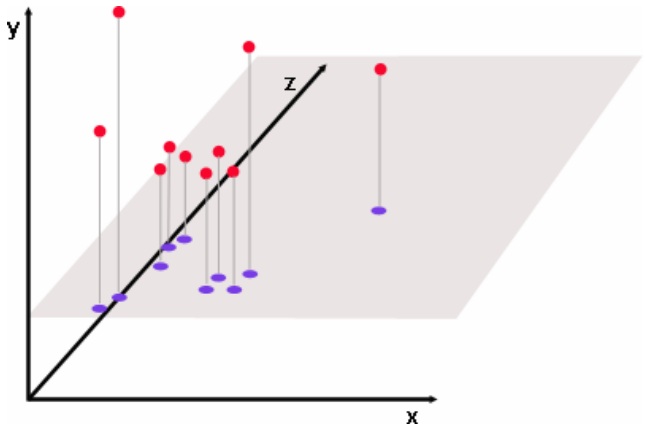


Figure 4.1 Projection of 3-dimensional points onto a plane.

4.1.1 Principal Component Analysis (PCA)

PCA is a classical statistical method and one of the most widely used projection techniques for dimension reduction [ED91]. It linearly combines correlated dimensions to produce a smaller set of uncorrelated dimensions under the premise that such correlations indicate redundant dimensions. The resulting derived axes are called *principal components*. The first principal component corresponds to the direction of greatest variance in the data and provides the first axis onto which the data can be projected. The second principal component is orthogonal to the first and the third is orthogonal to the first and second, and so on, progressively accounting for as much variance in the data as possible. For visualisation purposes, it is common to project the data onto the first two principal components thus providing a 2-d scatterplot that maximally preserves variance.

PCA is carried out via an eigenanalysis as follows. Suppose a data set is in the form of a population of vectors X where:

$$X = (x_1, x_2, ...x_n)^T \tag{4.1}$$

With mean given by:

$$\mu_x = E\{x_i\} \tag{4.2}$$

and the symmetric covariance matrix denoted as:

$$C_x = E\left\{(x_i - \mu_x)(x_i - \mu_x)^T\right\} \tag{4.3}$$

Elements of the covariance matrix are denoted $c_{i,j}$ and represent the covariance between variables *i* and *j* where $c_{i,i}$ is the variance of variable *i* – the amount of spread around its mean value. The projection space or *orthogonal basis* can then be determined by finding the eigenvectors e_i and corresponding scalar eigenvalues λ_i of the covariance matrix. These values are solutions of the following equation:

$$C_x e_i = \lambda_i e_i \quad i = 1,2,...,n \tag{4.4}$$

and can be found by solving the characteristic equation:

$$\left|C_x - \lambda I\right| = 0 \tag{4.5}$$

Where *I* is the identity matrix and |.| denotes the determinant.

Given a data vector *X* and a matrix *A* representing the eigenvectors as the rows, the data vector's projection coordinates *y* are obtained by the following equation:

$$y = A(X - \mu_x) \tag{4.6}$$

An eigenvalue is proportional to the amount of variance in the data set along the direction of the corresponding eigenvector. Thus, by composing the matrix *A* from the eigenvectors defined by the two highest eigenvalues, the 2-d projection of the data that maximally preserves variance is obtained.

However, there are two important drawbacks of PCA. The first is that in maximising variance as a global condition, interesting structure in the data that does not dominate the overall variance can be hidden. Figure 4.2 provides an example. The data in the figure is comprised of 10,000 3-d objects forming a 'swiss roll' shaped distribution. Since the greatest spread of the data is along the breadth of the swiss roll, the PCA projection onto the first two principal components provides a rather uninformative view. It is only when a cross-section of the data is taken, so that the greatest variance is across the diameter of the swiss roll, that PCA depicts a more informative view.

This issue has been indirectly addressed by Roweis and Saul who developed a dimension reduction technique called *locally linear embedding* [RS00]. Rather than trying to reduce dimensionality according to overall variance, samples are taken throughout the data and the localised regions of the samples are used to determine individual low-dimensional subspaces. When these local projections are patched together they can potentially depict global non-linear structure.

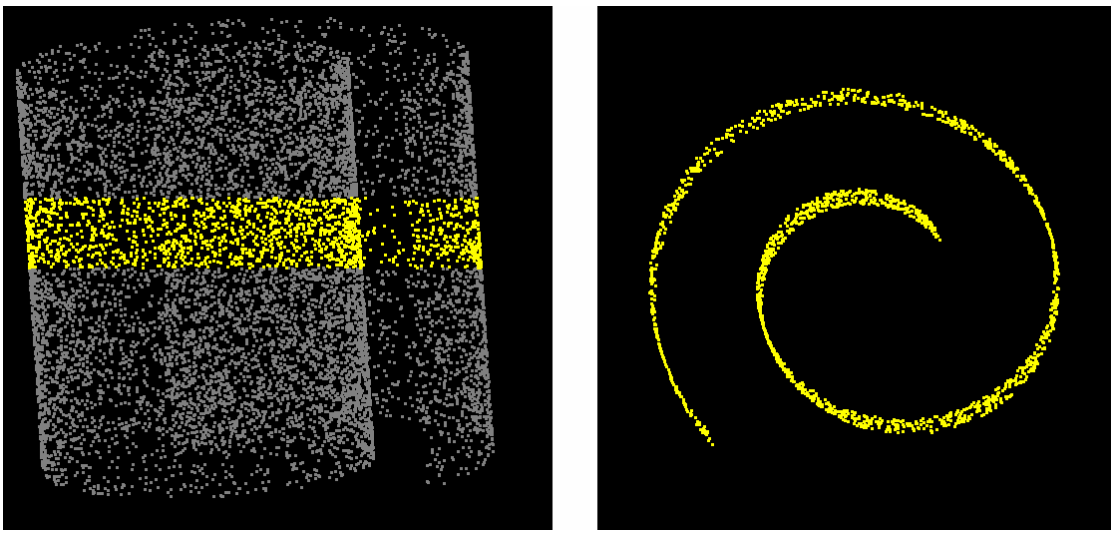


Figure 4.2 PCA projections of a 3-dimensional swiss roll-shaped data set. The image on the left is the projection of the whole set. The image on the right is a PCA projection of the highlighted cross-section of the image on the left. These projections were generated by the author's HIVE software.

Unfortunately, even a minority of outliers in the data can cause such a 'distraction' as shown in figure 4.3. Though, one way of overcoming this problem was proposed by Koren and Carmel [KC03] who developed a weighted PCA scheme. Here, pairwise distances are normalised in such a way that larger distances are less dominant in determining the projection.

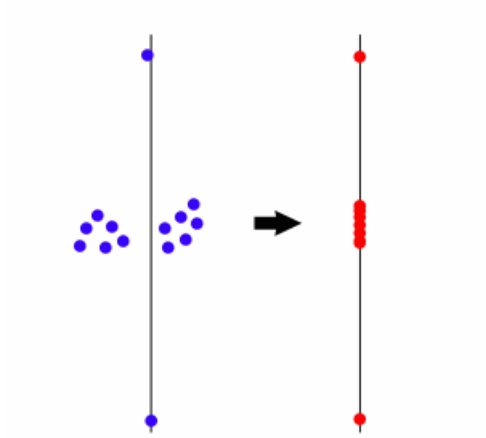


Figure 4.3 Since the first principal component is the direction of greatest variance in the data, the outliers shown in blue on the left, dominate the regression. This results in the projection onto the principal component (right) where possibly significant structure, such as the two clusters shown, has been lost.

The second drawback of PCA is due to the fact that the low-dimensional space is derived from a linear combination of the original dimensions. This means that unless any significant structure in the data lies upon a linear manifold, then it will not be adequately represented by PCA. These problems are also inherent in similar eigenanalysis methods such as *singular value decomposition* and *factor analysis*.

An advantage of PCA is that a neural network implementation can be quite fast. To train the network requires $O(LDP)$ time where $L >> N$ is the number of epochs (training iterations), D is the dimensionality of the training data and P is the dimensionality of the projection space. To produce the projection takes $O(NDP)$ where N is the cardinality of the data [Oja82].

4.1.2 Singular Value Decomposition (SVD)

Another well-known projection technique for dimension reduction is Singular Value Decomposition. The SVD of a rectangular data matrix X with n rows and d columns is obtained by its decomposition into three special matrices:

$$X = U_{[n\times r]} \cdot S_{[r\times r]} \cdot V^{T}_{[r\times d]} \tag{4.7}$$

Where U and V are composed of the left and right *singular vectors* of X respectively and have orthonormal columns. S is a square diagonal matrix and contains the *singular values*, the number of which is determined by the rank r of X.

A singular value is proportional to the amount of variance in the data set along the direction of the corresponding singular vector and therefore SVD is similar to PCA in that the data can be projected onto a lower, *k*-dimensional space by sorting the singular values (PCA eigenvalues) in descending order and taking the first *k* corresponding singular vectors (PCA eigenvectors). This results in a new decomposition showing the best least-squares-fit to *X*.

$$\tilde{X} = \tilde{U}_{[n\times k]} \cdot \tilde{S}_{[k\times k]} \cdot \tilde{V}^{T}_{[k\times d]} \tag{4.8}$$

For example, if $k = 2$ then *U* contains the 2-dimensional coordinates of the items (represented by rows) in the data set that minimise the sum of squares of projection errors. Like PCA, this can be thought of as rotating a plane in space to maximise the variance of the projected points.

SVD was employed for *Latent Semantic Analysis* (LSA) by Deerwester et al. in reducing the dimensionality of document collections to improve retrieval [DDF*90]. In this case the input is a term-by-document matrix, resulting in the left singular vectors being the coordinates of terms, and the right singular vectors representing the coordinates of the documents in k-dimensional space. Deerwester et al. provided an example where $k = 2$, using a scatter plot to show how the proximity of terms and documents exposes latent relationships in the data due to second and higher-order term co-occurrences.

SVD suffers from the same drawbacks as PCA, namely non-linear structure can elude analysis and outliers can hamper the detection of potentially interesting projections. SVD is also computationally intensive, running in $O(N^2D)$ time where *N* and *D* are the cardinality and dimensionality of the input data [LG03]. However, if the input data are sparse then the time complexity can be reduced to $O(NcD)$ time where *c* is the average number of non-zero entries in the input matrix [BM01].

4.1.3 Projection Pursuit

While the goal of PCA and SVD is to find a projection based upon the directions of greatest variance, the goal of Projection Pursuit is to find low-dimensional projections that optimise a different *projection index*. The projection index defines the "interest" of a direction and is typically a measure of departure from a Gaussian density. This is because the standard normal distribution does not contain much structure and is therefore not considered interesting in this context.

The negative Shannon entropy is commonly used as the projection index [Hub85] since it is minimised by the Gaussian distribution. Given a variable *x* with a probability density function *f*, its negative Shannon entropy is given by:

$$Q(x) = \int f(x) \log f(x) d(x) \tag{4.9}$$

When the projection space is 2-dimensional the outcome of projection pursuit is one or more static views portraying potentially interesting structure. However, Cook et al. [CBCH95] realised that these disparate views suffered from a lack of context. Inspired by Asimov's *Grand Tour* [Asi85], Cook et al. remedied this in the development of a technique of smoothly animating the transition from one projection to the next by interpolating a series of intermediate projections. They called this technique a *projection pursuit guided tour*. This process can be likened to smoothly rotating a 2-d viewing plane so that a salient aspect of structure is always visible in the projected point pattern, albeit from different angles.

While this approach helps alleviate the problem of not being able to expose any non-linear structure, which is common to all linear dimension reduction methods, projection pursuit still suffers from another drawback due to the necessity of *sphering* [CBCH95]. Sphering is a pre-processing step where data are conditioned to remove any effects of location (mean) and scale (variance) on the search for the projection pursuit index. If this is not carried out then differences between the distribution of variables with respect to location and scale might dominate other structure. However, as pointed out by Cook et al., sphering changes the *shape* of the data and as a result the projected views might present or elide structure as an artefact of this conditioning.

4.1.4 Random Projection (RP)

First proposed by Kaski [Kas98], Random Projection or Random Mapping, is one of the simplest dimension reduction techniques. In contrast to the techniques described above, RP does not use a *measure of interest* such as variance to identify a *good* projection [FB03]. It simply projects data through the origin onto a random subspace. Given a data matrix X with n rows and d columns, and a random matrix R, the projection onto a lower k-dimensional subspace is achieved by the following matrix multiplication (Equation 4.10):

$$A_{[n,k]} = X_{[n,d]} \cdot R_{[d,k]} \tag{4.10}$$

Where A is the matrix in which the rows are the k-dimensional coordinates of the input data.

Kaski showed that RP approximately preserves the mutual similarities in the original data, though the amount of distortion in the similarities is inversely proportional to $(d - k)$. This is based upon the Johnson-Lindenstrauss lemma [JL84] that states for n points in a d-

dimensional Euclidean space, there exists a k-dimensional projection where $k \geq O(\varepsilon^{-2} \log n)$, such that similarities will not be distorted more than a factor of $(1 \pm \varepsilon)$ with $0 < \varepsilon < 1$. Figure 4.4 shows a comparison between PCA and RP in projecting a 3-d cube. It can be seen that PCA preserves the similarities relatively well, while RP introduces some distortion.

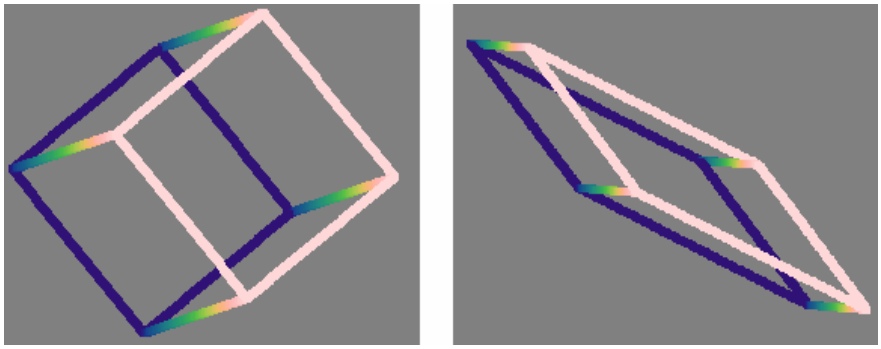


Figure 4.4 A PCA projection of a 3-d cube (left). A random projection of the cube distorts mutual similarities (right). Both of the above projections were produced by the HIVE software [RC03a, RC03b].

The susceptibility of RP to distorting mutual similarities led Kohonen et al. to use it as a pre-processing stage in the visualisation of a massive document collection [KKL*00]. Rather than attempting to reduce the dimensionality, which is in tens of thousands for such a corpus, down to two for visualisation, the authors opted to initially reduce it to several hundred so that distortion would be lessened. This compressed representation of the data then lessened the burden on a more computationally intensive process for further reduction and subsequent visualisation (see Section 5.2).

It should be mentioned that the distortion introduced by RP is not without its benefits, as Dasgupta discovered [Das00]. Dasgupta showed that the shape of clusters in a high-dimensional space is made more spherical after RP while the extent of their separation is maintained. This therefore suggests that random projection might be an appropriate pre-processing step for clustering algorithms such as K-means and model-based techniques that are naturally predisposed to finding spherical clusters. Dasgupta combined RP and the EM algorithm (see Section 3.6) and the results showed that the hybrid algorithm performed better than EM alone.

The major advantage of RP is that it is computationally very simple because it relies on a simple matrix multiplication. The random matrix R can be created in $O(dk)$ time and the projection of the data matrix X onto the k-dimensional subspace is $O(dkn)$. However, as with SVD (see Section 4.1.2) if X and/or R is sparse, then the time complexity can be reduced to

$O(ckn)$ where c is the average number of non-zero entries [BM01]. Even if X is not sparse, one can employ a reasonably sparse random matrix such as that suggested by Achlioptas [Ach01]:

$$r_{i,j} = \sqrt{3} \cdot \begin{cases} +1 & with\ probability\ 1/6 \\ 0 & with\ probability\ 2/3 \\ -1 & with\ probability\ 1/6 \end{cases} \tag{4.11}$$

Here, the element r at the i^{th} row and j^{th} column of the random matrix is assigned an integer value according to the given probabilities. With this random matrix, further time can be saved if computations are carried out using integer arithmetic.

4.1.5 FastMap

In 1995 Faloutsos and Lin proposed a very simple yet effective dimension reduction algorithm called *FastMap* [FL95]. Its name is taken from its achievement of "*a fast mapping of objects into points, so that distances are preserved well*." Like Random Mapping, FastMap does not explicitly try to optimise a measure of *interest* in a projection, but it does work in a similar, albeit simpler way to PCA and SVD.

The algorithm starts by identifying two objects in the data set that are relatively far apart in the original D-dimensional space. These items are referred to as *pivot objects* (O_a, O_b) and the line that passes through them is taken as the first axis onto which the remainder of the data set is initially projected. This projection is achieved by the Cosine Law where the triangle formed by the pivot objects and a data object O_i allows for that object's 1-d coordinate to be solved. See Figure 4.5.

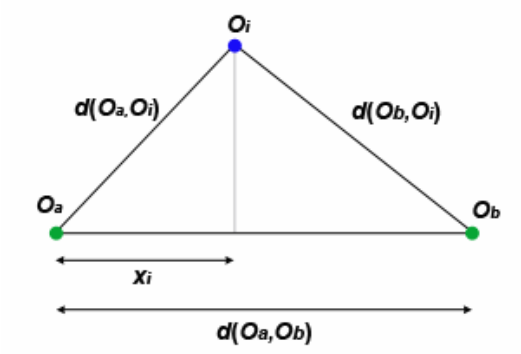


Figure 4.5 Object O_i is projected using the Cosine Law onto the line passing through the pivot objects O_a and O_b.

From Figure 4.5 it can be seen that the 1-d coordinate of object O_i is given by x_i. This value is derived from the Cosine Law as follows:

$$x_i = \frac{d(O_a, O_i)^2 + d(O_a, O_b)^2 - d(O_b, O_i)^2}{2d(O_a, O_b)} \tag{4.12}$$

Where $d(i,j)$ is the distance between objects i and j.

It is clear that the only information required to find the projection coordinates in the 1-d case is the distances between points. To extend this to the 2-d and eventually the k-d case, the authors realised that they had to be able to determine the distances upon consecutive orthogonal axes. This would enable the Cosine Law to determine the new projection coordinate.

To achieve this, the authors devised a way to measure distance on a (D-1) hyperplane H that is perpendicular to the line (O_a, O_b). Let O_i' denote object O_i when it is projected onto H, and $d'(O_i', O_j')$ represent the distance between two objects on H. When this distance function is used to find a second set of pivots, thus defining a second line orthogonal to the first (O_a, O_b), then it can be used to provide the second projection coordinate by Equation 4.12, substituting $d(O_i, O_j)$ with $d'(O_i', O_j')$. Figure 4.6 illustrates this reasoning.

By Pythagoras' theorem, $d'(O_i', O_j')$ is calculated as follows:

$$(d'(O_i', O_j'))^2 = (d(O_i, O_j))^2 - (x_i - x_j)^2 \tag{4.13}$$

This method can be extended to find k-d projections by recursively calculating the distances on consecutive orthogonal hyperplanes and applying the Cosine Law to find the projection coordinates.

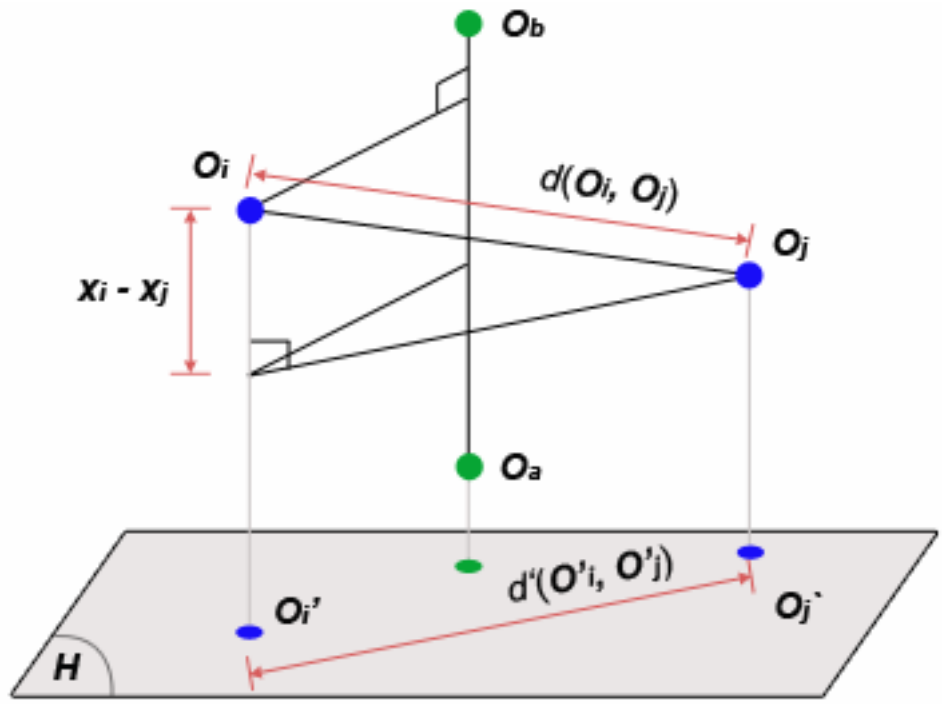


Figure 4.6 Objects O_i and O_j are projected onto the hyperplane H, perpendicular to the line through O_a and O_b.

Dimension reduction using FastMap can be achieved in $O(nk)$ time, where n is the number of items in the data set and k is the desired number of target dimensions. In comparison with a non-linear technique called *Multidimensional Scaling* (MDS), the authors also showed that FastMap produced a layout of almost equal quality in terms of layout stress – a measure of the residual sum of error. In plotting stress against time, the authors argue that the ideal scenario is to achieve zero stress in zero time and therefore the closer the trace to the origin, the better [FL95]. From this it was shown that FastMap's trace is indeed closer to this ideal origin than MDS and achieves almost an order of magnitude speed increase while maintaining comparable output quality. MDS and stress measures will be discussed more in Section 4.3.

The dimension reduction algorithms discussed above rely on a linear combination of the original dimensions to derive a projection basis. For this reason, they are commonly referred to as linear layout algorithms. While this approach can often be effective in representing data, potentially interesting non-linear structure might be lost because the linear combination of dimensions is not adequate for capturing non-linear relationships in the data. In the following sections several non-linear dimension reduction techniques will be discussed.

4.2 Kohonen's Self-Organising Feature Map

The inspiration for the Self-Organising Feature Map (SOM) comes from the topographic maps in the mammalian brain [RMS92], where closely related sensory stimuli activate topologically close regions in the brain. The SOM mimics this natural mapping by activating *neurons,* placed in a 2- or 3-d regular grid, according the input patterns that are presented. This grid (or map) is an Artificial Neural Network (ANN) and each neuron is represented by a reference vector that is of equal dimensionality to the input pattern vectors.

The SOM is an example of *competitive learning* and is trained in an unsupervised manner by presenting the input data $x \in \Re^n$ to the map of reference vectors $m_i \in \Re^n$. Reference vectors compete with each other to be allocated each input pattern and this is usually determined by the lowest Euclidean distance. When the winner or *best matching unit* is found, it is adjusted, along with neighbouring reference vectors, to be closer to the input. This representation is gradually refined by the learning process until the map provides an ordered non-linear regression of the reference vectors into the data space. The reference vectors m_i are updated at training step $t + 1$ by the following function.

$$m_i(t+1) = m_i(t) + h_{ci}(t)\left[x(t) - m_i(t)\right] \tag{4.14}$$

$h_{ci}(t)$ is a symmetric, monotonically decreasing function of the distance between the winning reference vector c and neighbour m_i on the map and is typically Gaussian. The outcome of this function is that the closer neighbours are to the winning vector, the more similar they will be made to the input. The function $h_{ci}(t)$, known as the *neighbourhood function*, is defined below:

$$h_{ci}(t) = \alpha(t)\exp\left(-\frac{\|m_i - c\|^2}{2\sigma^2(t)}\right) \tag{4.15}$$

Where $0 < \alpha(t) < 1$ determines the learning rate which decreases monotonically with t resulting in smaller updates to the reference vectors with time.

This process prompts the evolution or *self-organising* of a topologically ordered map with each reference vector being representative of one or more closely related points within the training set. This training stage can be said to classify the data set because the topological regions on the map conform to groups of similar items in the data set. Once training is complete, new data can be assigned to a class by finding the closest reference vector.

The SOM is trained by repeatedly inputting all of the training data and updating the reference vectors accordingly. Each of these cycles is called an *epoch* which can be carried out in $O(ND)$ time where N is the number of training items and D is their dimensionality. The number of epochs is bounded by the SOM's learning rate α, but a smaller number is often heuristically chosen.

The SOM effectively maps the data into a 2- or 3-d layout that can provide a useful visualisation because the neurons can be depicted as graphical structures and the map can be partitioned into regions allowing one to scan the view and gain insight into the data space being represented. One can therefore think of the SOM as providing clustering and dimension reduction simultaneously. In an implementation by Lin et al. [LSM91], the SOM was applied to the visualisation and retrieval of text documents where summaries of cluster contents were used to label regions of the layout called *concept areas*. Lin et al.'s idea was to provide a *semantic* map of the document associations akin to the mental or psychological map in the human brain. In another application, Lagus et al. [LHKK96] likened the neural clusters to document *traps* or *bins* which could be checked as the SOM evolves over time to see if new interesting texts had arrived, in a similar way to consulting the in-box of an email application.

However, it should be noted that the discrete layout of the SOM elides the proportional (dis)similarities between individual items that would otherwise be apparent in a continuous spatial layout such as in a scatterplot. This is mainly due to the neighbourhood function used in the construction of the SOM. Only local areas are adjusted in training and, as a result, the global relationships are coarsely represented.

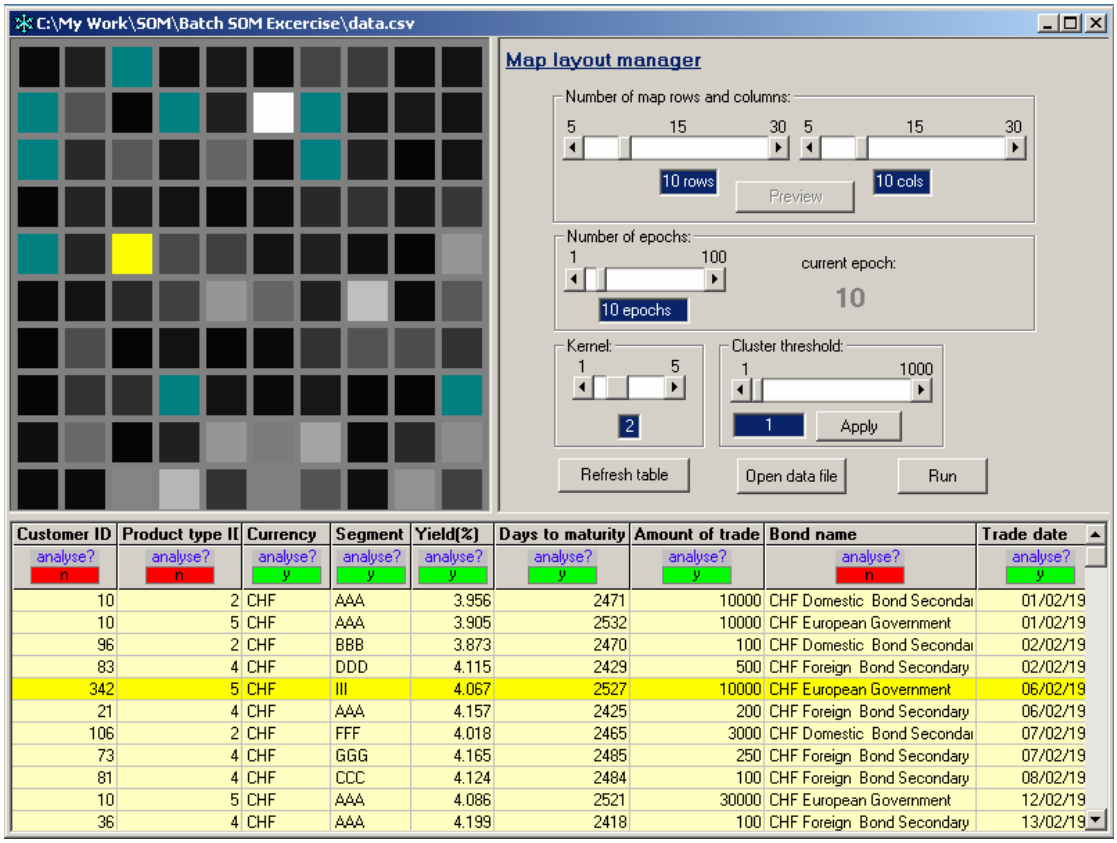
4.2.1 Batch-mode SOM

The learning strategy of the traditional SOM can be described as incremental because after each presentation of a training pattern, the weights of the best matching unit (BMU) and its neighbours are updated. Heskes et al. [HW96] describe the properties of a variant of the traditional SOM, called the batch-mode SOM. In this case the weights of the network are updated at the end of each algorithmic epoch rather than after the presentation of each training pattern. In this way, the batch-mode SOM is described as deterministic because running the algorithm repeatedly for the same initialisation of reference vectors will produce the same output, whereas the traditional SOM is stochastic in the arbitrary way in which the input patterns are presented to the network.

The batch-mode SOM, like its traditional counterpart, provides the same discrete visual structure depicting the competitive layer, but it can be quicker in converging to a solution

because there is a lesser number of weight updates. This makes it a promising algorithm to use for higher volumes of data [KKL*00].

This algorithm was implemented by the author to informally assess its performance and gain a feel for how the discrete visual output conveys information. The software was written in Microsoft Visual Basic 6.0 and the data set used was financial bond trade information consisting of 1000 records. In each record there are nine fields, and therefore these data are of relatively low dimensionality, but they still cannot be easily mapped directly onto a low-dimensional space for visualisation. Figure 4.7 shows a screen shot from the software.



Customer ID	Product type II	Currency	Segment	Yield(%)	Days to maturity	Amount of trade	Bond name	Trade date
analyse? n	analyse? n	analyse? y	analyse? y	analyse? y	analyse? y	analyse? y	analyse? n	analyse? y
10	2	CHF	AAA	3.956	2471	10000	CHF Domestic Bond Secondar	01/02/19
10	5	CHF	AAA	3.905	2532	10000	CHF European Government	01/02/19
96	2	CHF	BBB	3.873	2470	100	CHF Domestic Bond Secondar	02/02/19
83	4	CHF	DDD	4.115	2429	500	CHF Foreign Bond Secondary	02/02/19
342	5	CHF	III	4.067	2527	10000	CHF European Government	06/02/19
21	4	CHF	AAA	4.157	2425	200	CHF Foreign Bond Secondary	06/02/19
106	2	CHF	FFF	4.018	2465	3000	CHF Domestic Bond Secondar	07/02/19
73	4	CHF	GGG	4.165	2485	250	CHF Foreign Bond Secondary	07/02/19
81	4	CHF	CCC	4.124	2484	100	CHF Foreign Bond Secondary	08/02/19
10	5	CHF	AAA	4.086	2521	30000	CHF European Government	12/02/19
36	4	CHF	AAA	4.199	2418	100	CHF Foreign Bond Secondary	13/02/19

Figure 4.7 A screen shot from the author's batch-SOM implementation.

The discrete grid-like output of the batch SOM is shown in the top-left corner of the figure. Each square represents a neuron (potential cluster) and the lighter the colour, the more records it represents. The controls to the right of the grid are for setting parameters of the SOM. These include the size of the grid and the number of epochs. The control for *Kernel size* determines how many layers of neighbours to each winning neuron have their weights updated at the end of each epoch, and the *cluster threshold* control allows the view to elide neurons with a degree of membership below the set value. The table at the bottom of the view gives details of the members of the currently selected cluster, which is shown as the yellow

neuron on the layout. This table also allows the inclusion or exclusion of fields in the training of the SOM, i.e. this allows customisable feature selection. As shown, the red column headers represent features (dimensions) that have been excluded from the training phase, whereas the green column headers indicate features that have been taken into account.

From this exercise it is concluded that the batch-mode SOM could produce a rough intermediate layout for relatively large data sets. With the appropriate interaction controls, such as more advanced visual filtering, and better use of graphical structures representing the neurons, more information can be gleaned from this technique. The implementation of the batch-SOM is relatively simple, implying that it could be easily incorporated as a pre-processing stage or visualisation in an existing system. When the algorithm was run on the data set, it was found to converge to a solution much more quickly than a canonical spring model (see Section 4.4). However, the discrete layout produced can be hard to interpret even though it can capture non-linear relationships in the data – it tends to require more graphical embellishments to convey structural information because the position of clusters do not vary. In the author's implementation, this was alleviated somewhat by using colouring, and an interlinked table to *brush* clusters and subsequently reveal their contents.

The following section will, however, describe other methods where individual points can be represented in 2-d scatterplots, engaging Gestalt to reveal non-linear structure.

4.3 Multidimensional Scaling

Most of the dimension reduction techniques described above are applied in the process of feature extraction – they help identify, out of a number of pre-existing features (dimensions), the ones that convey salient information while removing or combining those that are irrelevant or redundant. There exists, however, a class of data which is not quantified by a set of features but by mutual (dis)similarities based upon people's judgements. Rather than being represented by *true distances* in a high-dimensional space, they can be considered as *comparative distances* in a space of unknown dimensionality [Tor52]. Such data are known as *proximity data* [She62] because only their subjective similarities are given, i.e. their mutual closeness or nearness is defined by some hitherto unknown mental model. Such data are commonly generated by psychophysical experimentation [BG97]. For example, a group of human subjects might be presented with a set of stimuli – some physical or abstract entities – and be asked to comment upon the similarities between all pairwise or triadic combinations. The goal then, is to find a Euclidean space of the minimum dimensionality into which the data can be

fitted so that the proportions of similarity are preserved. These dimensions might then be interpreted as the underlying principles of a theoretical mental or physiological model.

Multidimensional Scaling (MDS) is a tool that has evolved to transform proximity data into such a geometric representation. A good example of the efficacy of MDS was provided by Shepard in 1962 [She62]. The data from Shepard's experiment were acquired from a study by Ekman [Ekm54] where subjects were presented with 14 colours of varying hue – the stimuli. Each subject was presented with the colours, two at a time and asked to judge how similar they were on a five-step scale. The mean ratings were then transformed to lie on a scale between 0 (for "no similarity at all") and 1 (for "identity") in a 14 x 14 matrix. When Shepard fed these data into his MDS routine, a 2-d configuration of points was obtained that bore a striking resemblance to the familiar colour circle. The comparison is illustrated in figure 4.8.

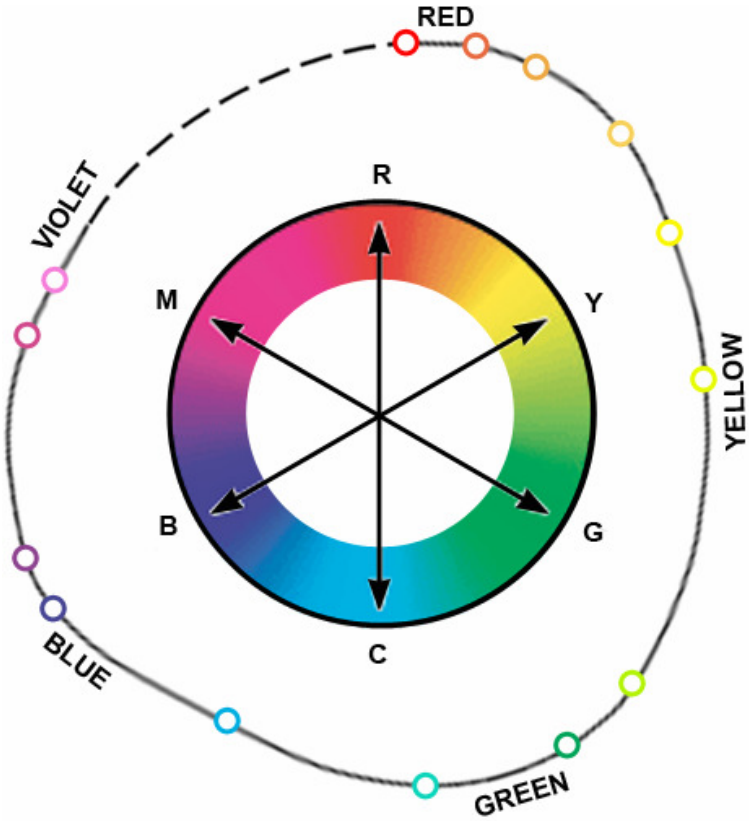


Figure 4.8 The coloured points bounding the figure, ranging from red to violet show the configuration obtained by Shepard's MDS algorithm when run on the colour-similarity data. It is clear that this closely follows the familiar colour circle (centre). The original figure [She62] has been rotated and flipped in this reproduction for ease of comparison.

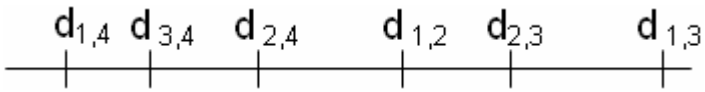
The colour circle was discovered by the physicist and mathematician Isaac Newton three centuries ago and contributes to today's theories on the psychological structure of colour. Opposing hues complement one another when mixed and one axis spans the perceived *warm* colours (from red to yellow) to the *cool* colours (from blue-green to blue-violet). The underlying physiological and psychological principles of this are well-understood, but it is intriguing that the MDS routine is able to reproduce the model just from the similarity judgments of human subjects.

In the above example, the interpretation of Shepard's point configuration was straightforward because of the prior knowledge of the colour circle. However, it is of interest to consider what an observer's interpretation would be if there was not such an *a priori* model. One might relate the vertical axis to the perceived difference between the warm and cool hues, but what about the horizontal axis? In this case there appears to be no definite cognitive or physical significance explained solely by this axis. Instead, it is a non-linear combination of both orthogonal axes that is important. The increasing wavelength from violet through to red is conveyed by this, hence the circle. This example suggests a warning in the general case of interpreting such MDS layouts. While it might be tempting to only attribute the axes to trivial or perhaps obvious factors, it should always be borne in mind that there might be more complex relationships represented by their combination.

4.3.1 Torgerson's classical metric MDS

One of the first MDS routines was proposed by Torgerson in 1952 [Tor52] and operated by transforming similarities into distances. This transformation was essential because similarities are not often symmetric, nor do they obey the triangular inequality. As a result, they do not fit into a Euclidean space. In the traditional *unidimensional* methods of analysis performed in psychophysical and psychometrics, subjects are required to specify similarity judgements between stimuli along one particular dimension such as brightness or weight. These similarities are then conditioned to lie upon a scale to reflect the psychological distance or difference between stimuli. As an example (provided by Torgerson), consider four stimuli S_1, S_2, S_3 and S_4 on the 1-d scale shown below:

Rather than scaling the stimuli to obtain a 1-d solution, Torgerson's multidimensional approach involved initially obtaining a scale of the similarities between all stimuli. So, for the four stimuli, a 1-d scale of the six inter-stimulus similarities might be as follows:



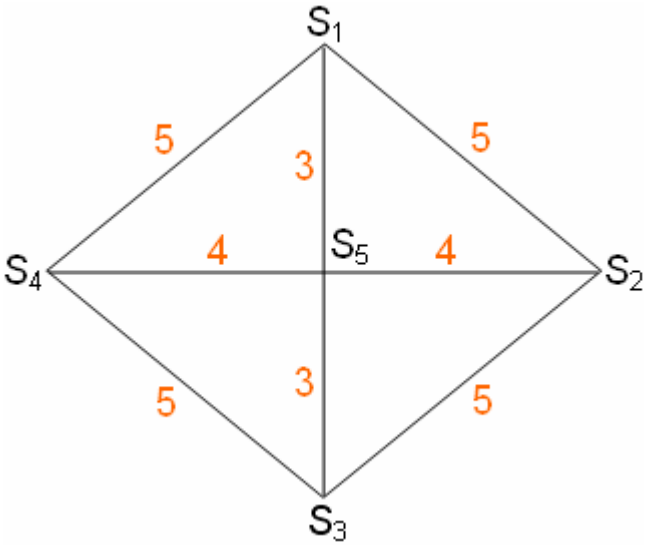
Torgerson pointed out that these comparative distances are not absolute distances. This is important because it is the absolute distances that are essential in finding the Euclidean space of the smallest dimensionality which might accurately represent the data. To obtain the distance d_{ij} between stimuli, Torgerson stated that a constant C must be added to the comparative distances h_{ij}:

$$d_{ij} = h_{ij} + C \tag{4.16}$$

He also stated that finding C is analogous to finding the true zero point of the scale of comparative distances, and that this would then permit the stimuli to be fitted by a Euclidean space of the smallest possible dimensionality. To continue with Torgerson's example, consider the comparative distances between a set of five stimuli:

$$h_{12} = 1, \quad h_{14} = 1, \quad h_{23} = 1, \quad h_{25} = 0, \quad h_{35} = -1,$$
$$h_{13} = 2, \quad h_{15} = -1, \quad h_{24} = 4, \quad h_{34} = 1, \quad h_{45} = 0.$$

In this case the additive constant required is 4, resulting in the stimuli fitting into a 2-d space as shown below:



If any number other than 4 was taken as C then a space of higher dimensionality would be required to fit the derived distances.

Having developed a routine for deriving absolute distances from similarities, Torgerson's MDS algorithm then drew upon Young and Householder's method for confirming whether data do indeed lie in a Euclidean space and if so, how to determine their dimensionality [YH38]. This is achieved by finding a matrix of lower rank than that which holds the distance information – a technique akin to PCA and SVD. In other words, once the Euclidean space is found in which the data fit, the space is rotated to produce a projection onto a lower number of *meaningful* axes.

Torgerson's approach was one of the first breakthroughs in multidimensional scaling and is known as *classical metric* MDS. The term metric is applied because the routine works solely on quantitative similarities. It is a virtue of MDS that the meaningful dimensionality of information can be uncovered from only a set of similarities. Recall from Section 4.1.5 that FastMap has the ability to produce a projection of data from a set of distances alone. While this provides an extremely quick reduction in dimensionality, it is worth noting that it cannot handle the proximity data for which MDS has evolved.

Apart from being computationally intensive, requiring $O(n^3)$ time for transforming similarities to distances alone, Torgerson's technique can be problematic when working with fallible data. Errors might be made in observing, recording and encoding subjects' similarity judgments and therefore the proportions of mutual similarity might evade a Euclidean space of suitably low dimensionality. This is because the routine seeks the absolute distances, conforming as closely as possible to the proportions of mutual similarities. The MDS methods described in the following subsections avoid this shortcoming.

4.3.2 Non-metric MDS

The second breakthrough in MDS was made by Shepard in 1962 [She62] whose algorithm is classified as non-metric [Krus64] because it can handle qualitative as well as quantitative measures of similarity. In embracing the analogy of similarity measures with representative physical distances, Shepard named such observations *proximity data* and his technique as *the analysis of proximities*. Shepard's approach treats input data types as ordinal by considering only their rank ordering when reducing their dimensionality. He showed that the rank order of mutual proximities is sufficient for metrically recovering a Euclidean configuration rather than Torgerson's method of first transforming quantitative similarities into absolute quantitative distances.

Shepard's algorithm starts by ordering the $N(N-1)/2$ proximities between N objects, from the smallest to the largest values. This produces a scale analogous to that of Torgerson's scale of comparative distances. The goal thereafter is to arrange N points in Euclidean space so that their mutual distances obey an inverse ranking of the proximities. For example, if objects A and B have a high mutual proximity, then their representative points in Euclidean space should have a low mutual distance. That is, the distances are constrained only to the extent that they have a monotonic relationship with the original proximities; the proportions of the distances do not have to comply exactly with those of the proximities. Thus, proximity is treated as an unknown monotonic function of distance and once transformed by this function, the data can be arranged in a Euclidean space.

To achieve a monotonic relationship, points are initially arranged on the vertices of an $(N-1)$-dimensional simplex centred at the origin and with edges of unit length. This ensures that all inter-point distances are initially equal to unity thus removing any bias from the final configuration. Also, when N points are placed in an $(N-1)$-dimensional space, they can be arranged to obtain any desired ordering of the mutual distances. These facts become clear if one imagines the case where $N = 3$, with each of the points lying on the vertices of an equilateral triangle – the 2-d simplex.

After the N points have been organised on the vertices of the simplex, the routine iteratively updates their positions by comparing the rank order of their distances to the rank order of their proximities, shrinking distances that are too large and stretching those that are too small. This produces a set of displacement vectors that are added to the point coordinates to move them closer to a monotonic relationship with the original proximities. Since the points reside in a relatively high dimensional space, they are free to move quickly to positions satisfying the monotonic requirement. However, as the goal of MDS is to reduce the

representative dimensionality of the input data, clearly the (N – 1)-d configuration must be *flattened* out to enable projection of the points into a space of lower dimensionality.

Shepard noted that dimension reduction is generally accompanied with an increase in variance of the point coordinates. In initialising the points to lie on the simplex vertices, the variance is at its minimum possible value because the inter-point distances are unity, thus to reduce the dimensionality this variance must be increased. This is accomplished by introducing a second set of displacement vectors, complementing those for attaining monotonicity. Rather than shrinking the larger distances and stretching the smaller distances, the opposite approach is required to increase variance and therefore force the configuration into a space of smaller dimensionality. That is, the smaller distances are further reduced while stretching out the larger distances. While the algorithm iteratively updates the point positions, the two sets of displacement vectors gradually balance out to the stage where the routine converges to a configuration of points where an optimal compromise between monotonicity and dimensionality is accomplished. However, the points still reside in a space of $(N - 1)$ dimensionality somewhat analogous to a plane in a 3-d space. To attain the final configuration in the desired number of dimensions, the points are rotated to their principal axes before being projected into the smaller space.

Shepard demonstrated how the monotonic function relating proximity to distance can also be recovered. When the original proximities between pairs of stimuli are plotted against the recovered pairwise distances in the low-d point configuration, the resulting trend shows the shape of the function. This graph is known today as a Shepard plot. An example is given in Figure 4.9. A set of 2-d data consisting of 300 points was converted into a set of proximities (specifically dissimilarities) by a Gaussian transformation based on that used by Shepard in his sequel paper [She62]. These points were then fed into the routine so that the following configuration and Shepard plot were produced:

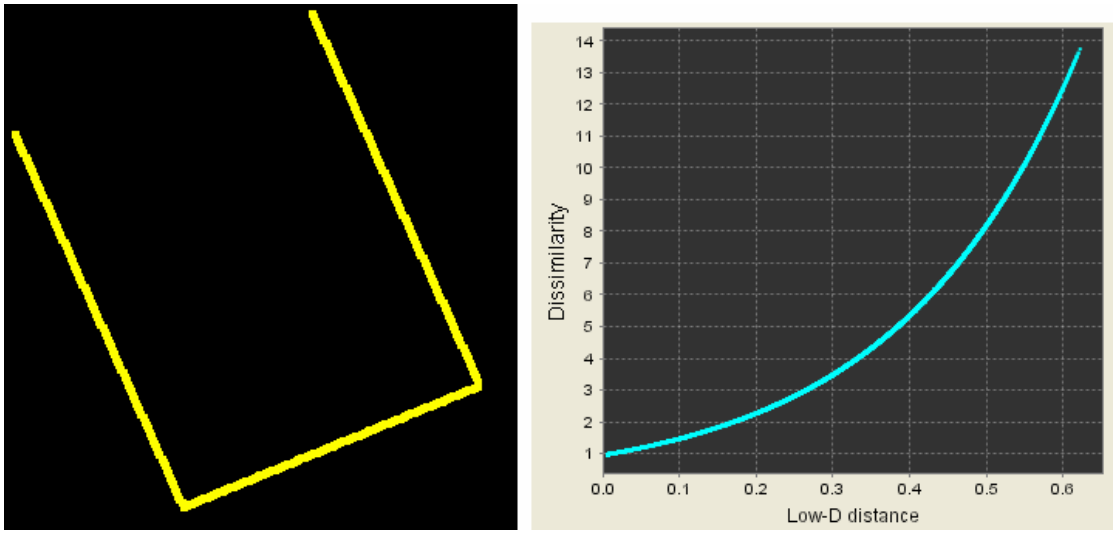


Figure 4.9 The point configuration on the left was recovered from the original 2-d data after they were transformed into proximities. The Shepard plot on the right shows the shape of the function used in the transformation. These images were produced in HIVE.

The function used to transform the original data into proximities is as follows:

$$s_{ij} = exp[1.4 * d_{ij}] \tag{4.17}$$

Where d_{ij} represents the Euclidean distance between points i and j. In real-life data, the function that allows proximity data to fit into a real Euclidean space can be of arbitrary shape, however, using the above technique, the function can still be recovered no matter what the shape. Note that if the original data had not been transformed, then the Shepard plot would show a straight 45 degree line representing a one-to-one relationship between the original inter-object distances and those recovered by the routine.

This MDS routine represented a breakthrough because it proved that quantitative distances could be recovered from qualitative (non-metric) proximity data and the monotonic function that relates proximities to the Euclidean distances could also be simultaneously recovered. Shepard's routine has the advantage over Torgerson's approach in that it can handle missing or erroneous data and it can be generalised to work with distance metrics other than Euclidean. However, its downfall is its computational complexity. The routine requires that $N(N-1)/2$ distances must be calculated over the $(N-1)$-d points for each update thus taking $O(N^3)$ time per iteration. The algorithm requires $O(N)$ iterations to converge resulting in an overall time complexity of $O(N^4)$ although the author has recently reworked Shepard's algorithm to attain convergence in $O(N^3)$ time. This is detailed in Section 5.4.

The criterion Shepard used for determining when to stop the iterative process, and project the data into a space of lower dimensionality, is the overall departure from monotonicity:

$$\delta = \frac{2\sum_{i=0}^{N-1}\sum_{j=i+1}^{N}[s_{ij} - s(d_{ij})]^2}{N(N-1)} \tag{4.18}$$

Where s_{ij} is the proximity of objects i and j and $s(d_{ij})$ is the proximity at the rank of the corresponding configuration distance between the same objects. This is the closest Shepard goes to providing a measure of the goodness of fit of the recovered configuration from the given proximities. In 1964, Kruskal took this further to define a measure that the MDS routine explicitly attempts to minimise [Krus64]. Instead of directly using the proximities, Kruskal defines the distances required to maintain the monotonic relationship with the proximities and takes the deviation of the current configuration distances from these ideal distances as a measure of fit. This measure of the goodness of fit is called *stress* and is defined as follows:

$$stress = \frac{\sqrt{\sum_{i<j}\left(d_{ij} - \hat{d}_{ij}\right)^2}}{\sum_{i<j} d_{ij}^2} \tag{4.19}$$

Stress is the residual sum of squares of the difference between the actual distance d_{ij} and the desired distance $\hat{d}_{ij}$. While it is invariant to rotation and translation of the configuration, it must also be normalised to make it invariant to uniform dilation.

Kruskal's version of the non-metric MDS algorithm employed a *steepest descent* routine where the goal of each successive iteration is to reduce the measured stress as much as possible. The routine ends when stress does not decrease further. While Kruskal's stress measure provides a more solid theoretical foundation for MDS, one disadvantage of this approach is due to the possibility that the steepest descent method might get stuck in a local minimum. The fact that stress does not reduce in a successive iteration does not necessarily mean that this is its lowest possible value; it might simply be resting in a *valley* in the curve of the stress function.

The time complexity of Kruskal's routine is $O(N^4)$ because stress takes $O(N^3)$ to compute when the dimensionality before projection is equal to $(N - 1)$ and approximately N iterations are needed to attain a local minimum of the stress function.

It is interesting to note that both Shepard's and Kruskal's approaches initially increase the dimensionality of the data to $(N - 1)$. The primary purpose of this is to quickly achieve monotonicity through the increased freedom of the points in this large space. As such, both methods rely on a linear projection technique such as PCA or SVD to collapse this representation into a space of the lower target dimensionality. While this last step is a linear transformation, the routine is still a non-linear technique in that it can represent non-linear relationships within the data.

4.3.3 MDS for feature selection

In the introduction to this chapter, the distinction between feature selection and feature extraction was provided. All of the examples of dimension reduction discussed hereto have described feature extraction which is the derivation of a set of features (dimensions) that are pertinent in succinctly representing the structure inherent in a set of data. However, MDS can be used for feature selection. That is, a method of selecting a subset of the original variables to use as dimensions for subsequent analysis of data.

Guttman [Gut68] developed a non-metric MDS routine called *smallest space analysis* (SSA), known today as *similarity structure analysis*. This is a form of MDS that became popular with practitioners in the fields of psychometrics and investigative psychology [Can85]. The example that will be provided here is concerned with the latter which is very closely related to the author's current occupation.

In the field of investigative psychology, various observations of criminal activity made at crime scenes are used to build behavioural models in order to profile offenders. Such observations are collectively called *modus operandi* (MO) and are treated as variables in a multivariate data set of crimes. These data are initially represented by a raw data matrix (RDM) where crimes are allocated to the rows and variables to the columns. In the first step of building a model of the typical MO for a particular type of crime, analysts apply SSA to the data to attain a layout of the variables, rather than a layout of the crimes. Thus the starting dimensionality is not the number of variables but the number of crimes. Formally, a data set of N items and d dimensions is represented by d vectors $A_i (0 \leq i < d)$ comprised of N elements $a_{ij} (0 \leq j < N)$ [ABK98]. SSA then places variables that are highly correlated close to each other while uncorrelated variables are placed further apart.

The output of this process is a scatterplot depicting the variables and the analyst manually partitions this plot into thematic regions. This can be thought of as a kind of spatial categorisation. For example, in the case of arson the variables might include the time of the offence, its location and the type of accelerant used. The underlying themes, represented by

groups of adjacent variables, might reflect whether the offence is manifested in the offender expressing him or herself, or whether it is instrumental for some personal gain. Figure 4.10 provides an example extracted from Canter and Fritzon [CF98]. The definition of these themes does, of course require diligent consideration of what the variables mean and their psychological significance. While it is rare for well-formed clusters to appear, the statistical properties such as co-occurrence and correlation are manifest in the MDS output in such a way that the psychological significance of their positions can be strongly explained and justified. It is generally also observed from such a layout that the centroid of the configuration is surrounded by the most frequent variables (usually the absence or presence of a variable at a crime scene is encoded as a Boolean value and therefore the sum of values for a particular variable is taken across the entire data set and considered as its frequency). Canter hypothesises that these high-frequency variables indicate the significance of the offences to the offenders.

Once such a layout and its partition have been obtained, analysts can select a subset of variables for further analysis such as MDS of the crimes where each point in the layout represents an offence. That is, in the first stage of analysis, MDS is employed for feature selection and then MDS is applied for feature extraction.

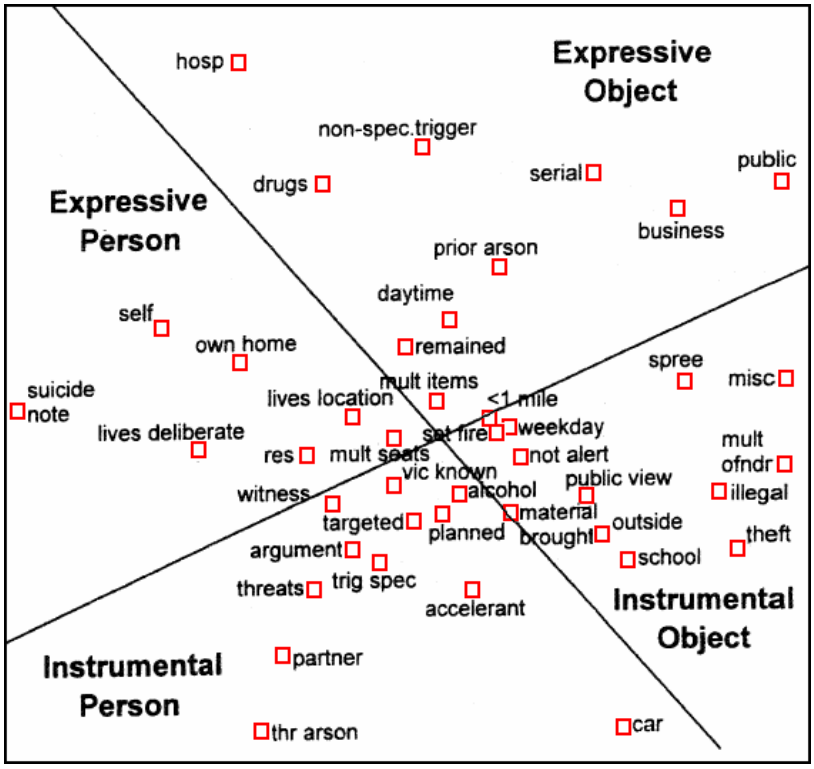


Figure 4.10 An SSA layout of variables representing the MOs of a set of arson offences.

This approach can also be applied to scientific data to identify redundant variables. Using MDS to layout variables essentially depicts their correlations – proximity in the layout is proportional to correlation between the variables concerned and if two variables are highly correlated then perhaps including just one of them would suffice in a representation of the data. To test this hypothesis, the author ran SSA on a data set gathered from an eScience project within the Equator Interdisciplinary Research Collaboration (www.equator.ac.uk). The eScience team set up a remote sensing probe at a frozen lake in the Antarctic, which transmits data including ice thickness, water temperature, UV radiation levels etc. to environmental scientists at the University of Nottingham. The aim of this is to learn about carbon cycling processes. The data set was composed of 2202 probe measurements, each consisting of 14 variables measured at five-minute intervals between 17th January 2003 and 31st January 2003. This was converted into CSV format before importing it into the author's HIVE software for analysis (discussed further in Chapter 7).

SSA was run on the variables of the data to gain the layout shown on the left frame of Figure 4.11. One would expect variables such as data-logger temperature and air temperature to be correlated and this is indeed the case, as can be seen from their proximity in the layout. From each group of highly correlated variables, one representative variable was selected resulting in a subset of 6 variables from the original 14. This 6-d representation of the data was then projected onto a 2-d layout using PCA as illustrated in the middle frame of the figure. This was then compared with a PCA projection of the full 14-d data set (right-hand frame of the figure). It can be seen that even though the number of variables has been reduced to less than half, the layouts are very similar indicating that there was indeed a high degree of redundancy.

Feature selection can also be achieved by applying clustering algorithms to the dimensions of a data set. Yang et al. [YPWR03] employ an agglomerative hierarchical clustering routine and visualise the result in their *radial space filling* (RSF) visualisation called InterRing [YWR02]. Yang et al. also show that dimension-clustering and filtering can facilitate the diligent ordering and spacing of dimensions in visualisations such as parallel coordinates [Ins85] to greatly improve their efficacy. Ankerst, Berchtold and Keim [ABK98] provide an extensive account of how to define similarity measures for dimension clustering. They also show that obtaining an optimal ordering of dimensions (according to similarity) is an *NP*-complete problem, although they provide heuristics for speeding up this operation. Another alternative for dimension-ordering and clustering is provided by Guo [Guo03]. In this case the *d*-dimensional data space is partitioned into *nested means* and the *maximal conditional entropy* of the partition cells is computed and used as a measure of dissimilarity.

Guo applies the minimum spanning tree in a graph-based clustering routine (see Section 3.4.1) to form the dimension clusters and derive an ordering.

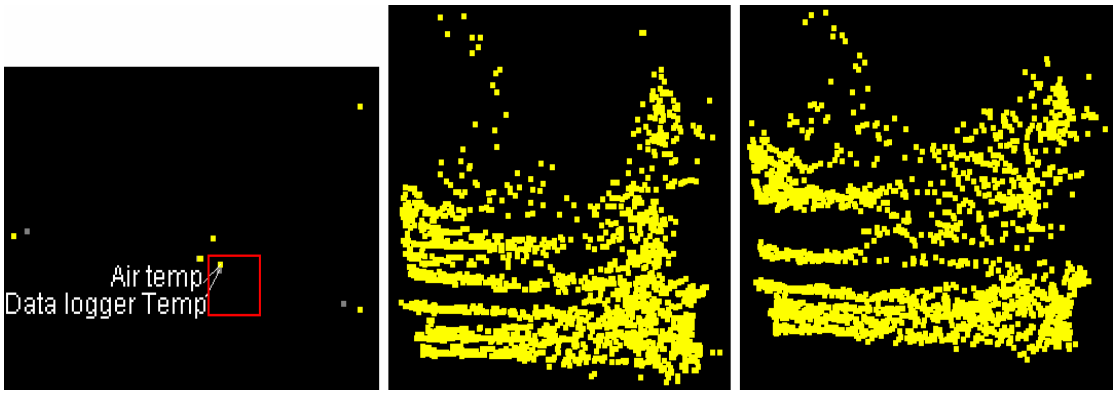


Figure 4.11 An SSA layout of the variables of a scientific data set is used to select a subset (highlighted in yellow) for subsequent dimension reduction (two left-hand frames). The projection of the full data set is shown on the right-hand frame.

4.4 Force-directed placement

For many years, researchers have been devising algorithms for the automatic layout of graphs given a set of vertices and edges [CT98]. Usually the goal is to obtain a graph where some predefined criteria is to be fulfilled. For example, in some cases it is desirable to minimise the number of edge-crossings and make the graph as near to planar as possible; also, it might be desirable to have the graph as symmetrical as possible or constrain the edges to be of unit length. Generally speaking, a graph-drawing algorithm reproduces a visual representation of a graph according to some desired aesthetic properties. A popular basis for such an algorithm is called *force-directed placement* (FDP) – a term coined by Fruchterman and Reingold [FR91] because the general technique is based upon a simulation of forces and motion in a physical system.

A seminal paper by Eades [Ead84] describes a heuristic technique for the aesthetic layout of general undirected graphs through the physical analogy of a system of steel rings connected by springs. The basic idea is that a graph, $G = (V, E)$, where V represents the vertices and E represents the edges, can have its vertices replaced by steel rings and its edges replaced by springs to represent a mechanical system. This system is initialised so that the vertices are placed in random positions and therefore the springs connecting them are stretched or compressed. When the system is *let go* the attractive and repulsive forces exerted by the springs move the system to a state of minimum energy or *equilibrium* (see Figure 4.12).

Such a system can be simulated using Hooke's law or Newton's law of motion, yielding a set of differential equations that can be solved numerically using a method such as Runge-Kutta [CDH92], although in this case, Eades defined his own formula for relating forces to the analogous springs. This technique was found to produce good results with regard to aesthetics of symmetry and uniform edge lengths in graph drawing but has since been applied in dimension reduction scenarios where the layout of high-dimensional objects in a low-dimensional space is the goal. Because the analogy of forces and springs is explicit, this type of algorithm is often referred to as a *spring model*.

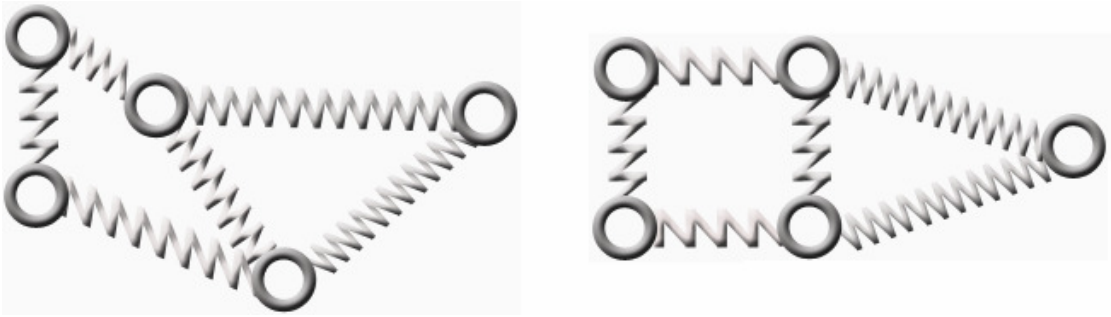


Figure 4.12 An illustration of Eades' concept of the spring model. The image on the left shows steel rings held in random positions causing the connecting springs to be stretched or compressed. The image on the right depicts the system in a state of minimal energy after the rings have been *let go* resulting in the springs reverting to their rest lengths. For clarity, only springs connecting adjacent rings are shown.

When applied to the low-dimensional embedding of a set of abstract data, this model can be considered as a combinatorial optimisation algorithm and is an example of the well-known *N-body* problem. The forces in the system are computed as being proportional to the difference between the high-dimensional (desired) distance and the low-dimensional (layout) distance and therefore a loss function can be derived which indicates the amount of energy in the system [Coh97, Cha96]. This loss function is related to Kruskal's stress function described in Section 4.3.2 and is a measure of the sum-of-squared errors of inter-object distances. Thus the objective of the spring model is to minimise the stress (Equation 4.20). It can be seen that the spring model is a form of metric MDS. It works directly upon the values of the dissimilarities (distances) to provide a layout.

$$stress = \frac{\sum_{i<j} (d_{ij} - g_{ij})^2}{\sum_{i<j} g_{ij}^2} \tag{4.20}$$

Where d_{ij} is the desired high-dimensional distance and g_{ij} is the current layout distance. It should be noted that the stress measure is widely used to describe the quality of layouts, however, it should be used with caution. A low stress value indicates a close fit of the low-dimensional layout to the high-dimensional space, but it states nothing of the layout's interpretability. In fact, from the author's experience even small differences in stress can reflect large differences between layouts often resulting in familiar or expected structure being hidden.

In its basic form, there are $N(N-1)/2$ pairwise object interactions to take into account and therefore it can be computationally infeasible when a large volume of data is to be considered, especially since it is usually implemented as an iterative algorithm. The number of iterations required for the system to reach a state of equilibrium tends to be proportional to N and therefore the overall time complexity could potentially be $O(N^3)$.

In a paper by Chalmers [Cha96] an algorithm is proposed in which stochastic sampling is employed to derive an array of neighbours V and an array of samples S for each data item. The sizes of both arrays are held constant. At the start of each iteration the sample arrays are filled with random items before calculating the forces between each item and only those items in the respective neighbour and sample sets. This bounds the number of distance calculations in each iteration of the algorithm to $N(|V| + |S|)$ and therefore results in overall time complexity of $O(N^2)$ in achieving equilibrium. At the end of each iteration, the neighbour sets are updated by replacing items that are further away with closer items that are in the sample set. That is, if a neighbour is further away from its parent item than the closest sample item, then the sample item replaces the neighbour. This improves the accuracy of the neighbours as the algorithm progresses. Chalmers' algorithm was one of the fastest non-linear dimension reduction techniques at the time of its publication.

The spring model can be used to provide visually intuitive views of a data set. Items that are similar are placed close together and items that are dissimilar are placed farther apart in a continuous fashion. This approach, as with MDS, provides an improved layout over that of the discrete SOM layout. It also has the advantage over the projection-based techniques in that it can uncover non-linear relationships within data. A user can perceive how similar or dissimilar groups or individual items are because the topology and the proportional inter-object distances are preserved to some extent. The simplicity of the spring model and its openness to heuristic improvements such as that of Chalmers make it a good component for hybrid dimension reduction algorithms, as will be seen in the next chapter.

A major difference between FDP and non-metric MDS is that FDP cannot yield a solution which is invariant under all monotonic transformations of the input data. While non-

metric MDS tries to minimise departure from monotonicity, the spring model attempts to recreate the distances perfectly and is therefore not readily applicable to proximity data where there exists no Euclidean space in which they would fit. Recall the example illustrated above in Figure 4.9. A data set consisting of 300 2-d points was transformed into proximity measures and non-metric MDS was shown to exactly recover the Euclidean configuration as well as the function relating proximity to Euclidean distance. The author applied Chalmers' spring model to the same data. The result is shown in Figure 4.13.

Although Chalmers' spring model produced a 2-d layout relatively faithful to the original 2-d configuration (see Figure 4.9), some distortion is evident. The corresponding Shepard plot implies the shape of the function but is not as clear as that in Figure 4.9 as produced by Shepard's non-metric MDS. The Shepard plot also shows that there is more discrepancy between the original data and the layout. This is evident in the roughness of the curve and is due to the distortion in the layout.

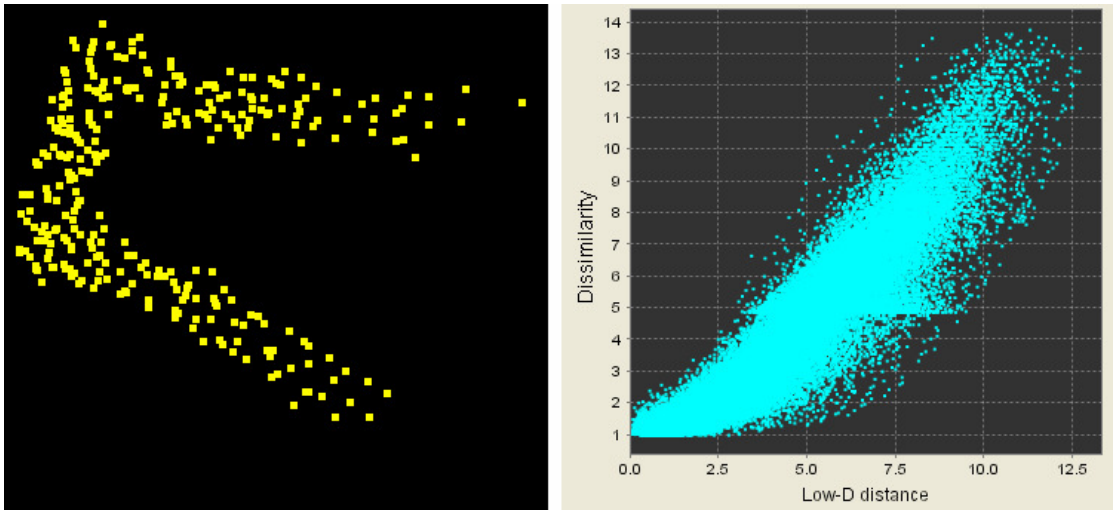


Figure 4.13 The same data set as in Figure 4.9 is fed into Chalmers' spring model and the layout is shown on the left. The corresponding Shepard plot is shown on the right.

4.5 Conclusions

This chapter has discussed a number of techniques for dimension reduction, many of which have been implemented by the author to help clarify their operation and gain a deeper understanding of their behaviour. Several of the figures were produced by a system called HIVE that was developed by the author to explore, combine and utilise such algorithms. HIVE will be the focus of later chapters where its utility will be demonstrated in the creation and evaluation of several novel algorithms for dimension reduction and data clustering.

Dimension reduction can be classified according to two dichotomies. The first distinguishes between feature extraction, where a new smaller set of uncorrelated dimensions is derived from the original data, and feature selection, where the dimensions are carefully filtered to reduce redundancy in the data. The second dichotomy is a sub-class of feature extraction and includes the family of projection techniques in which a linear combination of the variables provides a new set of derived dimensions. An advantage of this approach is that the resulting axes are easier to interpret because of their linear relationship and the routines producing them tend to be quite fast. The second class of this dichotomy pertains to non-linear methods which tend to be more powerful than the projection techniques because they can expose latent non-linear relationships in data. This latter class does, however, tend to provide more computationally expensive solutions and the reduced set of dimensions can be harder to interpret. Note that while the two (or three) geometric dimensions of a layout do not mean anything in themselves, non-linear techniques can still be regarded as a type feature extraction – the onus is put on the user to look at any resulting patterns such as trends and clusters and try to figure out the underlying meaningful dimensions of the data.

While this taxonomy (see Figure 4.14) of dimension reduction techniques is by no means definitive, it has served its purpose in providing a structured account of the prominent routines found in the literature.

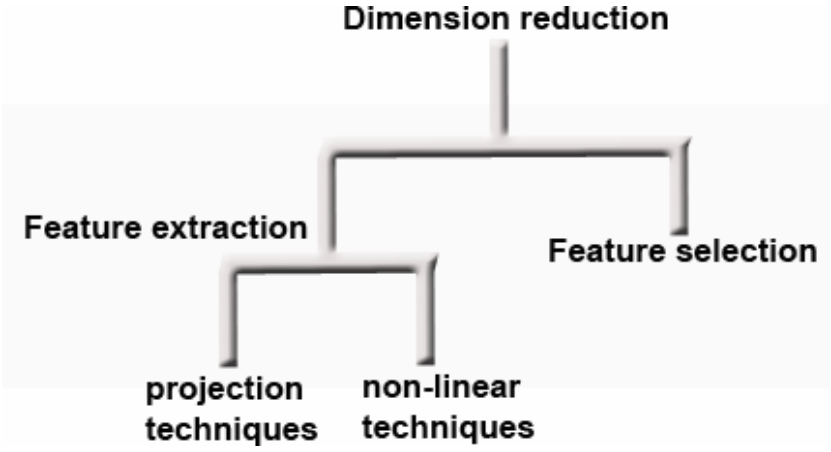


Figure 4.14 A taxonomy of dimension reduction techniques.

As with clustering algorithms, dimension reduction routines provide a reduced representation of data. Dimension reduction can be considered as an orthogonal approach to clustering – it works primarily on data dimensionality *d* whereas clustering most commonly tackles cardinality *N*, with the exception of the special case of dimension clustering as described in Section 4.3.3. However, both approaches can work together to reduce data to a

concise form suitable for visualisation or further processing that would otherwise be unfeasible due to insufficient computational resources. The next chapter is concerned with hybrid algorithms for clustering and dimension reduction.

5. Hybrid clustering and layout algorithms

The previous two chapters have shown how data can be reduced to a smaller number of representative units (clustering) with fewer variables (dimension reduction) without losing much information. This makes visualisation easier because there are fewer entities to code into graphical structures; it also makes the resulting visualisations easier to comprehend. The fact that the two approaches described for producing such a reduced representation often go hand-in-hand has also been demonstrated. Clustering can be utilised to aid dimension reduction and dimension reduction can be used to help define clusters. One may also argue that the two approaches are equivalent.

This chapter is concerned with hybrid algorithms for attaining a reduced representation of data. Such combinations, including both clustering and dimension reduction algorithms have been touched upon in the previous chapter. For example, the SOM (Section 4.2) performs clustering and dimension reduction simultaneously. In another example (Section 4.3.3), dimension clustering can be used to expose redundancy in variables and therefore facilitates feature selection. Section 4.1.4 describes how Random Projection can enhance the effectiveness of clustering algorithms such as Expectation Maximisation and K-means that are predisposed to detecting spherically shaped clusters. While these observations imply that hybrid combinations of particular clustering and dimension reduction algorithms can be useful in attaining reduced representations of data, many such algorithms still suffer from the drawback of being computationally expensive operations. However, various clustering algorithms and dimension reduction techniques exhibit their individual advantages and disadvantages and hybrid compositions can be exploited to counter the shortcomings of the individual components. Generally speaking, some cheap algorithms (with respect to time) can be used as pre-processing stages that effectively provide a shortcut for more expensive routines, thus reducing overall running time while maintaining layout quality. This chapter will provide evidence of the potential effectiveness of hybrid algorithms through examples from the literature and from the author's own research.

5.1 Hybrid algorithms for clustering

In Section 3.2.1, it was pointed out that the K-means algorithm is prone to converge to a local minimum of the clustering cost function and is especially sensitive to the initial cluster centres which are usually randomly selected. The Buckshot algorithm [CKPT92] can be used to find good initial cluster centres that can help K-means avoid local minima. The Buckshot method works by randomly choosing $\sqrt{N}$ samples from the data set (of size N) before applying a hierarchical clustering routine such as HAC (see Section 3.1). Since the time complexity of HAC is $O(N^2)$, the dendrogram of the $\sqrt{N}$ sample can be attained in linear time. K-means can then be initialised and run on the full data set using the mean vectors of a selected subset of the clusters obtained from the HAC routine. Since the chosen centres already approximate potential clusters, K-means is more likely to avoid local minima. The advantage of the Buckshot algorithm is its stochastic sampling. This reduces the time complexity of the initial (computationally expensive) clustering routine which subsequently provides a head-start for the second (computationally cheaper) clustering stage. In principle, the clustering routines used with Buckshot do not have to be HAC and K-means as described above.

Another way of improving the quality of a clustering solution while decreasing the time complexity is to apply Random Projection dimension reduction (see Section 4.1.4) as a pre-processing stage for a clustering algorithm that is predisposed to finding spherical clusters. The tendency for clustering algorithms such as K-means and model-based techniques such as Expectation Maximisation (EM) to detect predominantly spherical clusters is obviously a disadvantage because it is unlikely for real data to contain only spherically shaped clusters. However, Random Projection has been shown to distort clusters of arbitrary shape by making them more spherical but it still maintains their separation [Das00]. This means that a clustering algorithm such as K-means or EM has the potential to find clusters of different shapes. Another benefit arises because Random Projection reduces the dimensionality of the data, and therefore the time taken by the clustering routine is reduced.

The research areas of cluster analysis and pattern classification are closely related. In data mining and exploratory data analysis, clusters are often sought in the attempt to classify (or categorise) existing data and to subsequently classify new data. A new pattern (data item) can be classified by finding the most similar cluster in an existing data set and this has had a major impact in the fields of pattern recognition and artificial intelligence [LS97a]. In fact the unsupervised clustering algorithms detailed in Chapter 3 can be considered as *machine learning* techniques because they can be applied in the unsupervised classification of data. A relatively recent development in pattern classification involves the use of multiple classifiers

to improve the accuracy of the resulting classification[1]. This is based upon the *Condorcet jury theorem* - proposed by the Marquis of Condorcet in 1784, this theorem provides evidence showing that the judgement of a group of people is superior to that of an individual [LS97b]. This theorem can easily be transposed to pattern classification by running a classifier on a data set several times and considering the allocation of a pattern to a class as a 'vote' vouching for that pattern as being a true member of the class. If a pattern x has a majority of votes for class A over another class B then it is considered as more likely to be a true member of class A. A simple example of this would be to apply multiple runs of K-means clustering on a data set and then derive the clusters from the most consistent allocation of the data [Fre02].

Formally, the Condorcet jury theorem can be defined as follows. If each classifier has a probability p of being correct and the probability of the majority of classifiers being correct is m, then:

- $p > 0.5$ implies $m > p$
- and m approaches 1, for all $p > 0.5$ as the number of classifiers approaches infinity

This reasoning, as employed in pattern classification, is referred to as classifier *ensemble*, *combination* or *fusion* [Lam00, RK02, Str02] and can be considered as a special case of a hybrid algorithmic approach to clustering. There are several possible manifestations (or topologies) of classifier ensembles [Lam00]. The *conditional* topology works by using a primary classifier and when certain patterns are rejected or allocated to a class with low confidence, then another different classifier can be used to see if it can do better. This approach can be very efficient, especially if the primary classifier is computationally cheap. The *hierarchical* topology involves a set of (possibly diverse) classifiers applied successively to patterns. Each classifier produces a smaller number of classes which are in turn used by the next classifier thus gradually reducing the classification problem and focussing the process. The *hybrid* topology systematically chooses a particular type of classifier according to the values of the pattern features. If, for example, there are missing features in a pattern vector, then a classifier that can cope with this would be employed. The *multiple* topology has already been demonstrated by the K-means example above. This approach employs the simple majority voting rule to determine classes from the consistent allocation of patterns.

---

[1] In the context of this section, the term *classifier* will be used synonymously with *clustering algorithm* as will *class* with *cluster*.

5.2 Hybrid algorithms for dimension reduction

This section will describe how some researchers have exploited hybrid algorithms to improve the quality and speed of dimension reduction routines. One of the most popular algorithmic components used is the SOM due to its reasonable time complexity and its ability to detect clusters and reduce dimensionality simultaneously.

In the attempt to make the SOM faster, Su and Chang [SC00] employed K-means to gather k clusters from the data set. The representative centroids of these clusters are then organised into a discrete $\sqrt{k}$ by $\sqrt{k}$ grid and a SOM is subsequently used to fine-tune the layout. Su and Chang suggest that this hybrid approach is much faster than the traditional on-line SOM because K-means has a lower time complexity than the SOM and it provides an initialisation that is reasonably close to the final solution.

Kohonen et al. [KKL*00] demonstrated another way of reducing the running time of the SOM. In this case the objective was to use the SOM to classify and visualise over six million text documents. After discounting words that appeared less than 50 times in the whole corpus and removing stopwords (terms, such as articles and connectives, that would normally be considered to bear little content), the remaining vocabulary – and therefore the dimensionality – of the data was 43,222. Kohonen et al. then employed Random Projection to reduce the dimensionality to 500 before applying the SOM. Random Projection can be achieved in time linear with N while still retaining much of the original information. In this example, it was instrumental in making the application of the SOM to such a large data set feasible.

In another example, Brodbeck and Girardin [BG98] reduced the running time of a canonical spring model algorithm (Section 4.4) by using the SOM as a pre-processing step. Consisting of a discrete grid of cells (or neurons), SOMs cannot show as much intuitive structure or detail as a spring model layout but are often quicker to produce and scale to larger data sets. In this example, Brodbeck and Girardin used the SOM to acquire a set of clusters, much in the same way as Su and Chang used K-means, however, in the next step a spring model was employed to either lay out the contents of one of the clusters or to lay out the set of vectors representing the SOM neurons. Whichever step is taken is up to the discretion of the user. However, to obtain a full layout of the data set, the latter option is taken before applying a novel interpolation routine to add the cluster contents to the layout of neurons. Since the spring model is consequently run upon a reduced data set (the representative SOM neurons), it

converges far more quickly while the interpolation of the remainder of the data set takes only $O(N)$ time. The accuracy of the interpolation depends upon a set of constants used to govern the process; the higher the values for these constants, the longer the process takes but the better the positioning obtained. Example figures showed that layouts could be produced that were strikingly similar to those generated by a spring model run on the entire data set. Brodbeck and Girardin also stated that an improvement in run time was gained, requiring hours rather than days to complete.

While the SOM can be an effective component of a hybrid algorithm for dimension reduction, other methods have been explored. Schroeder and Katopodis [SK02] ran experiments to find a good method of initialising the point positions for a canonical spring model. A common heuristic for initialising a spring model is to place the layout points in a random configuration before allowing the spring model to iteratively refine their positions until the final solution is obtained. Schroeder and Katopodis compared this technique along with several others including initially placing all points at the origin of the 2-d layout, placing the points on a circle, and using a hierarchical clustering algorithm to dictate placement. They found that the clustering approach provided the best results. Initially placing points in the layout according to their relative positions in the clustering dendrogram improved the performance of the spring model resulting in lower stress levels in a smaller number of iterations. This is due to the points being placed in positions that approximated the final solution.

The previous two sections have indicated that the hybrid combination of algorithms can provide more efficient and effective clustering and layout algorithms. The approach generally adheres to an *algorithmic symbiosis* – a cheap process roughly reduces the problem to hasten a more expensive one which improves the output. The following sections will outline some of the author's own work in this area.

5.3 A novel hybrid algorithm for dimension reduction

This section presents an original algorithm for reducing high-dimensional data to a 2-dimensional layout. The routine is based upon a hybrid combination of Chalmers' spring model [Cha96], stochastic sampling and an improved version of Brodbeck and Girardin's interpolation [BG98]. Results of the algorithm's evaluation show that it is successful in obtaining non-linear dimension reduction in sub-quadratic time. The algorithm and the results

were published in the proceedings of InfoVis 2002 [MRC02] and in the Journal of Information Visualization [MRC03].

Chalmers' spring model algorithm was one of the fastest non-linear dimension reduction techniques (see Section 4.4). Taking $O(N^2)$ time to provide a solution, it outpaces the traditional MDS routines while providing more detail in its output than the SOM. It also has the potential to expose more interesting structure than linear techniques such as PCA and SVD. It was for these reasons that this algorithm was chosen as one of the components to be included in the experimental hybrid conjunction.

The goal was to reduce the time complexity of Chalmers' spring model without bearing any detrimental effects on the resulting layout quality. This was achieved by initially sampling $\sqrt{N}$ items from the input data to obtain a spring model layout of the subset in $O(\sqrt{N}\,\sqrt{N})$, hence $O(N)$ time. Assuming that this layout provides a reasonable representation of the full data set, the remaining $(N - \sqrt{N})$ items can be interpolated onto the layout to provide a good approximation of that produced by running the spring model on the entire data set. The results, as will be seen, indicate that this is indeed a reasonable assumption.

Interpolation is carried out by placing each of the remaining $(N - \sqrt{N})$ items near to the closest item in the subset and therefore takes $O(N\sqrt{N})$ time. Even though the interpolation can exactly recover the inter-object distances in 2-d data, it might not always provide the desired results, especially for data of higher dimensionality. If, for example, the spring model of the subset terminated prematurely with some points misplaced, then this would have a knock-on effect on the interpolation, resulting in segments of the layout lying askew. To overcome this, the spring model is run for a constant number of iterations over the full data set to refine the layout. In practice, this constant is between 20 and 35 iterations because it was frequently observed that this was long enough to significantly reduce layout stress.

Since each iteration of Chalmers' algorithm is achieved in linear time, the overall time complexity is dominated by the interpolation stage and is therefore $O(N\sqrt{N})$. The outline of the algorithm is given in Figure 5.1. and its operation is illustrated in Figures 5.2 to 5.4.

1. take sample of $\sqrt{N}$ items from data set
2. run Chalmers' spring model on the sample
3. interpolate remaining ($N - \sqrt{N}$) items onto layout
4. refine layout by applying Chalmers' spring model to the full data set for a constant number of iterations

Figure 5.1 The sub-quadratic ($O(N\sqrt{N})$) algorithm for non-linear dimension reduction.

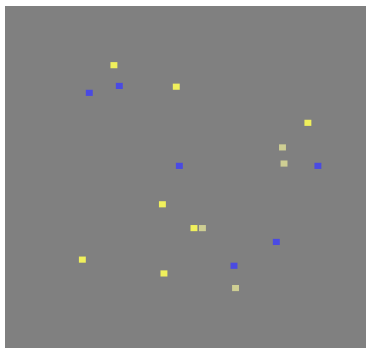


Figure 5.2 Chalmers' spring model is initialised by randomly positioning the $\sqrt{N}$ sample in 2-d.

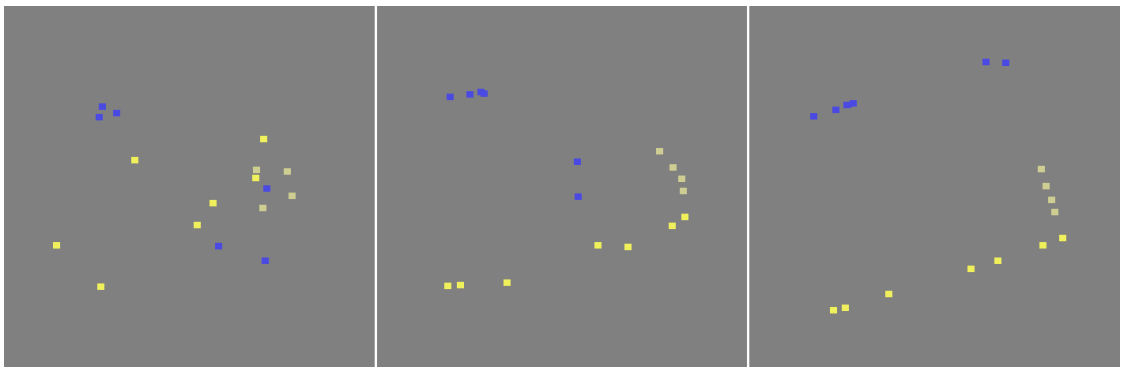


Figure 5.3 The spring model interatively produces an accurate layout of the sample.

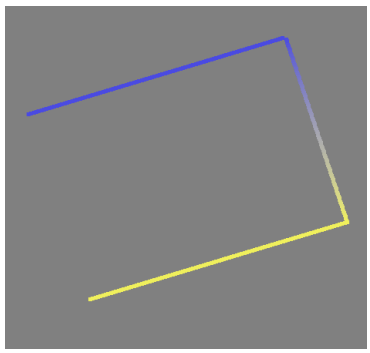


Figure 5.4 The remaining (N - $\sqrt{N}$) items are interpolated onto the sample layout. The spring model can then be run for a constant number of iterations to refine the layout. In this case that is not necessary.

Figures 5.2 to 5.4 show screenshots demonstrating the algorithm working on a small 2-d data set consisting of 300 items. While this provides an intuitive example of how the algorithm works, it also demonstrates an advantage of this multistage technique. The user can quickly observe an overview of the data and then decide whether to proceed with the more time-consuming full layout or to halt the process. Furthermore, the views provided by individual algorithmic stages can enrich interaction with data and therefore provide more insight. This will be demonstrated in later chapters where an interactive environment developed by the author for building and using such algorithms is discussed.

It should be noted that there are several methods for determining when to terminate the execution of the initial spring model run on the sample. One commonly used method for terminating force-directed placement routines is to measure the stress after each iteration and terminate execution when its value no longer decreases [Krus64]. However, this method suffers from a major disadvantage due to the fact that measuring stress requires calculating all inter-object distances and therefore takes $O(N^2)$ time. Although, the initial stage of the algorithm works on a $\sqrt{N}$ sample of the data, the additional time for calculating stress can still be prohibitive with large data sets. Another method of terminating execution is based upon the rule-of-thumb that force-directed placement requires $O(N)$ iterations on average to reach equilibrium. The initial spring model can thus be run for $\sqrt{N}$ iterations before applying interpolation. In practice, however, an alternative criterion for termination based upon *velocity* is used. Since the spring model simulates a system of objects with forces acting between them, the velocity of an object (its speed and direction) can be measured with respect to these forces. When the difference in overall velocity between the current iteration and the preceding iteration drops below a scalar threshold, the spring model is terminated and interpolation begins. Measuring velocity does not require much additional time because it is calculated as part of the spring model routine. Furthermore, it provides a dynamic account of how the algorithm is progressing and therefore provides a more effective means of termination than that based upon the assumption that convergence is achieved after N iterations.

5.3.1 Distance metric

The operation of the algorithm is pivotal on the measurement of dissimilarity between data items. By measuring the dissimilarity between items in their high-dimensional space and comparing this to their representative *distances* in the 2-d layout, the algorithm progressively refines the layout by moving items closer or further apart in the layout in accordance with this comparison (see Section 4.4 for a description of the spring model). However, since the spring model is based upon a physical analogy, it attempts to reproduce the high-dimensional

dissimilarities as well as possible. This predisposes the algorithm to the use of a dissimilarity measure that obeys the triangle inequality, is non-negative and is symmetric. If these criteria are not met then the model cannot be expected to find a layout in a Euclidean space of any dimensionality.

For this reason, the choice of dissimilarity used belongs to the family of metrics known as the Minkowski (or Lebesgue) metrics [JMF99], which operate on a vector representation of each datum. The general form of this family of metrics is as follows:

$$d_L(x_i, x_j) = \left(\sum_{k=1}^{d} \left| x_{i,k} - x_{j,k} \right|^L \right)^{\frac{1}{L}} \tag{5.1}$$

Where L is a parameter that takes on some value in the interval [1, ∞] and d represents dimensionality. The most commonly used instance of this metric is Euclidean distance, where $L = 2$. This has been adopted as the measure of dissimilarity in this algorithm because of its metric qualities as described above and also because it is widely used in measuring distances in lower dimensional space (such as the physical environment), and intuitively generalises to higher dimensional spaces.

Inter-object distances are calculated on the fly during execution of the algorithm. However, to prevent specific dimensions from dominating the distance, the values for each dimension are normalised by dividing them by two standard deviations of their distribution.

5.3.2 Brodbeck and Girardin's Interpolation algorithm

The interpolation stage of the algorithm was derived from Brodbeck and Girardin's [BG98] technique of introducing points onto a layout of a subset of the data. They achieve this by finding for each item that is to be added, the point in the layout representing an item that is closest in the high-dimensional space. Herein, such a point will be referred to as a *parent* point. The algorithm progressively improves the position of the item so that the difference between its high- and low-dimensional distances to a sample set of layout points is reduced. Figure 5.5. provides an outline of this algorithm.

1. centre a circle of radius proportional to the high-dimensional distance, at the closest layout point
2. define a random subset of layout points S
3. repeat n_c times:
    a. take a random position on the circle's circumference.
    b. sum the discrepancy between the high-dimensional and the low-dimensional distances between this position and the points in S
4. place the new item at the position on the circumference that provides the lowest sum of discrepancies
5. repeat n_r times:
    a. define a force vector between the current position and the sample of layout points
    b. randomly sample a number n_f of positions along the vector's direction
    c. sum the discrepancies between high- and low-dimensional distances between this position and the points in S
    d. place the new item at the position that provides the least discrepancy

Figure 5.5 Brodbeck and Girardin's interpolation routine [BG98].

The quantities n_c, n_r and n_f are fixed and therefore the time complexity of this algorithm is linear with respect to N. Increasing these values will result in more accurate placement but will increase the time taken.

It was found that this technique often produced sub-optimal layouts, even for 2-d data. This is due to the stochastic sampling of positions in the vicinity of the parent point. Figure 5.6 shows a layout of a 2-d data set after running Chalmers' spring model on a $\sqrt{N}$ subset and subsequent interpolation by Brodbeck and Girardin's method. It can be seen that although the general shape of the data has been recovered, it remains rather rough. The Shepard plot in Figure 5.7 confirms this by the deviation from the 45 degree diagonal. An accurate layout of 2-d data in two dimensions should result in a perfect 45 degree line appearing in the Shepard plot due to the one-to-one relationship between the original distances and the recovered layout distances.

One might be tempted to blame the initial spring model for throwing the interpolation off course. However, in this case (as is usual for 2-d data), the spring model produced a layout of the $\sqrt{N}$ subset with near zero stress, that is, an almost perfect layout.

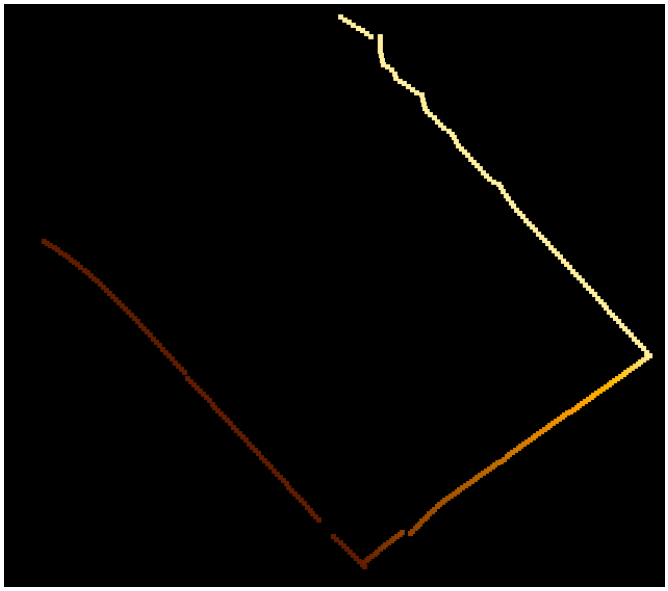


Figure 5.6 Brodbeck and Girardin's interpolation often provides a sub-optimal layout, even for 2-d data.

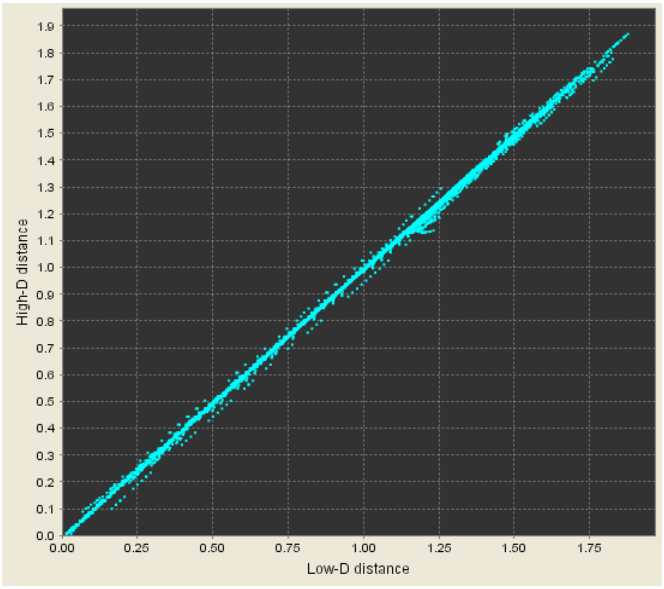


Figure 5.7 For a 2-d layout of 2-d data, the recovered distances should exactly match the original distances. This one-to-one relation should manifest itself as a 45 degree slope in the Shepard plot, however, in this case it can be seen that the interpolation routine has resulted in deviation from this slope.

5.3.3 An improved interpolation technique

Prompted by the shortcomings described above, the author designed and implemented a new routine to improve upon Brodbeck and Girardin's. As with the original, the new routine starts with finding a parent point in the layout which is closest in high-dimensional space to an item that is to be added. Again, a circle of radius proportional to this high-dimensional distance is centred on the parent. However, in the next stage, rather than randomly sampling points on the circumference of the circle to find one that has a small discrepancy between high- and low-dimensional distances, a quadrant and binary search is applied instead. The pseudocode for this new routine is shown in Figure 5.8.

1. centre a circle of radius proportional to the high-dimensional distance, at the closest layout point
2. define a random subset of layout points S of size $N^{1/4}$
3. at positions on the circumference 0, $\pi/2$, π and $3\pi/2$ radians from the horizontal norm, sum the discrepancies between the high- and low-dimensional distances. The position that provides the lowest discrepancy defines the quadrant of the circle's circumference upon which the new item will be placed.
4. apply a binary search to the chosen quadrant to find the position that minimises the discrepancy between high- and low-dimensional distances
6. repeat n_r times:
    a. define a force vector between the current position and the sample of layout points S
    b. add the force vector to the current position

Figure 5.8 An outline of the new interpolation algorithm.

After defining a circle around the parent point in the layout, the algorithm finds the quadrant on the circumference that would minimise the difference between high- and low-dimensional distances of a point on that quadrant, and the $N^{1/4}$ sample of layout points. This narrows the space of a binary search to the $\pi/2$ radians on that quadrant. The binary search finds the position that minimises the discrepancy between distances before force calculations are performed to refine the position of the interpolated item. Figure 5.9 illustrates this diagrammatically.

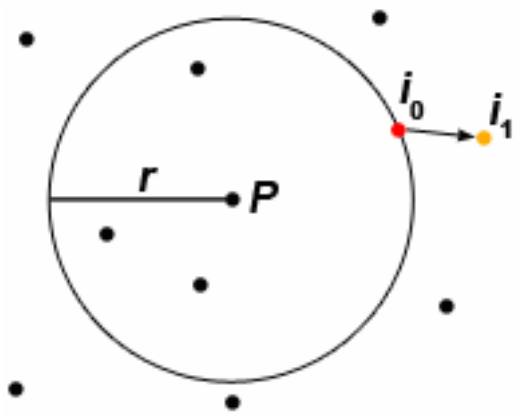


Figure 5.9 The placement of item *i* begins with finding its parent point *P* in the initial layout. A circle of radius *r* (proportional to the high-dimensional distance between *i* and the item represented by *P*) is centred on *P*. Quadrant and binary search over the circle's circumference finds the position i_0 that minimises the summed discrepancy between the high- and low-dimensional distances to the subset of layout points. This position is then refined by iteratively adding an aggregate force vector, moving the item to its final position i_1.

After finding an item's parent in the layout, its interpolation requires $O(N^{1/4})$ time because a subset of size $N^{1/4}$ is used to calculate its position and all other quantities are constant – finding the best position on the circle's circumference requires 4 steps for the quadrant search and 6, at most, for the binary search. The aggregate force vector for refining this position is achieved in n_r distance comparisons. However, since there are initially $\sqrt{N}$ items in the initial layout, this leaves the remaining (N - $\sqrt{N}$) items to interpolate. There is no information gained *a priori* as to which layout point is an item's parent, and so a brute force search is required. This search dominates the time taken by the routine so the overall time complexity is $O(N\sqrt{N})$.

A visual comparison of the output of this interpolation routine shows that it is more accurate than Brodbeck and Girardin's routine. Figure 5.10 illustrates the 2-d test data set after interpolation and the associated Shepard plot. Comparison with Figure 5.6 shows that the layout has reached a better representation of the data. The Shepard plot depicts a straight 45 degree line with no points deviating from the diagonal indicating that the original distances have been perfectly reproduced by the algorithm.

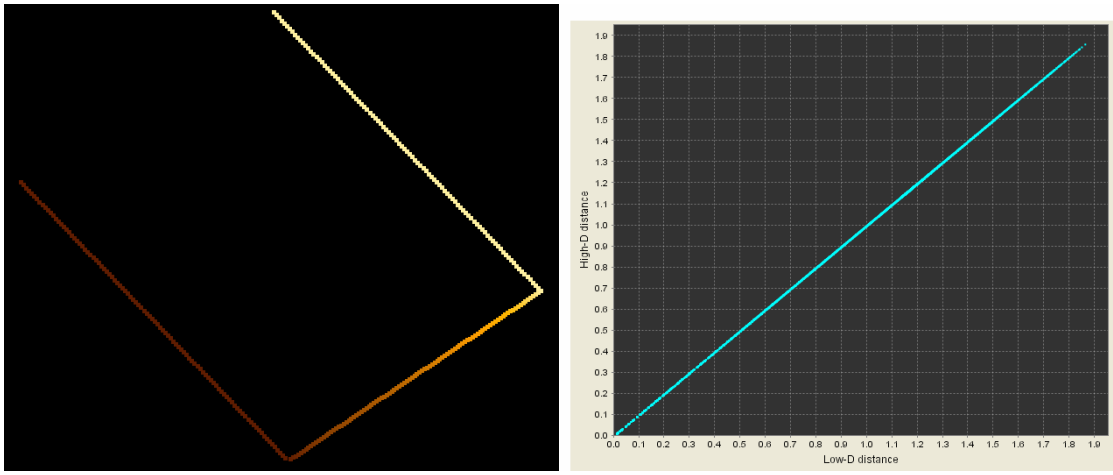


Figure 5.10 The new interpolation routine has produced a much more accurate layout of the 2-d data.

The author designed and implemented this interpolation algorithm, however, its evaluation was undertaken by a colleague, Alistair Morrison [Mor04]. Morrison employed the author's HIVE software (from which all of the screenshots in this chapter have been taken) to automate multiple runs of the algorithm and record run time and stress values. Figure 5.11 shows the results of a comparison between Brodbeck and Girardin's interpolation and the new routine. Experiments were carried out on a PC with 504 MB RAM and a Pentium 4 2.41GHz processor, running Windows XP Professional.

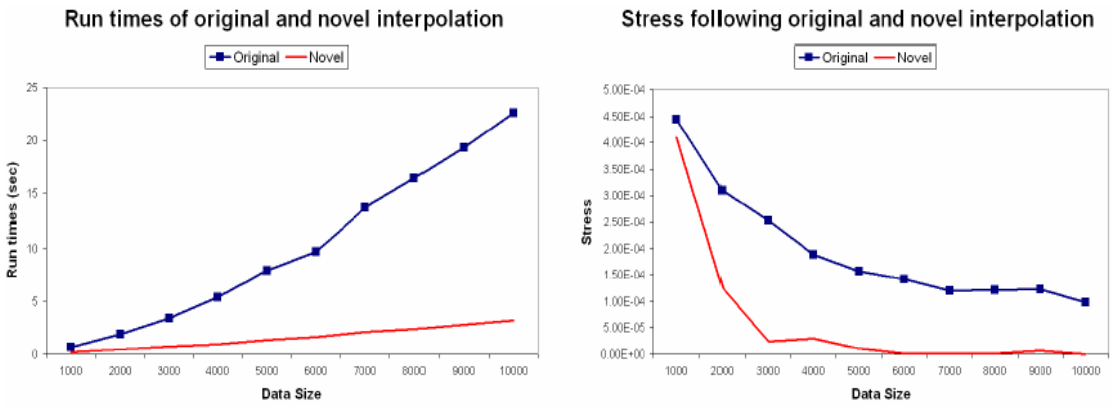


Figure 5.11 A comparison of Brodbeck and Girardin's original routine with the new interpolation algorithm. These results were obtained by Morrison [Mor04] using the author's HIVE software. The results for each algorithm were averaged over 10 runs for 2-d data sets ranging from 1000 to 10,000 items.

These results show a significant improvement over Brodbeck and Girardin's interpolation routine in terms of both run time and stress. It is interesting to observe that for smaller data sets, the stress levels are higher than those gained with the larger sets. This can be

explained by the fact that as N increases, so does the size of the $N^{1/4}$ sample used in the force calculations. This results in a more accurate placement. Although smaller data sets result in higher stress, this is not important because as a final stage of the layout algorithm, a spring model is used to refine the layout and therefore reduce the stress. Also, since the data sets are smaller, the final spring stage converges quickly. Figure 5.12 illustrates graphs of run time and stress for these data.

5.3.4 Evaluation of the full layout algorithm

The initial spring model stage and interpolation as described above comprise the first two stages of the new hybrid algorithm – the final stage involves the use of a spring model to refine the layout produced after interpolation. In this section, Chalmers' algorithm is used as a benchmark to which the new routine (including the final stage) is compared. Both algorithms were run on subsets of a 3-d 's' shaped data set. The size of the subsets range from 5000 to 50,000 items and results are averaged for the subsets over 10 runs for each algorithm.

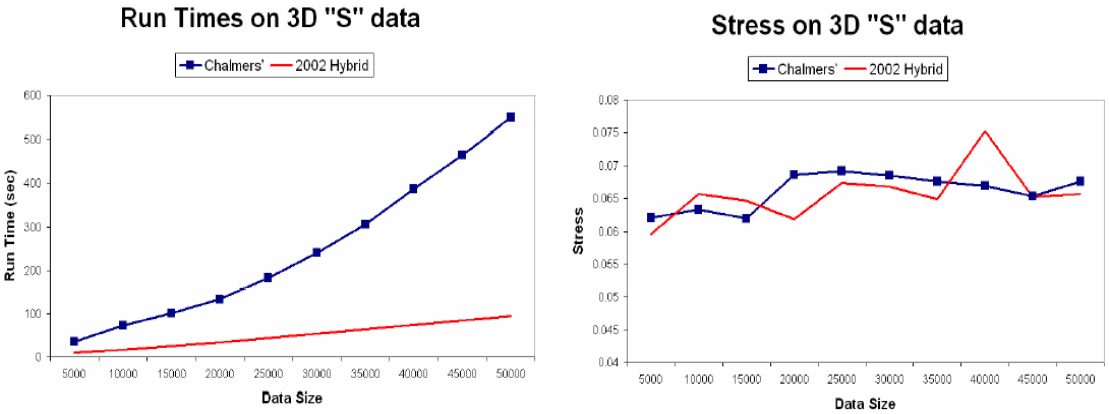


Figure 5.12 Results obtained by Morrison [Mor04] showing a comparison of Chalmers' spring model with the new algorithm.

Up until the publication of the new algorithm, Chalmers' spring model was one of the fastest non-linear reduction algorithms based upon a spring model. From the above results, it is evident that the new algorithm significantly outperforms Chalmers' routine with respect to run time while maintaining comparable layout quality.

The initial spring model stage has time complexity of $O(N)$ because it operates on a sample of the data and the final spring model stage also has $O(N)$ time complexity because it is run for a maximum of 35 iterations to refine the full layout. This means that the overall time complexity is dominated by the interpolation routine because a brute force search is required to locate the parent points. The overall time complexity is therefore $O(N\sqrt{N})$.

5.3.5 A hybrid variant based upon K-means

During the development of the above algorithm, the author investigated an alternative approach based upon the K-means clustering algorithm (see Section 3.2.1). The reasoning behind this is as follows. Recall that the K-means algorithm partitions the data into clusters by identifying and refining centroids. The centroids are representative of clusters and each datum is allocated to one cluster according to their proximity. After each iteration the values of the centroid vectors are updated according to the average of the data vectors allocated to them. This has two implications. The first is that the centroids will gradually disperse throughout the data set as their vector values will increasingly reflect local regions of the data. Thus the centroids can provide an overview of the data distribution. The second implication is that all items in the data set are allocated to their nearest centroid and therefore the simple heuristic, described in Section 3.2.3, for finding the nearest neighbour of a given item can be employed. This heuristic is as follows. Given a K-means clustered data set X and a datum $x_i \in X$, its approximate nearest neighbour $x_j \neq x_i$ can be found by searching through the cluster members of the centroid to which x_i has been allocated.

With this in mind it can be seen that this is applicable in the interpolation stage of the novel hybrid algorithm described in the previous section. The interpolation stage is the computational bottleneck because of the brute force search required for parent finding, however, if the full data set is initially clustered using K-means, then the centroids and their cluster members can be used to find interpolation parents in much less time. Also, since the initial spring model runs upon a random subset of data, there is a small chance that this sample is confined to one local portion of the distribution. This would have a disastrous effect on the remaining stages of the algorithm. However, if the initial subset is chosen to be the K-means centroids, then the subset is guaranteed to be more representative of the true distribution thus providing a more reliable basis for interpolation. Figure 5.13 illustrates this concept.

To test this theory the author initialised the K-means algorithm with a randomly selected $\sqrt{N}$ subset of data. K-means was then run until the centroid vectors ceased to change value. The items in the data set that were closest to the centroids were then fed in to a spring model before the interpolation algorithm was deployed, using the nearest neighbour heuristic to find parent points. Finally, a spring model was used to refine the full layout. The results of similar experiments were recorded by Alistair Morrison as discussed below.

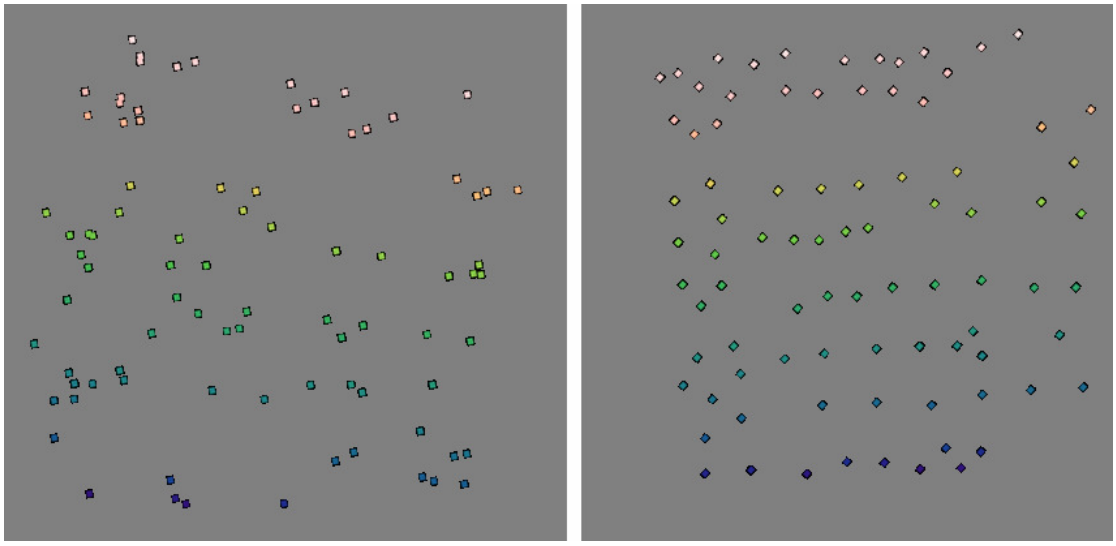


Figure 5.13 The image on the left is a layout of a random data sample. The image on the right is a layout of items that are closest to the K-means centroids of the data. The original data used are 2-d and consist of 7239 items. Both layouts contain 85 (√7239) points from the data set.

Experiments were run on two data sets, each of which were synthetically created. The first collection was created by sampling points from a 2-d set – the logo for a company owned by Brodbeck and Girardin [BG05]. Subsets of 10 different cardinalities were created from this 'logo' set, from 1000 to 10,000 items. The second collection was similarly sampled from a band curving in an 'S' shape through three dimensions. Again, 10 subsets were derived, this time from 2000 to 20,000 items. The K-means based algorithm (hereafter referred to as *K-means+interpolation*) was then compared to both Chalmers' spring model and the novel algorithm (hereafter referred to as *sample+interpolation*) described in the preceding sections.

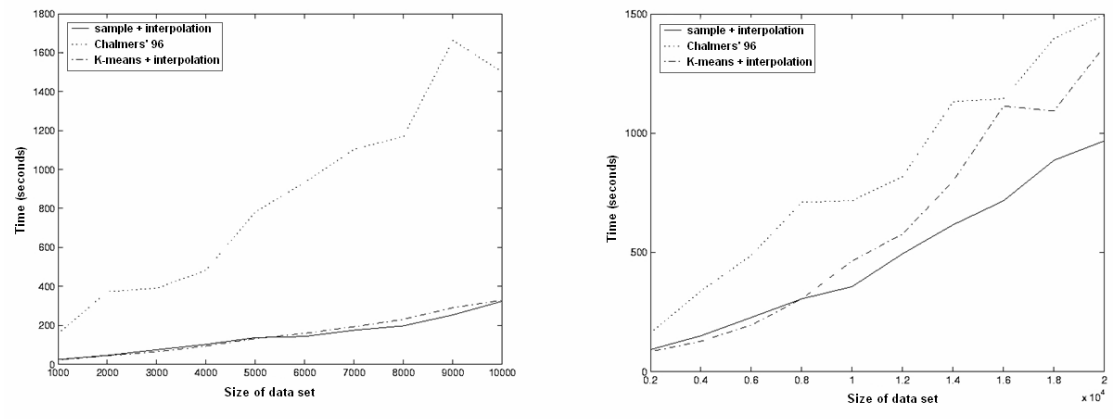


Figure 5.14 A comparison of the K-means based algorithm with the sample+interpolation and Chalmers' algorithms.

Figure 5.14 compares the stress and run time of Chalmers', sample+interpolation and K-means+interpolation algorithms. Tests were run on a PC with an Intel Pentium 3 731MHz and 256MB RAM running Microsoft Windows 2000 Professional. As expected, Chalmers' algorithm took longer to converge than both of the hybrid algorithms, over all of the data subsets. It can also be seen that the sample+interpolation and K-means+interpolation algorithms exhibit comparable times. For the smaller data sets, K-means appears slightly faster. The K-means algorithm converges quickly for smaller subsets, however, as the size of the data set increases, the cost of using K-means would seem to outweigh its benefit over simple sampling.

The stress recorded for Chalmers' algorithm is significantly higher than the others. This is because, as the algorithm proceeds, the stress tends to fall sharply and then level out for a considerable time before the layout becomes stable. The stress levels exhibited by the hybrid algorithms are lower for the logo set because the interpolation can achieve near-optimal positioning of the points in two dimensions. To enhance this, Chalmers' algorithm is finally run on this interpolated structure for a small constant number of iterations to refine the layout and minimise stress.

Since K-means was run on a $\sqrt{N}$ subset of the data it dominates the time complexity, taking $O(N\sqrt{N})$ time. Although the two hybrid algorithms exhibit the same time complexity, the difference in observed times is due to overheads in the K-means algorithm being greater than those for the brute force search for the sample+interpolation approach. It can therefore be concluded that the only benefit of using the K-means routine is to ensure that a good representative subset of the data is gained to provide a basis for interpolation. However, in practice it is very unlikely that random sampling would result in the initial subset residing in a local area of the data and therefore the sample+interpolation algorithm is preferred.

5.4 Fast non-metric multidimensional scaling

The two novel hybrid algorithms discussed above are essentially spring models. They model the differences between data items as Euclidean distances in a high-dimensional space and progressively refine their positions on a low-dimensional layout to reflect those distances as well as possible. However, since this approach is a metric one, it cannot be readily applied to non-metric proximity data, that is, measures that do not necessarily satisfy the triangle inequality, symmetry and non-negativity. It is more difficult to model such data in a low-dimensional Euclidean space and as such the spring model is often inadequate.

In Section 4.3.2 Shepard's non-metric multidimensional scaling (NMDS) was discussed and shown to be applicable to proximity data. Recall that Shepard's solution to dimension reduction has a very special property in that it can obtain a metric low-dimensional configuration from non-metric information. This is achieved by ranking proximities and iteratively forming a layout in which the (inverse) rank of the inter-point distances matches the rank of proximities as well as possible. The condition in which the ranks perfectly match is known as *monotonicity*. However, the main drawback of Shepard's NMDS is its $O(N^4)$ time complexity making it prohibitive in its application to larger data sets.

In this section, the author will discuss his work on a faster $O(N^3)$ version of Shepard's algorithm. This algorithm is outlined in Figure 5.15.

1. obtain a ranked list of all (N(N–1)/2) inter-object proximities
2. randomly place N points in a 2-d layout
3. instantiate neighbour and sample arrays for every point (as in Chalmers' spring model)
4. repeat until departure from local monotonicity < threshold m_l:
    a. for each object 1…N:
        i. rank distances between object and its neighbour and sample arrays
        ii. calculate displacement vector based upon discrepancy of ranks
        iii. update point positions in the layout
        iv. update the neighbour and sample arrays
5. repeat until departure from global monotonicity < threshold m_g:
    a. for each object 1…N:
        i. obtain a ranked list of all (N(N–1)/2) inter-point Euclidean distances
        ii. compare object to its neighbour and sample arrays and calculate displacement vector according to discrepancy in overall rank
        iii. update the neighbour and sample arrays

Figure 5.15 An outline of the fast NMDS algorithm.

Recall that in the initialisation step of Shepard's algorithm, all points are placed upon the vertices of an $(N-1)$-d simplex. As Shepard's algorithm progresses, this $(N-1)$-d space is gradually collapsed to a dimensionality in which monotonicity is satisfactorily maintained. There are therefore two conditions that Shepard's algorithm seeks to optimise. These being minimum dimensionality and monotonicity. This requires two displacement vectors to update the positions of points. One to reduce dimensionality and one to increase monotonicity.

However, since in the context of this thesis dimension reduction is used for visualising data in 2-d layouts, the points are initialised in a 2-d space instead of the (N – 1)-d space. Thus, only the displacement vector for increasing monotonicity needs to be calculated. Also, since the configuration space is reduced to 2-d and no longer O(N), the algorithm is reduced to $O(N^2)$ time complexity per iteration.

The speed of the algorithm is improved further by using Chalmers' neighbour and sample approach to reduce the number of inter-object and rank comparisons. This is employed in two stages: in the first stage of the algorithm, the distances between each object and its neighbour and sample arrays are ranked and the discrepancy between the ranks is used to calculate the displacement vector for maximising monotonicity. The displacement vector is calculated using Equation 5.2.

$$\alpha_{ija} = \frac{\alpha[s_{ij} - s(d_{ij})]\left(x_{ja} - x_{ia}\right)}{d_{ij}} \tag{5.2}$$

This is the same equation defined by Shepard [She62]. α_{ija} is the a^{th} element of the 2-d vector directed from point i to another point j. The proximity between i and j is denoted by s_{ij} and α (without subscripts) is a parameter for determining the length of the displacement vector. The Euclidean distance between points in the layout space is given by d_{ij}, and x_{ia} denotes the a^{th} element of the vector representing point i. The quantity $s(d_{\text{ij}})$ represents the proximity at the rank of d_{ij}. When points are too close or too far away such that monotonicity is compromised, this equation pushes them apart or moves them closer to rectify this.

Since in the first stage of the new algorithm this equation is applied only to distances between each object and its neighbour and sample arrays, it serves to reduce departure from local monotonicity rather than global monotonicity. This is emphasised by making sure that the neighbour array is larger than the sample array ensuring that the rank of distances in local portions of the layout approximate those of the corresponding proximities. Figure 5.16 illustrates this concept. The figure shows a layout of a 3-d cube data set consisting of 1200 items obtained by the first stage of the fast NMDS algorithm and layout of the same data using

a spring model. It can be seen that with NMDS local structure is preserved while the global structure has been dramatically distorted. The spring model, on the other hand preserves global structure at the expense of local detail.

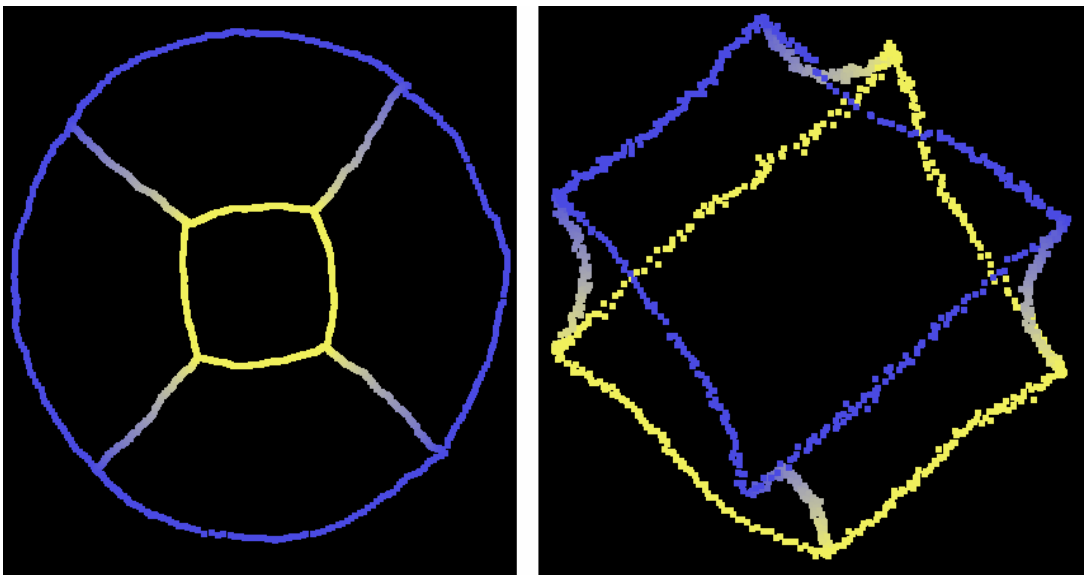


Figure 5.16 The image on the left shows the layout of a 3-d cube obtained by the first stage of the fast NMDS algorithm. This shows how monotonicity is only locally preserved. The image on the right shows a layout obtained by Chalmers' spring model on the same data. Here the overall structure is better preserved but local regions remain rough.

The first stage of the algorithm only requires $O(N)$ time per iteration because the layout is 2-d and because each point's neighbour and sample arrays are of fixed size. This first stage serves to quickly obtain an approximate layout and terminates when the departure from local monotonicity falls below a threshold m_l. Using monotonicity in this instance is analogous to using velocity to determine when to terminate the initial stage of the algorithm in Section 5.3. Monotonicity can be derived at little expense from calculations used in the layout process and it provides a dynamic account of the algorithm's progress.

The second stage of the algorithm is more similar to Shepard's original approach in that all inter-point distances are ranked and the displacement vector is derived according to these. In other words, $s(d_{ij})$ from Equation 5.2 is derived from the full set of proximities and therefore aims to improve on global monotonicity. However, to improve performance, the displacement vector is obtained only from a comparison of each object to the neighbour and sample arrays. That is, all other quantities in Equation 5.2 are obtained from a representative subset of the data. Since all inter-point distances are calculated in each iteration of the second stage, its time complexity is increased to $O(N^2)$ time per iteration and therefore $O(N^3)$ overall.

It can be seen that this is a hybrid algorithm. The first stage provides an approximate layout to speed up the operation of the more time consuming second stage.

5.4.1 Evaluation of fast NMDS

Evaluation of the fast NMDS algorithm was carried out in comparison to Shepard's original algorithm. The first experiment investigated the layout stress obtained by both algorithms on a small 2-d data set consisting of 120 items. Such a small set was used because Shepard's algorithm is too time consuming when applied to larger sets. Results were averaged over 5 runs of each algorithm and tests were carried out on a PC with 256 MB of RAM and a 1.4 GHz Pentium M processor, running Windows XP professional. Figure 5.17 shows the results obtained from measuring stress after every 5 iterations up to a total of 70.

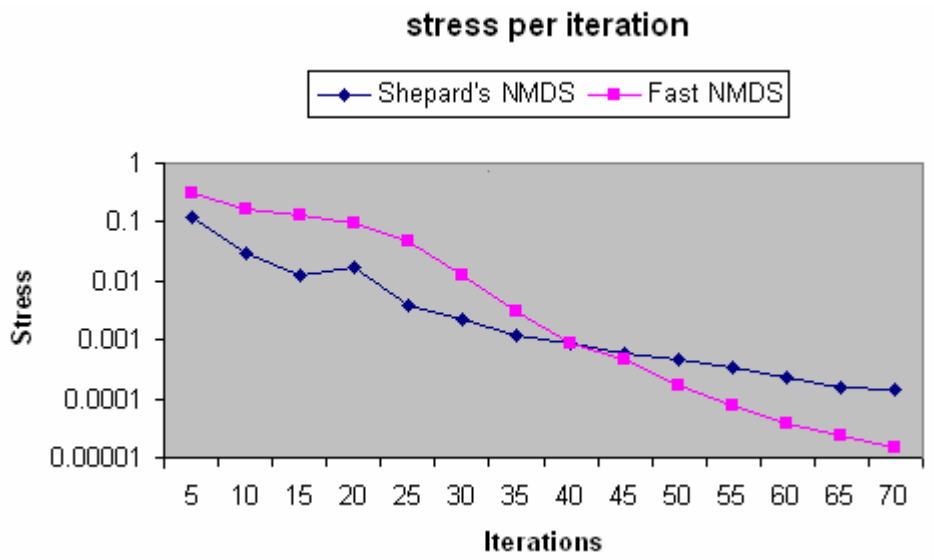


Figure 5.17 Stress is plotted after every 5 iterations for Shepard's NMDS and the fast NMDS algorithms.

Stress is shown on a logarithmic scale on the y-axis to clarify the comparison between both algorithms. For the fast NMDS algorithm α was set to 0.2, and for Shepard's algorithm its was set to 0.035. This discrepancy is due to the fact that Shepard's algorithm balances the displacement vector for achieving monotonicity with another displacement vector for reducing dimensionality. This is not required in the fast NMDS routine.

It can be seen that the fast NMDS algorithm initially displays a higher global stress level. This can be explained by the first stage of the algorithm achieving only local monotonicity. At around iteration 30, the first stage terminates and the second stage kicks in to achieve global monotonicity. This results in a sharper descent in stress per iteration,

eventually surpassing that of Shepard's algorithm. The lower stress for the fast NMDS algorithm comes as a result of only requiring the tuning of one parameter α. Shepard's algorithm requires a second parameter to determine the length of a displacement vector for collapsing the $(N-1)$-d configuration space. This means that a very fine balance between the two parameters must be maintained to approach monotonicity in 2-d. This can be difficult to attain and in this case Shepard's algorithm favours a reduction in dimensionality to the extent that stress suffers.

A second experiment was run to compare the run times of Shepard's algorithm to the new fast NMDS. The same 2-d data set used in the first experiment was taken as the test data, however, 6 subsets were sampled ranging in size from 20 to 120 items and the time taken for both algorithms to converge to a minimum in departure from monotonicity was measured. Again, results for each data set are averaged over 5 runs of each algorithm. Figure 5.18 show the results.

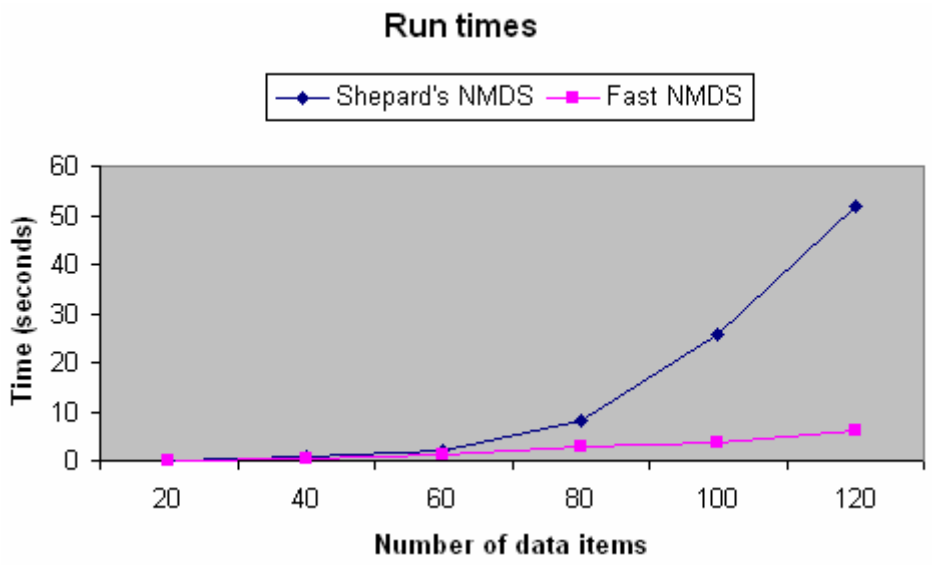


Figure 5.18 Run times for Shepard's algorithm and the fast NMDS algorithm.

It can be clearly seen that the new algorithm is much faster than Shepard's algorithm. However, it is still too time consuming to be applicable to the size of data sets to which the novel spring model described in Section 5.3 can be applied. It does, however, outperform the hybrid spring model algorithm when applied to proximity data. Recall from the discussion of Shepard's algorithm in Section 4.3.2 that when applied to proximity data, the routine can recover a monotone function relating proximities to Euclidean distances. Figure 4.9 showed a Shepard plot of a layout obtained using Shepard's NMDS run on data that were transformed into proximities using Equation 4.17. The plot showed that the routine recovered the shape of

this function. When the same data were fed into a spring model, it was found that the function could not be recovered with the same accuracy (see Figure 4.13).

It was found that the new NMDS algorithm is also capable of recovering such a function. To illustrate this, a 2-d data set consisting of 350 items was transformed into 61,075 ($N(N-1)/2$) proximities using equation 4.17. The data were then fed into both the new NMDS algorithm and the novel spring model of Section 5.3. Figure 5.19 shows that the fast NMDS algorithm produces a more accurate layout than the hybrid spring model. The data were sampled from a set of concentric circles and, as can be seen, the configuration has been recovered by the fast NMDS algorithm, while the hybrid spring model has only produced a rough layout.

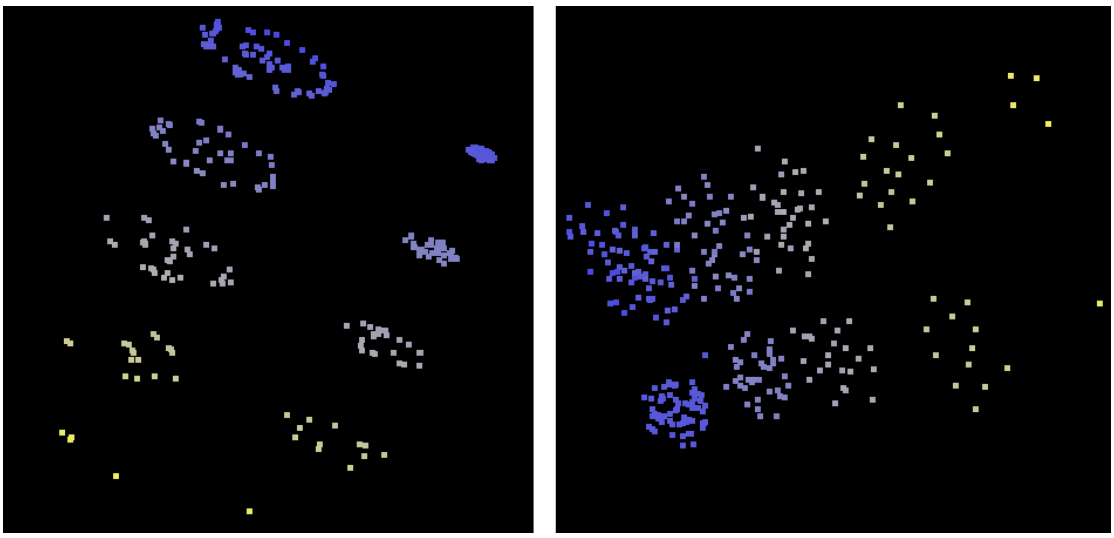


Figure 5.19 The image on the left shows a layout of proximity data obtained by the fast NMDS algorithm. The layout on the right was produced by the novel hybrid algorithm described in Section 5.3.

The corresponding Shepard plots for the above layouts are shown in Figure 5.20. The plot on the left shows that the fast NMDS algorithm has recovered the shape of the function given by Equation 4.17, whereas the plot for the hybrid spring model has not recovered it with the same accuracy. Overall these results indicate that when presented with a large amount of proximity data, the hybrid spring model algorithm can be applied to quickly gain an overview, albeit at the expense of a less accurate layout. However, to gain a more accurate layout, the fast NMDS algorithm can be run on a sample of the data to get a glimpse of the finer local detail.

To carry on with the demonstration of the efficacy of hybrid algorithms, the next section describes a novel hybrid clustering algorithm developed by the author.

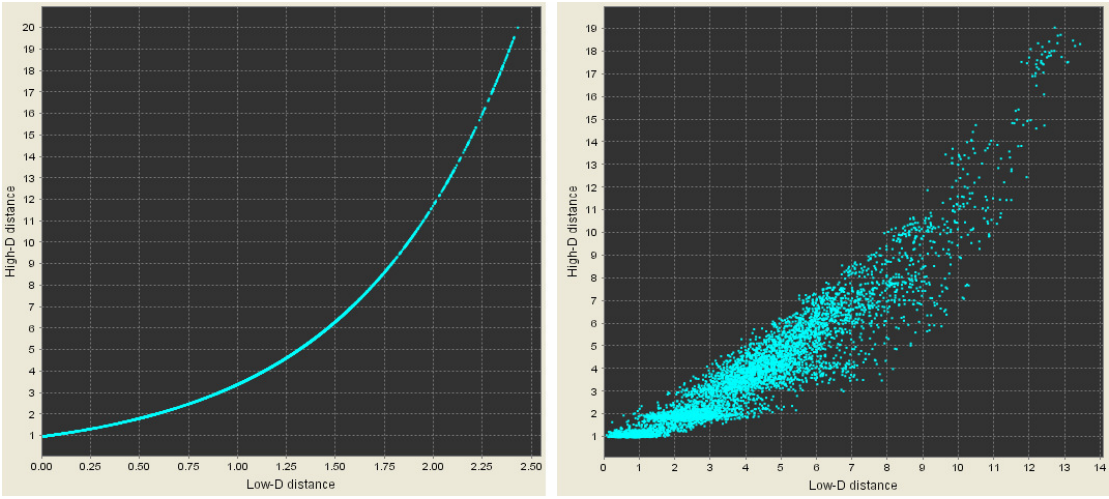


Figure 5.20 The Shepard plot on the left shows that the fast NMDS algorithm has recovered the function relating proximities to real Euclidean distances. The plot on the right shows that the hybrid spring model was not able to recover this function with the same degree of accuracy.

5.5 Voronoi-based clustering algorithm

This section illustrates the author's design and implementation of a novel clustering algorithm. The algorithm is a hybrid combination of density-based and graph-theoretic clustering routines and is similar to CHEMELEON [KHK99], DBSCAN [EKSX96] and Ertöz et al.'s algorithm [ESK03] (see Chapter 3) in that it addresses the challenge of identifying clusters of different shapes, sizes and densities by identifying a small subset of points before 'growing' the clusters from them.

5.5.1 Preattentive cluster identification

The scatterplot is the predominant visualisation technique used in this thesis. Groups of similar data points are immediately recognisable because their proximity forms clusters and other structures that appeal to our perception. This is because the human visual system has the ability to automatically process the visual stimuli. Certain patterns and Gestalt qualities stand out without requiring attention to any specific part of the visual field. This unconscious and preattentive processing is what allows information visualisations to accelerate the rate at which humans perceive information.

By reducing the dimensionality of data so that they can be represented in a spatial substrate such as a scatterplot, preattentive processing allows the user to recognise structure and patterns very quickly. Colour and other retinal properties can also be used to augment this basic property of the scatterplot by providing visual stimuli that attract attention.

The combination of preattentive and attentive processing in visualisation also allows the user to build a cognitive map of a layout. The visual stimuli become part of the users' internal representation that aids orientation, navigation and browsing between pertinent regions. This is the underlying concept of spatial location memory.

In Rob Ingram's work [IB95a, IB95b], this quality is referred to as *legibility*, a term borrowed from the context of city planning. Ingram's work was inspired by Kevin Lynch's book "The Image of the City" [Lyn60] in which a study reveals that urban dwellers' development of cognitive maps of a city is enhanced by features such as landmarks, nodes, and paths. By transposing this theory into the domain of information visualisation, Ingram's aim was to accelerate the users' creation of a cognitive map of the information.

Other researchers including Chalmers [Cha93, CIP96], Brodbeck et al. [BCLC97], and Wise et al. [WTP*95] have adopted a similar approach by representing data via this landscape metaphor. It is for these reasons that a clustering algorithm has been developed to automatically detect potentially interesting structure in the layouts produced by the algorithms discussed above. The layouts are segmented into clusters and then coloured to distinguish them. This helps preattentive processing guide the user to interesting parts of the layouts before any conscious effort is required to examine them.

5.5.2 A novel clustering algorithm

Layout algorithms tend to produce a 2-d point configuration in which it is left to the observer to perceive any patterns of interest. To aid the user in this respect, a clustering scheme is applied to 2-d layouts to help to differentiate points that contribute to interesting structure. This partitions the configuration of points, explicitly highlighting existing patterns. It also enables the user to easily select clusters for brushing and linking with other views, or to extract a subset for further processing. To achieve this, a partitioning algorithm is required that is unsupervised, able to detect clusters of varying size, shape and density, and sufficiently fast for user interaction not to be hampered. Many partitioning algorithms are described in the literature. However, each behaves differently according to characteristics of the data set (such as distribution and variable types) and input parameters (for example, the number of centroids used in K-means [Mac67]).

The author has introduced a visual module in his HIVE software that implements a novel Voronoi-based clustering algorithm. The goal was to produce a clustering scheme where the user does not have to select or adjust abstract parameter values to obtain an effective partition, nor wait too long for it to complete. This is a two-stage algorithm, drawing from concepts underlying density-based and graph-theoretic clustering. The first stage finds significant areas of similar density in the point configuration. These 'hotspots' are then fed into the second stage of the clustering algorithm where their neighbourhood relationships are used to expand the groups in an agglomerative way. The algorithm effectively extracts small contiguous groups of points from regions of similar density and then uses these as the seeds from which the clusters are grown.

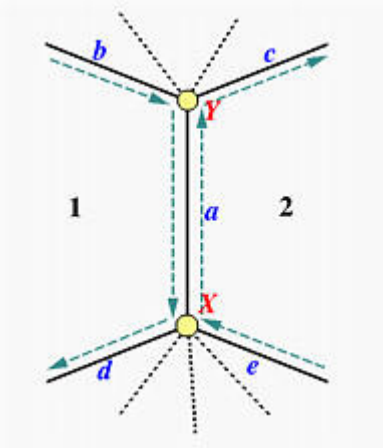
The next sub-section describes and justifies the Voronoi algorithm that was implemented as the basis for the clustering. The last two sub-sections detail the 'first cut' version of the algorithm and the final implementation along with experimental results.

5.5.2.1 Voronoi diagrams

The point pattern of a 2-d scatterplot can be segmented into different areas so that each point is contained by a convex polygon. If the polygons are contiguous and the portion of space within each polygon is closer to the contained point than any other then these regions form a tessellation of the plane called the *planar ordinary Voronoi diagram* [OBSC00, Aur91].

In HIVE, the Voronoi diagram is implemented using an incremental method. A seed Voronoi diagram is first drawn for three dummy points that are arranged as an equilateral triangle. To build the Voronoi diagram for the point pattern, new points are added, creating new edges and vertices to gradually fill out the structure. This approach was chosen because it is the most robust technique for handling degeneracies such as those due to co-circularities in the point pattern. The incremental method is also one of the fastest techniques, capable of achieving $O(N)$ time complexity on average.

The Voronoi diagram was used as the clustering basis for two reasons. The first is because the underlying data structure can be used to efficiently store and retrieve information such as that for quickly finding points within a region of the plane, e.g. nearest-neighbour searching, and for easily determining incidence relations between Voronoi edges, vertices and polygons. In the author's implementation, a standard structure used in geometric modelling called the *winged-edge* data structure [Bau75] is employed (see Figure 5.21).



vertices		faces		left traverse		right traverse	
start	end	left	right	pred.	succ.	pred.	succ.
x	y	1	2	b	d	e	c

Figure 5.21 The winged-edge data structure. This maintains a compromise between a compact representation of the Voronoi geometry and fast retrieval of vertex, edge and polygon incidence relations. Edges are used to keep track of the geometry and are represented by arrays of start and end vertices, polygon faces, predecessors and successors.

The second reason for utilising the Voronoi diagram is because it provides a very flexible platform for experimenting with different clustering routines. The Voronoi diagram has a dual graph called the Delaunay triangulation, which is formed by drawing arcs between points contained in adjacent Voronoi regions. Among the sub-graphs of the Delaunay triangulation is the Minimum Spanning Tree (MST) from which familiar graph-theoretic clustering schemes such as single-link, complete-link and average-link are derived.

Since the Voronoi diagram represents the point pattern as a set of areas, it is easy to devise density-based clustering. It is also relatively easy to derive the Delaunay triangulation from the Voronoi diagram and vice versa. This means that the Voronoi diagram bridges the gap between the two families of clustering techniques: density-based and graph theoretic and therefore lends itself to experimenting with hybrid conjunctions. See Figure 5.22 below.

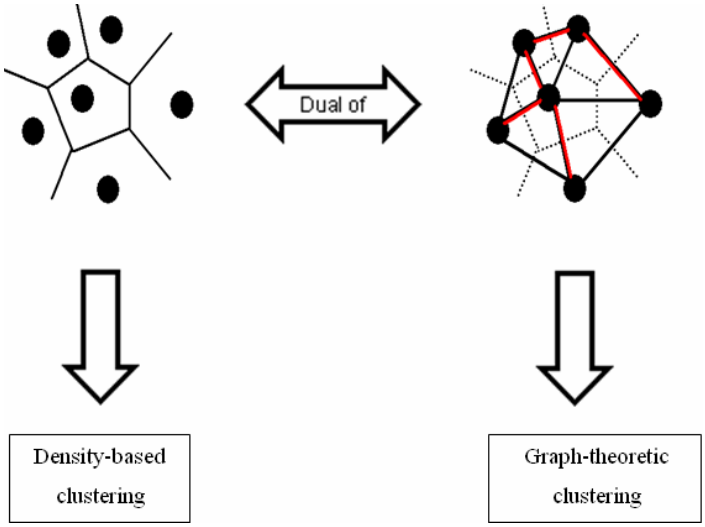


Figure 5.22 The top-left graph is the Voronoi diagram, which can be used for density-based clustering. The top-right graph is the dual of the Voronoi diagram, the Delaunay tessellation, whose sub-graphs include the minimum spanning tree (shown here in red) and therefore is suitable for graph-theoretic clustering applications.

Promising results have been obtained from clustering algorithms based upon the Voronoi tessellation and its dual, the Delaunay triangulation [OBSC00]. Duyckaerts et al. [DGH94] describe an algorithm where a 2-d partition is obtained from the Voronoi tessellation by first selecting the Voronoi polygon with the smallest area, then creating a new empty cluster, and then appending neighbouring polygons whose areas are less than a pre-specified threshold. This threshold is proportional to the area of the smallest polygon. Once the remaining polygons exceed the threshold, and the cluster therefore cannot expand any further, it is removed and the process is repeated, starting with the smallest of the remaining polygons. This approach can be classed as a density-based clustering because the clusters are defined by patterns of points that are similarly spaced. An alternative approach has been taken by Estivill-Castro [CL02], where the Delaunay diagram is used in a graph-based clustering. Here local and global point neighbourhood relations are considered when producing the clustering. This method is desirable because it is an 'argument-free' approach i.e. it does not require the user to estimate any algorithm parameters or make assumptions about the data set. Another advantage is that graph-theoretic clustering is not sensitive to the input order of the data. This approach, again, is applied to partitioning points on a plane. Interestingly, this notion of taking both global and local connectivity into account in graph-theoretic clustering is not often explicitly addressed in the clustering literature. However, in their exposition of the density-based OPTICS algorithm [ABKS99], Ankerst et al. acknowledge the fact that with

most real-world data sets global density or connectivity parameters cannot be used to pick out clusters of varying density.

Andrews et al. [AKS*02] have taken another novel approach to clustering via the Voronoi diagram. However, their use of the diagram is to use multidimensional scaling to lay out different levels of a predefined document hierarchy and then encapsulate whole clusters within individual Voronoi regions. It should be noted that this is different from the approach adopted here. The goal of the algorithms described in the following sub-sections is to obtain the clustering without any *a priori* class information.

5.5.2.2 Interactive density exploration

To examine how the Voronoi diagram can depict varying density in a point pattern, a slider was added to the Voronoi visual module in HIVE to apply an (optional) area or perimeter threshold. As the user changes the value with the slider, Voronoi polygons whose area or perimeter is smaller than the threshold are filled with a colour. Duyckaerts et al. [DGH94] developed a density-based clustering algorithm using this threshold technique (Section 3.3).

It can be seen that as the threshold is reduced, the highlighted regions recede to areas of higher density and clusters of various shapes and sizes stand out. However, because of the global threshold, clusters of different density cannot be highlighted simultaneously when their densities are at either side of the threshold. This creates an effect where clusters appear, shrink and then blink out of existence as the user moves the slider to reduce the threshold value. This is illustrated in Figure 5.23.

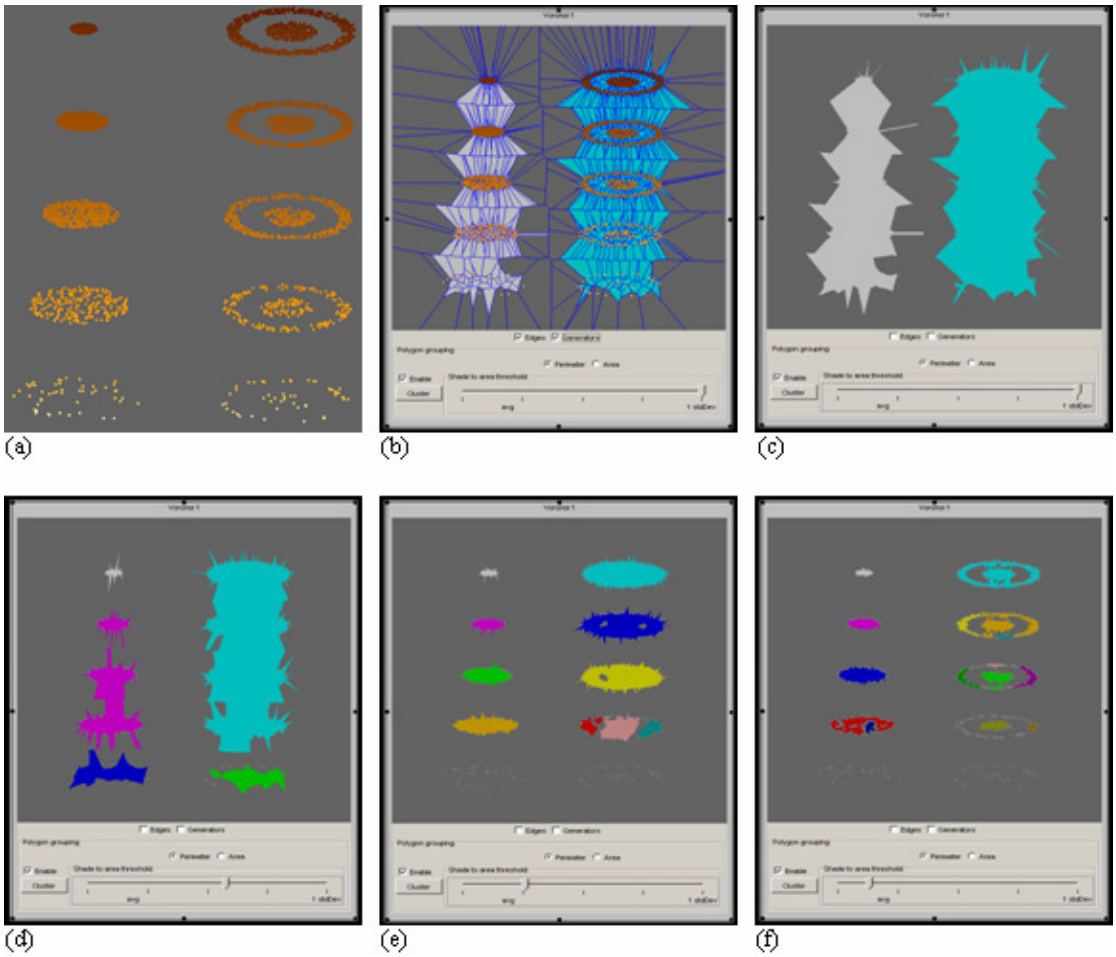


Figure 5.23 Data set used for clustering (a). Perimeter threshold reduced gradually (b) - (f).

Figure 5.23(a) shows the data set used to demonstrate the effects of modifying the perimeter and area thresholds. Dense clusters appear at the top of the layout, and density gradually reduces towards the bottom. Figure 5.23(b) shows the points, Voronoi regions and shading when the user initially reduces the perimeter threshold just below the maximum value. Figures 5.23(c) to (f) show how clusters of lower density gradually fragment then vanish leaving only the densest clusters as the threshold is reduced. For example, the blue and green clusters visible at the bottom of Figure 5.23(d) disappear for the threshold set for Figures (e) and (f).

The author's colleague at Nickleby HFE Ltd. has used this interactive mechanism in exploring text data in HIVE (see Chapter 7). He creates layouts representing text documents, and then adjusts the density slider until various clusters appear. He then selects these clusters to view the text they represent. This allows him to explore the themes present in the corpus.

The next two sub-sections detail a clustering scheme that automatically reduces the perimeter threshold while keeping track of clusters as they appear and diminish. This information is then used to resolve clusters of varying density as well as shape and size.

5.5.2.3 Clustering algorithm - first version

The new clustering algorithm consists of two steps. The purpose of the first step is to find groups of points that lie in areas of similar density. Recall that one does not want to burden the user with the task of specifying any data-dependent parameters. A recursive threshold-reduction approach is therefore applied in finding the density 'hotspots'. This is achieved by first calculating the smallest Voronoi polygon perimeter and then the standard deviation of polygon perimeters. These are then used as the smallest (t_{min}) and largest (t_{max}) thresholds for perimeter size in determining whether to append a polygon to the cluster. The author has found in most cases that when the t_{max} threshold is set to the standard deviation of polygon perimeters, all of the points are grouped into one cluster. By then slowly reducing the threshold value by a small amount a, this large cluster eventually breaks up into smaller clusters, each representing an area of similar local density. Pseudocode for this approach is provided below in Figure 5.24.

It should be noted that the Voronoi polygon perimeter is used as the dynamic threshold value because it tends to have a narrower distribution than area and therefore it takes less time to move the threshold across its range while keeping the decrement steps at low values. Also, t_{max} is initially set to one standard deviation of polygon perimeters because from the author's experience with lots of data sets, this initially includes all of the points in one large cluster.

When a contiguous region of similar density is found that contains at least twice the minimum number of points for a cluster ($2 * c_{min}$), the procedure is recursively applied with a reduced perimeter threshold to further split this cluster. The amount by which the perimeter threshold decreases, a, is equal to $(t_{max} - t_{min}) * 0.01$. This decrement is made locally within the recursive function. When a cluster is smaller than $2 * c_{min}$, it is added to the list of clusters found so far, and after the recursive base case is satisfied, the list of clusters is returned and input to the second stage of the clustering algorithm.

```
1. set t_max = st. dev. of Voronoi polygon perimeters
2. set t_min = smallest Voronoi polygon perimeter
3. set minimum number of contiguous polygons that represent a
   cluster, c_min, (e.g. c_min = 10)
4. set current perimeter threshold t_c = t_max
5. make list of all polygons: pList, and for clusters: cList
6. find contiguous groups of polygons within pList whose
   perimeters are less than t_c, and each group having at least c_min
   polygons. Create a list of these groups, gList.
7. while (t_c > t_min)
  7.1 for each group in gList
    7.1.1 if group.size ≥ (2 * c_min )
        make new pList, add group's polygons
        set t_c = t_c – a
        go to step 6 (recursive call)
    7.1.2 else
        add group to cList
8. return cList
```

Figure 5.24 Pseudocode for the first stage of the clustering algorithm.

The value *a* ensures that the threshold will always decrease in a constant number of steps. However, since *gList* initially contains all Voronoi polygons (in one large cluster) and each group of polygons is recursively split, the time complexity of this stage of the algorithm is $O(N \log N)$.

The first stage of the clustering algorithm returns lots of small clusters representing areas of similar local density. In the second stage, these cluster seeds are grown into larger clusters, which form a partition of the point configuration. This is achieved by starting with the smallest Voronoi polygon, in the cluster with lowest average inter-object distance, and cumulatively expanding the cluster by adding neighbouring points whose distance to the point in the polygon is less than or equal to the average inter-point distance. When no more points

are within this distance to any point in the cluster, the cluster is complete and the next cluster seed is examined. Figure 5.25 provides pseudocode for this step.

```
1. sort list of cluster seeds, cList, from stage 1 in ascending
   order by average inter-object distance, d_av
2. delete all points, except that with the smallest Voronoi
   polygon, from each cluster seed
3. for each cluster seed, c_s, in cList
   3.1. set Δ = c_s's smallest polygon point
   3.2. for each neighbouring polygon point, ν of Δ
      3.2.1 if dist(ν, Δ) ≤ d_av
          add ν to c_s
          set Δ = ν
          go to step 3.2 (recursive call)
4. return cList
```

Figure 5.25 Pseudocode for the second stage of the clustering algorithm.

This stage of the algorithm returns the partitioning of the point configuration. Since the average inter-object distance of the cluster seed is used to determine whether the nearest neighbours of each member point are added to the cluster, clusters of different shapes can be found. The time complexity of this stage is $O(N)$ because it is not necessary to examine a neighbouring polygon once it is a cluster member. The time complexity of the Voronoi algorithm and stage 1 is $O(N \log N)$ and therefore the overall time complexity of the clustering algorithm is $O(N \log N)$.

To evaluate this algorithm's performance, it was run on the data shown in Figure 5.19 as well as benchmark data sets that were used to test the CHAMELEON [KHK99] clustering algorithm and Ertöz et al.'s approach [ESK03]. The latter data sets contain clusters of different shapes, sizes and densities along with noise points in between them; see Figure 5.26.

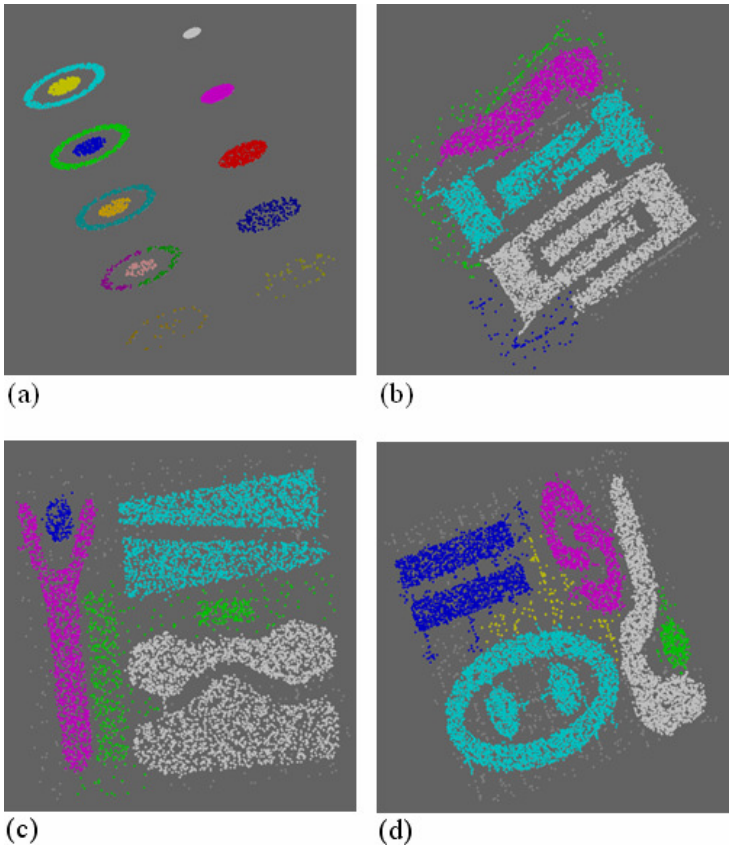


Figure 5.26 Clustering results of the first version of the clustering algorithm on benchmark data sets. Points of the same colour are deemed to be in the same cluster.

From Figure 5.26(a) it can be seen that all of the clusters have been identified, although one has been overly fragmented. However, the benchmark data sets (Figures 5.26 (b) to (c)) indicate that the noise between clusters affects the algorithm's ability to differentiate between some of the clusters. This is due to a chaining effect – a problem of which the familiar single-link algorithm is susceptible. The distance calculation in step 3.2 of the pseudocode (Figure 5.25) causes the clusters to merge across *bridges* built by the noise points.

5.5.2.4 Clustering algorithm - final version

To overcome the fragmentation shown in Figure 5.26(a) and the chaining effect evident from the benchmark data sets discussed above, it was realised that the probability of a point's membership of a cluster could be used to help decide whether it is included in that cluster. This probability is determined by examining the dendrogram created by the first stage of the clustering algorithm described above. The result of reducing the perimeter threshold in the above clustering strategy produces a hierarchy of clusters (dendrogram), the top level being

one cluster that contains all points when the threshold is set to t_{max} and the lowest levels (leaves) consist of clusters of size c_{min}.

Given a pair of points i and j, the probability of them being in the same cluster $p(i \cap j)$ is determined as follows:

$$P(i \cap j) = \begin{cases} 1 & \textit{If i and j appear in the same leaf} \\ \sqrt{L / L_d} & \textit{If i and j do not appear in the same leaf} \end{cases}$$

Where L is the dendrogram level in which i and j diverge, and L_d is the deepest level of i or j. To determine whether two points i and j should be part of a cluster, $p(i \cap j)$ is multiplied by the average inter-object distance of the cluster seed d_{av}, and if the distance between the points is below this value then they are deemed to belong to the same cluster. This modifies step 3.2 of the pseudocode (Figure 5.25) as follows:

```
if (dist(i, j) * p(i ∩ j)) ≤ dav
    // i and j belong to the same cluster
```

This strategy improves the clustering because it uses density information of the point pattern to vary the distance threshold. The results of this modification are depicted in Figure 5.27.

Note that this clustering routine is opposite to that of CHAMELEON [KHK99]. Instead of initially partitioning the graph before merging sub-clusters via an agglomerative hierarchical algorithm, the current technique first uses a divisive hierarchical algorithm and then creates the partition by growing the cluster seeds found in the lowest level of the dendrogram.

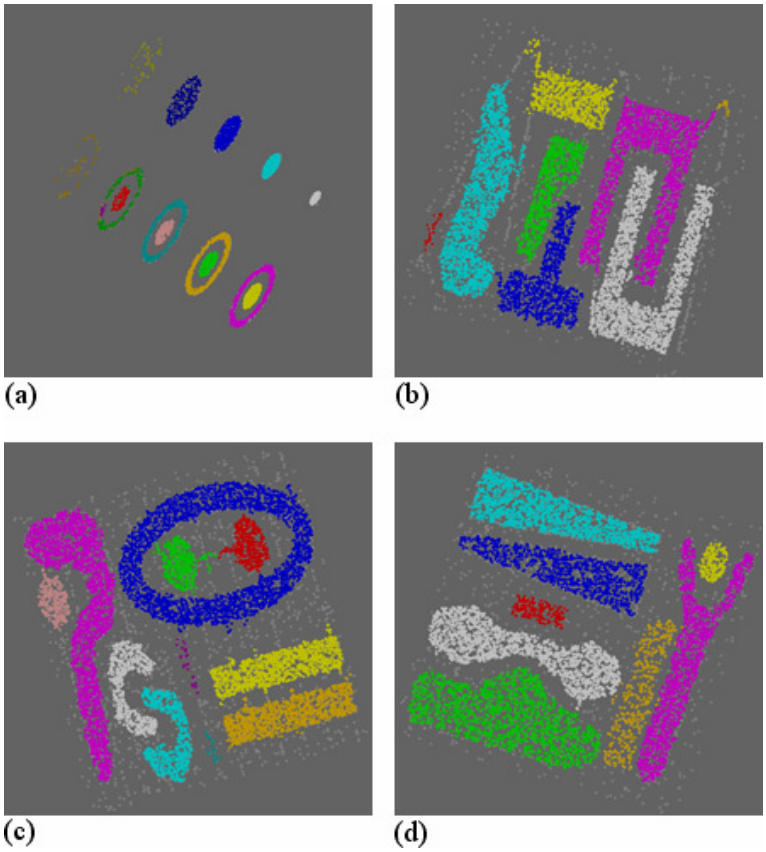


Figure 5.27 Clustering results of the final version of the clustering algorithm on benchmark data sets. The modified algorithm correctly identifies most clusters.

From these results it can be seen that this algorithm performs very well on the data sets. It seems that its objective has been met as discussed in Section 5.5.2: The time complexity of the algorithm remains at $O(N \log N)$, the user does not have to manually adjust any parameters and it can detect clusters of varying shape, size and density. The only drawback of this algorithm is that its time complexity increases exponentially with data dimensionality. This overhead is incurred because of the computation of the Voronoi diagram. However, since the intended application of this algorithm is to augment 2-d layouts produced by dimension reduction algorithms, this is of little concern.

This algorithm has a valuable role in the hybrid approach to algorithm development. As well as being a hybrid algorithm, consisting of a density-based stage and a graph-theoretic stage, it can comprise a useful component in a larger hybrid algorithm. The clusters that are produced are data aggregates and the routine can therefore be considered as a method for cardinality reduction. This would suggest that its output – a transformation of the input data – would be suitable for feeding into other algorithmic stages to carry out further processing. Worked examples of this are illustrated in Chapter 8.

5.6 Conclusions

This chapter began by describing how hybrid algorithms found in the literature are able to overcome many of the shortcomings of individual algorithms. This was followed by a demonstration and discussion of two new hybrid spring model algorithms, a fast version of Shepard's non-metric multidimensional scaling and a novel clustering algorithm. It has been shown how the hybrid approach to designing clustering and layout algorithms can improve performance in terms of run times, layout quality and cluster detection.

The author's work in this area prompted the development of a software environment for creating, evaluating and using such algorithms. This system is called HIVE (Hybrid Information Visualisation Environment) and is designed to be extremely flexible for prototyping dimension reduction and clustering routines. Most of the figures used in the preceding chapters were generated by HIVE. The next chapter provides a discussion of visualisation environments and their design. This will provide a basis for describing the design and implementation of HIVE in Chapter 7.

6. Visualisation environments

This chapter details a review of the literature regarding the design and development of visualisation environments as a requirements gathering phase and precursor to the implementation of HIVE. Information visualisation environments, like their scientific visualisation counterparts, are abundant. Although the common goal of these applications is to turn data into visual information, there are substantial differences in the typical architectures and modes of use associated with each. The aim of this chapter is to draw, from the fields of information visualisation and scientific visualisation, a discussion of the major design concepts and interaction mechanisms.

One of the goals of the author's work was to produce a hybrid algorithmic framework (HIVE) that supports adaptive and interactive visual information seeking. The key features of this framework are:

Adaptability to data cardinality and dimensionality – to cope with evolving databases and diverse data sets, the core of the framework consists of a hybrid algorithmic architecture. This architecture borrows from the modular data-flow model familiar in scientific visualisation applications and is the inspiration for the name of the framework, HIVE: Hybrid Information Visualisation Environment. The hybrid algorithmic approach essentially uses the complexity of the data, in terms of cardinality and dimensionality, to efficiently help steer the execution through a network of algorithmic components to produce the visualisations.

Adaptability to diverse variable types and heterogeneous data – the proposed system is able to work with different types of variables in the input data including nominal, ordinal and quantitative and a mixture of these. A variable-type transformation process to produce continuous vectors as required by the algorithmic architecture facilitates this.

Interactivity – the framework is comprised of a multiple-view system and tight coupling between views because a consistent flow of interaction is essential in data exploration. Direct manipulation of visual structures is also important in providing flexibility in exploratory analysis.

Abstraction management - to maintain integrity of the visual representations, abstraction management is required to manage the relationship between the rendered information across different abstractions and the underlying data.

HIVE allows analysts to gain insight into latent complex relations within abstract data and helps algorithm designers in building novel hybrid algorithms. The framework combines information visualisation techniques together with interactivity and computational algorithms to produce the visualisations, thus affording visual information seeking. Although the theory of Knowledge Management (KM) is beyond the scope of this thesis, the overall intention of the framework is to provide a system that will form part of the knowledge management process. This is by transforming data into information – by adding value to them – and helping people transform information into knowledge by prompting better understanding [Spi00].

Before embarking upon the design and development of HIVE an extensive review of the literature was carried out to determine how HIVE should look and feel and how interactive mechanisms should be formed to effectively guide the user in his or her work. The study, which was essentially a requirements gathering phase, began by looking at the data-flow architecture adopted in many scientific visualisation systems (Section 6.1). Such a model, in tandem with visual programming, has proven to be extremely flexible for creating visualisation applications and for computational steering. This discussion is followed by an exposition of some useful facets of information visualisation systems (Section 6.2) such as the use of multiple views, and the notion of the *information workspace*. Finally, Section 6.3 describes some fundamental concepts in designing information visualisation environments.

6.1 Data-flow model

Computer-based visualisation was first established in its application in the fields of engineering and the physical sciences. The very nature of these fields, in dealing primarily with physical data, meant that computer visualisation was a natural step in the evolution of tools for engineers and scientists – since the data are intrinsically spatial, it would be appropriate to represent them spatially and therefore graphically for their analysis. It is known that for humans, space is perceptually dominant [CMS99]. The elements of the physical data often map directly onto a set of two or three axes in graphical plots that make the myriad data 'jump out' at the beholder and naturally appeal to human perception. In light of this,

visualisation was given formal recognition of its application in computing to amplify cognition.

In this section, some examples of scientific visualisation systems that utilise visual programming and a data-flow model for execution are given. Systems based specifically on the data-flow architecture are the focus here, primarily because the underlying model is extremely flexible and has been widely adopted. The key features of these systems are discussed to demonstrate their relevance to the HIVE framework.

6.1.1 Visual programming

At around the time when scientific visualisation was being established, the concept of *visual programming* was also becoming established. Conventional programming languages, whether high-level or low-level, tend to be built around a vocabulary where the 'words' consist of primitives (characters). Such primitives in the *English* language are the letters of the alphabet. With this in mind, it can be said that the language constructs are essentially lists of primitives and are therefore one-dimensional representations.

Visual programming languages, on the other hand, are at a higher level of abstraction than conventional languages. Haeberli [Hae88] states that a visual programming environment is any system that has adopted a graphical 2-d notation for the creation of programs. The visual structures that make up the vocabulary of these programs are essentially representations of well-defined aggregates and the (direct) manipulation of these aggregates means that complex programs can be produced more easily than with conventional languages. This is because the abstraction allows a greater degree of code or function reuse and the workings of the programs themselves are more readily understood and communicated due to their visual and spatial properties. It can also be argued that if the manipulation of the visual constructs is flexible enough, for example, the user may wish to place them arbitrarily on the display surface, then this allows a larger margin of freedom for externalising the plans and thoughts of the user. This 'informality' in the notation is the basis for one of Green's *cognitive dimensions of notations* [Gre89] called *secondary notation*. Secondary notation will be discussed in greater detail in Section 6.3.

Visual programming has become a significant aspect of scientific visualisation applications. It has been recognised that many monolithic precompiled applications with static interfaces tend not to be flexible enough to cater for many of the needs of scientists and engineers. The functionality and the user interface are 'set in stone' and as a result the system is only applicable to a few specialised tasks. The application designer cannot anticipate all of the tasks that the target audience will want to perform. It is also often the case that these

systems do not support interoperability between applications and where they are extensible they require significant amounts of coding on the part of the user. However, visual programming has been used in a number of systems to address these issues and allows the user to build his or her own task-specific applications without the need for conventional programming expertise. By breaking applications down into sets of functional modules, and rendering them as graphical representations, the user may use direct manipulation to join them together into networks, where the links are data-flows. With a variety of different modules for data input, transformations, rendering and parameter control etc., fully functional applications can be 'thrown' together relatively easily by the user to provide custom visualisation solutions.

This data-flow model is interesting because it reinforces the merits of visualisation in computing. With regard to the means-end relationship of scientific and information visualisation, the *means* are a visual process and the *end* result is a tool that produces the visual information originally sought after – visualisations are very useful for producing other visualisations.

6.1.2 Data-flow architecture

Before visual programming was available in scientific visualisation tools, the functional components of the tools were hidden from the users and they had no control of the flow of data between them. The stream of data from input through calculation functions to mapping, filtering and rendering graphics and their control was pre-set and the scientists and engineers had to make do as best as they could for their tasks. In the words of Haeberli [Hae88], *"Instead of the user driving an application, the user is often driven and constrained by the application."*

The concept of visual programming came as a solution to these problems and became a paradigm of moving away from these monolithic and static applications, providing integrated environments where the user could customise his or her applications without programming expertise. The visual programming in the application design cycle took the form of a data-flow architecture. In this architecture, users are presented with a library of modules, application components, that have specific functions. The users can select which modules will be useful in their application and draw, via direct manipulation of graphical representations, a block diagram and create connections between modules for the data to flow through (see Figure 6.1). This quick and easy process meant that the scientists and engineers could spend most of their resources on the problems being studied instead of dealing with the overhead of re-coding and configuring monolithic applications.

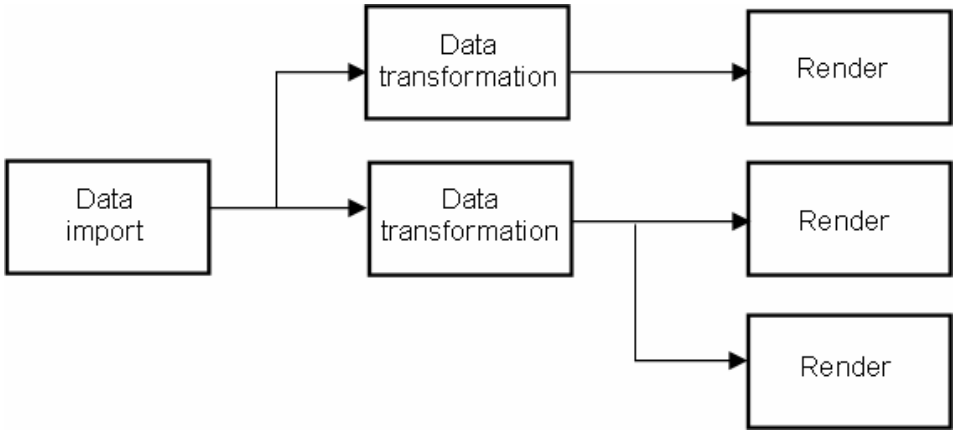


Figure 6.1 An example of a data-flow architecture. Blocks represent functional modules and the arrows represent the flow of data.

The data-flow model is appealing because it is visual and it is analogous to water flowing through a pipe, hence its description as *programming by plumbing* in [AT95a]. The mental model it instils in the user is intuitive and makes it easier to learn. It also simplifies control and navigation through the resulting applications.

The visual design cycle produces an *executable flow network* [UFK*89], which is essentially a directed acyclic graph. In this graph the modules (nodes) are at a higher level of abstraction than conventional program procedures but lower than a complete application. This abstraction borrows from Object Oriented Programming (OOP) and as a result also benefits from code re-use, polymorphism and extensibility. Integrated visualisation environments are more readily extensible because new modules can be produced by third parties for use in the data-flow architecture, provided that an appropriate Application Programming Interface (API) has been implemented. Some more advantages of the data-flow paradigm are as follows:

- ***Parallel processing*** – the architecture naturally lends itself to implementation on parallel processing platforms and distributed environments. Branches of execution in the flow network may be carried out simultaneously resulting in speed-ups for calculation and simulation. Research in this area was carried out for the Vipar (Visualisation in Parallel) project in the Computer Graphics Unit (CGU) at the University of Manchester [LGH02].

- ***Interoperability*** – systems implementing the architecture are more likely to be compatible with other data processing and visualisation applications. Inter-application communication modules can be part of the framework. In [AT95a] a uniform data model is in place for unifying access to imported and exported data.

- ***Collaboration* and communication** – if the data-flow architecture's implementation is distributed across multiple platforms for processing then collaboration between users comes as a natural extension to the framework.

- ***Animated, interactive simulation*** – previous scientific visualisation systems often carried out calculations in batches and the results were only available when the calculations were finished. In the data-flow paradigm, modules for changing calculation parameters and caching intermediate results for animations afford the user interactive intervention to *steer* simulations of the processes being visualised [AT95a, Hae88].

- ***Traceable computation*** – in relation to the above point, while simulations are being run, the branches of execution can be recorded as the calculations are made. When interesting anomalies arise in the generated visualisations, the calculations can be halted and backtracked to a previous point in the audit trail, allowing the user to restart from that point and modify the calculation parameters to gain more insight. This is implemented as a *history tree* in [BBB*93].

- ***Subjunctive presentation*** – with mechanisms such as a *history tree*, simulations can be run simultaneously from various points in the path of execution but with different parameter settings. This facilitates the *what-if* scenarios described in [Lun99] for comparison of different outcomes.

- ***Appropriation*** – by allowing the users to carry out the *plumbing* in their applications, they have the ability to match the tool to their tasks. This goes beyond mere customisation because the users can implement applications that the original system designer did not anticipate.

- ***Visualisation*** – by using visualisation in the design cycle the notational constructs appeal more to human perception. The user is afforded a higher awareness of the propagation of data in the application and is therefore made more efficient in constructing and debugging.

Having described most of the virtues of the data-flow architecture adopted in scientific visualisation environments, it must be said that there are some limitations as a result of the architecture. However, as will be seen later, these are not intractable.

In [AT95a] the authors pointed out the fact that there are cases where some modules in the data-flow network executed needlessly. Such instances arise when a module is in the path of execution but its output is not needed at that time, i.e. it should be bypassed. Another case is when a module on the path of execution will not change the state of its output from its last execution and therefore its current execution would be superfluous. These points imply that the data-flow network should not be static. Internal states and user actions can, and should, influence the path of execution which, as well as creating computational overheads, can also provide opportunities for optimisation.

Another concern in the data-flow architecture relates to visual programming and the obvious overhead incurred by the user in creating the application. The resulting applications may be tailored precisely to the user's needs but they require work in getting them into this state. The data-flow architecture promotes exploratory application composition, but it is only useful if the user has a good understanding of the functions offered by the system. Designers of visual programming languages must be careful in ensuring that the system is usable. Traditional and typical static-interface applications, on the other hand, allow the user to explore, experiment and concentrate on his or her tasks almost immediately albeit at the expense of being less flexible for a wider variety of tasks.

In [UFK*89] an important limitation is highlighted regarding the visual representation of the data-flow. Upson et al. point out that in some applications the data-flow graph can become very large and unmanageable if the functional level of abstraction in the modules is too low. This would present a problem if the data-flow representation were to be relied upon for navigating through the visualisation application per se. There are, however, practical means of getting around problems such as these. For example, aggregation of visual components could reduce the clutter, or, zooming could provide overview and detail on demand. In Schroeder et al.'s visualization toolkit (VTK) [SML96] this problem is avoided by manipulating and representing the data-flow model purely in code, rather than visually. Written in C++, VTK is an open source library of classes that can be hooked together using its C++, tcl, Python and Java APIs. Although this combination of compiled and interpreted components provides for fast and flexible application creation, its lack of visual programming means that its data-flow network is not as readily perceivable.

6.1.3 Some examples

A brief discussion of three papers describing scientific visualisation systems that employ visual programming and a data-flow model will now follow. These systems are discussed in chronological order of their respective publication in the literature to illustrate the evolution of the data-flow concept. Although it may seem tautological, the purpose of this section is to highlight some of the approaches and motivation researchers have in adopting the data-flow architecture.

6.1.3.1 ConMan: Connection Manager

In 1988 Haeberli [Hae88] published a paper describing a system called ConMan (Connection Manager). Haeberli realised that existing systems were not flexible enough to cater for many of the diverse needs of their users and as a result he developed ConMan to address this issue.

In presenting his system, Haeberli adopted a culinary metaphor of making a sandwich in describing this situation. He likened the typical monolithic application to that of a pre-prepared sandwich, and in contrast to this, he went on to describe ConMan as a way of obtaining the ingredients and making your own sandwich. Through this metaphor Haeberli highlights one of the points made in the previous section: visual programming incurs an overhead for the user, i.e. she might be a bad cook and is unable to appropriate the ingredients to make a sandwich she likes.

This system is described as a visual programming language that uses functional modules (implied as verbs) and data-types (implied as nouns) as its vocabulary. Inspired by the UNIX pipe command for channelling data between processes (one-way inter-process communication), the modules in the system can be connected so that data can flow between them and the configuration of the pipes and modules is determined by the user's ultimate task. The types of module included in the system include render controllers for geometric shapes and bitmaps, and a tape module to record and play back animations of the generated views. Haeberli implies that the system is easily extended, as modules only have to declare their input and output port properties to the core system, which in turn maintains a queue of events to be dispatched to the modules concerned.

Implemented in C and running on the Silicon Graphics Iris Workstation, the ConMan system is an early example of the data-flow architecture and was limited in its application. Its applicable data-types were restricted to transformations, geometric shapes, RGB colours and bitmaps, and the data were piped in textual interchange format between modules. Hence it was not applicable to many scientific visualisation challenges involving large amounts of diverse data, but it did influence further research into data-flow architecture [AT95a, UFK*89].

6.1.3.2 Application Visualisation System

In a paper published by Upson et al. [UFK*89] in 1989, the Application Visualisation System (AVS) is described. The authors realised that at that time while graphics and computing hardware were becoming ever advanced, software development was not keeping up to take full advantage of the new powerful capabilities on offer. The typical scientific visualisation tools available to researchers were in the form of expensive inflexible monolithic applications, graphics libraries or animation packages. Graphics libraries required lots of low-level programming to use and animation packages worked only on pre-computed data and sacrificed processing power for high-quality graphics rendering rather than maintaining a balance between rendering and providing better interaction mechanisms. The proposed

solution to this was found in applying the notion of object-oriented visual programming and an interactive GUI in a system that allowed researchers flexible access to the hardware power without requiring programming expertise. It was intended that this would allow scientists and engineers to quickly build applications supporting 3-d interactive graphics and high computational power without low-level programming, thus allowing them to focus more on their area of study. The authors gave an example of how, for one user of AVS, it took a day to create a visualisation application for a task, whereas previously without AVS the user had spent two weeks coding an application for the same task.

In the development of AVS, Upson et al. determined the requirements that AVS would need to satisfy by modelling the typical process through which a scientist or engineer would go about simulating a physical system using scientific visualisation. Inspired by the uncovered requirements and other work in data-flow architectures such as ConMan, Upson et al. recognised many of the advantages to be found in the data-flow model, such as animation of simulations and being appropriate for parallel processing platforms. The direct manipulation in visual programming meant that it would be easier to use. The system would also be extensible by allowing new modules to be developed under AVS by making use of module templates that provide the housekeeping routines and only requiring the algorithms to be defined by the user. Third parties could also code modules in C, C++ or FORTRAN independently of the AVS system. The resulting applications would be cheaper because only new modules would have to be bought instead of entire new monolithic applications.

Going beyond the scope of ConMan and animation packages, AVS offered a more complete application in that it could be the producer as well as the consumer of data rather than just manipulating pre-calculated data. It integrates visualisation with the processes that create the data. Also, to promote inter-application compatibility, the range of applicable data-types is also extensible. Another advance was to make the data-flow model in AVS demand-driven to increase efficiency. Modules pull the data through the data-flow network as they are needed.

The authors of the AVS paper did however find a limitation of the data-flow visual programming paradigm. They realised that in complex applications the number of modules and pipes in the data-flow network could become very large and therefore unmanageable and that it would lose its value in providing an overview of the application. This, they said, would be affected by the level of abstraction used in the modules and implied that by making the abstraction higher without being too generic would help alleviate this problem.

Implemented in C++ on the X Window System, AVS provides three types of view: a data-flow network diagram for shaping the application; control panels, many of which are

created automatically from the parameter descriptions of the network modules; and output windows for visualisations.

Through the given application example in the paper by Upson et al., AVS was shown to be one of the first systems implementing the data-flow architecture that could tackle real scientific visualisation challenges.

6.1.3.3 IBM Data Explorer

IBM Corporation's Data Explorer (DX) and its extension of the data-flow model is described in a 1995 paper by Abram and Treinish [AT95a]. This is another system utilising graphical programming to allow users to create visualisation applications. By the time this paper was published, the full advantages of the visual programming data-flow architecture were well known and, as this paper shows, work was now underway to refine and optimise such systems to overcome some of the data-flow model limitations that have been described. The goal of this paper was to show how the data-flow model could be extended to support greater scientific visualisation challenges while maintaining its fundamental virtues.

The predominant issue that the paper addresses is that in the traditional data-flow architecture, some modules in the network may needlessly be executed. This superfluous processing obviously makes the system less efficient. The paper describes *side-effect* modules in the data-flow as being sources of output external to the network, for example, a visualisation on the screen or a data file for exporting. In some instances, in the flow of execution not all modules in the network will feed into these side-effect modules and will therefore not require execution. The DX system employed *graph evaluation* of the data-flow network and *conditional execution* strategies to determine which modules needed to be executed and which ones did not.

Another instance when a module does not need to be executed is when its inputs have not changed from the last time it was executed. This can happen when the user modifies another module's parameters, prompting it to execute and the network to process the new results. Modules in the execution path before the modified module will have the same input states as before and it will therefore be unnecessary to re-execute them. To overcome this, DX uses caching of partial results so that when a module is sent input, it determines whether it is different from the last time it was executed. If the input is the same then the module will merely retrieve the last results from the cache otherwise it will re-execute to produce new results. This caching is also the used to create a history tree similar to that of [BBB*93] which allows previous network states in a simulation to be re-visited and re-run with modified

parameters from that point. This naturally extends to creating animated simulations of the physical processes under study.

These modifications to the data-flow model mean that its performance is optimised and therefore more applicable to scientific visualisation challenges that present large amounts of data. As well as this, the DX system outlines a number of further extensions. Adaptability to new applications and data-types is facilitated by *a uniform data model* providing uniform access services to imported data, generated data and exported data of standard scientific classes described by shape, spatial location, rank and type, etc. This addresses the need to integrate diverse data sets, which is a prevalent issue in scientific visualisation. The visualisations per se have also been endowed with a richer variety of interactive mechanisms such as those for location probing, object selection and user-defined annotation.

DX and OpenDX (the open source version of DX) come with a comprehensive toolkit of pre-made modules for a large range of applications, and module polymorphism enhances reusability. It runs on Unix workstations and on PCs with Windows and represents a mature example of the visual programming data-flow paradigm.

6.1.4 Relevance of data-flow based scientific visualisation to the HIVE framework

The key features of the systems described above have provided inspiration in many areas of the HIVE framework. Those areas include the hybrid algorithmic architecture, the interaction model, extensibility and the completeness of the system with regard to data production as well as consumption.

At the core of the HIVE framework is a hybrid algorithmic architecture. The complexity of the input data, in terms of cardinality and dimensionality, steers the flow of execution through the various algorithmic stages in a similar fashion to the executable flow network apparent in the scientific visualisation systems discussed above. However, there are two points of departure from these systems. The first is where the data-flow network is formed. In the traditional data-flow model, the user explicitly designs the network via visual programming, whereas in the HIVE framework, the system can use the complexity of the data to determine the network. In this sense, it is more accurate to say that the HIVE architecture can *automatically* form a graph with the vertices being algorithmic modules and the edges being the data pipes. The second and most prominent point of departure from the traditional data-flow model is that the HIVE algorithmic architecture works to increase the computational efficiency of the system not just its flexibility.

In the outline of the hybrid architecture shown in Figure 6.2, the node labels refer to the input combination of the ranges of data dimensionality and cardinality respectively, e.g. *HH* means high dimensionality *D* and high cardinality *N*. The dotted lines represent a path through the algorithmic modules for performing transformations of the data. This will be discussed in depth in Chapter 7.

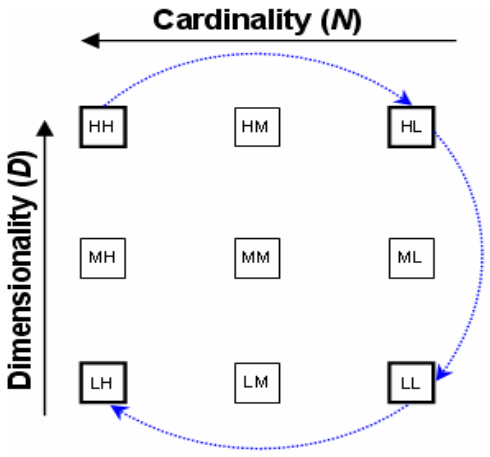


Figure 6.2 An outline of the hybrid algorithmic architecture.

It should also be mentioned that the HIVE framework is specifically aimed at information visualisation rather than purely scientific visualisation. With these differences aside, however, and by borrowing from the data-flow paradigm in scientific visualisation, potentially all of the advantages listed in section 6.1.2 are available with the bonus of greater algorithmic efficiency. As an example, consider extensibility: an Application Programming Interface (API) can be used to allow the extension of the framework without extensive programming. Algorithmic modules are 'pluggable' by declaring complexity in cardinality and dimensionality and also describing any natural visualisations that result from them. New algorithms slot into the framework to improve efficiency as well as to provide new visualisations.

Although the visual programming aspect of the systems described in this chapter does not apply so strongly as mediator in the algorithmic architecture of HIVE, the notion of visual programming is useful in considering how a user may interact with the system. North [Nor01] and North and Shneiderman [NS00a, NS00b, NS01] have evolved the concept of *snap-together visualisation* in a system where the user can link views so that specific actions in one view can propagate to change the appearance of connected views. Although snap-together visualisation is based upon the premise that multiple views in visualisation provides more insight [MS90, Rob98, BWK00] and that tight coupling between views is useful for filtering

data and maintaining consistency [AS94b , AT95b], the system is essentially utilising visual programming albeit at a higher level of abstraction. This notion of an *interaction pipe* coupled with the hybrid algorithmic architecture could be of benefit to the proposed HIVE system. The data-flow model will provide efficiency in the computational stages and the interaction-flow strategy will reflect this and will also provide a vehicle for appropriation to allow the user greater flexibility in the system's application. It should be noted that GeoVISTA Studio [TG02] is very close to this concept. Based upon JavaBeans™, this is a geocomputational system that utilises visualisation and visual programming to help analyse geospatial data. It includes an extensible library of algorithmic and visualisation components including k-means, SOM, parallel coordinates and visual classifiers. The user can select these components and connect them together into data-flow networks. However, it does not place as much emphasis on the flow of interaction between views, as provided in HIVE. While views can be coordinated, the flexibility of snap-together is absent. Also, unlike HIVE's proposed hybrid algorithmic framework there is no provision for the semi-automatic generation of algorithms.

6.2 Information visualisation environments

Information visualisation has its roots in scientific visualisation where the data typically are physically based and of low dimensionality. Information visualisation is applied to abstract data of arbitrary dimensionality – from very low to ultra high. There is also a greater variety of data types that are the focus of information visualisation (as Shneiderman has noted [Shn96]) and can be characterised by dimensionality (1-d, 2-d, 3-d, 4-d+), temporality (e.g. LifeLines [PMR*96], Gantt charts), hierarchy (trees) and relationship (networks/graphs). Therefore, more novel techniques of visualisation are required beyond those of the simple direct mapping of variables to a set of two or three orthogonal axes, as typically encountered in scientific visualisation. The literature provides a wealth of novel applications of spatial substrates, retinal variables and visual structures drawing from the fields of cognitive, gestalt and ecological psychology. Even though progress in these areas has allowed visualisation users to glean more information from the data under study, interaction has to be added to allow users to look at the data from different perspectives and to detect latent structure (or *see the unseen* [Rob98]). An example interaction technique is *tight coupling* [AS94b] between multiple views which provides a flow of interaction propagating from one view to the next.

Having interactive visualisations is all very well but these informative views of data must also be considered within the context of the user's task environment. The environment with its human and informational resources frames the users, their tasks, activities, goals and

the information together with the tools to nurture knowledge which may result in new actions applied in that task environment [Spi00]. To couple information visualisation with the user's task environment the concept of the *information workspace* was proposed in [CRM91]. An information workspace is a macro-environment that may reflect the resources in the user's physical environment but with the informational resources providing more nourishment through the effective abstraction and manipulation afforded by visualisation.

Information visualisations are very data-dependent and can be extremely complex. Designing effective visualisations is still a hit-or-miss craft. The visualisations themselves should be seen as part of a suite of tools in a workspace where people can efficiently make sense of and work with information to gain knowledge. The following subsections will focus on multiple-view visualisations, information workspaces and the process of knowledge discovery. The relevance of these areas to the proposed HIVE framework will also be discussed.

6.2.1 Multiple views for information visualisation

While a single static presentation can offer some insight into the structure of data, there are great benefits of utilising more than one view of the same data. Visualisations tend to be abstract representations of data and therefore multiple views serve to provide multiple representations, each of which offers a different perspective. These different perspectives can prevent the user from making misinterpretations from the data [Rob98].

There are two obvious choices in determining the configuration of multiple views in visualising information. They correspond to whether the views are displayed simultaneously or singularly over time. This is addressed by Baldonado et al.'s *rule of space/time resource optimisation* in their guidelines on using multiple views [BWK00]. A common example of the latter is found in animation where *frames* are displayed sequentially and rapidly to give the viewer a sense of motion. This sense of motion blends a whole series of perspectives on the data into one continuous view allowing the information to unfold with time. This could be interpreted as a special case of using multiple views and is most naturally applied to temporal data.

In the former case, however, the diligent use of screen real estate and the co-ordination of the visual spatial substrates is paramount in affording insight into non-temporal data. While animation might be effective in showing how data evolve over time, the use of simultaneous views can have several purposes depending upon the technique employed and the desired end result. The techniques of *focus-plus-context* and *overview-plus-detail* [CMS99] use one view as an overview of a large set or subset of data and another view as a

window into a specific, more detailed portion of this set. This is to afford the user an understanding of the data by the context and the detail within the wider context. Another related technique is to use *previews-and-overviews* [GMPS00]. Again, one view provides an overview while other views provide different levels of representation of parts of the overview. The purpose of this technique is to allow the user to rapidly probe the overview to seek out relevant information before deciding to look at it in more detail. A third example of effectively using multiple views is in *brushing-and-linking*. In this case data are represented as before, as visual artefacts across several views, and by selecting one or more of these artefacts in one view, the corresponding artefact(s) can be highlighted in the context of other views. See Chapter 2 for a detailed discussion of these methods.

The above examples are of co-ordinated views that rely heavily upon interaction to root out information and enhance navigation through data. However, even the mere spatial positioning of static views can be used to reinforce understanding as well as making efficient use of screen real estate. In the field of Ecological Interface Design (EID) [RS98], the importance of multiple views is realised in the Proximity Compatibility Principle (PCP). In describing PCP, Burns [Bur00] states that *"things are related to each other when they must be used sequentially within a task"* and this is subsequently used to justify the positioning of views. This is exactly in line with one of Green's cognitive dimensions, *side-by-side-ability* [HH99]. This describes the ability of multiple views to allow comparison without unduly burdening the user's working memory. This is reflected in [EW95] and [LRB*97] where the juxtaposition of views that have a common axis provides a visual *join* between the representations. Superimposition of transparent views can also enrich visualisation, although occlusion can be a problem.

Although multiple-view environments have many advantages as discussed above, they are not to be utilised without realising some inherent drawbacks. Typical examples of these drawbacks are in time overheads incurred in setting up task environments and switching between tasks. For an analysis task a user may have to open and configure several windows. This not only takes time but when many such windows are open, system resources may also be severely drained and the user must also bear a greater cognitive load. This is a common experience in GUI-based operating systems such as Microsoft's Windows. Being motivated by these overheads, Kandogan and Shneiderman proposed a system called *Elastic Windows* [KS97] to make efficient use of screen space and allow the user to invoke operations over several windows simultaneously to speed up task environment set-up, context switching and task execution. In the context of information visualisation, these cognitive-load and system-resource overheads were also realised by Baldonado et al. [BWK00] and merited a tentative

set of guidelines to the proper design of multiple-view systems. These guidelines serve to clarify when and how to use multiple views.

6.2.1.1 Animation

In stereotypical temporal animation the time dimension tends to be the primary independent variable of the data set. This means that other (dependent) variables can be mapped onto it providing a flow of transition in the view that seems natural. However, for non-temporal data distributions, i.e. time is not a major contributor to the principal components, animation of its distribution along one of its potentially many components may not be very informative. If, for example, one rotates a 3-d scatterplot, containing clusters, in arbitrary directions, many directions would not be good for revealing the clusters and their relationships. Techniques such as grand tours [CBCH95] and co-ordinated views can overcome this.

Cook et al. [CBCH95] describe a grand tour as being a dynamic view that presents a sequence of many multidimensionally scaled projections of the data. Rotating the projections smoothly across the data distribution produces animation with the 'smoothness' determined by the granularity of the sampling and interpolation employed over the data. This shows that time can be used to show the distribution of non-temporal data unfolding. However, since the search space of potentially meaningful projections is so large, animated views of random paths through this projection space may fail to show anything meaningful.

A special case of interactive animation that is similar to a grand tour is facilitated in force-directed layout algorithms such as the spring model [Ead84] for multidimensional scaling. With this type of algorithm, a two or three-dimensional embedding of the higher-dimensional data is gained by iteratively refining the low dimensional representations. As opposed to a grand tour, the 'projections' in this case are the low-dimensional positions of the data that continuously move towards an optimal layout. This motion is made with regard to a measure of global minima in the spatial embedding. With each iteration step, the low dimensional view of the data can be updated to animate the progress of the algorithm. In [BSLD98, RMC05], the user can intervene in this animation by dragging visual representations of the data to different positions and see how the algorithm reacts. Buja et al. [BSLD98] use this as *sensitivity analysis* because the stability of the layout can be observed with respect to the amount of movement of the data points. The author has also published in this area – by interactively adding energy to a spring model layout, the user can help the point configuration *bounce* out of a local minimum [RMC05].

The term *animation* tends to instil a notion of a smoothly changing picture where the transitions between views (frames) are imperceptible. However, there is another side to the

word ‘animation’. High levels of human-computer interaction such as in *direct manipulation* of visual artefacts and co-ordinated views can be enough to animate static presentations into being dynamic visualisations. In this case, the user’s actions at the interface directly control the view transformations. Instead of the transitions in the views being imperceptible, they are now visual feedback in the cause-effect relation given by interaction over time. Animation, i.e. multiple changing views, can reveal hidden structure within data. When multiple views are shown in temporal and spatial context, latent patterns can be found more easily.

6.2.1.2 Tight coupling and flow of interaction

The configuration of multiple views determines whether the data are presented simultaneously or singularly over time within each view. Interaction however, fruitfully marries the above two types of view configuration. It allows two or more views to be concurrent as well as to show visual representations evolving as the user participates in their transformations. The flow of visual transformations in views occurs with the flow of interaction. By co-ordinating multiple views so that changes made in one view are reflected in other views, interaction can be said to flow between them. This provides the user with the ability of focussing in on specific parts of the data set and seeing them within the context of other representations. In evaluating their snap-together visualisation system, North and Shneiderman have found that this enhances user-performance in data analysis tasks [NS00a].

The notion of co-ordinated views is expressed in the concept of *tight coupling* [AT95b, AS94b]. Tight coupling between visual interface components such as controls and views allow actions invoked on these components to affect the state of other components. Take, for example, a series of form windows with button controls for moving backwards and forwards through the sequence of forms. By using tight coupling between the fields (for entering information) and the “Next” button (for progressing to the next form), the system can grey out the button until the user has filled in all of the necessary fields. This effectively constrains the user’s actions and can be used in general to guide the user through interacting with the interface in the direction of possible goals. When applied to graphical query controls such as range sliders, tight coupling can also alleviate the *all-or-nothing phenomenon* typically associated with Boolean queries in information retrieval. When multiple controls such as range sliders and check-boxes are used to enter query parameters, the selection of a value on one control can influence the range of values that are selectable in the others. This can prevent zero-result situations where the data space being searched has no items matching specific queries. Conversely it can also prevent the user from being overwhelmed by too many results. By displaying result previews as queries are being constructed, users can quickly refine

queries until a manageable and relevant set of results is obtained. This is also known as *dynamic querying* [Shn94].

Co-ordination between multiple views has been around since the 1970s. A popular and well-established technique used by statisticians is to use a matrix of scatterplots to gain several simultaneous perspectives of their multivariate data [BC87]. The scatterplots represent layouts of each possible pairwise combination of the variables comprising the data set. This means that for a data set of dimensionality d, the number of scatterplots required is equal to $d(d-1)/2$. Obviously, the downside to this approach is in its limited applicability to high-dimensional data sets. As dimensionality increases, screen space for the scatterplots rapidly diminishes. However, when the dimensionality of the data permits such a configuration of multiple views of the data, the collage of views can be enhanced by interactive mechanisms. This interaction typically allows the selection of arbitrary points in any of the plots and highlighting the corresponding points in the other views. The highlighting usually takes the form of using different colours to distinguish selections and has therefore led to the adoption of the term *painting* to describe such interaction [MS90]. It is also known as *brushing-and-linking* [BC87] and *location probing* [CMS99]. In statistical analysis this interactive mechanism is referred to as part of *exploratory data analysis* (EDA).

The insight into data that EDA provided statisticians inspired researchers to experiment in its use in visualisations other than scatterplot matrices. Generally the concept can be applied to any multiple-view system where visual entities have different abstract representations across the views. One example is in the linked views employed in the Apple Dylan programming environment [DP95]. In this case, views are connected by what Dumas calls *hot-links* that cause the selection of an object in one view to effect prominence of related objects in the other linked views. Another example is in the spreadsheet approach to visualisation. In [CKBR97], a spreadsheet layout of views allows efficient use of space, multiple operations, side-by-side-ability etc. and supports conjunctive analysis by allowing 'what if' questions to be easily posed and answered (like a normal spreadsheet). The notion of *brushing* has also been extended. Buja et al. [BSLD98] make a distinction between two types of brushing. They call the temporary highlighting of objects, as the mouse brushes them, *transient brushing*. Whereas for highlighting that remains after direct manipulation of the visual artefacts, they use the phrase *persistent brushing*. This interaction was implemented in the XGobi system [SCB98] which provides a workspace for exploring multidimensional data. In XGobi, brushing can be applied to link scatterplots and projections but is limited by the fact that the software's set of algorithms and views is not extensible.

So far, a number of terms have been introduced in this section: *tight coupling, co-ordinated views*, *brushing and linking, location probing, dynamic querying*; but whatever it is called or part of, the co-ordination of activity across multiple views gives the user greater control over the visual representations of the data. This ultimately nurtures discovery. In [BCS96] it is described as linking *"…a graphical query to a graphical response"*, and in [EW95] it is stated that it gives users the impression that they are *touching* the data.

Such systems do, however, incur overheads in terms of search time and memory, and therefore efficient data-structures must be employed to keep these down [JS94]. This issue, along with more novel applications of interaction with multiple views, remains a challenge in information visualisation research. Fekete addresses this with his InfoVis Toolkit [Fek04]. This is a Java library, consisting of many visualisation components such as scatterplots, tables and trees that can be linked together and incorporated into java-based visualisation applications. These components can also be supplemented with interactive mechanisms such as fisheye views and dynamic labelling and therefore efficient data structures are implemented to help maintain the computation speeds required by dynamic queries and multiple coordinated views. However, Fekete's toolkit is similar, in a sense, to Schroeder et al.'s VTK [SML96]. It does not provide a visual method of dynamically coordinating data-flows between the toolkit components and, as such, lacks the flexibility that visual programming can provide.

Prefuse [HCL05], which is also implemented in Java, is similar to Fekete's toolkit, however, it utilises a lower level of abstraction for the composition of visualisation applications. Rather than modularising whole visualisations, Prefuse adheres to a model akin to the data-flow paradigm. Again, the data-flow is established only by writing Java code, albeit a far smaller amount than would be necessary for creating applications from scratch.

6.2.2 Information workspaces

The notion of the *information workspace* addresses two important issues in information visualisation. One is the cost structure of information. The other, which essentially supplements the first, is in providing the user with a suite of tools for carrying out different tasks within the work domain. With regard to the first issue, Card et al. made one of the earliest proposals of an information workspace called the Information Visualizer [CRM91]. The inspiration for this system came from the ways in which people organise their resources within their working environment. For example, in an office, information that is not needed all of the time is stored in filing cabinets (secondary storage), whereas information that is in current or constant use is placed on the worker's desk for convenient access (immediate

storage). Thus, in a sense the information in the office has a cost structure – the cost of retrieving a document from the filing cabinet is higher than that of taking a document from the desk. Hence, for efficient use, the user tunes this structure to his or her current working situation to reduce the overall cost of accessing the information required. Card et al. argued that such a structure is also evident in electronic information processing systems. They made several observations of this structure and implemented their system as an electronic information workspace where the cost of information in cognitive and temporal terms could be minimised.

The second issue that the information workspace addresses is the provision of a variety of tools applicable within the worker's task domain. For example, consider a programming environment for software development such as Microsoft's Visual Studio Integrated Development Environment (IDE). This provides packaged bundles of many disparate tools for utilisation and appropriation in the programmer's task processes such as multiple views for source code files, plug-in extensions and database configuration tools. Because many of the tasks and goals of programmers are not known in advance, the system designer cannot anticipate them. For this reason such software environments tend to provide this extensive suite of tools and also allow a great deal of customisation of the environment itself. This is also reflected in the information-centric visualisation environment Visage, implemented by Roth et al. [RCK*97]. With Visage, the primary objective was to integrate visualisation tools for information retrieval, analysis and communication (via presentation) in a flexible system where the user could arrange these information resources in any way he or she desired.

The workspace metaphor is also prominent in the field of computer supported co-operative work (CSCW), although in [HD96] the word 'space' was contested by 'place' because it was argued that people only really understand and interpret spaces as places. Also, the term *media space* [Gav92] has been adopted in CSCW to denote a video-based virtual environment as a medium of communication and social interaction. That aside, the inspiration ultimately comes from physical spaces (or places) that often have more than one person present at any given time and promote communication such as face-to-face speech and gesture. This implies that a virtual workspace should also provide a means of collaboration and awareness of other people working in the domain. This is in line with the theory of distributed cognition [HHK00], where system boundaries are expanded to include the activities of other people and processes that bear on one's current task and work domain. Notification systems such as Elvin [FMK*99] attempt to enhance awareness of other people and events and can supplement a virtual workspace. Recording histories of user interaction with the information artefacts as in [HHWM92] also provides awareness and generates useful

information. This suggests that channels for communication and collaboration should be amongst the tools provided by an information workspace.

Physical workspaces such as a joiner's workshop are not entirely specialised applications – there might be communication facilities such as phones and an office area, i.e. the workspaces are not geared purely for woodwork – but are macro-environments that are appropriated by the human worker(s). The virtual workspace, to be true to its physical counterpart, must provide intuitive means of retrieving and organising the required information from a variety of people and processes in an efficient manner that promotes productivity and/or satisfaction. The virtual workspace should not be an encapsulated environment but should be extensible and be able to broaden its boundaries to accommodate other tools and resources within the user's domain.

6.2.3 Data, information and knowledge

When multiple views portraying abstract representations of data are brought together, possibly intentionally adhering to the notion of an information workspace, the prime motive is to gain knowledge from this arrangement. In [Spi00], Spiegler defines knowledge as being "the process of knowing". This is described as a cyclic process that takes data, information, social context and experience etc. to produce knowledge, which can in turn generate more data, information and even more knowledge.

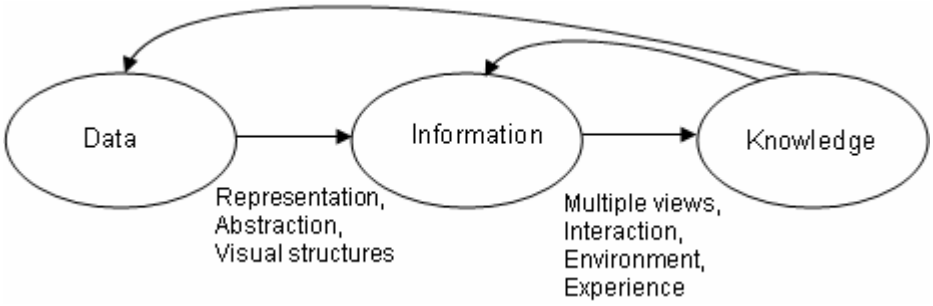


Figure 6.3 A conceptual view of the "process of knowing".

In [Sow00] it is implied that data and information are lower and higher levels of knowledge representation. They are abstractions, and in the context of this chapter it can be seen that visualisations serve to increase the level of abstraction of data into the higher level of information. Visualisations naturally appeal more to one's perception than raw data. When this occurs in an information workspace where the system boundaries are broad enough to incorporate many resources in the user's environment, including the above-mentioned social context and experience, meaning can be found in information and foster new insight and

ultimately knowledge (see Figure 6.3). Spiegler states that this transformation of data to information to knowledge has a strong inherent time dimension. The process takes data, generated in the past, to create information in the present and foster knowledge in the future.

The relationship between data, information and knowledge can also be tentatively mapped onto a *Meaning Triangle* [Sow00]. A meaning triangle is a triadic structure similar to Peirce's ontology [Sow00] of *firstness*, *secondness* and *thirdness*, based on entities, signs and meaning (or concept). In this case, the entity is described at the lowest level by data, the sign is the information gleaned from the data and the meaning/concept is derived from a person's perception of the representation. These meaning triangles can be created for different levels of representation of knowledge and when joined together, where the sign of one triangle becomes the entity of the next, an arbitrary number of levels of representation can be gained. This is the basis for Peirce's idea of *unlimited semiosis* and can explain how Spiegler's transformation of data to information to knowledge can indeed generate more data, information and knowledge in an infinite cycle. See Figure 6.4 below:

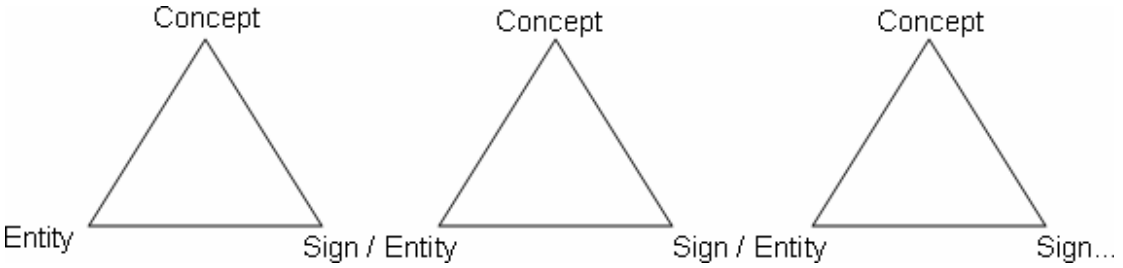


Figure 6.4 Linked meaning triangles.

Data mining and *data archaeology* are knowledge-discovery paradigms that ideally adhere to the concept of the process of knowing. By incrementally increasing the level of representation from data to information, typically via visualisation, it is their goal to uncover hidden knowledge. There is, however, a conceptual difference between data mining and data archaeology. In [BST*94], Brachman et al. established data archaeology – aptly named when one considers Spiegler's notion of temporality in data. In data archaeology, Brachman et al. put human intervention and interaction at the centre of the knowledge discovery process. Rather than allowing some automated and unsupervised classification algorithm to try and find patterns in the data as in data mining, data archaeology dictates that the user should instead be in the driver's seat. By giving the user a suite of flexible tools for representation, manipulation and analysis of data and information, the user can form and test hypotheses. As emphasised in the previous paragraphs, different levels of representation in data and information are involved in describing knowledge and, in realising this, Brachman et al.

implemented a formal knowledge representation (KR) language as the core of their system for data archaeology. By giving the user a powerful means of representing concepts in the target problem domain, the user can use queries and inferences to segment (abstract) data into interesting subsets for analysis and hypothesis testing. With respect to artificial intelligence and machine learning, data archaeology can be more generally thought of as a form of supervised learning, whereas data mining, on the other hand, predominantly employs unsupervised learning algorithms for the discovery of potentially interesting subsets within data.

Representation plays a critical role in information visualisation. From the internal bit-wise representations of knowledge in the form of data for the machine to manipulate, to the visual abstractions that encode information for the human to perceive, the representations potentially serve as a resource for the creation of knowledge upon which he or she can act.

6.2.4 Relevance of the information workspace concept to the HIVE framework

At the core of the proposed HIVE framework is the hybrid algorithmic architecture as discussed in Section 6.1.4, however, this architecture merely transforms the data, which are the low-level representations of knowledge, into a slightly higher level of abstraction. The architecture may accurately classify and segment the data but this may be seen as producing meta-data, an informed synopsis of the original data set. A rich variety of interactive views of this transformed data are required not only to allow the user to perceive Gestalt qualities but also to allow the user to steer the computations of the hybrid architecture. This serves to help form hypotheses based upon the uncovering and manipulation of the higher-level visual representations.

North and Shneiderman propose a classification of visualisation systems into four levels of flexibility in data, visualisation and co-ordination [NS00a, NS00b]. They state that level 0 systems are the least flexible in that they work only with one type of data and proffer perhaps only one form of visualisation with no co-ordination between multiple views (if any). Level 1 and 2 systems are described as being progressively more flexible up to the point where the level 3 system emerges. Level 3 systems permit the user to apply a variety of visualisation techniques to disparate data sets and also to define how the interaction with one view can transform the visual representations that are the other connected views. This level of user-defined co-ordination between views is akin to the visual programming paradigm discussed in Section 6.1.1, but at the same time, the range of visualisations provides a comprehensive suite

of tools that somewhat resemble the notion of an information workspace. This suggests that it would be beneficial to make the HIVE framework extensible not only in the underlying algorithms but also in the visualisation tools, as this would allow the system to address the two issues behind the information workspace as described in section 6.2.2. The extensibility of the algorithmic architecture addresses Card et al.'s cost structure of information [CRM91] by making the transition from data to meta-data or information more efficient. This efficiency is to be achieved by matching algorithms of appropriate complexity to the data in such a way that transformations in the data are produced in as little time as possible. Also, the provision of a variety of visualisation tools makes it easier for the user to get his or her hands on the data. By approaching the HIVE framework from this workspace perspective, the author can strategically map out the transformation, through abstraction and representation of data to information and to potential knowledge discovery.

A limitation that is generally encountered in information visualisation-based workspaces is that the original data being transformed cannot be modified in terms of adding or deleting elements. While systems such as IVEE [AW95], XGvis [BSLD98], DEVise [LRB*97] and Visage [RLS*96] offer flexible workspaces for the visualisation of information and the creation of meta-data, they do not allow the user to use any of their tools for modifying the composition of the underlying data set. It may be for this reason that these systems have not seen the widespread use anticipated by the designers, because in real-world situations the consumers of data are also often the producers as well. In scientific visualisation, however, the Application Visualisation System [UFK*89] and the GRASPARC system [BBB*93] have addressed this limitation. Here, the processes that generate the data are part of the system. Also, as mentioned in section 6.2.2, programming language environments can be seen as information workspaces, but this type of non-visualisation based workspace inherently facilitates production of the data it contains, i.e. the source code. As far as can be seen in the literature, the closest information workspaces get to expanding the data upon which they work is by deriving meta-data such as interesting classes, or recording use histories such as in [HHWM92].

It is the nature of scientific visualisation and programming workspaces that makes the production of data a natural aspect of their anticipated use. In scientific visualisation, data tend to be derived from the simulation of physical processes where subjunctive analysis is desired and therefore requires the generation of simulation data on the fly. In programming IDEs the goal is to produce data, the source code. However, in typical information visualisation systems, the data being transformed are abstract, generated in the past and might not lend themselves to subjunctive analysis. This serves to understate the importance of on-line

modifications to the data. With this in mind it is considered that if the HIVE framework accommodates modification of the underlying data set, this will be an important asset because it will make it a more useful system for a wider range of people (the information producers as well as the consumers). The fluctuation in data complexity should not be a problem because the proposed hybrid algorithmic architecture is designed to adapt in such situations. The adaptability of the algorithmic architecture comes from the notion of matching the appropriate algorithms for clustering, layout and variable transformations to the data, as they are needed (see Chapter 7 for a detailed discussion).

6.3 Visualisation system design theory

The previous two sections have outlined potentially useful concept options for system architecture and applicable theories for turning abstract data into information (visualisations) with the intent to foster knowledge. However, this alone is not enough to ensure the effectiveness of a target system. Petre et al. describe such a system as a *cognitive technology* [PRR*01], characterised as a means of externally representing information residing in the user's short-term memory, essentially externalising his or her plans. This has the advantages of reducing cognitive load, and also making the externalised information easier to communicate to others and to interact with in order to create more information. For this to be effective, the system obviously must be usable.

Only recently has there been a flurry of activity in the empirical evaluation and meta-analyses of information visualisations [CY00], but there are several long-standing proposals of design theories for information artefacts in the literature. These include ecological interface design (EID) [Bur00, FTM*98, RS98], cognitive dimensions of notations [Gre89, HH99, PRR*01] and Roth et al.'s dimensions of expression [RCK*97]. Each of these theories presents a comprehensive set of guidelines that intrinsically overlap and complement each other. In this section each theory will be discussed in turn and the common ground between them will be uncovered.

6.3.1 Ecological interface design (EID)

There are many approaches to information system design. Their constraints and the aspects of the system that are emphasised generally distinguish them, for example:

- ***Technology-centred*** – this approach focuses primarily on the limitations and the capabilities of the technology underlying the system. The user interface is designed in such a way as to fully utilise the functionality provided. This is often referred to as the *single-sensor-single-indicator* (SSSI) approach in EID [FTM*98, RS98].

- ***User-centred*** – this emphasises the capabilities of the users. Operation of the system should not exceed the capabilities of the user. The Visage workspace [RCK*97] is user-centred because it aims to make information accessible to users by breaking down the barriers between applications and processes that externalise the required information.

- ***Control-centred*** – this concentrates on the stability of the human-machine control loop in terms of time delays and order of operations [FTM*98]. Systems that try to minimise delays in user response to information often aim to help the user anticipate future states such as in subjunctive displays [Lun99] and situation awareness displays [AC98].

- ***Application-centred*** – with this approach, the information is stored in files and has no other useful representation external to the application that is required to manipulate the files [RCK*97].

- ***Document-centred*** – this is at a higher level of abstraction than the application-centred approach. Documents provide a more useful abstraction and organisation of information [RCK*97].

- ***Information-centred*** – this approach provides yet a higher level of abstraction than the document-centred approach. Here the system allows individual data elements to be treated as manipulable objects. The Visage workspace [RCK*97] also provides an example of this.

- ***Activity-centred*** – systems that adopt this approach generally aim to enhance the awareness and/or visibility of other users and their actions, as well as one's own previous activity. Examples include systems that record use-histories [HHWM92] and *paths* through informational resources [CRB98] to help guide the activities of others in attaining their goals.

- ***Property-centred*** – this approach is used in the Presto document system at Xerox PARC [DELS99]. Here users can attach arbitrary property value-pairs to documents to facilitate objective and subjective categorisation for retrieval of information. This approach provides a flexible level of representation for information, based upon the abstract notion of documents.

The first three of the above design approaches were detailed in [FTM*98] by Flach et al., where they were considered in the design of human-machine systems such as flight crews and their aircraft. They are however, also applicable in the design of abstract information systems as is implied by their descriptions. Flach et al. stated that these three approaches have a common *image* of the system, i.e. that of the user and the machine. The addition of the other

design approaches with the exception of activity and property-centred types do not violate this image.

It was asserted by Flach et al. that in designing a system, the initial analysis of the user's activities in the work domain could best be understood by considering them within the broader context of the workspace, i.e. the ecology. This served to shift the focus of attention from purely human-machine interaction, with situation-based constraints (technology, control, user, etc.), to human-work interaction where all of the above constraints are situated. This notion lead to the *use-centred* (not user-centred) design approach that is now known as ecological interface design. "Ecological" is the term used because the design approach concentrates on the (work) ecology and the (worker's) *niche*. "Environment", on the other hand, was too broad a notion as this implies *habitat* and things external to the system rather than niche [Gib79].

Traditionally EID is applied to tangible systems with an underlying physical theory such as aircraft and control rooms, to address problems from the likes of SSSI. EID can, however, be applied to situations where one must make sense from abstract data, for example, the information visualisation equivalent of SSSI could be regarded as a table of figures or a range slider control for every attribute in a query view. Also, the burgeoning application of distributed cognition theory and CSCW that inherently recognise ecological issues in information system design, show that EID can be transposed to virtual workspaces as well as their physical counterparts. This is also reinforced by the inclusion of the activity-centred and property-centred approaches in the list above. Whether it is used for enhancing analysis in scientific visualisation or control for system operators, EID is also entirely applicable to the design of information visualisation workspaces.

6.3.2 Cognitive dimensions of notations

For over a decade the 'cognitive dimensions' framework, originally proposed by Green [Gre89], has been developed as a framework to help designers create *notational systems* and *information artefacts*. Notational systems are defined as being any means by which information can be structured by a user via manipulation of (in the visual case) graphical properties and retinal variables. Examples of notational systems include word processors, programming language IDEs and CAD tools etc. Information artefacts, on the other hand, are self-contained notational systems such as radios and clocks.

The dimensions are a list of issues the designer should consider and they provide a vocabulary for design decisions and their cognitive implications for the end users. Each dimension addresses specific activities that the target user may engage in and a particular

aspect of the system involved, and by conforming to the dimensions, they also provide a basis for design evaluation. The following is a description of the set of cognitive dimensions [PRR*01]:

Viscosity: *resistance to change.* This pertains to how difficult or time-consuming it is to modify a construct in a notation to achieve a goal. For example, in a programming IDE, source code markers bookmark specific lines of code. If the programmer wanted to remove all of the markers then he or she would have to go through all of the code to select and remove each marker if there was no function to 'remove all'.

Visibility: *ability to view components easily.* Too high a level of abstraction could make information important to the task at hand invisible

Premature commitment: *constraints on the order of doing things.* The *history tree* [BBB*93] mechanism used in some scientific visualisation systems, as described in chapter three, addresses this. In this context a user can set parameter values then run a simulation. When something interesting appears, the user can backtrack to a previous point in time and modify parameter values to gain more understanding of the interesting phenomenon.

Role-expressiveness: *the purpose of an entity is easily inferred.* This pertains to how visible the relationship is between two or more elements in a notation.

Error proneness: *the notation invites mistakes and the system gives little protection.* Slips and errors specific to the notation should be anticipated so that adequate means of recovery are provided by the system.

Abstraction: *types and availability of abstraction mechanisms.* If the system permits modification of abstract representations then some form of *abstraction management* should be provided (see section 6.3.4).

Secondary notation: *extra information in a means other than formal syntax.* This is a vehicle for appropriation. Comments in a program's source code are an example. Here the user increases comprehensibility without having to adhere to the formal program language syntax.

Closeness of mapping: *closeness of representation to domain.* This regards how close the representation of entities in the notation is to the notional entities in the user's problem domain.

Consistency: *similar semantics are expressed in similar syntactic forms.* Similar information should have similar representation.

Diffuseness: *verbosity of language.* The notation should be concise. Visual structures should not use more screen space than is necessary. 'Chart junk' should be obviated.

Hard mental operations: *high demand on cognitive resources.* Information encoded in visual structures should be readily perceived.

Provisionality: *degree of commitment to actions or marks.* Subjunctive interfaces [Lun99] address this by facilitating "what-if" scenarios.

Progressive evaluation: *work-to-date can be checked at any time.* The history trees in scientific visualisation address this by allowing simulations to be halted and re-run. Software written in an interpreted language can be run before its completion to check for bugs and conformance to requirements.

Side-by-side-ability: *providing comparison of notational structures.* The positioning of visual structure in information visualisations can convey meaning per se. Graphs that share a common axis can be placed so that screen space is utilised efficiently and related information is perceived.

The influence each of the dimensions has on design decisions is determined by the nature of the target system. Trade-offs also become apparent because if one dimension is addressed this can have an impact on other dimensions.

The cognitive dimensions framework has been shown to be applicable to the design of information workspaces. Hendry and Harper [HH99] emphasised secondary notation in the design of SketchTrieve, an information seeking workspace that allows users to organise their queries and results in a very flexible manner to increase comprehensibility and support opportunistic searching. Cognitive dimensions are in line with ecological interface design because they essentially provide a use-centred design approach. By considering the target

audience and the type of system under analysis, the dimensions provide situated constraints that can be traded off with each other to *tune* the design.

6.3.3 Dimensions of expression

In the design of the information visualisation workspace Visage, Roth et al. utilised a set of design guidelines they called *dimensions of expression* [RCK*97]. These dimensions are similar to cognitive dimensions but have been developed specifically with information visualisation in mind. Each dimension pertains to how the user may express his or her intent in interacting with information visualisations for exploratory tasks. The four dimensions of expression are as follows:

Set description: *how people describe data sets of interest.* There are two means of determining set memberships: *set extension* and *set intension*. In set extension, users may define a set by pointing and clicking with a mouse to select visual structures, for example, the members of a cluster on a scatterplot may be selected by dragging a bounding rectangle over the cluster. However, set intension can be used when the interface makes it difficult for direct selection of objects and therefore the user must specify criteria for set membership. Examples of set intension are SQL queries, range sliders and forms. Set description is related to the cognitive dimension of 'secondary notation' with regard to organisation of visual structures into spatial sets to enhance comprehension.

Granularity and composibility of actions: *whether people communicate via composing primitives or with higher level, abstract expressions.* Ideally there should be a trade-off between the constraining *appliance-like* interface and *composition-by-primitives* approach to allow greater flexibility at the same time as reducing the number of steps in carrying out an operation. This relates to the cognitive dimension of 'viscosity' because the granularity of actions (resolution of representation) is inversely proportional to the viscosity of a notation.

Continuity of action: *whether the communication process is about a continuous process or discrete action.* Continuous communication with the system is used where the user is unsure of what intermediate stages will yield in the process. An example is in using smooth semantic zooming such as in Pad++ [BH94] to progressively uncover detail and provide feedback to see entity relationships emerge. Discrete actions, on the other hand, are used when the user knows what the outcome of the action will be and does not require intermediate visual feedback. An example is in closing a view. Continuity of action is related to the cognitive

dimension of 'hard mental operations'. This is because the degree of continuity should be chosen to alleviate cognitive load as in not having to remember previous states or lose context in the representation.

Consistency with domain vocabulary: *whether communication can reflect familiar domain vocabulary.* This refers to the vocabulary (representations) used in interacting with the system and how similar or relevant they are to the familiar vocabulary of the target user in his or her work domain. This is most strongly related to the cognitive dimension of 'closeness of mapping'.

As can be seen from the above, the dimensions of expression have much in common with the cognitive dimensions described in section 6.3.2. Roth et al. state that in designing information visualisation systems, the above dimensions repeatedly arise, and like the cognitive dimensions, they are situated constraints that have a degree of influence dependent upon the type of system under development and the context of its use. The dimension pertaining to the consistency with domain vocabulary can be seen as a human-work interaction issue. This emphasis on the work ecology is one of the essential ingredients in information workspace design and is in line with the theory of ecological interface design.

6.3.4 Abstraction

Abstraction in information visualisation is about the representation of data in such a way that the user's visual perception is utilised to transform the representation into meaningful information. The lower levels of abstraction pertain to how visual structures are composed of graphical properties such as marks (which are the geometrical structures of points, lines, planes and volumes) and retinal properties (characterised by shape, size, colour, orientation and texture). These visual structures serve to encode either individual data elements or compound aggregates in a meaningful way. When the visual structures are presented in concert, as in points on a scatterplot or nodes in a network diagram, a higher level of abstraction is achieved where the context of visual structures brings to bear more (relational) information. The low-level abstraction was the focus of a paper by Card and Mackinlay [CM97] where contemporary information visualisation techniques were analysed with respect to visual structures and the use of spatial substrates. A paper by Lohse et al. [LBWR94] focussed on the higher level of abstraction used in visual representations. Here, a taxonomy of graph types was developed to distinguish between the various techniques of providing informative views of data sets. However, both of these papers only marginally addressed the

fact that there can be many different representations for encoding data and finding the most effective is by no means trivial.

Russell et al. [RSPC93] have addressed this issue of providing the most effective means of representation in a paper written in 1993, where they outline the *cost structure of sensemaking*. They state that finding the most effective representation, i.e. one from which the right information is perceived and which facilitates task-specific operations, can be difficult and they define this process of searching for this representation as sensemaking. It should be noted that the cost structure of sensemaking is different from the cost structure of information (see section 6.2.2). This is because in sensemaking the search for the correct representation and evaluating it with respect to the problem domain is a cyclic process with each step having an associated cost. In the cost structure of information, informational resource availability and the associated cognitive work in disseminating information incur the costs. However, this aside, sensemaking is important to getting abstraction right in information visualisation – abstractions that make the most sense, given the task domain, are desired. The encoded data must be in a form where the user can immediately make sense of them and become more informed.

6.3.4.1 Abstraction management

In describing abstraction as a cognitive dimension, Petre et al. asserted that systems that permit many abstractions are potentially difficult to learn. They also stated that when the user is allowed to modify the abstractions, some form of *abstraction manager* is required. In information visualisation the proliferation of abstractions can be difficult to avoid because the user is usually provided with interactive functions to transform and co-ordinate views. The transformation of views is in itself modification of the abstractions.

Abstraction management is required mainly to maintain consistency. It should ensure that the visual representations accurately represent the state of the system and the transformations of the abstract data under analysis. For example, when the system is carrying out heavyweight calculations that may be unavoidable due to extremely high volume data, it might be appropriate to give feedback on the progress of the computations. An example of abstraction management is provided by the model-view-controller (MVC) architecture[1] which is used in Sun Microsystems' Java [Sun02]. The *model* represents the data and rules for accessing and modifying the data. The *view* corresponds to how the data are visually rendered and it makes sure that changes to the data made through the model are reflected. Finally, the

[1] MVC was first developed at Xerox PARC in the late 1970s for the Smalltalk-80 programming system [KP88].

controller translates the interactions with the views into actions for the model. In Java, the view and controller mechanisms are collapsed into one entity called a *user interface delegate*. This is to make their management easier when implementing interfaces. See Figure 6.5.

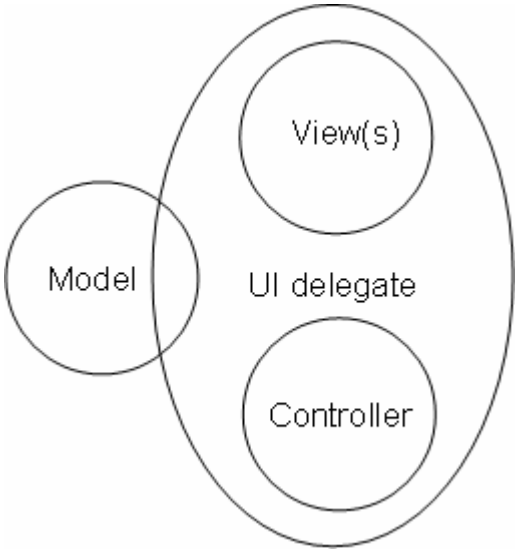


Figure 6.5 The Java model-view-controller architecture.

The Java MVC essentially ties together the data, rules for its manipulation and its visualisation. The MVC can be seen as an abstraction manager because the views are abstractions of the data and by formalising the link between the data and their representation, consistency is maintained across views and through interaction.

With respect to sensemaking, an abstraction manager should ensure that the generated abstractions remain meaningful in the context of a given problem domain. In keeping with the dynamic process of sensemaking, abstraction management should be flexible enough to allow the exploration of different abstractions as well as maintaining their consistency. If the representations are not consistent and appropriate, contextual information can be lost and the visualisations may become open to misinterpretation.

6.3.5 Relevance to the HIVE framework

In Shneiderman's task-by-data-type taxonomy [Shn96], he outlined his visual information seeking mantra: *"overview first, zoom and filter, then details on demand"*. This is an intuitive and useful procedure in information visualisation and one that was adhered to in the implementation of the proposed HIVE framework. However, this is a top-level description of what the system offers the user in terms of functionality. It states nothing of the situated

constraints and the work ecology addressed by the design approaches described in this chapter. For this reason the design approaches described above have been utilised to some extent to flesh out the object of Shneiderman's mantra.

Of Green's cognitive dimensions, secondary notation as employed in SketchTrieve [HH99] is an important characteristic of the framework's user interface. With respect to secondary notation, users are able to store and retrieve visual structures by using space and layout to externalise their plans. This allows the display to be used for both prospective (future plans) and retrospective memory. The informality of secondary notation allows the user greater expression and flexibility in the management of the information space. Secondary notation can also be implemented by allowing the user to mark/paint parts of individual visualisations as in labelling and persistent brushing in XGvis [BSLD98]. These notation instances along with their context may be saved and retrieved as the user desires. Secondary notation should ideally provide a vehicle for appropriation. However, one drawback of secondary notation is the cognitive overhead incurred as a result of re-arranging and placing items on the display [Rob98]. This can be alleviated by trading off secondary notation for increased viscosity to decrease the granularity of actions.

It has been suggested that the HIVE framework should address producers of data as well as consumers. It should provide a means for not only visualising information but also modifying the underlying data set. This will obviously require updating current abstractions to maintain consistency and therefore some form of abstraction management has been utilised. The framework has been implemented in Java and therefore this abstraction management is based upon the model-view-controller architecture. The *Observer* software pattern [GHJV95] has also been adopted for data- and interaction-flow. MVC handles the link between the underlying data and views of them, while the Observer pattern manages the interaction between different stages in hybrid algorithms.

6.4 Conclusions

Section 6.1 discussed scientific visualisation systems that use the concept of visual programming and data-flow models. Their relevance to the proposed HIVE framework has been outlined and it appears that the inspiration they provide could be fruitful.

The data-flow metaphor is appealing when designing and using applications. It helps the user/designer form a mental model of the processes involved in the transformation of data into information. This has also been recognised and implemented in disciplines other than scientific visualisation. One example is in information retrieval where Young and

Shneiderman [YS93] used a guise of the data-flow model they called *filter/flow*. This system was based explicitly on the metaphor of water flowing through pipes and filters to represent information flowing through successive Boolean queries.

The scientific visualisation systems described in this chapter make use of abstraction of computational processes and as a result make the components easier to manipulate and the composite system more versatile and understandable. The same applies when the abstraction is carried forth into interaction as in North and Shneiderman's snap-together concept. By breaking down systems into user-definable configurations of modules, truly flexible applications can be built that might not have otherwise been anticipated by the system designers. The importance of functional decomposition has long been realised in the field of Artificial Intelligence. Luger and Stubblefield [LS97a] state: *"Modern AI programs generally consist of a collection of modular components, or rules of behaviour, that do not execute in a rigid order but rather are invoked as needed in response to the structure of a particular problem instance"*. This applies directly to the basis of the hybrid algorithmic framework in HIVE where the complexity of the data determines the order of algorithmic execution. HIVE adds computational efficiency to the list of advantages of the traditional data-flow metaphor. By extending this with the concept of visual programming at the user interface to determine the flow of interaction, HIVE lets the user drive the application, rather than the application drive the user.

In Section 6.2, facets of information visualisation environments were discussed. Such systems transform data into information and ultimately information into actionable knowledge. Data are the low-level representation of knowledge, but in their raw form humans can find it impossible to perceive latent patterns within them. By using visual representations as surrogates for data structures, the level of abstraction is increased to convey information to the user. These static presentations of information must be supplemented by interaction mechanisms to animate the views into visualisations from which the user can gain more information, and form and test hypotheses about the data as well as the application. Finally, by providing multiple visualisations, supplementary tools and resources from the working environment within a virtual workspace, users are afforded a flexible means of organising, manipulating and sharing their information.

Inspired by this process of discovering knowledge and the limitations of contemporary visualisation workspaces, the HIVE framework is implemented to provide its users with a means of minimising the cost structure of information. It is an extensible system developed from an ecological perspective to broaden its boundaries and incorporate resources from the user's environment. The visualisations provided by the HIVE framework not only provide a

front-end to a database and hybrid architecture, but are instrumental in the transformation of information to knowledge.

The design theories that have been discussed in Section 6.3 have addressed situated constraints and their role in work ecology for information system design. The situated constraints pertain to how the design facilitates use of the system in certain situations. Technology, user, control, information-centred design approaches etc. address these. So do Roth et al.'s first three dimensions of expression. On the other hand, the theory of ecological interface design and the other related dimensions of design take the work ecology into account. EID provides a setting for the situated constraints to be applied. It broadens the boundaries of the system so that they not only include the human and the machine but also parts of the human's working environment as well – a predominant concept in the notion of information workspaces, at least to the extent of application interoperability and electronic communication.

These design approaches provide the designer with a vocabulary for articulating design decisions and understanding the trade-offs apparent when considering specific parts of a design. Although the design approaches have been developed independently of one another, they have been shown to have considerable overlap in their underlying theories. A unification of these approaches is beyond the scope of this thesis, but it is considered that this may be worthwhile.

The next chapter will outline the development of the HIVE framework.

7. The hybrid information visualisation environment (HIVE)

There is a multitude of algorithms available for dimension reduction and clustering abstract data. The different algorithmic approaches seem to be tailored to specific types of data. Some algorithms perform well with low cardinality and dimensionality, such as the canonical spring model [Ead84], and these would suggest practical application in the likes of small to medium size, general graph drawing and dimension reduction. Other algorithms work well with only high cardinality data. An example of which is the *self organising map* [RMS92] where, in training, a substantial training set allows the SOM to learn how to classify a very large body of data.

In a working environment, corporate memory and project-specific databases tend to start off small and gradually evolve into large information repositories. While it would be feasible to visualise the inter-object relationships with a force-directed layout algorithm in the infancy of such a database, it would become less and less effective as the database matures and demands a more computationally feasible solution. Previous work has shown that hybrid algorithmic approaches to visualisation scale up to relatively high-volume data sets, even though some of the constituent algorithms would be too costly on their own if applied to the entire set (see Chapter 5). This would suggest that when applied to a growing database, algorithmic steps could be bypassed in the repository's infancy and incorporated as it approaches maturity. Or, in the case that volume fluctuates, the hybrid algorithm could fluctuate and adapt with it.

This chapter presents an implemented system and framework called HIVE (Hybrid Information Visualisation Environment) that utilises direct manipulation to allow users to interactively create and explore hybrid dimension reduction and clustering algorithms. Figure 7.1 shows screen-shots of the system. Visual programming and a novel algorithmic architecture are proposed as a means to let the user semi–automatically co-ordinate multiple views and define data-flows.

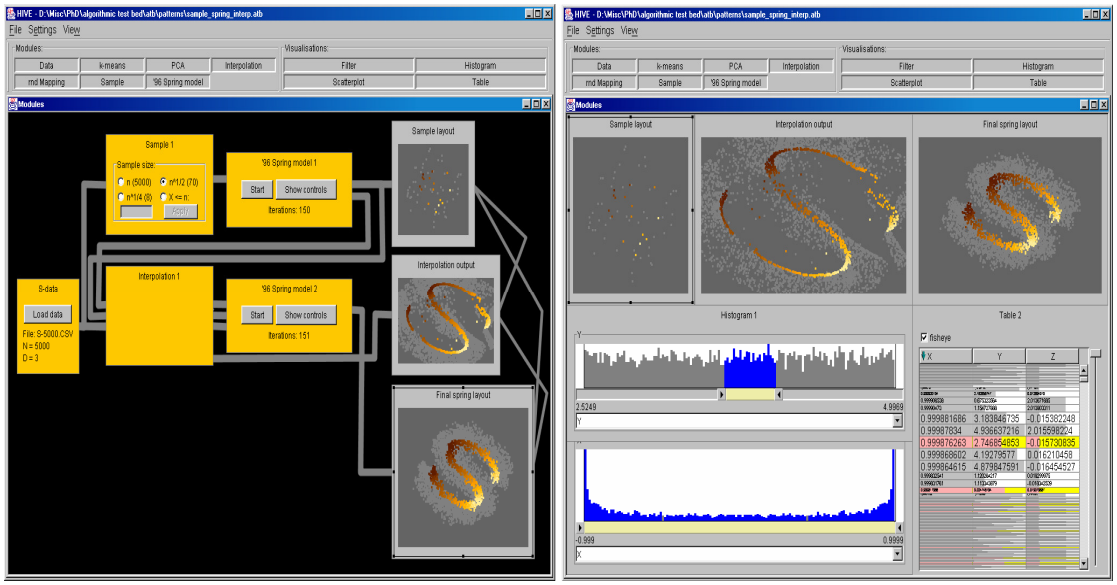


Figure 7.1 Two screen-shots of the HIVE interface. The image on the left illustrates interconnected components that import, transform and render multidimensional data. The algorithmic components collectively represent the $O(N\sqrt{N})$ hybrid algorithm of Section 5.3. Thick lines that link modules represent data-flows while thin ones, connecting scatterplots and other visualisations, represent the connections between interlinked interactive views. The image on the right shows the same scatterplots enlarged and supplemented with a fisheye table component (bottom-right) and histograms (bottom-left). The data consist of 5000 points sampled from a 3-d 'S' shaped distribution.

7.1 Multiple-view co-ordination

HIVE takes advantage of the data-flow model and visual programming. To create a hybrid algorithm, a user drags components from the system's tool bar into the drawing region (see Figure 7.1) and then interconnects them by dragging links between ports on the components. Not only is the data-flow set up in this manner, but the view co-ordination can also be defined this way. After connecting visualisation tools such as scatterplots to the output ports of algorithmic components, 'Select' ports can be linked between view components to establish 'brush and link' functionality.

Hybrid algorithms can exhibit a lower run time than spring models run upon the whole data set, as discussed in [MRC03] and Chapter 5, but they also lend themselves to the production of intermediate visualisations. The benefits of this hybrid approach are two-fold: efficiency is enhanced and intermediate views provide more insight into the data. For example, the hybrid algorithm depicted in Figure 7.1 (left) uses a spring model of a sample of the full data set, to gain an initial small-scale 2-d layout. In the left frame of Figure 7.1,

scatterplots have been hooked up to intermediate stages of the hybrid algorithm to allow for comparison. The three layouts have been positioned by the user on the right hand side of the frame. The sample layout is fed into another module, which interpolates the remainder of the set to produce a second scatterplot. The third and final scatterplot, shown in the right of the frame shows this layout after spring model refinement. In the right hand frame in the figure, the fisheye table shows the layout points sorted on the *x* dimension and histogram views have also been connected to depict the *x* and *y* distributions of the 3-d set. If one then uses brushing to select a range of rows in the table or a region in a histogram, this highlights the corresponding points in the scatterplots and reveals more of the structure of the data. An extension of the work presented in [RC03a] allows for the neatly tiled layout of visualisation components, as in the right hand frame of the figure.

7.2 Combinatorial hybrid approach

HIVE has been inspired by some of the existing data-flow and visual programming systems that are prominent in the literature and common in the marketplace. Upson et al.'s Application Visualisation System (AVS) [UFK*89] and North and Shneiderman's snap-together system [NS00a] are two good examples. AVS is predominantly aimed at scientific visualisation, for modelling or simulating physical processes such as fluid dynamics, and concentrates on channelling data through algorithmic processes for transformation and rendering. The emphasis here is on the data-flow. North and Shneiderman's snap-together system, on the other hand, is concerned with information visualisation. In this system there is less emphasis upon the algorithmic processes for transforming data and more on the transformation of graphical representations by way of multiple interconnected views. Here the flow of interaction takes precedence.

HIVE borrows from the data-flow model of AVS to be flexible in creating efficient algorithms for the visualisations. However, to be in line with the goal of information visualisation, it concentrates on exploration rather than simulation. This is achieved by supplementing the data-flow with interaction flow across multiple views, rather like the snap-together system. It must be said, however, that this approach does not come without drawbacks. It is important to note that if the level of abstraction used in the visual programming language is too low then there might be too many visual modules, in that programming would become complicated and the flow networks too large and hard to manage in the available screen space. One solution being considered is to allow the user to

dynamically increase the level of abstraction by aggregating groups of modules, simplifying the graph of interconnected modules and the programming task.

As well as implementing visual programming to steer data-flows and co-ordinate multiple views, HIVE has at its core a novel hybrid algorithmic framework, exploring a general approach to the composition of efficient and flexible hybrid algorithms. The choice of each algorithmic component is influenced by many characteristics including computational cost, the cardinality, dimensionality and distribution of the data, and the other interaction components that might be used within a larger workspace, such as scatterplots and fisheye tables. The author suggests that these choices can be made incrementally, so that users can employ intermediate representations as they work with and explore their data. The author also suggests that the system can assist the user by using a pre-authored classification of data – based on, initially, cardinality and dimensionality of data sets – and a corresponding classification of available algorithmic components based on the classes of data each is suited for. This offers an incremental and combinatorial approach to the creation of efficient and informative hybrid visualisations.

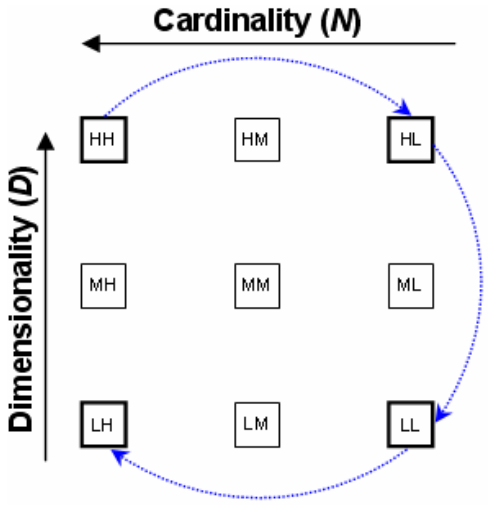


Figure 7.2 Data input to components in a hybrid algorithmic architecture can be categorised by the ranges of dimensionality and cardinality they are best suited for – high, medium or low. Each component transforms the data, effectively moving across the 3x3 grid. The hybrid spring model in Section 5.3 produces a low-dimensional layout of a large high-dimensional data set i.e. a move from (*H*, *H*) to (*L*, *H*) that involves several steps shown as dotted lines in the figure: sampling, which reduces *N*, then a spring model of the sample, which reduces *D*, and then interpolation, which increases *N*.

The author's work has focused on data set cardinality, *N*, and the dimensionality or number of variables associated with each object: *D*. A rough categorisation of *D* and *N* using an ordinal range (high, medium and low), permits the categorisation of an algorithmic

component with values of *D* and *N* for 'good' inputs and for the component's outputs, effectively stating that the component is best suited to such combinations of *D* and *N*. For example, it is considered that the input to K-means clustering should be medium to high in *D* and *N*, whereas a canonical $O(N^3)$ spring model algorithm can only handle low *N* and low to medium *D*.

As shown in Figure 7.2, the choice of components and how they are connected allows one to solve familiar problems in new ways. The hybrid algorithm of Section 5.3 transforms a large set of data of high *D* to low *D*. It can be thought of as a move across the grid of combinations of *D* and *N*, stepping from (*H*, *H*) to (*L*, *H*) – but taking an indirect route via (*H*, *L*) and (*L*, *L*) that involves sampling, spring model layout of the sample, and interpolation based on that intermediate representation.

Tentative default values for these ordinal categories of data are as shown in Table 7.1. The author derived these values from his own experience of constructing hybrid algorithms, however, HIVE allows the user to tailor them:

Cardinality	**Dimensionality**
Low $N < 1000$	Low $D < 3$
$1000 <=$ Moderate $N <= 25000$	$3 <=$ Moderate $D <= 100$
High $N > 25000$	High $D > 100$

Table 7.1 Ordinal categories of cardinality and dimensionality.

The HIVE system has been designed and implemented with this hybrid algorithmic approach in mind, and serves to provide a workspace for experimental algorithm design and exploratory data analysis. The visual modules that have been implemented so far include a CSV data-importer (imports comma-separated-value data files into HIVE), Chalmers' 1996 spring model, radial interpolation (see Section 5.3.3), K-means, neural PCA [Oja82], stochastic sampling, scatterplot, histogram and fisheye table (Appendix A provides a full list). These components are the ingredients used in an algorithmic 'cookbook', in which components deemed to suit particular data characteristics can be automatically connected to form hybrid algorithmic paths that span the grid of Figure 7.2 (see Appendix B).

7.2.1 3-stage hybrid approach

It is not yet known whether there is a full set of algorithms available that would be suitable for the hybrid framework. However, as a starting point to test whether the framework would

work, it is possible to implement part of the above model and evaluate its performance over the ordinal range of data cardinalities and dimensionalities. Supposing the outer loop in the conceptual diagram of Figure 7.2 is implemented then there need only be three types of algorithms. It may then be assumed that some tolerance in the allowable input of the algorithms is present, i.e. the algorithm for taking high dimensionality and/or high cardinality could work adequately with medium cardinality and/or medium dimensional data.

It is therefore proposed that hybrid algorithms consisting of three incremental stages can be created, to determine whether the larger framework is feasible and therefore justify the search and/or development of new algorithms that would fit into place in the larger framework. Such algorithms have already been demonstrated in Section 5.3 and it will be shown here how they fit into the proposed algorithmic framework.

Note that the computational complexity of the individual algorithmic stages should be, informally speaking, inversely proportional to the cardinality and dimensionality of the input patterns. This is supported by the results given in Chapter 5. A hybrid approach to dimension reduction is taken, where K-means or stochastic sampling (linear algorithmic time complexity in *N*) is initially applied to high-cardinality data to reduce the representative cardinality. Then, the more complex (quadratic algorithmic time complexity in *N*) spring model is applied to this subset to reduce dimensionality and therefore obtain a 2-d spatial layout. The results show that this reduces overall time complexity in *N* and increases layout quality with respect to stress. Figure 7.3 illustrates the 3-stage approach.

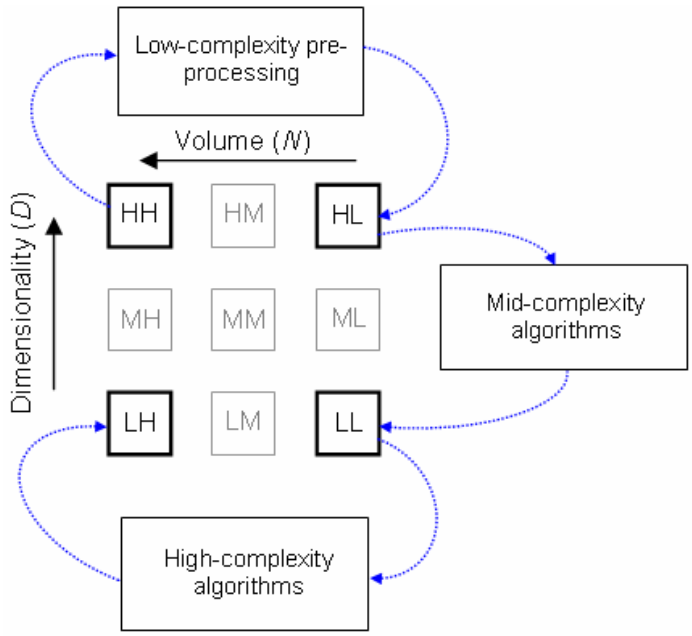


Figure 7.3 The proposed model of the hybrid approach for scalability and adaptability.

Investigation into some of these algorithmic steps has already been detailed; however, there are some aspects that will require a deeper explanation and more research.

Note that in Figures 7.2 and 7.3, the hybrid process starts by clustering or sampling and then performs dimensional reduction. This ordering is enforced because the hybrid algorithms described in Section 5.3 performed sampling or clustering first, then dimensional reduction, and produced promising results. In the future, work will need to be carried out to determine whether the proportions of N to D might impact on this process order. New algorithms that are fitted into the framework may work linearly in N but be quadratic in D, and in this case it may be appropriate to reduce dimensionality first.

7.2.1.1 Low algorithmic complexity pre-processing

Clustering at the initial or pre-processing stage should be performed in $O(N)$ time complexity. Otherwise, stochastic sampling should be employed to gain a representative subset of the data. As can be seen from the above, the output of this pre-processing step, consisting of the cluster centroids or samples, is passed to the next stage of the hybrid algorithm. As with each of the hybrid algorithmic stages, each preceding stage serves to give the next a head start by producing intermediate results and/or a reduced representation of the data that can be refined by successively more expensive stages.

7.2.1.2 Medium algorithmic complexity

In the hybrid algorithms detailed in chapter 5, Chalmers' spring model was used for the intermediate stage. Having reduced cardinality, the aim is now to reduce dimensionality. This stage further reduces the representation of the data but does so in a way that it provides an accurate overview. As will be seen later, such intermediate algorithmic stages also serve to provide extra visualisations of the data that can be effective in gaining insight.

Although it has overall time complexity of $O(N^2)$, the spring model's complexity is effectively reduced to $O(N)$ because the preceding algorithmic stage provides it with a sample or small set of cluster centroids. Other possibilities for this stage in HIVE are neural PCA, random mapping and the first stage of the fast NMDS algorithm described in Section 5.4. In fact, just about any dimension reduction algorithm that converges in quadratic or sub-quadratic time can be used.

7.2.1.3 High algorithmic complexity

The input data of this stage of the hybrid approach should be of relatively low cardinality and dimensionality – a reduced and representative subset of the data. It was shown in Section 5.3 that interpolation can be used to increase the number of represented objects because initially

only the cluster centres or samples are placed in the layout. This then forms the lower arrow on Figure 7.1.

7.3 Adaptability to different variable types and heterogeneous data

As well as being able to cope with varying cardinalities and dimensionalities of data sets, HIVE also works with different types of variables in the data including nominal, ordinal and real and a mixture of these. This is handled by a modular stage in the system that transforms different types of data into continuous vectors. This essentially forms part of the pre-processing illustrated in Figure 7.2. This is essential because algorithms such as K-means can only process numerically continuous vectors because they are often represented within Euclidean space.

There are existing strategies for transforming nominal and ordinal variables into continuous numerical quantities [RB99, RRBW03]. In text processing the *tf-idf* weighting scheme is often applied, where *tf* means the intra-document frequency of unique terms and *idf* is the 'inverse document frequency'. This is commonly used in the field of information retrieval and can also be used as a pre-processing stage in the proposed system. Chapter 9 demonstrates a strategy for this in HIVE.

Techniques such as this can be automatically called on depending upon the variable composition of the input pattern vectors, in order to transform the data set into a collection of continuous values. For example, when the data are Boolean, HIVE automatically uses the Jaccard similarity coefficient instead of Euclidean distance when measuring dissimilarities on the fly. Similarly, when a text corpus is the input, dissimilarities are calculated using the cosine measure.

7.4 Implementation of HIVE

In light of the literature review summarised in Chapter 6 of this thesis, a well-defined image of the system that conveys the HIVE framework was developed. Development of the software is now at an advanced stage and HIVE has been adopted by several researchers in their work on new algorithms, novel interaction techniques and exploring their data.

The software has been implemented in Java SDK 1.4. The system architecture (Figure 7.4) has been designed to let users compose visualisation tools. In general terms, the architecture involves a graph manager that supports the user's composition of a flow of data

through components such as scatterplots, K-means clustering, spring model layouts, table views and so forth. Additionally, a hybrid algorithm generator allows HIVE to semi-automatically load and connect algorithmic components.

7.4.1 System architecture

The architecture, illustrated in Figure 7.4, has been designed with visual programming and the data-flow model in mind. Users can compose visualisation tools using modular components for importing data, algorithmic processing and graphical rendering. Information workspace issues such as inter-application operation, interaction flow and their relation to the underlying hybrid algorithmic architecture have also been taken into account.

Before implementation of the system had commenced, alternatives other than creating a system in Java from the ground up were considered. One alternative option was to expand on an existing environment such as AVS or Snap-Together. This would have circumvented the need to write and debug lots of new code. However, it was felt that this approach would not be flexible enough in achieving the interface's look and feel, and the system behaviour that was desired. The author's approach allows much more breathing space for experimenting with design options. There was, of course, more work in implementing the system this way, but ready-to-use implementations and examples of useful Java programs are prolific and therefore much of the code for HIVE was derived from these.

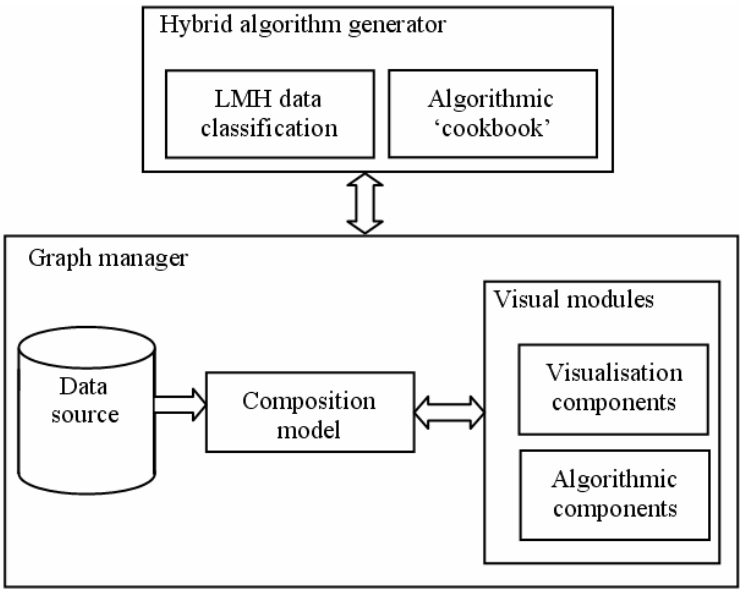


Figure 7.4 The system architecture of the HIVE framework.

7.4.2 Graph manager

The graph manager allows the user to incrementally create executable networks of components. It employs a scripting/composition model [NTMS91] to impose constraints upon which modules can be connected and through which 'ports', depending upon factors such as the categorisation of data types mentioned in Section 7.2, as well as graph structure and port polarity (input only, output only, two-way). A user can manually connect together components, and is warned of potentially unsuitable or inefficient connections. Another mode offers an automatically generated default path through the grid of Figure 7.2, instantiating components, from an algorithmic 'cookbook', based on the system's classification of the input data set.

7.4.3 Visual modules

The graph manager defines three types of components to support the construction of hybrid visualisations. These are (1) a data source component to allow the import of CSV files, free text and lower triangle data matrices, performing the required variable type transformations; (2) algorithmic components to transform data into metadata and intermediate representations; and finally, (3) visualisation components for rendering. It should be noted that this system is not strictly a data-flow model since it is not the original data that are passed between components through links and ports, but references to the data and any transformations that are applied. The primary benefit of this is the more efficient support for tightly coupled interaction such as brushing and linking.

To facilitate extensibility, the visual modules that represent algorithmic processes and visualisations are all derived from a common Java class. This means that to accommodate new algorithms and visualisations, the programmer need only extend the base class and implement his or her own specific methods. The base class exhibits default behaviour such as allowing the user to resize, transpose and rename modules via keyboard or mouse commands. This class also contains the routines that handle port declarations.

The Java Reflection API [Sun03] has been employed in HIVE to dynamically load algorithmic and visualisation components at run time. Compiled visual module classes reside within a specific folder in the system's directory structure. Periodically and without unduly impacting performance, HIVE checks this folder for any new modules – any class, that is, having the default visual module as its superclass. If any are detected, the software creates a new drag-label for it in the toolbar and the component is ready for use.

Within the Department of Computing Science at the University of Glasgow, users of HIVE are already implementing their own extended modules – one user has created diagnostic components to measure layout stresses and run times exhibited by hybrid algorithms. With the ability to dynamically load visual modules, users can now share their algorithmic or visualisation components and incorporate them into HIVE while actively using it. A list of the implemented visual modules and their descriptions in the author's version of HIVE is given in Appendix A.

7.4.4 Ports

Visual components 'listen' to each other by way of their ports (illustrated in Figure 7.5). When a programmer writes a component, he or she must declare the ports that are necessary for the functioning and communication of the component. There are five types of port that a visual component can implement. These consist of the one-way data-in, data-out, trigger-in and trigger-out ports, as well as the two-way 'select' port. When declaring ports, this type must be defined. However, data-in and data-out ports may also define the structure of the data that will pass through them as well as the variable types comprising those data. Two forms of data structure that the ports cater for are high-dimensional feature vectors that can consist of real, integer, string and date variables, and 2-d real-valued co-ordinate vectors. Trigger-in and trigger-out ports can convey arbitrary data structures. Their purpose is to allow algorithmic modules to signify convergence and pass control to other modules – rather like control constructs in a conventional programming language. Selection ports pass integer arrays of selected datum indices between visualisation components.

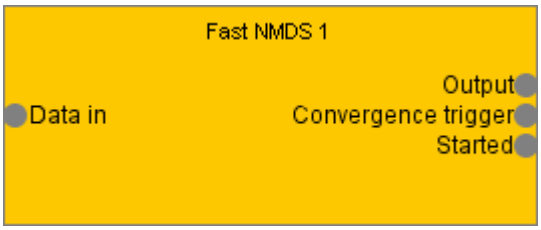


Figure 7.5 When HIVE is in link mode, all Swing components are hidden while port representations are rendered.

7.4.5 Linking and the composition model

The system's composition model is responsible for laying down the rules for which ports can be connected, based upon their port types. These rules comprise the default composition

model, however visual component implementations can override them to tighten or loosen connection constraints when required. An overview of these rules is shown in table 7.2.

Port linking rule	Description
polarity	one-way ports can only be connected to their complement
self-connection	ports on the same component cannot be connected
fan-in	one input port can be linked to only one output port
fan-out	one output port can be linked to many input ports
data-structure compatibility	data-in and data-out ports can only be connected when they are declared to handle the same data structure
data-variable compatibility	data-in and data-out ports can only be connected when they are declared to handle the same variable types

Table 7.2 Linking rules for HIVE's composition model.

The rules of the composition model constrain the user to create only legal and sensible connections between modules. To create a link, the user must place the system in 'link mode'. This is achieved via menu selection or by double-clicking the black background of the drawing canvas. When in link mode, HIVE hides all Java Swing GUI controls on each visual module, such as buttons and sliders, before rendering the ports as grey circles shown in Figure 7.5 (both links and ports are rendered using the Java2D API). Input ports are drawn on the left-hand and output ports on the right-hand side of each module and all ports are labelled as to their purpose. Ports are not visible during the normal mode of operation so that more space on the visual modules can be allocated to GUI controls useful in controlling algorithmic and visualisation parameters. While it would have been possible to render ports outside the edges of modules, it was felt that this would complicate the placement of port labels and would have made the resulting networks more cluttered.

The user creates a link between two modules by first placing the mouse pointer over a port and holding down the mouse's left button. This changes the selected port's colour to blue. There might be several modules on the drawing canvas and each might have several ports. To prevent the user from trying to make illegal connections and to save time, HIVE looks at all other ports and consults the composition model to see if a valid connection can be made from the selected port. If so, each potential target port's colour is changed to pink. This visual

feedback guides the user in connecting modules. To complete the link's creation, the user simply drags the mouse from the selected port to one of the highlighted ports. While doing this, HIVE provides additional feedback by rendering a link from the initially selected port to the current mouse position. When the mouse is dragged over a legal terminating port, that port turns green, signifying that the user can now release the mouse and the link will be made. Both data-flow links (between algorithmic modules) and view co-ordination links (between visualisation modules) are made in this way. However, to distinguish between them, data-flow links are rendered as thicker lines while co-ordination links are thinner (Figure 7.1).

A link can be selected by clicking on it with the mouse, which causes the link to turn red. When in link mode this causes the corresponding ports to be highlighted to identify the link's start and destination ports. Once selected, the link can be deleted or it can be dragged to bend it. Bending links allows the user to clarify connections and tidy up the resulting graph. See Figure 7.6.

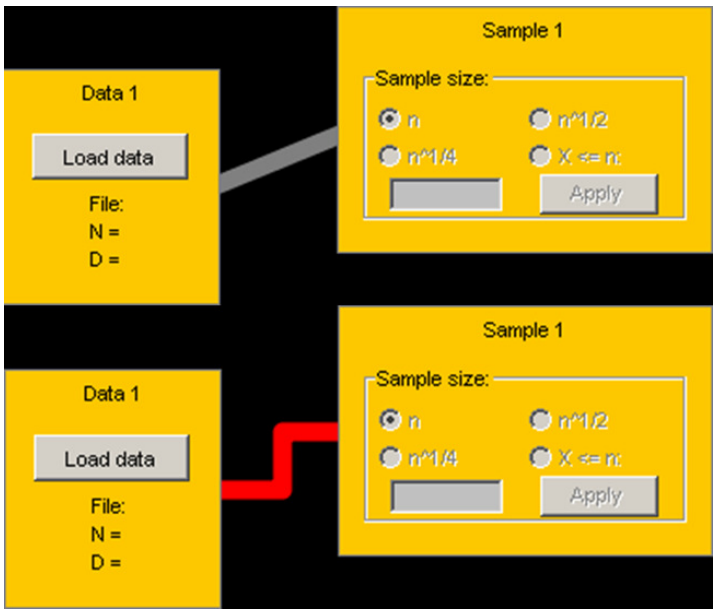


Figure 7.6 The top part of the image shows a link from a data source to a sample module. The bottom half of the image shows a link after it has been selected and bent by the user.

7.4.6 Hybrid algorithm generation

There is one exception to the data-structure compatibility rule above. This is to facilitate the semi-automatic generation of hybrid algorithms and occurs when the user connects a high dimensional output port such as the output of a data source component, to a 2-d input port such as the input to a scatterplot. In this case HIVE classifies the data on the output port according to the ordinal ranges of dimensionality and cardinality as described in Section 7.2.

Once this is complete, HIVE loads the appropriate algorithm from a default set of hybrid algorithms – the algorithmic 'cookbook'. These algorithms have been pre-classified in their applicability in spanning the grid of Figure 7.2, and are inserted between the two components that the user had originally connected, thus restoring adherence to the data-structure compatibility rule described above. Appendix B illustrates the cookbook in HIVE.

When HIVE has finished this process the user can run or modify the algorithm and visualise his or her data. It is suggested that this functionality might aid inexperienced users of the system, as well as encourage experimentation with hybrid algorithmic conjunctions.

HIVE allows users to save algorithms and visualisations by serialising module, link and port instances and writing them to file. The default set of algorithms in the 'cookbook' is also stored in this way in a 'patterns' folder within the system directory structure. If the user modifies the HIVE-generated algorithm, he/she can save it to this directory and specify that this should be used the next time HIVE is prompted to generate an algorithm under the same circumstance. That is, the data to be visualised are in the same LMH categories and the same type of visualisation is requested.

Overall, this architecture is inspired by the subject matter of Chapter 6. The data-flow model and visual programming used in scientific visualisation (Section 6.1) is used in conjunction with the design dimensions of secondary notation and side-by-side-ability (Section 6.3). HIVE also has functions to select data in scatterplot views and export to Microsoft Excel or export to PNG format graphics files. This awareness of interoperation with existing information tools is in line with the notion of information workspaces described in Section 6.2.

7.5 Examples

The system currently holds a limited number of composition components for creating visualisation applications and hybrid algorithms. Figure 7.7 shows a simple network of data, algorithmic and visualisation components. The data set used in this case is in the form of a CSV file containing 300 2-d co-ordinates representing a box that is open at one side. Note that the data are fed into the spring model and the table simultaneously. This creates a cyclic graph but in this case the scripting/composition model allows this because no conflicts between modules can arise due to the only output of the table being an interaction connection.

In this view, the spring model has finished laying out the data, however, during the iterative process, the spring model has output to the scatterplot the 2-d layout co-ordinates of the set after every ten iterations. This allowed the scatterplot to display an animation of the

layout process so that the user could watch the layout form. The link that is highlighted in red (link between table and scatter plot) is an interaction link. This means that by selecting rows on the table or points in the scatterplot, the corresponding items are highlighted in the other view. This location probing and its representation is an example of the flow of interaction possible within the system.

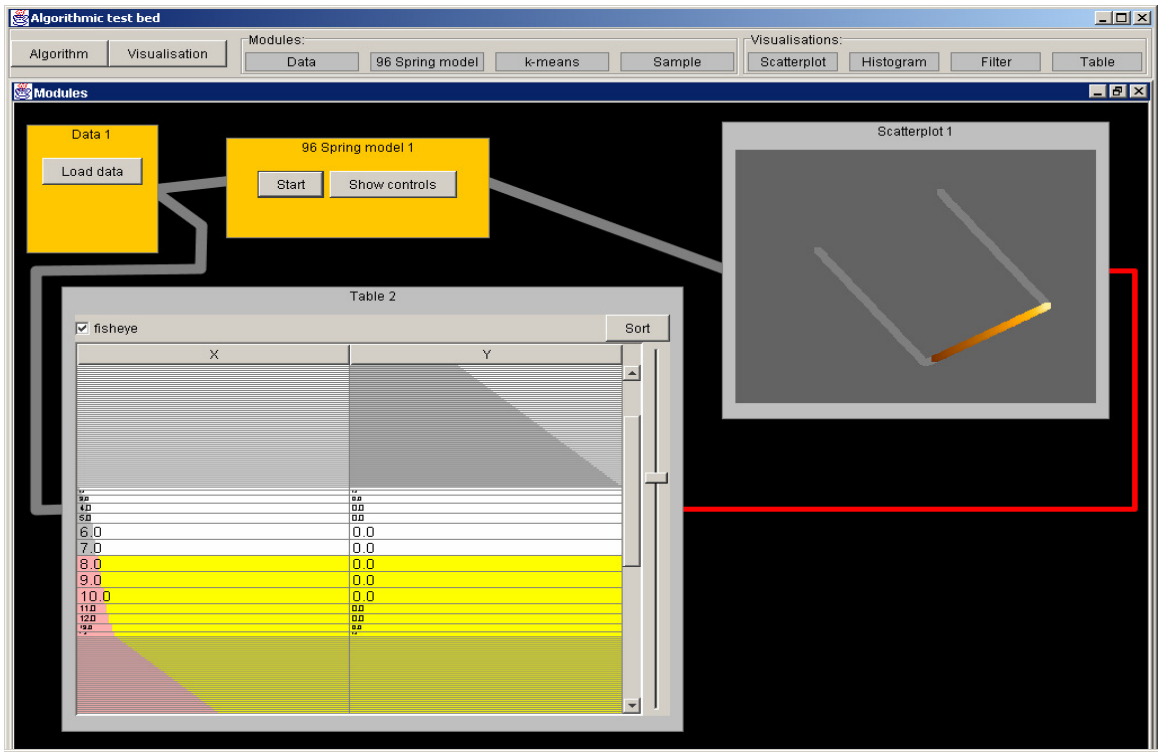


Figure 7.7 A network of the three types of components: data source (2-d geometric data), algorithm (Chalmers' spring model) and visualisation (fisheye table and scatterplot).

7.5.1 Comparison of spring model layouts

In the next screenshot (Figure 7.8), two data sources, two spring models and two scatterplots are wired together to provide a side-by-side comparison of the spring model layouts. The data used in this instance are a financial data set containing historical performance and volatility information on investment funds. In this set there are 1000 items, each of which has 13 dimensions.

By placing the scatterplots next to each other and connecting them with an interaction link it easy to identify the differences in the two layouts. We see that the model has converged upon the two major clusters but by *brushing* one of the plots we also see that these layouts have been flipped round with respect to each other. These differences exist because of the

non-deterministic convergence of the spring model algorithm – when used on multidimensional data, two layouts of the same data are seldom the same.

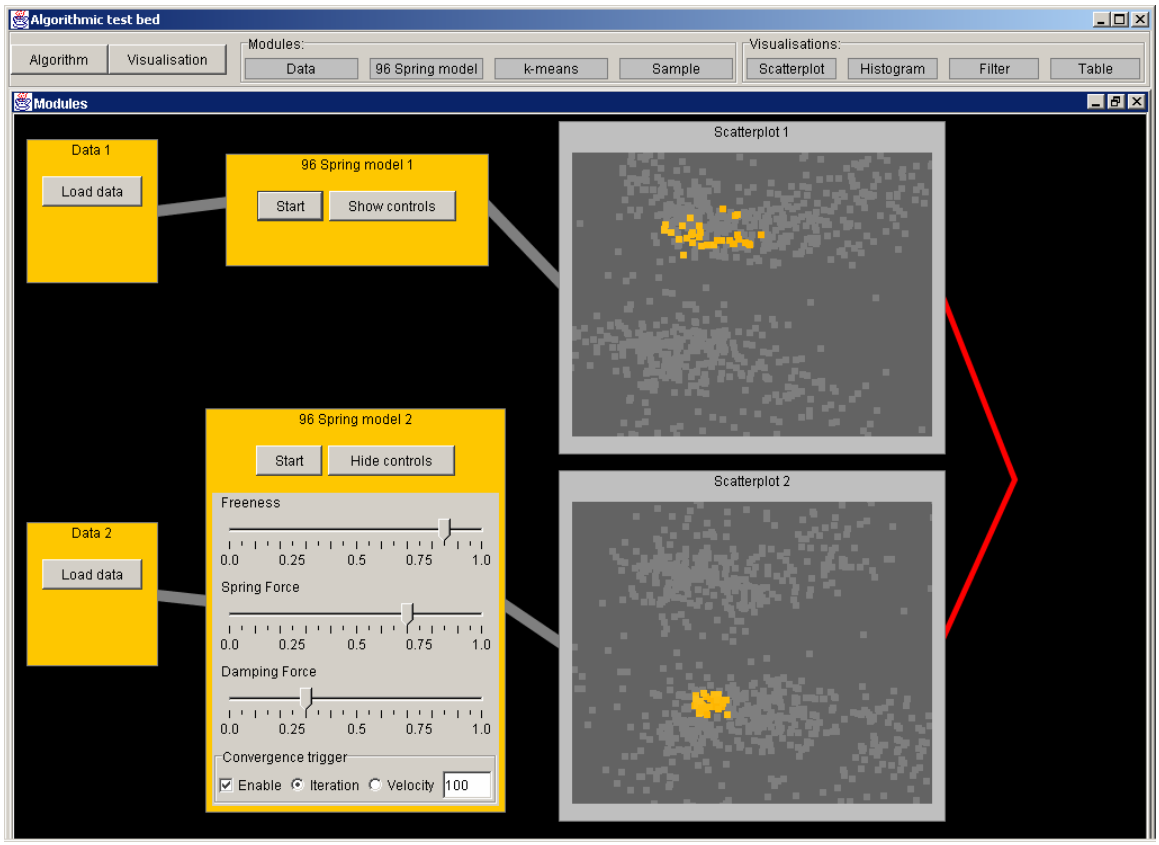


Figure 7.8 An example demonstrating the non-deterministic nature of the spring model. The expanded view of the bottom spring model component shows controls for changing parameters such as freeness, velocity and damping as well as controls for setting convergence criteria.

7.5.2 Exploration of a real data set

HIVE was used to explore a data set gathered from an eScience project within the Equator Interdisciplinary Research Collaboration (www.equator.ac.uk). The eScience team set up a remote sensing probe at a frozen lake in the Antarctic, which transmits data including ice thickness, water temperature, UV radiation levels etc. to environmental scientists at the University of Nottingham. The aim of this is to learn about carbon cycling processes. The data set was composed of 2202 probe measurements, each consisting of 16 variables measured at five-minute intervals between 17th January 2003 and 31st January 2003. This was converted into CSV format before importing it into HIVE.

Two algorithms were set up in parallel in HIVE and used to perform dimensional reduction of the data so that they could be rendered as a point distribution in scatterplots. One algorithm consisted of a neural PCA component and the other was generated automatically after the user specified the data set and visualisation tool, in this case a scatterplot. This latter algorithm was similar to the hybrid algorithm illustrated in Figure 7.1 with the exception that it used K-means instead of stochastic sampling in initially reducing the representative cardinality. Both algorithms took less than five seconds to run. By setting up these two algorithmic paths in parallel, it was possible to directly compare the visualisations produced (Figure 7.9).

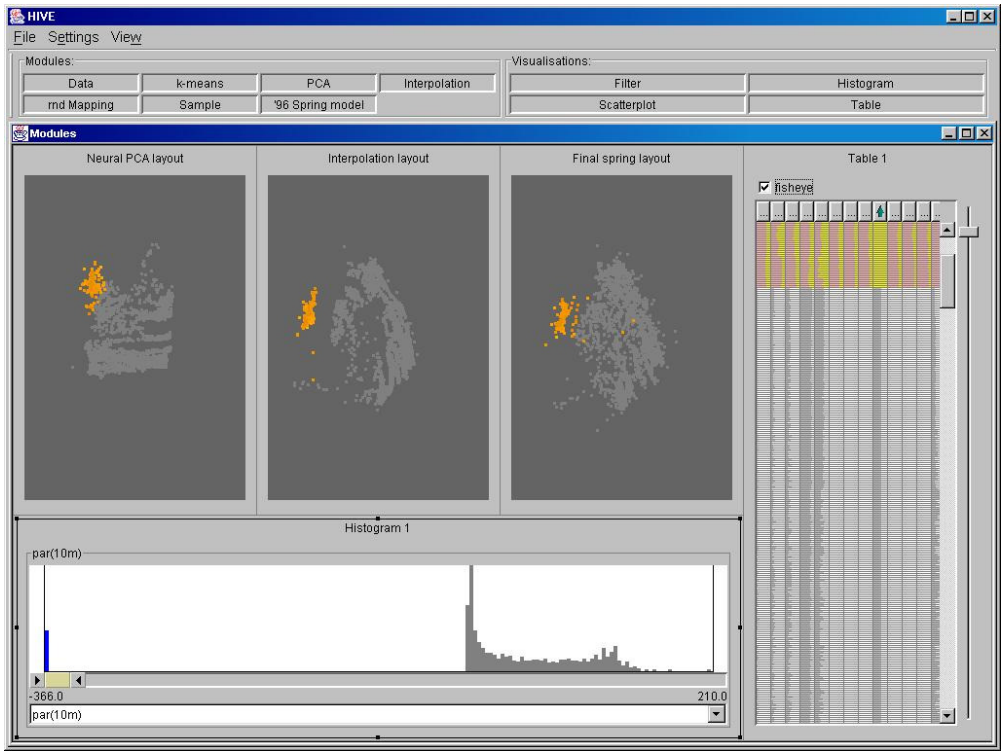


Figure 7.9 The leftmost scatterplot shows the output of neural PCA. The middle scatterplot shows the data after interpolation around the K-means centroids while the right scatterplot illustrates the output of the final spring model component. The highlighted cluster is a small subset of erroneous PAR measurements. These clusters are much clearer in the hybrid algorithm's plots than with PCA. The histogram shows the PAR distribution at a depth of 10 metres. The outlying peak (far-left) has been selected and this highlights the clusters in the scatterplots.

One notable difference between the visualisations was a small cluster made prominent by the hybrid spring model, especially in the intermediate view after the interpolation phase, which was not apparent in the PCA output. By linking a histogram and table to the scatterplots it was found that this cluster of points represented data where the photosynthetically active radiation (PAR) measurements at a depth of 10 metres were invalid. It turned out that these

erroneous measurements were caused by the light level exceeding the sensor's maximum input threshold.

The fisheye table view in Figure 7.9 has been sorted on PAR at 10m. The rows that correspond to the selection in the histogram and scatterplots are highlighted. This table depicts the data distribution over individual variables by colouring areas of each cell proportional to the value it contains. In its application here, it can be seen that the highlighted block of rows show that the distribution of values they represent is uncharacteristic of the other non-highlighted rows below them – the two regions appear disjointed. Although this clearly reflects the erroneous data, they would have been harder to identify without the help of the connected scatterplot. This is because without the scatterplot the user would have to sort each column in turn to look for such uncharacteristic distributions. Fortunately in this case, the low-dimensional representation provided by the scatterplot (and underlying hybrid algorithm) immediately caught the author's attention and made it easier to manipulate the table to take a closer look.

The two algorithms used here are examples of 'recipes' that are in the algorithmic cookbook mentioned in Section 7.2. Since the data set used here is deemed to be of moderate cardinality and dimensionality, K-means is applicable in reducing the representative cardinality (centroids) to make it low enough for Chalmers' spring model to converge very quickly and reduce the dimensionality to 2-d. From here, the rest of the data set is interpolated onto the layout to restore the representative cardinality. A final spring model step is added to run for a small constant number of iterations to refine the final layout. This algorithm was generated by HIVE to span the grid in Figure 7.10 from (*M*, *M*) to (*L*, *M*). If however, the cardinality of the data set was high, the algorithm would have had to span from (*M*, *H*) to (*L*, *H*), in which case HIVE would have utilised stochastic sampling instead of K-means in the initial phase, to speed things up. The other algorithm used in the exercise, neural PCA, was composed manually and can be regarded as a direct jump from (*M*, *M*) to (*L*, *M*) with respect to the algorithmic space in Figure 7.10.

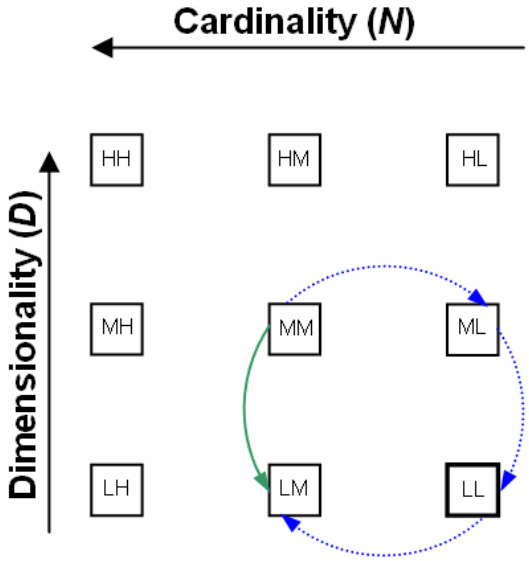


Figure 7.10 Dashed arrows represent the HIVE-generated hybrid algorithm spanning the space from (*M*, *M*) to (*L*, *M*) via K-means, Chalmers' spring model and Interpolation (clockwise). The solid arrow represents the manually instantiated PCA module.

7.5.3 Using MDS for feature selection

Recall that in Section 4.3.3 the use of MDS to help an analyst select features (variables) from a data set was described. This section will provide an example of how HIVE can be used to do this. The first step is to load a data set into a data source module. In this case the data were collated by the authors' colleagues at the Centre for Investigative Psychology (CIP) at the University of Liverpool. The data consist of 115 items each with 35 binary variables. Each item represents a person and each variable represents a particular crime to which the person has admitted. Figure 7.11 shows the modules and port connections required to allow the user to lay out the variables (rather than items) and select a subset to use in laying out the items.

The data source is shown on the top-left of the figure and feeds into a transpose module. The transpose module effectively turns the data matrix on its side so that the rows become columns and therefore each output item represents a vector of values for one variable across all of the 115 original items, i.e. the dimensionality is now 115 and the cardinality is 35. The transpose module also offers the user the option of standardising the value scales of the variables. The transposed data are then fed into an SSA (Section 4.3.3) module for dimension reduction, which in turn feeds into a scatterplot (lower-left) to display the layout of variables. In this layout, the proximity of variables reflects their co-occurrence. The "Data out" port of this scatterplot feeds the user's selection to another transpose module which restores the cardinality of the data, however, the dimensionality is now determined by the variables

selected in the scatterplot. These data are then fed into a spring model that provides a layout of the items in the second scatterplot. The thin link, shown in red between the scatterplots represents an interaction link. When the data are binary, as in this case, this means that if the user selects points in the variable scatterplot (lower-left) then the items that contain a '1' for those variables will be highlighted in the other scatterplot. Similarly, selecting items in the lower-right scatterplot will highlight the variables that have a value of '1' for the selected items.

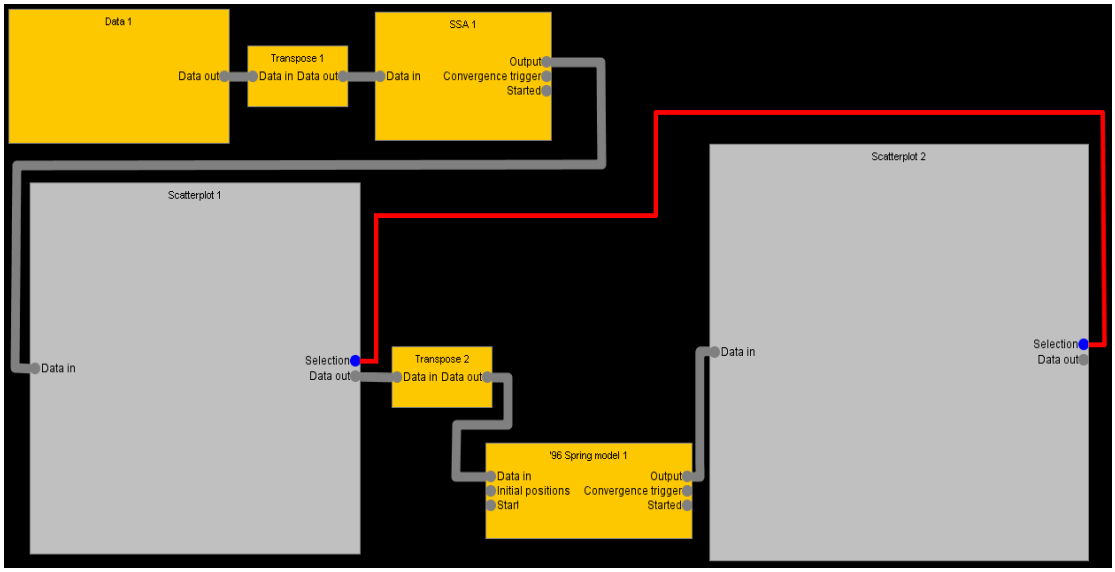


Figure 7.11 Port connections for using MDS for feature selection and subsequent analysis.

Figure 7.12 shows the above network after loading data and running the dimension reduction algorithms.

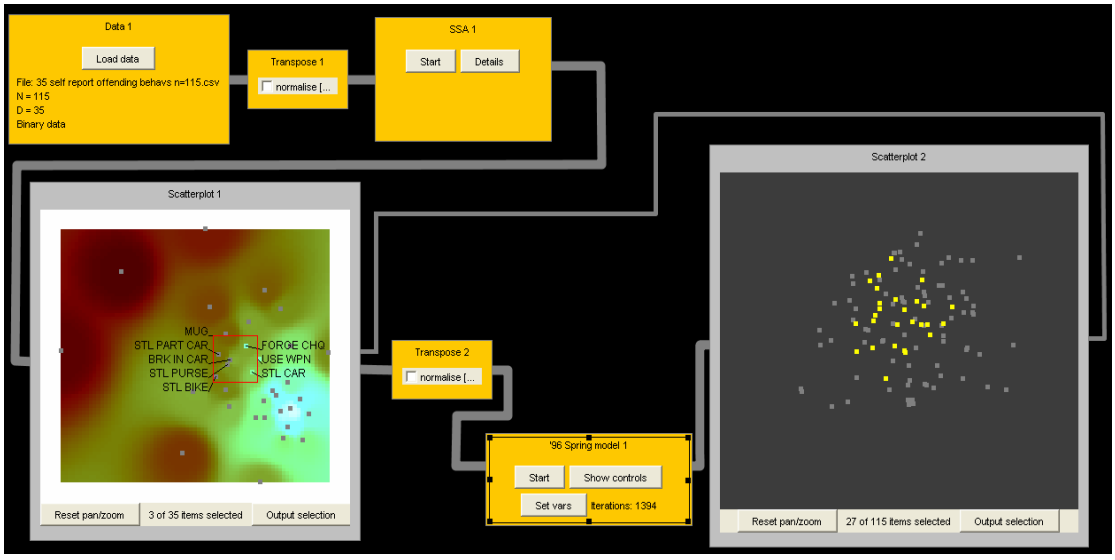


Figure 7.12 MDS for feature selection in action.

Given that the data are binary, the variable layout in the lower-left of Figure 7.12 has been embellished with a frequency surface – the lighter the area, then the higher the frequency of the variables represented by that region in the data set. The scatterplot also shows dynamic variable labels for the points near the mouse. The scatterplots in HIVE have several view-changing functions including custom annotation, changing background brightness, various colour schemes and displaying all point labels (if any) simultaneously.

In the figure, the author has selected several of the variables before pressing the "Output selection" button to send them to the second transpose module and then to the spring model. The highlighted points in the lower-right layout of items (people) show all those who have forged cheques, stolen cars and used a weapon. This is the type of functionality that researchers at CIP have been using to investigate whether the specific subsets of variables explain patterns in the population of items. Chapter 10 provides an account of this.

7.6 Design review

In the previous chapter each of the three main sections – Section 6.1 (data-flow model), Section 6.2 (information visualisation environments) and Section 6.3 (visualisation system design theory) – provided a subsection detailing the relevance of the reported observations from the literature to the design and implementation of the HIVE framework. These observations were made during what can be considered as a requirements gathering phase. This section will describe how these reflections on the literature have influenced HIVE. Finally, subsection 7.6.4 provides a complete list of the key features in HIVE.

7.6.1 Data-flow model

In scientific visualisation the data-flow model coupled with the visual programming paradigm can potentially facilitate the following advantageous features:

- *Parallel processing*
- *Interoperability*
- *Collaboration and communication*
- *Animated, interactive simulation*
- *Traceable computation*
- *Subjunctive presentation*
- *Appropriation*

It was suggested that the HIVE framework could also attain these advantages because the underlying algorithmic architecture acts as a data-flow model. Indeed, the nature of the proposed architecture should add increased efficiency to the above list. This coupled with co-ordinated views – which are akin to the visual programming paradigm – provides the flexibility that allows the user to drive the application rather than the other way around.

An additional feature that has been inspired by the review of data-flow models in scientific visualisation is that to open up the applicability of the system, it should allow the production as well as the consumption of data as in [BBB*93, UFK*89]. If users can modify the underlying data set through the same interface as for its analysis then the system might be more beneficial in real world scenarios. This was put forward in section 6.2.4 as a potential reason for the failure of widespread adoption of contemporary information visualisation systems such as Visage. This has been implemented to an extent in that the user can interactively apply feature selection and extraction, as well as filter and sample the data, essentially modifying their composition and meta-data.

From the description of the framework in Section 7.2 it is implied that it is data-driven in the sense that the complexity of the data determines their *path* through the hybrid algorithmic architecture. The data are *pushed* through the hybrid algorithm. However, it is highly likely that the algorithms on the most efficient path through the model may not be capable of producing the visual representations that the user desires. This means that there must be a trade-off between the complexities of the data that 'push' them through, and the chosen representations (views) that essentially 'pull' them through. This means that the HIVE framework is essentially data-driven and goal-driven. The data-driven aspect is automatically determined by the system's implicit suggestion of a path through the hybrid architecture, and the goal-driven aspect is determined by the user's visualisation requirements. This is a departure from the typical approach in the scientific visualisation system described in Section 6.1. As well as using visual programming and the desired visualisations to explicitly determine the data-flow in the system, the HIVE framework also uses the data-complexity and the required visualisations to implicitly define the data-flow.

7.6.2 Information visualisation environments

The role of information visualisation environments is to transform abstract data into information to supplement the user's perception in creating knowledge from the data. This can be achieved by enhancing visualisations by using multiple co-ordinated views for interaction flow, effective representations via abstraction management and sensemaking, by and within the views. The hybrid architecture addresses Card et al.'s cost structure of information by

increasing the efficiency in generating the data segments that will be the basis of visual representation. These issues address the notion of a workspace that is an information-rich environment for problem solving.

The HIVE framework is as flexible as North's level-3 system (defined in section 6.2.4) by allowing a variety of visualisations of different types of data, and by providing the user with the means to connect multiple views to enhance interaction. This increases the applicability of the system across different work domains and essentially allows the user to help steer the transformation of abstract data into information. If collaboration and communication services are also built into the system then it will be closer to being an effective virtual workspace. This has been partially addressed by using Java's reflection framework to allow users to swap and integrate visual modules at run time. The ability to export data and save analyses is also a step towards this ideal.

7.6.3 Visualisation system design theory

Shneiderman's visual information seeking mantra hints at the useful interaction techniques that are applicable in gaining more information from visualisations. This, however, is not enough to ensure the effectiveness of an information system in facilitating the discovery of knowledge. Design theories such as EID, cognitive dimensions of notations, dimensions of expression and abstraction management must supplement visual information seeking.

Ecological interface design has taken a different perspective on the role of information systems. EID suggests that by expanding the boundaries of the system to include aspects of the work ecology, the target system is much more likely to be useful in the work domain. In addressing the notion of a virtual workspace, the HIVE framework has been designed with not only the interaction mechanisms (situated constraints) in mind but with how it will be used within the working environment and how it may be augmented by services for sharing information (and knowledge) with co-workers.

The design principles provided by Green's cognitive dimensions of notation and Roth et al.'s dimensions of expression provide a richer design vocabulary for describing situated constraints and the trade-offs between them. Of the cognitive dimensions, secondary notation and abstraction greatly appealed in considering the design of the HIVE framework. The informality of interfaces that employ secondary notation (as with HIVE) provides a vehicle for appropriation and encourages opportunistic searching of information. Abstraction is important because, as addressed by Russell et al.'s cost structure of sensemaking, some representations (abstractions) of information are more effective than others. Also, if the system allows the user to modify abstractions such as by making changes to the underlying data set and by

transforming views, then some form of abstraction management must be in place to efficiently maintain consistency in the information visualisations produced. In HIVE's architecture, the graph manager and its composition model regulate the higher-level abstraction of the visualisation application itself.

7.6.4 HIVE features

The key features that have been built into HIVE are:

- user-defined and directly manipulable data flows
- user-defined and directly manipulable interaction flows
- an algorithmic framework for semi-automatically generating hybrid algorithms
- an extensible palette of algorithmic, visualisation and profiling components
- histogram range-selection provides dynamic querying of other views
- visual and dynamic profiling of hybrid algorithms
- user-defined colour schemes, annotations and Excentric labelling [FP99] in scatterplots
- capability for indexing and mining raw text
- export data to MS Excel
- zoom and pan scatterplots
- automatic layout segmentation
- caters for CSV, lower triangle and raw text input data
- tabular data views provide focus + context
- surface plots of variables
- cross platform – developed in Java

7.7 Conclusions

The flexible algorithmic framework of HIVE adds power to visualisations because the time taken to generate the visualisations, and therefore also the view transformations via efficient hybrid algorithms, is reduced. This helps maintain the cause and effect relation between a user's actions at the interface and the subsequent visualisations. In the context of figure 7.2, interaction brings the human into the loop, effectively closing it by allowing exploration of high-dimensional abstract data.

The work documented up to this point has answered, at least to some extent, the first two research questions posed at the beginning of this thesis: Which algorithmic components

should be combined? When should the different types of algorithms be used? The former is answered in the experiments with hybrid algorithms and the development of the hybrid algorithmic framework in HIVE. The framework dictates that a series of algorithmic stages should be matched to the complexity of the data set as it is transformed by them, successively refining and improving its representation for visualisation. The algorithmic components that have been used include K-means clustering, stochastic sampling, novel radial interpolation, Chalmers' spring model, SSA, PCA, fast NMDS and Voronoi clustering. Note that some of these components are themselves hybrid algorithms. The latter question, again, pertains to the hybrid algorithmic framework. The order of successive algorithmic stages is very important to the outcome of a hybrid algorithm and depends upon the components used. The novel algorithms for dimension reduction described in Chapter 5 suggest that an inexpensive algorithm should first be used to reduce data cardinality (such as K-means clustering) before a more expensive algorithm (such as NMDS or a spring model) works upon this reduced representation of the data to further reduce it – this time by dimensionality. Finally, the representative cardinality can be restored by fast interpolation before fine tuning with restricted application of a dimension reduction routine such as Chalmers' spring model.

The composition of hybrid algorithms by direct manipulation of visual modules produces a system schema that can be understood more easily. The flow of data and interaction can be visually traced through the system. Another benefit of this interactive approach is that hybrid algorithms can be set up in parallel in a similar fashion, as in Figures 7.8 and 7.9 above, i.e. two or more runs can be made simultaneously. This is useful for testing the robustness of the algorithms with respect to different types of data and starting conditions. This approach can also be used to determine and compare the run times and output quality of composite algorithms as well as the individual components. The interactivity of the system is very useful when it comes to evaluating and using hybrid algorithms, as will be seen in the next chapter.

8. Algorithmic profiling

HIVE has been demonstrated to be an effective environment within which to create hybrid algorithms and explore high dimensional data sets. A palette of algorithmic components and visualisation tools provides the user with several disparate views of a data set and allows a number of different aspects to be explored. Further insight is supported via coordination between these views. Novel combinations of modules may be experimented with, and the extensible nature of the algorithmic palette permits the simple addition of new components.

In addition to this, it is proposed that HIVE is a useful tool for profiling and evaluation of hybrid algorithms. A number of HIVE modules have been implemented to measure and display performance characteristics of other HIVE components. Such profiling modules permit algorithm evaluation to be tightly and interactively coupled with the algorithms being run. Linking together profiling modules can be carried out using the same visual metaphors, and the choice of algorithmic properties to measure can be made and altered at run time. Profiling tools may also be linked to existing visual modules, with their coordinated use providing insight into data sets that would go unnoticed in a sole visualisation. Examples of such coordination are provided in Section 8.4.

This chapter introduces the profiling modules implemented in HIVE. All of these modules, with the exception of the interactive Shepard plot were implemented by the author's colleague, Alistair Morrison. However, the author did assist Alistair with this implementation.

8.1 Multiple runs module

When evaluating an algorithm, several runs over several data sets must be executed, often with different algorithmic parameter settings. Manually coordinating such algorithmic executions can be a laborious and time-consuming task. To alleviate this, a Multiple Runs (MR) module has been implemented as a central controller for automating such operations.

The MR module passes data sets and algorithmic parameters through its output ports to a connected algorithm. At the start of each run the MR module tells the algorithm to begin execution and once execution is complete the algorithm notifies the MR module to start the next run. Instructions for the module are specified by the user via a text field and are in the following format:

(*DataFile,*[*NumRuns*, <*ModuleID*, (*parameters*)>,<>..])

The *DataFile* parameter specifies the input data file to pass to the algorithm and the *NumRuns* parameter determines how many times the algorithm will be run on this data set. The <*moduleID*, (*parameters*)> tuples specify an algorithmic component and its relevant parameters. A spring model, for example might be identified by *moduleID* and parameters such as the number of iterations and damping might also be specified. Many data sets and experimental conditions can be specified in this way and batch runs of more than one algorithm can be executed simultaneously using multiple MR modules.

8.2 Stress and clock modules

Recall from Chapter 4 that stress (Equation 4.20) provides a measure of the discrepancy between layout distances and high dimensional relationships. It quantifies the *goodness of fit* of the layout to the high-dimensional data space. To analyse dimension reduction algorithms in HIVE with respect to stress, a new module was implemented. This module simply takes the output from an algorithm and performs the stress calculations. It can be used in conjunction with the MR module by measuring stress after (and during) algorithmic execution. Since it is a self-contained module, it can have multiple instantiations and can be used to measure stress at intermediate algorithmic stages, such as after the initial spring model layout in the algorithm described in Section 5.3.

Algorithms should also be evaluated with respect to running time. To this end, a clock module was implemented with trigger ports to commence and terminate timing of algorithms. Like the stress module, clock modules can be used to take measurements at intermediate stages of hybrid algorithms.

The output of both stress modules and clock modules can be used in several ways. One approach is to write the values to a file for exporting to Excel to produce graphs such as those shown for the evaluation of algorithms in Chapter 5. Another use of the modules is to build performance charts while algorithms are executed (see Figure 8.1). By charting performance measures while watching a layout form, the user can ascertain whether the algorithm has reached a local minimum and whether more iterations are required. A final, more novel use introduced by the author is to collate performance measures, along with algorithmic parameters and data sets sizes for individual algorithmic runs, into new multidimensional data sets and use algorithms in HIVE to visualise them [RMC05]. Morrison [Mor04] used this technique to distinguish between the behaviour of several types of dimension reduction algorithms in HIVE. Each algorithm was applied to 9 data sets, and both run time and stress

was recorded for each execution using the profiling modules. With each algorithm represented by an 18-d vector, a spring model was then used to visualise the results.

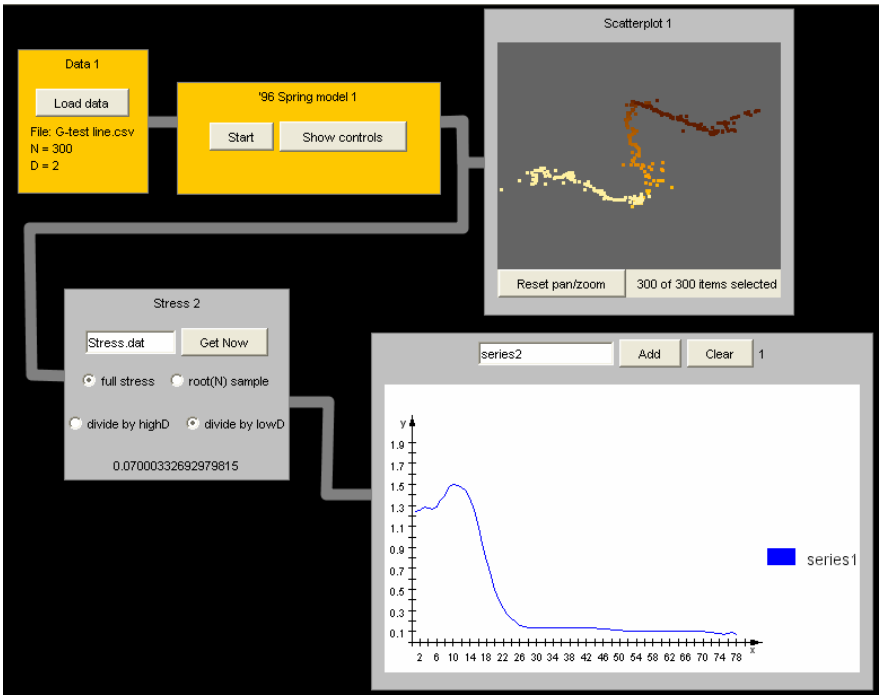


Figure 8.1 As a spring model runs, stress is measured and charted against each iteration. The plateau, after around 26 iterations, shows that the algorithm has fallen into a local minimum while the layout (of the 2-d data) shows that more iterations are required to break out of the minimum configuration.

8.3 Shepard plot

The Shepard plot [She62], as demonstrated in Chapters 4 and 5, is another tool that can be used to illustrate the quality of a dimension reduction solution. The Shepard plot shows the relationship of the high-dimensional distances (between each pair of items in the data set) and the corresponding low-dimensional layout distances. Given an ideal fit of high-dimensional data into a space of lower dimensionality, the points in the Shepard diagram form a 45 degree diagonal. As the fit becomes less representative, points will start to deviate from this diagonal. Figure 8.2 shows the Shepard diagram in HIVE. High dimensional distances are plotted along the y-axis and layout distances are plotted along the x-axis. Points that lie above the diagonal represent items that are too close in the low-dimensional space, while points lying below the diagonal represent items that have been placed too far apart. Outliers in the Shepard plot indicate pairs of objects that are likely to contribute more to layout stress.

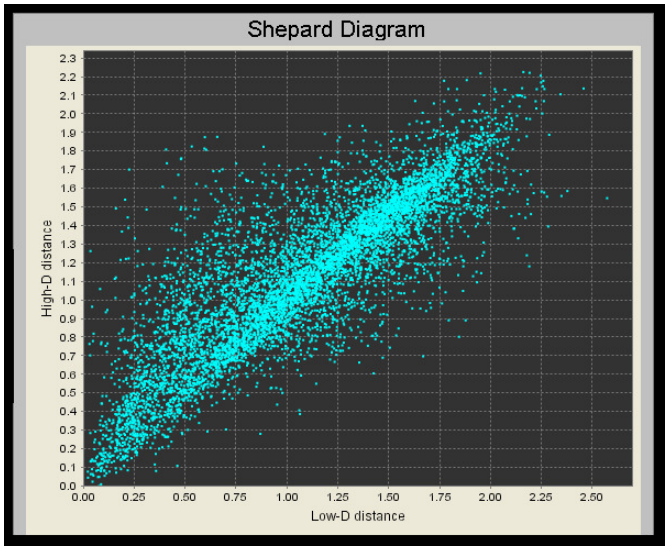


Figure 8.2 The Shepard plot module.

Since the calculation of the Shepard plot requires $O(N^2)$ time, controls have been added to allow the user to specify the size of a random data sample over which the plot is created. This plot can also be coordinated with the corresponding scatterplot representing the dimension reduction's output. This allows one to select points in the Shepard plot and highlight the corresponding points in the layout.

The Shepard plot has been used for decades for evaluating the output of MDS routines. Traditionally it is a static presentation of the output of an MDS routine. However, its novel interactive incarnation in HIVE gives it more analytical power and increases its applicability to larger data sets.

8.4 Coordination of profiling modules in HIVE

The previous sections have described isolated instances of HIVE's new profiling modules. The real power of such techniques, however, comes in their combination and interaction with existing components within the HIVE environment. Histograms, fisheye tables and scatterplots all have interactive functionality, allowing coordination with the profiling components.

For example, the Shepard plot is traditionally a static presentation technique for qualitatively evaluating a low-dimensional representation of high-dimensional data. By incorporating it into the HIVE framework, the plot can have as many instantiations as

necessary, with each instance connected to a different part of the visualisation's data-flow. It should also be noted that the Shepard plot is traditionally employed after the completion of a layout, however, in HIVE the Shepard plot can be used during the layout process to give an incremental account of the quality of the layout.

By connecting a Shepard plot to a scatterplot, one creates an interactive link between the two plots. Making a selection in the Shepard plot will therefore highlight those objects in the scatterplot layout whose pairwise distances correspond to the selected points. For example, in Figure 8.3, Shepard plots are used to compare layouts obtained from PCA and a spring model. It is apparent that the Shepard plot of the PCA layout has a distinct diagonal edge, below which no points are plotted. This may be explained by the fact that PCA functions via a projection of the high dimensional space onto a 2-d plane. In contrast, the spring model has no linear constraints and attempts to position objects to best preserve high-dimensional relationships. This results in a Shepard plot where there are points both above and below the diagonal, as well as the points that represent items ideally placed in the plane.

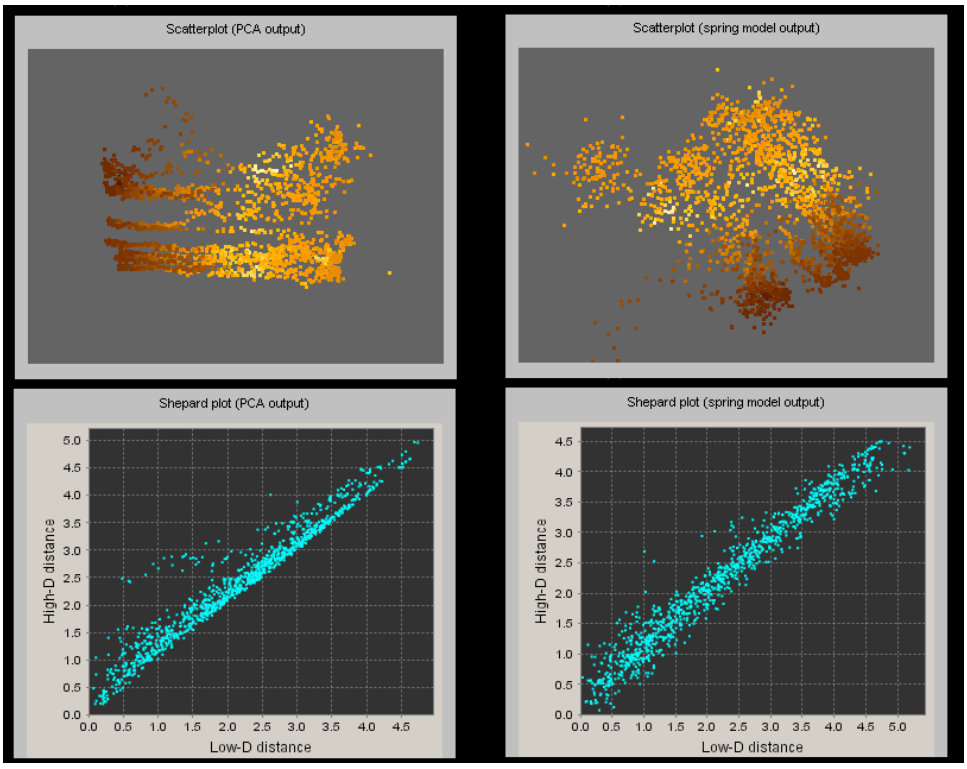


Figure 8.3 The top two images show 2-d layouts using linear PCA (left) and spring model (right). Below each layout is the associated Shepard plot. The PCA layout has no pair of objects at a greater distance from each other than in high-dimensional space. This is confirmed by the fact that no objects appear below the diagonal in the PCA's Shepard plot.

Another implemented component that may be used in interactive combination with the profiling modules is a module for Voronoi tessellation and clustering (see Section 5.5). This module may be used to partition a completed layout. Each point is contained by a convex polygon so that the portion of space contained within the polygon is closer to that point than any other. Clustering may then be performed by finding contiguous groups of polygons where the density of points is similar. The module may be used in combination with profiling modules to detect clusters that may benefit from closer examination.

The following section provides more concrete examples of the coordination of components with a series of case studies.

8.5 Case studies

This section documents several case studies, demonstrating the coordinated use of existing components with the novel profiling modules within HIVE.

8.5.1 Batch job of executions for algorithm evaluation

The first case examined is the evaluation of a novel algorithm. The use of the profiling modules described in the previous section allows such an evaluation to be performed simply and in an intuitive manner. The following describes the evaluation process undertaken in the writing of a paper that presents a novel hybrid layout algorithm that was implemented and evaluated in HIVE [MC04].

In performing such algorithmic profiling, a large number of executions is necessary. Several models might be evaluated on several different data sets. In addition, results should be averaged over multiple runs: an especially important consideration in the case of iterative models, which can occasionally become stuck in local minima. It is common to require several hundred executions for a thorough evaluation, and it is therefore clear that an automated profiling process is a useful aid to the designer. Figure 8.4 illustrates the configuration of components required for such an evaluation. To avoid unnecessarily describing the specific model in depth, details such as the names of individual components have been omitted from the figure.

Having built a hybrid algorithm (composed of the modules shaded in yellow in Figure 8.4), it is desirable to examine its performance in comparison with an alternative technique. Profiling modules (grey) may be added to the module configuration at the user's discretion. Here, the author has elected to measure the run time of two stages. Stress is also measured at

the conclusion of stage four. By specifying file names on each of these components, separate output files are generated by each, allowing the performance characteristics to be explored further in other applications such as Microsoft Excel.

The MR module (top-left) is provided with a list of data files and parameter commands. It systematically loads in each file, and passes the instructions for the current execution to each algorithm component. The MR module passes a start trigger into stage one, and receives another trigger from the final stage to indicate that the algorithm has terminated: the cue to reset all modules and begin another run.

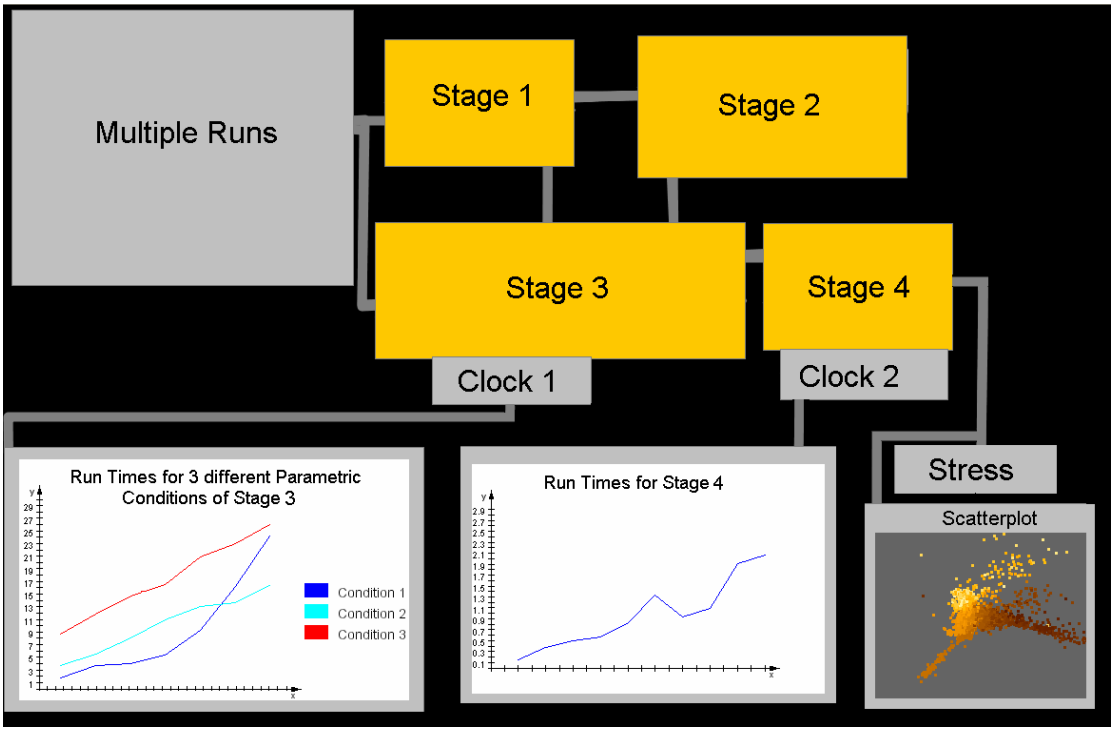


Figure 8.4 The yellow modules represent different stages of a hybrid algorithm. The Multiple Runs module coordinates a sequence of executions, loading data and parameters into each component. Charts plot run time against data set size at various stages of the algorithm. The bottom left chart shows run times under three separate sets of parameters for stage three. Having connected the various components and provided instructions to the MR module, the algorithm executions and chart plotting may proceed unsupervised.

Finally, charts at the bottom of the figure display experimental results. A test data collection has been sampled to create data sets of varying size. The charts show run times against data set sizes under different algorithmic conditions, with each line on the chart representing a different condition. Each chart is connected to a different clock module, and therefore displays times taken by different algorithmic components. For example, the chart at

the left-hand side displays the run times required by the third stage of the model. Three separate approaches were experimented with for this stage (as specified by the experimenter in the MR module and passed to stage three via parameters), as indicated by the three lines on the chart. It can be deduced that on small data sets, condition one executed in the least time, whereas condition two becomes optimal as data size increases.

Charts of this type formed the basis of the results section of a paper by Morrison and Chalmers [MC04]. What could have been a laborious evaluation procedure was undertaken via a simple, unsupervised process. The algorithms can be left to run overnight, and the generated charts can be exported as bitmaps or JPEG files.

8.5.2 Exploratory analysis of synthetic data

This example illustrates the interactive combination of the Shepard plot with other HIVE visualisations. As mentioned before, brush-and-link coordination has been incorporated in the Shepard plot to allow interaction with other components. For example, linking the Shepard plot and scatterplot views allows insight into the relationships between quality of positioning and objects' placement within the layout.

To illustrate the utility of this interactive capability, an example is provided using a synthetic data set representing a 3-d cube. Such a data set is a useful test case as it is impossible to represent perfectly in a 2-d space, and no 2-d projection of the data is much better or worse than any other. To begin, PCA is used to obtain a scatterplot layout (Figure 8.5(a)). Linear projection-based layout techniques such as PCA and SVD, although fast, provide a layout based upon global data properties (e.g. variance). It is therefore the case that certain local areas might be poorly represented. This example illustrates how interactive use of the Shepard plot can help a user to resolve inaccuracies in these areas, and thereby enhance understanding of the structure of the data.

The cube structure is clearly visible from Figure 8.5(a), coloured dark to light from top to bottom. Figure 8.5(b) shows a Shepard plot of the layout generated by PCA. Each point in the Shepard diagram represents a distance between a pair of objects. In Figure 8.5(b), the author has highlighted a section of points in the upper left of the layout: those points corresponding to the relationships worst represented in the PCA layout. Figure 8.5(c) shows how this selection affects the scatterplot display. The linking between views indicates that the objects worst represented in the layout appear in the centre. These objects represent points at opposite corners of the cube, forced together in the projected layout.

Having identified such a poorly represented area of the layout, it may be desirable to extract the subset of objects in that region and lay them out separately. In doing so, it is

possible to remove the influence of the full data set, and examine only relationships between the objects in that subset. The selected region was therefore fed into another PCA module and re-projected, yielding the layout shown in Figure 8.5(d). It can be clearly seen that the inter-object distances are now more accurate; the two corners of the cube have been separated. As a quantitative measure of the quality of the layouts, the stress of the full layout was measured and compared to that of the sub-layout. As expected the stress of the sub-layout was much less than that of the global layout: 0.001 compared to 0.149.

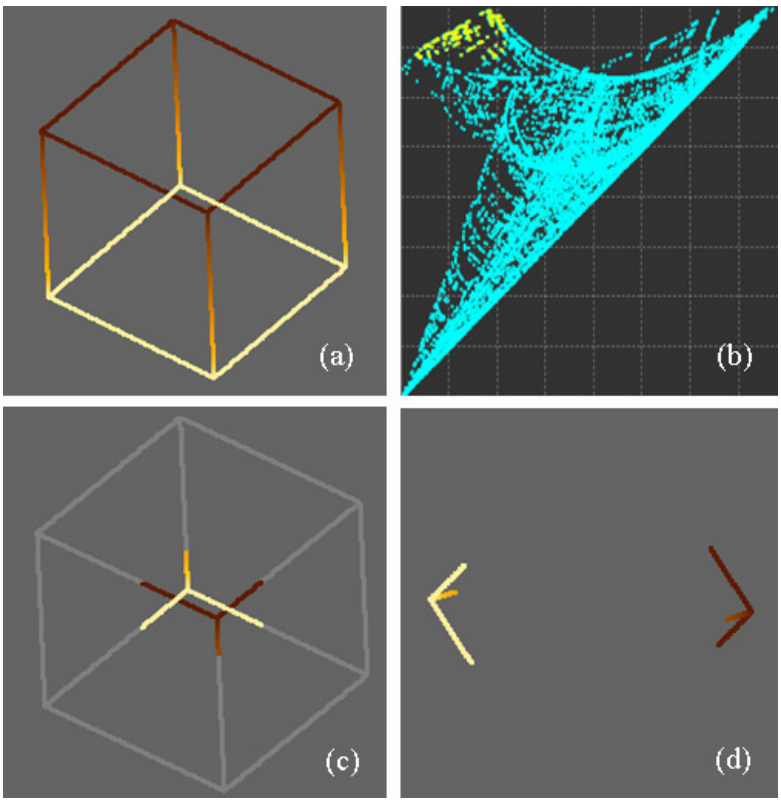


Figure 8.5 PCA layout and Shepard plot working together interactively to help build user understanding of a data set. The PCA layout (a). A selection is made in the Shepard plot of points corresponding to distances in the layout that may benefit from further analysis (highlighted region) (b). The selection in the Shepard diagram is also highlighted in the PCA layout (c). A re-projection of the selected points and their immediate region confirms their misrepresentation in the original layout (d).

8.5.3 Exploratory analysis of real data

The above scenario demonstrated, via a simple example on synthetic data, how a profiling component could be used interactively in combination with other views to encourage further exploration of a data set. A similar example will now follow to illustrate the usefulness of such techniques in a real-world setting. The data used here are the same as in Section 7.5.2

and were gathered with a remote sensor probe during an investigation into carbon cycling in Antarctic lakes (www.equator.ac.uk). They represent a number of properties measured over time, such as water temperature and the level of photosynthetically active radiation.

The data set was initially fed into a PCA module with the Shepard plot used to identify a local region of items that were potentially badly placed. In a manner similar to the previous example, points far away from the diagonal trend were selected in the Shepard plot, which resulted in the contributing items being highlighted in the connected PCA layout. The leftmost two components of Figure 8.6 illustrate the scatterplot and Shepard diagram following this selection.

Having identified these poorly represented objects, they appear to be localised to a specific region in the top-right of the layout. One might hypothesise that this area represents a distinct cluster within the data, which has not been made apparent by the PCA layout. A Voronoi clustering component (see Section 5.5) can be employed to gain a clearer understanding of the partitions within the data. The output from PCA is fed into a Voronoi component, which identifies five separate clusters in the layout.

The Voronoi component is illustrated in the centre of Figure 8.6 and is shown in detail in Figure 8.7. Five clusters were found, and shown in different colours. Outlying objects not identified as belonging to a specific cluster were coloured grey. It can be seen that the objects highlighted in the PCA layout all belong to the yellow cluster. This subset is selected and overlays the Voronoi tessellation.

Having now identified a cluster of the data within which certain distances are poorly represented, it is possible to extract it for further exploration to determine why this is the case. Through connecting to the Voronoi output port, another component may take as input the selected cluster. The figure illustrates how the cluster is passed to a spring model (FDP). This non-linear technique is able to discover further detail that PCA could not identify: two clear sub-clusters are found within the selected data.

The PCA layout had clearly failed to adequately separate these two sub-clusters, which explains the large discrepancy between high- and low-dimensional distances observed from the Shepard plot. The measures calculated by the stress module confirm the findings, with the PCA layout exhibiting 0.031 and the spring model layout of the extracted cluster giving 0.025.

Having discovered the presence of two sub-clusters, it is interesting to see how they are depicted in the original PCA layout. Comparing the spring model layout and the Voronoi display, it might seem as if the smaller of the two sub-clusters appears on the left of the yellow Voronoi region, with the larger C-shaped sub-cluster appearing on the right. Had the two images been produced independently, one may have made this assumption. HIVE's interactive

and coordinated view framework, however, allows the user to compare the location of the same objects in different layouts. Figure 8.8 shows the selected C-shaped sub-cluster in the spring model layout and the resultant highlighting of the corresponding objects in the PCA projection. It can be seen that the division between the two sub-clusters actually occurs in the top-right corner of the PCA layout.

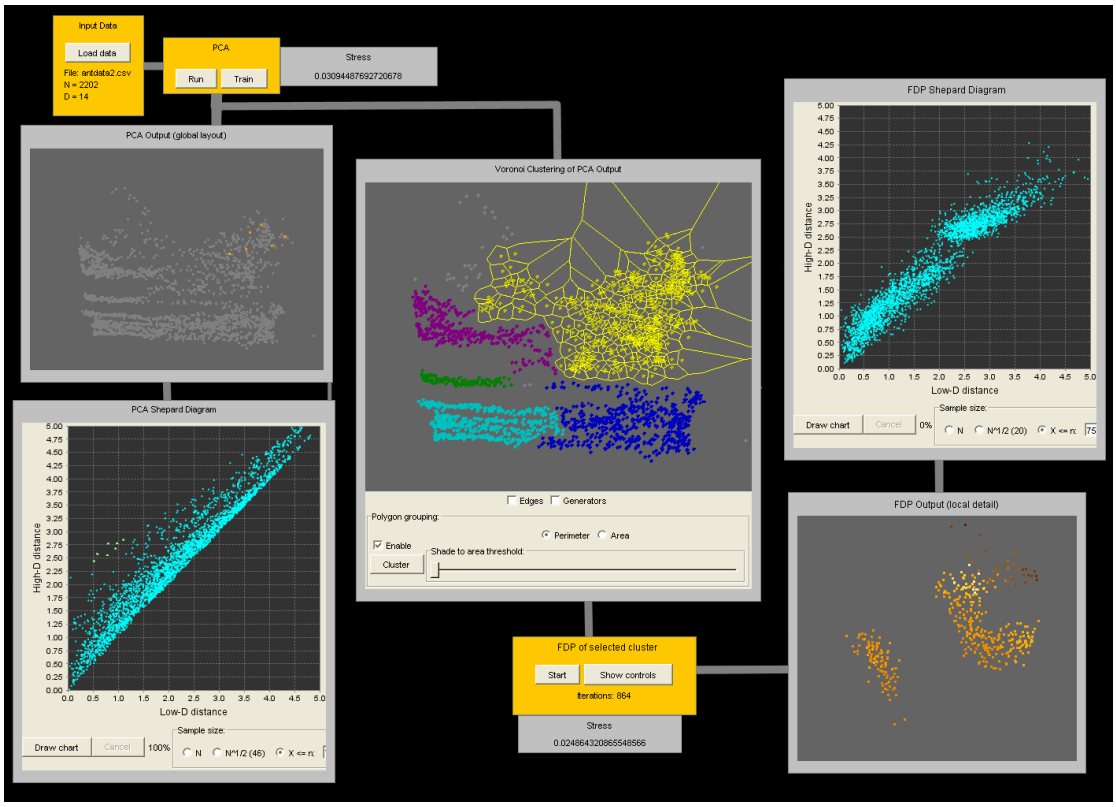


Figure 8.6 Various algorithmic and visualisation components working together in a coordinated environment. A PCA layout is made and the associated Shepard diagram is used to detect a local area that might be better represented if considered separately. A Voronoi tessellation component is used to cluster the data, and extract the cluster containing the previously identified local area (the yellow objects in the central component). This cluster is processed with a spring model (FDP) routine, which uncovers two sub-clusters that the author had not previously been able to identify.

The final stage of processing undertaken in Figure 8.6 is the creation of the Shepard plot of the spring model layout, shown in the top right corner. In tandem with the stress calculations, this allows a visual comparison of the degree to which the PCA and spring model layouts preserved high-dimensional relationships.

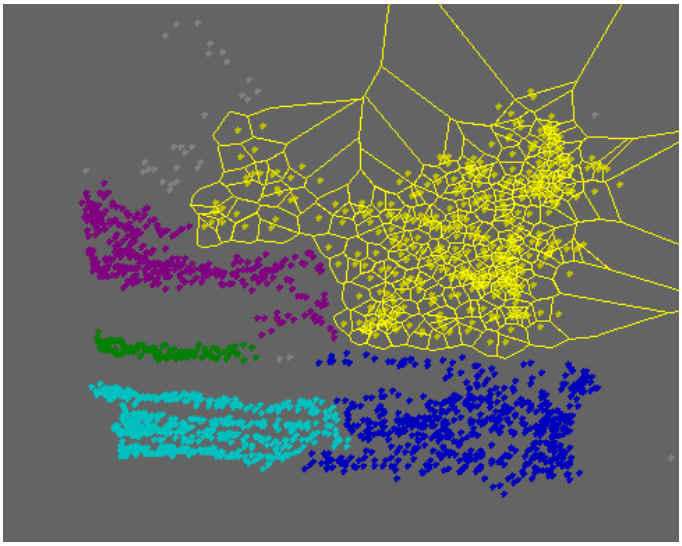


Figure 8.7 . A close-up view of the Voronoi module in Figure 8.6.

The Shepard diagram resulting from the PCA layout exhibits a 'cleaner' line on the 45 degree diagonal, as would be expected from the discussion in section 8.4. It is the spring model's Shepard plot, however, that appears to show less deviation from the diagonal overall. This is perhaps better illustrated in Figure 8.9, where the 45 degree diagonal line has been drawn.

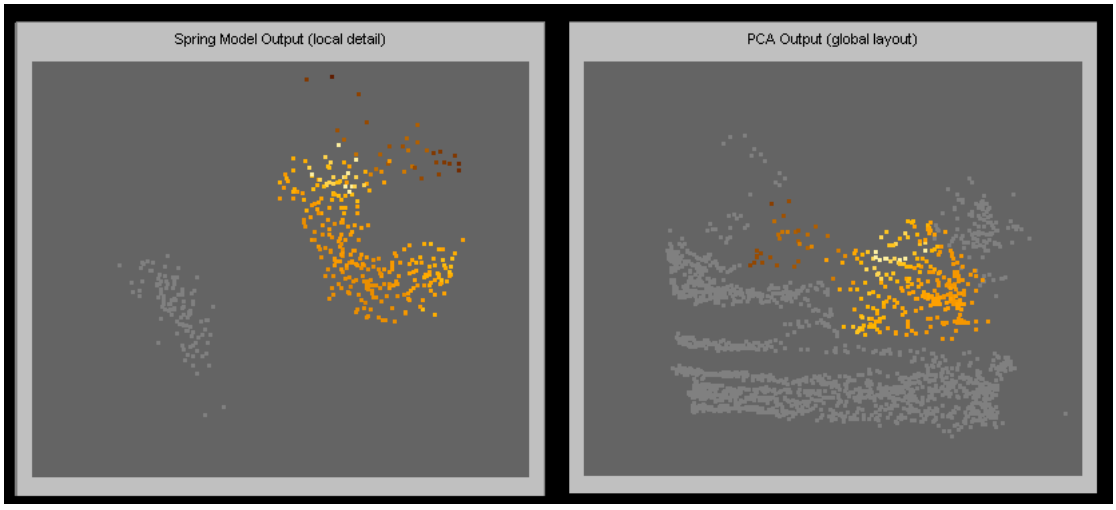


Figure 8.8 The left image shows the spring model layout of a selected Voronoi cluster within a PCA layout. On the right is the original PCA layout. Selecting the C-shaped sub-cluster on the left highlights the corresponding objects in the PCA layout, helping one understand the overlap or separation of sub-clusters in the PCA.

It may be noted that two separate clusters of points exist on the Shepard diagram of the spring model layout. Again, an initial reaction may be to assume that each of these

corresponds to one of the sub-clusters identified in the spring model layout. This is not the case, although the 'clustering' of the Shepard diagram is due to the presence of two clusters in the spring model layout. The Shepard diagram plots distances, and therefore the two apparent distinct groups of points on the Shepard diagram correspond to distinct ranges of distances within the layout. The lower of the two groups refers to pairwise distances between objects in the same sub-cluster, whereas the higher group represents inter-cluster pairs.

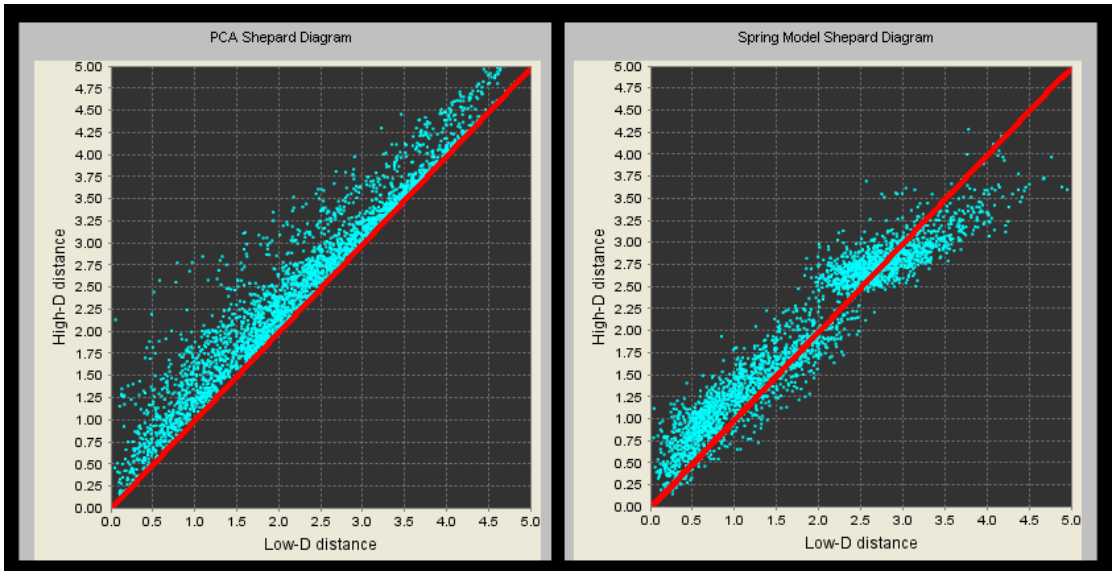


Figure 8.9 Shepard diagrams based upon the PCA layout of the full data set (left) and the spring model layout of the selected cluster (right). Red lines are drawn at 45 degrees to help detect the extent to which points deviate from this diagonal.

These examples have illustrated the advantages of HIVE's multi-view framework over solo static layouts. The provision of profiling modules provides further insight into data sets. Linking and interactive coordination between such views encourages further exploration and leads to greater understanding.

8.6 Conclusions

This chapter has presented a number of components within the HIVE system that are used to profile, understand and control other HIVE components. Their construction, and examples of their use and utility have been described. One of the advantages of bringing such components into a visualisation system is to support the process of understanding the strengths, weaknesses and interdependencies of algorithmic components. Through techniques such as recording and displaying the performance of ongoing runs, and linking layouts from complementary algorithms and from intermediate stages, it is suggested that designers and

users can explore not only the data but also the ways that the system represents, transforms and presents those data.

The work described in this chapter is based upon the premise that building visualisation applications that are tailored to one's data and interests, and which are comprised of a palette of algorithmic components, can be a complex task. However, this task may be aided by modern visualisation techniques such as those used by the profiling modules in HIVE. As data set sizes increase, so do the number of tools developed for visualising them and it can be frustrating for designers and for users if the tools for analysis and understanding data were themselves difficult to analyse and manage. Therefore the author suggests that the use of visualisation for visualisation – in the form of well-designed interaction with the components, processes and parameters of a visualisation system – may afford deeper insight into the visualised information itself.

9. Text-mining in HIVE

This chapter provides details on how HIVE can be used for mining unstructured text. Functionality for text analysis has been implemented to broaden the possible target audience in preparation for user engagement with HIVE – just about everyone deals with text in their work. Many researchers have developed ways to visualise document collections. Perhaps the most popular technique is the use of self-organising maps (SOMs) [HKLK96, HKLK97], however, with respect to HIVE, the force-directed placement approach such as that adopted by Chalmers [Cha96], Wise [Wis99] and Korfhage [Kor91] is the most relevant here. The influence of Wise's technique will be discussed later in this chapter.

To import text into HIVE, a new data-source visual module was written to apply the common text processing operations such as stop-word removal [Sal71] and stemming [Por80]. This was achieved by using the API provided by Lucene – an open source text search engine written by volunteers for the Apache Jakarta project [Jak04]. Lucene creates an inverted index to facilitate efficient searching and access to term and associated document frequencies.

One of the most important aspects of text mining is in finding an effective vector representation of the documents. The author experimented with popular techniques such as normalised term frequencies and tf-idf measures [RB99], but settled on a representation based upon the conditional probabilities of term occurrences, as will be discussed shortly. Another issue that was dealt with is the measurement of *similarity* between document vectors.

9.1 Vector representation of documents

To apply the layout algorithms in HIVE to a set of text documents, each document must be represented by a vector of numbers where the number of elements, D, is equal to the number of unique *content-bearing* terms across the collection. If there are N documents in the collection then the vectorisation produces a *D x N term-document matrix* (TDM) – columns represent documents and rows represent terms.

To help discriminate between documents, stop-words such as articles and connectives are removed and stemming is applied to normalise terms. In addition to this, the 5% of least frequent terms and the 4% of most frequent terms in the text collection are discounted. From the author's experience these values provided a set of content-bearing words that adequately discriminate documents while substantially reducing the dimensionality of the TDM.

Researchers have experimented with other ways of conducting this initial filtering of terms (feature selection). Huang et al., for example, use a visualisation called InterRing [HWR03] to cluster terms based on their correlation. Bookstein et al. [BKR98] observed that content-bearing words tend to appear in passages of text that clump together, and they used this *clumping* property to assign weights and subsequently remove terms from the analysis whose weights are below a threshold value. However, these techniques require considerable processing, and to keep HIVE as fast as possible for fluid user interaction, the simple frequency-based filtering as described above is used.

Experiments were carried out in methods for converting terms to numbers for the document vectors. Two types of quantification were used: normalised term frequency and tf-idf. In addition to this, experiments were carried out with two measures of similarity between documents: Euclidean distance and cosine similarity.

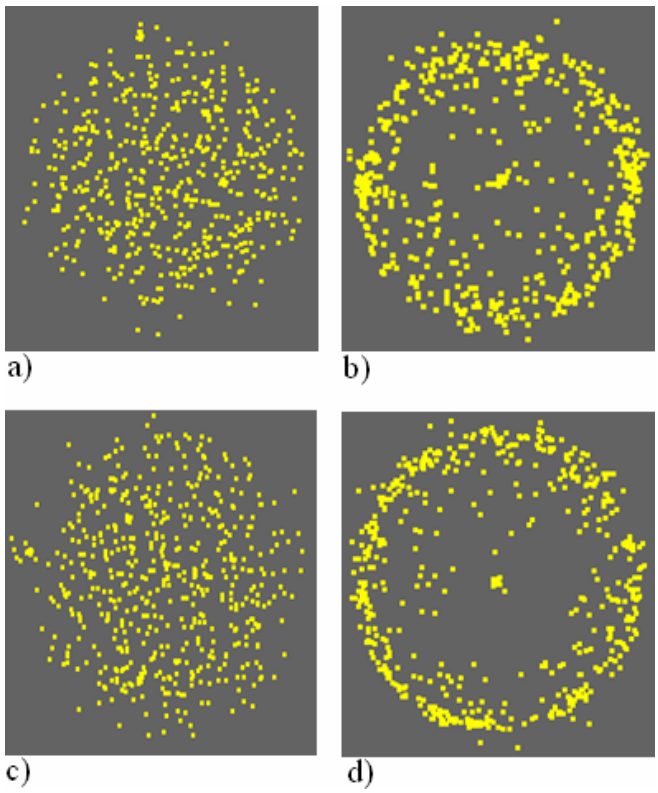


Figure 9.1 Initial experimental layouts. Cosine measure using normalised term frequency (tf) (a). Euclidean distance using tf (b). Cosine measure using tf-idf (c) and Euclidean distance using tf-idf (d).

It has been argued that Euclidean distance is not ideal for this application because even when documents do not have any terms in common, the Euclidean distance between them can be shorter than between documents that do share some common terms [ESK03]. However, the

author decided to experiment with this metric to observe this for himself. The cosine similarity measure, on the other hand, overcomes this shortcoming.

The above quantification and similarity types led to four experimental conditions: cosine measure using tf, Euclidean distance using tf, cosine measure using tf-idf and Euclidean distance using tf-idf. It should be noted that since the FDP routines in HIVE use distances in calculating layouts, cosine similarity was converted to *dissimilarity* by subtracting it from 1 and this was used to approximate distance.

Figure 9.1 illustrates screen shots of layouts produced by Chalmers' spring model when run on a collection of 538 abstracts. The abstracts were taken from the University of Glasgow's DCS bibliography, from the proceedings of InfoVis 2001 to 2003, and from other papers that cite publications about HIVE and the algorithms produced in it, as well as articles related to these. After stopping, and removing low and high frequency terms, the dimensionality of the TDM was 348.

From Figure 9.1 it is evident that the cosine measure does not show any interesting structure in the data. On the other hand, both layouts that use Euclidean distance to compare documents show structure that is highly suspicious. Upon closer inspection of the small cluster of points visible in the centre of Figure 9.1(b) and 9.1(d), it was found that these points represent documents that contained no content bearing words at all – no terms contained in these documents were used in the vectorisation. This means that when using Euclidean distance to approximate similarity, this small group of documents is much more dissimilar (and equally distant) from all other documents. This causes a high amount of repulsion of the remaining documents and explains the outer 'ring' of repelled points. This effect is not observed with the cosine measure of similarity because many other documents that do contain content-bearing words can still have zero similarity between them - e.g. when they have no terms in common. Hence the inclusion of the *empty* documents does not greatly affect how strongly the other points are repelled.

The empty documents were removed from the data set and new layouts were generated. The layouts are shown in Figure 9.2.

After removal of the empty documents, the layouts all appear very similar and there appears to be no interesting structure. To test if this was indeed the case, all documents that contain the phrase "multidimensional scaling" were highlighted to see if they clustered or appeared consistently within a particular region or configuration. The layouts of Figure 9.2 are shown with the highlighted points in Figure 9.3.

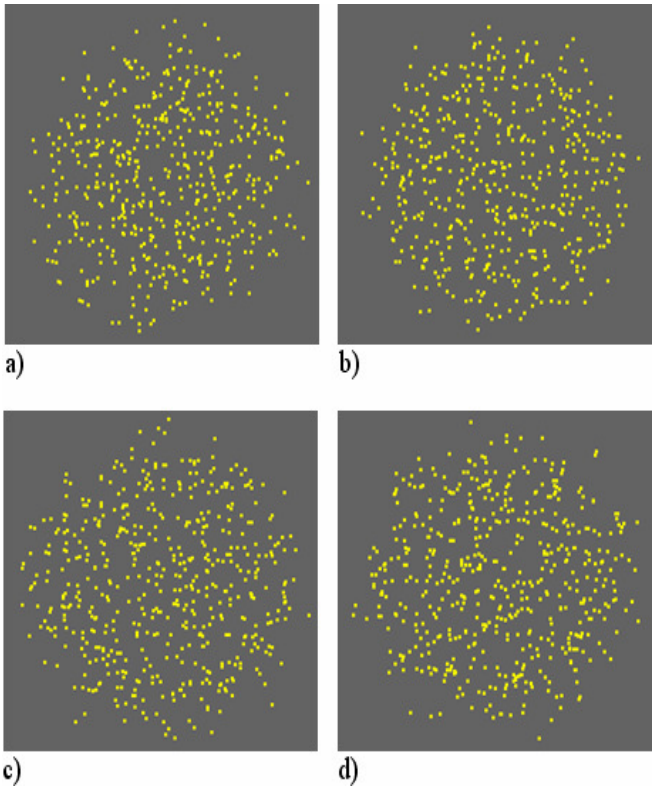


Figure 9.2 Layouts of the abstracts data set using the same measures as in Figure 9.1 but with the empty documents removed.

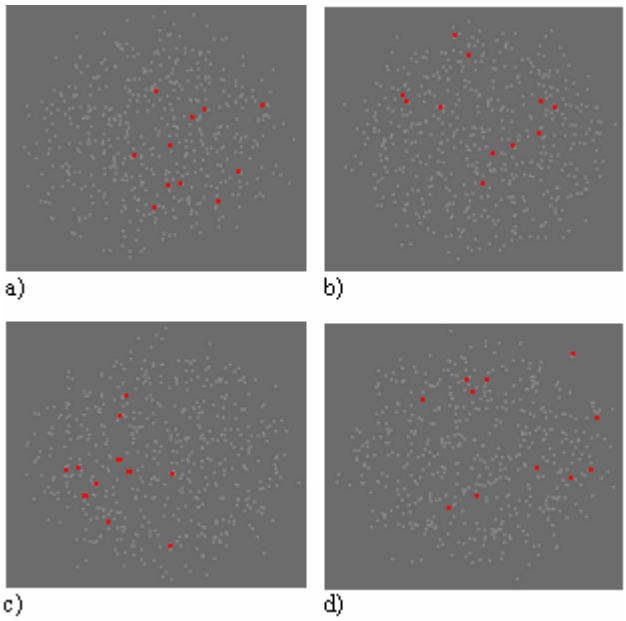


Figure 9.3 The points that are highlighted in red represent documents that contain the phrase "multidimensional scaling". No clustering seems to occur and upon reproducing the layouts, the points appear in different positions with respect to each other and the other points.

As suspected, the matching documents show hardly any tendency to cluster and upon recreating the layouts it was found that their positions varied greatly between different layouts using the same similarity measures and term-weighting.

Buja and Swayne [BS02] point out that in cases where the distribution of distances tightly clusters around a positive value, point configurations similar to the above circular layouts are produced. Buja and Swayne refer to this property as *indifferentiation*. To find out if this is indeed what is happening, 60,000 distance samples were taken under each of the experimental conditions that produced the layouts of Figure 9.2. Histograms of two of these samples are shown in Figure 9.4.

Only two of the histograms are shown here because the others are very similar. The histograms confirm that distances/dissimilarities do tightly gather around a non-zero value and therefore indifferentiation is the most probable cause of the degenerate layouts in Figure 9.2.

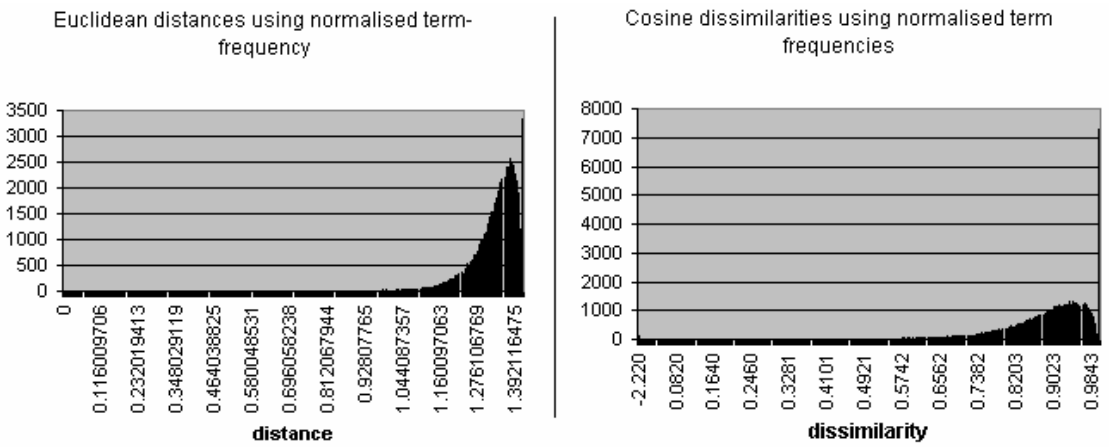


Figure 9.4 Histograms showing the distribution of sample Euclidean distances and cosine measures of dissimilarity. The histograms show that the samples do indeed cluster tightly around a high positive value.

The likely cause of this indifferentiation is the sparsity of the TDM. This results in many high values of dissimilarity between documents. Inspired by Wise's use of conditional probabilities for weighting content-bearing words [Wis99], it was realised that the zero values in a document vector of term frequencies could be replaced by the probability that they would occur given the terms that actually do occur in the document. This would result in a less sparse TDM and therefore greatly diminish the likelihood of indifferentiation. Figure 9.5 depicts the results of this approach using Bayes theorem for calculating the probabilities, and the cosine measure for dissimilarity.

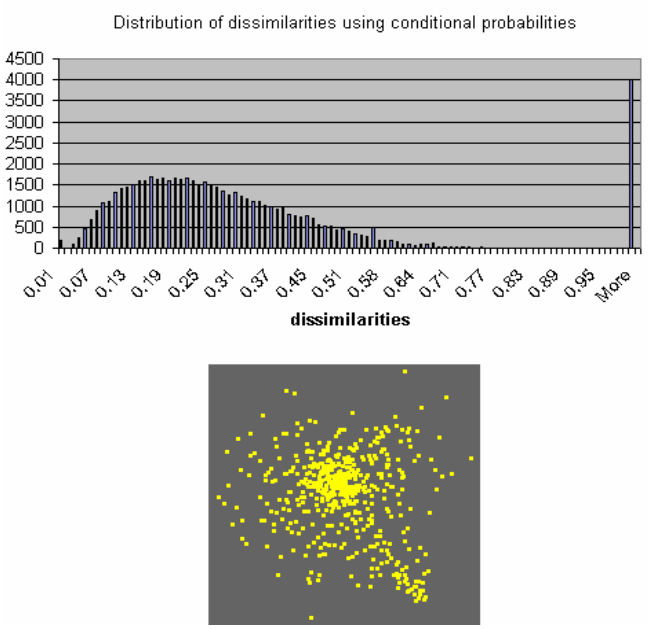


Figure 9.5 The histogram shows that the distribution of dissimilarities is now spread out more. The layout accordingly shows potentially interesting structure in the data.

While it can be seen that there is still a large proportion of distances tightly hugging the higher range of the distribution, the result of using conditional probabilities to fill in the missing terms in the TDM has resulted in a distribution with greater spread. When the conditioned TDM was fed in to a spring model, the layout produced looked much more promising (see Section 9.2 for details).

The experiments described above were also run on a collection of over 1000 issue statements gathered from various projects undertaken by the author and his colleagues at Nickleby HFE Ltd, a human factors and ergonomics consultancy. The results obtained from these experiments are consistent with those described above.

9.2 Where is particular literature within a HIVE layout?

The layout in Figure 9.5 looks promising, but in order to see if it actually means anything it is necessary to query it and view the contents of the documents that are represented. Two new visual modules were implemented in HIVE. One is to allow the user to search the layout for specific documents and one to view the contents of selected documents.

The search module uses the Lucene search engine to query the indexed set of documents. It allows free text search as well as regular and Boolean expressions. Since it is implemented as a visual module, the user can instantiate as many instances as required thus providing a flexible way to combine and store queries.

The search module has a *select* port so that when it returns the query results, the matching documents can be highlighted within the layout, or in any other view for that matter, because it conforms to the same interaction model as the other visual modules in HIVE.

There are two ways to view the text contained in documents with the text-viewer module. The first is to directly connect the text-viewer to the search module so that query results can be loaded. The other way is to connect the text-viewer to the data output port of a scatterplot containing a layout. The latter method is quite intuitive because in the course of exploring a layout, the user can use the search module to highlight documents that match a query and then select those documents, and others that are nearby, that might be relevant but not returned by the query.

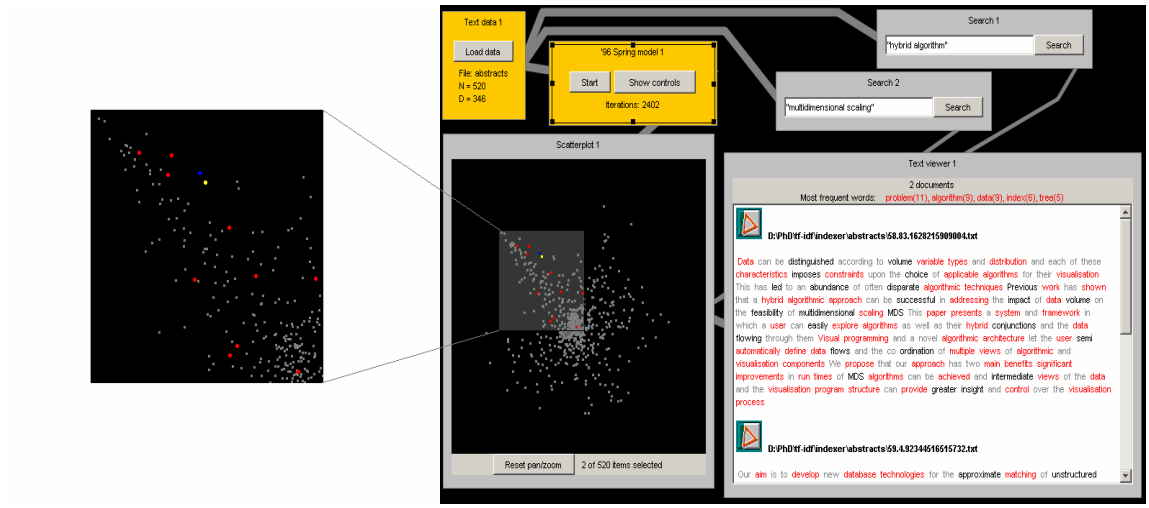


Figure 9.6 A screenshot from HIVE of a text-mining application (right) - the top-left visual module is the text collection. The text-viewer component is next to the scatterplot layout and two search modules appear in the top-right of the figure. For clarity, the layout is reproduced in the image to the left of the screenshot.

It was realised that a fitting way to test the text-mining strategy described above would be to lay out the abstracts data set (consisting of over 500 documents), then find where the HIVE and related publications are in the layout before exploring nearby documents to see if they are relevant to HIVE. The abstracts data set contains most abstracts of papers that were cited in HIVE publications as well as abstracts of articles that cite the HIVE and algorithm papers. Roughly 300 other computing science abstracts are also in the data set so as not to bias the

experiment and also to see if any other relevant papers that have not yet been read by the author appear close to the HIVE related articles.

Figure 9.6 is a screenshot from HIVE showing how a simple text-mining application can be set up in HIVE and used to query the abstracts document collection. Two query components have been connected, and the red and blue points in the scatterplot represent documents that were returned by the search on "multidimensional scaling". Red points represent unselected query results; blue points denote returned documents that the user has selected in the layout, and yellow points represent documents that are selected by the user but were not returned by the query. In this case, the author has selected one of the abstracts returned by the query (which represents a journal paper about HIVE [RC03b]) and one neighbouring point representing an abstract that was not returned by the query.

The scatterplot is connected to a text-view and therefore the text represented by selected points (blue and yellow) is visible. Words that are coloured red in this view represent content-bearing terms that are used in creating the layout. Words that are black are low or high frequency terms that were filtered out during the pre-processing stage described earlier. Functionality has also been provided to allow the user to select red and black terms, modify their weights and exclude or include them in the layout process. This can be done while the layout is forming and provides an interesting animation as groups of points move, forming and dissolving clusters as the user modifies the term values. Also, at the top of the text view, the top five most frequent terms in the selected documents are indicated along with their counts to give an idea of the subjects covered. The remaining grey terms in the text-viewer indicate stop-words that were removed in the indexing process.

The purpose of this experiment was to see if the layout does indeed indicate structure in the data and not just artefacts of the layout algorithm. By searching the layout using the "multidimensional scaling" query, the author knew that some of his papers and several others would be highlighted in the layout. It was found that the author's papers are close to each other, as would be expected, and the region of points that contains them pertains to abstracts of papers on algorithms, data and text mining, clustering and more generally information visualisation. Upon repeating the experiment, the same general configuration was generated. This provided to some extent a picture of where HIVE is in the context of the literature (see Figure 9.7).

In Section 7.5.3, the use of dimension reduction for feature selection was demonstrated. This method can also be applied to text. Figure 9.8 illustrates transposed textual data resulting in a layout of the terms in the abstracts document collection. This allows the user to select the

terms she is interested in and then layout a (sub)set of documents according to only the chosen terms.

These experiments were carried out using the cosine measure of dissimilarity. However, when Euclidean distance was used, it was found that although the topological structure of the layout was roughly the same, the salient structure that was made apparent with the cosine measure was not as pronounced using Euclidean distance. For this reason, the cosine measure is the preferred choice.

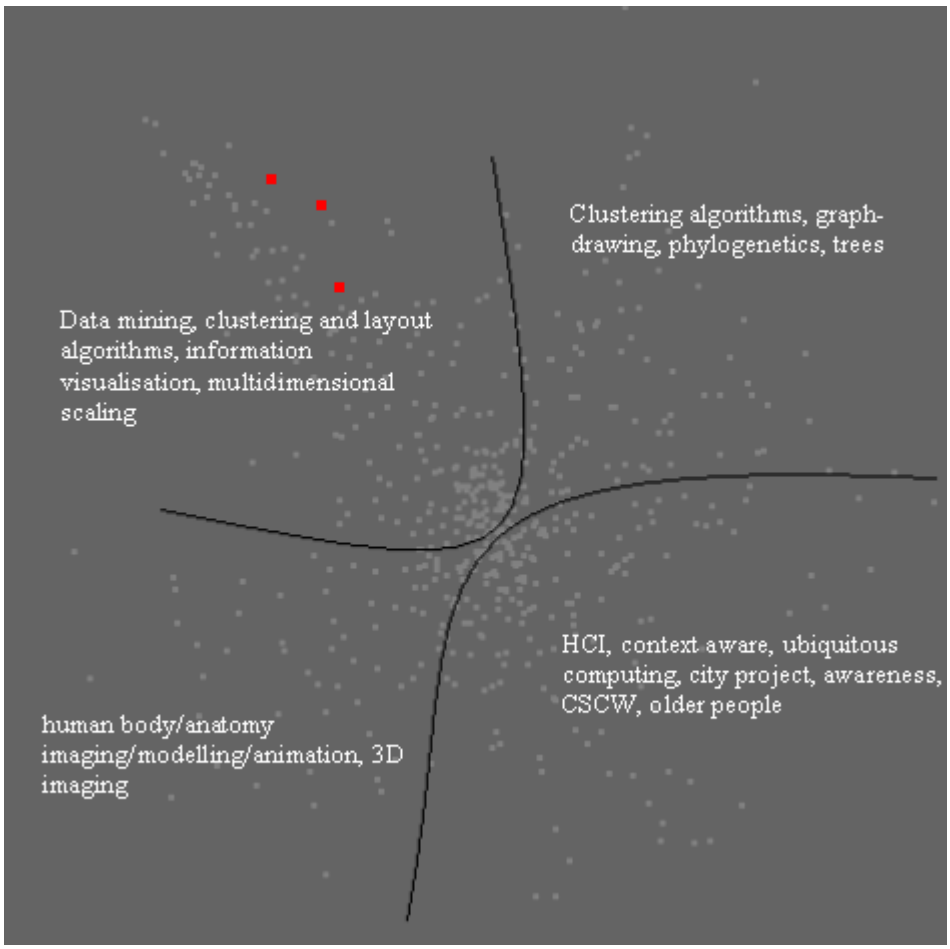


Figure 9.7 The layout from Figure 9.6 is labelled to indicate general themes running through the documents in different regions. Some HIVE papers are highlighted in the data mining and clustering region at the upper-left side of the layout.

This example uses a small document collection and therefore only a small number of possible query results can be returned; in this case eleven abstracts were returned for the query "multidimensional scaling". It would be interesting to see how this hands-on approach scales up to larger collections and to see whether second or higher order term-co-occurrences, such

as those in latent semantic indexing (LSI) [DDF*90], are being influenced by the conditional probabilities used in term-weighting. However, for now, the results from this example and other explorations appear promising. Psychologists at the University of Liverpool have recently applied HIVE to the analysis of text. The preliminary results will be described in Chapter 10.

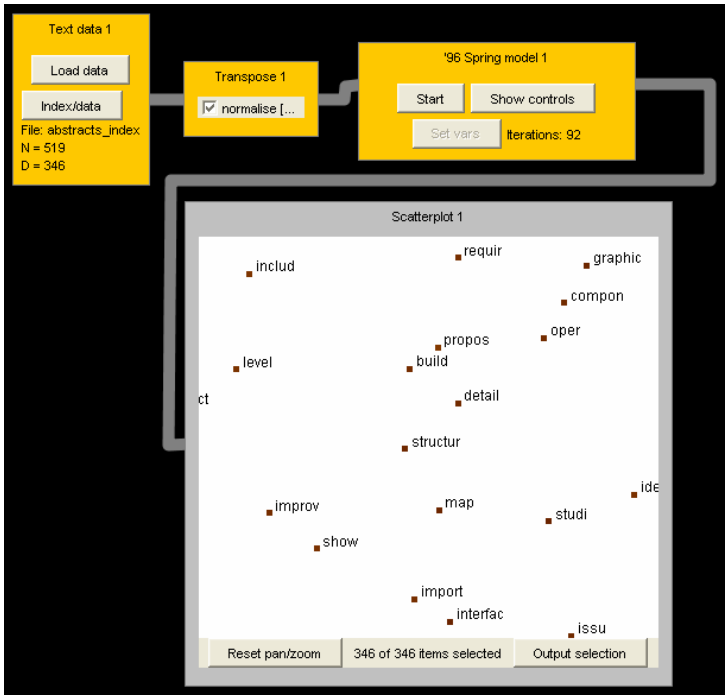


Figure 9.8 A layout of terms in HIVE. This can be used for feature selection as in Section 7.5.3.

9.3 Conclusions

This chapter has described how HIVE has been equipped with functionality to apply dimension reduction algorithms to unstructured textual data. An investigation of the effects of both Euclidean distance and cosine dissimilarity showed that there was little difference between layouts of documents represented by vectors of tf-idf values or normalised term frequencies. In fact the layouts produced did not reflect any interesting structure. It was found, however, that by representing documents by term frequencies and filling in any zero entries with conditional probabilities of the corresponding terms, that salient structure was made apparent. Exploring layouts produced by a spring model indicated that similar documents were placed close to each other and that the topology of the layouts were indeed meaningful.

It is worth mentioning that layout stress was not used to assess the solutions. This is because stress is mainly used to test the layout algorithm; in this instance, however, the author is interested in the vectorisation (or document representation) and the interpretability that it

conveys; not the layout algorithm per se. This was a more subjective challenge that could not be mediated by trying to reduce stress values alone.

HIVE has been shown to be capable of building simple and reconfigurable text-mining applications. An example was provided in which some of the author's HIVE-related publications were visualised in a wider frame of general computing science publications. While this provided promising results by the appearance of sensible and explicable structure, it also implied a novel means of profiling the HIVE framework per se. In the previous chapter, it was shown that profiling modules could be implemented in HIVE to help evaluate hybrid algorithms; it now seems that the text mining functionality could be used for a higher level of profiling – that of the overall HIVE framework with respect to its place in the literature.

10. HIVE user engagement

The author's work evolved from the investigation and development of novel hybrid clustering and layout algorithms to the development of a framework for their creation, evaluation and use. The initial target audience of this framework consisted of the author and his colleagues, however, as HIVE became more familiar through the literature and by word of mouth, the number of its users has considerably increased. HIVE is available as an open source code project under the provision of the Mozilla Public License and has been requested by researchers at several prominent institutions, both academic and commercial, from all around the world. HIVE cannot be downloaded directly from the author's website, instead it is available purely by request. In this way the author can engage in dialogue with potential users from the start and politely keep abreast of their progress.

The feedback provided by users has shown that HIVE has two main modes of use. One is to take the code, create new visual modules and therefore customise it for new tasks and work domains; several publications, written independently of the author have arisen from such work with the software [MC03, MC04, Mor04, WM04, Dar04]. The second mode of use is purely for exploring data and testing related hypotheses without creating new visual modules.

This chapter provides details of how various people have used HIVE and how its two modes of use have emerged. The evaluation of the software has been achieved by giving it to users with real tasks. Rather than being a usability study, this evaluation is based upon a demonstration of HIVE's efficacy and flexibility.

It should be noted that the study documented in this chapter is purely exploratory and therefore the validity of the results is not guaranteed. Possible impact factors on its validity include bias due to some of the users being familiar with both the author and his research – this has not been accounted for in the study; also, the questionnaire, described in the next section, was not piloted or evaluated in any other way prior to its distribution and therefore problems such as respondents misunderstanding some questions might have occurred. One might also consider the possibility that some of the users may have further distributed HIVE. Thus, the pool of users might be larger. In this case, the information obtained by observing and querying about user engagement might not be an entirely accurate account of the general view of all of those who have attempted to use HIVE.

10.1 Questionnaire

Shortly after the author published the first two publications on HIVE [RC03a, RC03b], it was made available to anyone who expressed interest. Within a few months, there were nine people actively using HIVE. Only two were internal to the Department of Computing Science at the University of Glasgow and of the remainder, two worked at commercial companies. Upon receiving the software, each of the users were provided with installation and operating instructions along with some examples of simple visualisation applications and hybrid algorithms that could be implemented.

The author desired to find out how HIVE had been used and whether it had been useful and effective. To determine this, an on-line questionnaire was carefully designed and several weeks after each user received the software, the author made contact to politely ask him or her for its completion. Fortunately all nine users participated in the questionnaire's completion.

The format of the questionnaire (shown in Appendix C) was prepared according to the guidelines provided by Sudman and Bradburn [SB82]. These guidelines include the following:

- clearly state the purpose of the questionnaire
- avoid loaded questions
- avoid appearing judgemental
- questions should be succinct and concise yet clear
- ask multiple choice (attitude) questions first
- ask long open-ended (behavioural) questions last
- keep it short

It should be noted that the questionnaire was not intended for statistical analysis – this would have required more users. Instead the author was concerned with gaining some preliminary feedback on the usefulness of the software. In total there were twenty-one questions; the first ten were on a Likert scale and required the user to choose one response from five possibilities ranging from "strongly agree" to "strongly disagree". These were followed by several "yes/no" closed questions. The last five questions were open-ended and prompted the user to enter free text.

The purpose of the first five questions was to determine to some extent whether the software was easy or difficult to use. All respondents with the exception of one agreed that it was, indeed, easy to use. The same number of respondents also agreed that it was flexible.

Four respondents agreed with the statement “it is easy to make the software do exactly what I want”. Four also agreed that HIVE’s design made it easy to modify the source code and six agreed that the software was satisfying to use.

The next five questions were aimed at gaining some feedback regarding HIVE’s data-flow model and its use of visual programming for interactively building applications. Five respondents agreed that the semi-automatic creation of algorithms was useful while the remaining four neither agreed nor disagreed. Four agreed that HIVE’s generation of a hybrid algorithm was not confusing while, again, the remainder neither agreed nor disagreed. Regarding the linking of visual modules for data- and interaction-flows, seven agreed that they were useful and intuitive. For the statement “the visual construction of algorithms is advantageous”, six respondents agreed and only one disagreed. All of the respondents, with the exception of one stated they had more than five years programming experience, however, only five modified the source code to implement new visual modules or for further customisation. All respondents stated that they would use HIVE again.

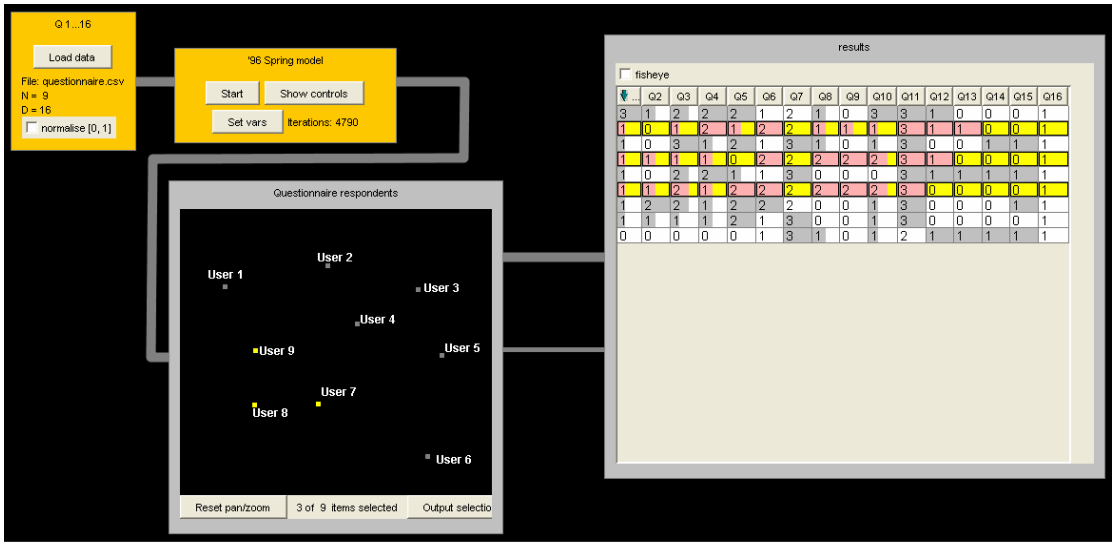


Figure 10.1 The first 16 questions coded and laid out by a spring model in HIVE. The responses of the selected users in the scatterplot are highlighted in the table. The stress in this configuration was measured as 0.087 indicating a good fit.

These results are heartening and, even though they are not overwhelmingly positive, they do imply that the visual programming adopted for data- and interaction-flow provides a flexible and intuitive method of building algorithms and visualisation applications. Out of curiosity and to further analyse the results, the author fed them into a spring model. Since the first 16 questions are multiple choice, they can be simply coded according to their response. For example, the scale from “strongly agree” to “strongly disagree” can be coded from 0 to 4.

This allowed the results to be represented by a 16-d data set. Figure 10.1 shows a screen shot of the results in HIVE.

It is possible to investigate the similarities between users' responses by selecting points in the scatterplot to highlight rows representing the responses in the table. This showed that users 2, 3, 4 and 9, who did not modify the source code, are adjacent in the plot. These users also answered the majority of the dichotomous questions with a 'no'. The remaining users were slightly more dispersed. From multiple runs, users 1 and 6 were found to consistently reside at opposite ends of the plot. Upon closer inspection it appeared that this difference was due to user 6 strongly agreeing that the software was easy to use and responding positively to more of the earlier questions.

The final five questions were open, providing space for the respondents to write answers in their own words. The first of these asked what aspects of the software did the users dislike. One respondent stated that the data-flows can *clutter the screen* and suggested that some form of aggregation might be useful so that a hybrid algorithm could be wrapped into a single derived visual module that could be opened up for inspection. For the same question, three respondents stated that there could have been more documentation provided on the default visual modules and their parameters; another wrote that the method of entering link-mode (for hooking visual modules together) was not obvious. One respondent commented on the *limited data types* that HIVE could handle and that it would also be better if the visual programming interface could be supplemented by a scripting language.

The author has since addressed several of these points. More documentation on the use of visual modules and examples of their use has been drawn up. It is now also possible to invoke link-mode via a view menu that incorporates several other useful functions such as tiling visualisation modules. Also, the diversity of data types that HIVE can take as input has been extended; unstructured text and lower-triangular matrices of proximity data can be loaded into data source modules for analysis. Owing to a lack of time, module aggregation has not been implemented yet.

The next open question asked what particular aspects of HIVE the users liked. The *ability to alter parameters interactively and perform comparisons at run time* was noted by one respondent. Another respondent said that the visual construction of algorithms was *interesting and helpful*, and another described this as *easy*. The *ability to handle large data sets* and *integration of many different data analysis tools* was another response. These answers imply that HIVE's design and its intended use are indeed valuable to others.

The third open question enquired into the use of HIVE. One respondent used it for the design and evaluation of hybrid algorithms for force-directed placement. Similarly, another

respondent used it to investigate how incremental (hybrid) MDS algorithms worked. Two others used it (independently) for the analysis of genetic data. HIVE was also used by one respondent for the creation of a new method of intervening in the process of FDP, while another user simply wanted to extract some of the hybrid algorithms for his own software. These answers indicated that HIVE has a dual role, as anticipated by the author: to analyse data and to create the processes for such analyses.

The next question asked what other software the respondents might use for the same tasks. It was felt that this might indicate target systems for comparison with HIVE for future evaluation. Two respondents left the field blank and one stated *none*. Another said that he would have had to write something similar *from scratch*. Others mentioned research systems such as FSMvis [MRC02] and xGobi [BCS96].

The final question simply asked for any other comments and the answers mainly reiterated what was already stated. Although HIVE had only nine users – fortunately all of whom answered the author's questions – their feedback has provided some valuable insight into how HIVE was adopted and the author has addressed some of this feedback to improve the system. On the whole, the results appear promising.

10.2 Examples of HIVE's use

This section will briefly describe how HIVE has been utilised by others. The first subsection describes how one user employed HIVE to develop and evaluate the fastest FDP algorithm to date [MC04]. The second subsection provides details on how the software was used by another researcher as a development platform for a tool called MDSteer [WM04]. HIVE was also used to develop a bioinformatics application for describing *chains* of data analysis components. This will be the focus of the third subsection. Finally, the fourth subsection describes work with HIVE by psychologists at the University of Liverpool.

10.2.1 Development and evaluation of a new FDP algorithm

Recall the novel hybrid spring model algorithm from section 5.3. At the time of its publication, this was the fastest force-directed placement routine that could produce layouts of quality comparable to that of Chalmers' algorithm. To recap, the algorithm works by first randomly selecting a sample of $\sqrt{N}$ items from the input data set consisting of N items. Chalmers' spring model is then applied to produce a layout of the sample, before a novel

interpolation technique is used to place the remaining items in the plot. The interpolation stage proved to be the bottleneck of the algorithm because each of the remaining non-sample items is placed according to its nearest neighbour (or parent) in the high-d space. This parent-finding is accomplished via a brute force search and therefore the time complexity of this stage is $O(N\sqrt{N})$. Since the other stages of the hybrid algorithm are executed in linear time with respect to N, the interpolation stage dominates the complexity of the algorithm as a whole.

Recently, Morrison [MC04] addressed this by reworking the algorithm in HIVE. Concentrating on parent-finding in the interpolation stage of the algorithm, Morrison successfully reduced the time complexity to $O(N^{5/4})$ and demonstrated the improvement over its predecessor by evaluating it with a data set consisting of 108,000 14-dimensional items.

The novel parent-finding strategy, implemented as a new visual module in HIVE, works by randomly selecting a subset of items from the initial sample and discretising the distances of points to members of the subset. This subset (of a subset) contains very few items – typically three – which are referred to as *pivots*. For each of these pivots, a set of *buckets* representing a range of distances is defined and each of the remaining items in the initial $\sqrt{N}$ sample is allocated to a bucket for every pivot. To find an interpolation parent in the $\sqrt{N}$ sample for an item, the distance from the item to each pivot is calculated, indicating the buckets to which the item belongs. The parent is then found by searching through all sample items contained in these buckets and selecting the closest.

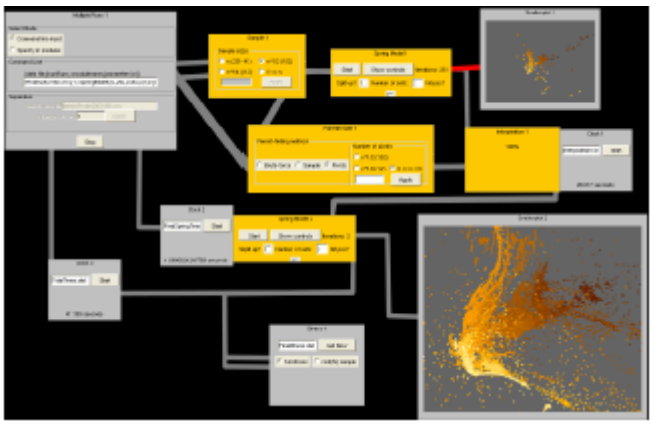


Figure 10.2 Morrison's hybrid algorithm in HIVE. The large component at the top-left is the multiple runs module. Morrison used this in conjunction with clock and stress modules to evaluate the algorithms.

In keeping the number of pivots constant and by defining $N^{1/4}$ buckets for each pivot, the time complexity of the interpolation stage, and therefore the whole hybrid algorithm is $O(N^{5/4})$. Morrison provided experimental results on the performance of the new algorithm by

using the profiling modules in HIVE as described in Chapter 8. The results reflected the reduction in time complexity and average layout stress comparable to the previous hybrid algorithm. Figure 10.2 shows a screenshot of Morrison's configuration of visual modules in HIVE for the implementation and evaluation of his algorithm [MC04].

Since completing this work, Morrison continued to use HIVE. For the successful completion of his PhD, he developed numerous novel visual modules and several experimental hybrid FDP algorithms, all of which were profiled in HIVE. As a final analysis of these algorithms, his thesis [Mor04] demonstrates how a dimension reduction algorithm is used to produce a layout of the characteristics of the novel designs.

Now working as a research associate in the department of Computing Science at the University of Glasgow, Morrison continues to develop visualisation solutions via HIVE. So far, he has implemented HIVE modules for time series analysis based upon mutual information, a parallel coordinates visual module and a matrix histogram module.

10.2.2 Steerable dimension reduction

In 2004 Williams and Munzner developed a means of intervening in the layout process of Chalmers' spring model. This tool, called MDSteer [WM04] was built in HIVE and allows the user to direct the computation of the layout algorithm to user-selected areas of interest in the layout. This approach is highly visual and interactive. It starts by progressively laying out $\sqrt{N}$ points where N is the number of items in a data set. Once this initial layout is obtained, the viewing area is subdivided into two regions. The user can then select one of these regions to specify that the algorithm concentrates only on this area of the layout. After a subset of points that should lie in this area have been placed, the two regions (called *bins*) are further subdivided and all points in the data set are allocated to the appropriate bin. This process is repeated until all items in the data set have been laid out. The hierarchy of bins is shown on the layout as wire frame boxes subtending progressively smaller regions. Figure 10.3 illustrates MDSteer in HIVE. The figure shows layouts of an environmental data set consisting of 40,000 294-dimensional items. Each dimension pertains to a measurement such as water and air quality. The image on the right shows MDSteer in action where each box on the layout can be selected, effectively stating the user's intention that the layout algorithm fills out that region.

Williams and Munzner argue that this approach can be used to quickly obtain overviews of very large data sets and also drill down into areas of potential interest. In fact, the authors used the tool to obtain partial layouts of up to one million items. Rather than only selecting a sample for laying out, Williams' and Munzner's approach allows the user to

determine potentially interesting regions of layouts and therefore allocate more computation to them. This method supplements the incremental layout process that is typical of FDP routines with user-interaction so that exploration is enhanced.

The authors stated that HIVE was used because it was one of the only tools available that could handle very large data sets. It also allowed them to experiment with the dimension reduction algorithms for producing the incremental layouts used with the MDSteer technique.

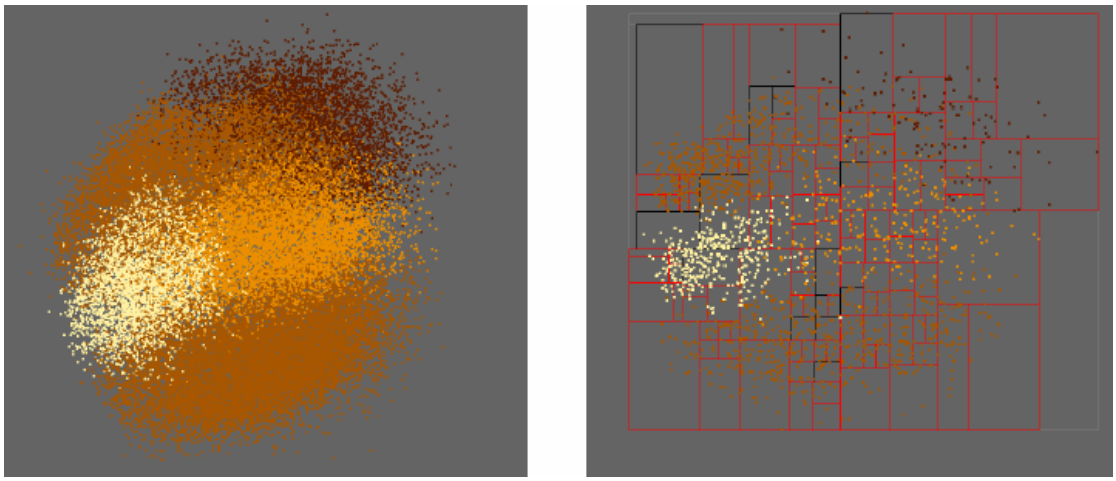


Figure 10.3 The image on the left shows a layout obtained by Williams and Munzner using the novel layout algorithm of Section 5.3. The image on the right shows an overview of the same data set using MDSteer. Black boxes represent bins that have no unplaced points while red boxes represent bins in which there are points still to be placed.

10.2.3 A bioinformatics chain description tool

In the analysis of data, scientists often use several programs, taking the output of one as the input to another and so on. The use of programs in this manually driven pipeline is akin to the hybrid clustering and layout algorithms for which HIVE has been designed. For his PhD, Darroch [Dar04] re-branded HIVE as a Chain Description Tool (CDT) for use in the domain of bioinformatics. The term “chain” refers to the serial connection of computational components to form an application. Darroch defined a chain description language in XML whereby chains of programs for the incremental analysis of biological data can be specified. However, rather than having users resort to the cumbersome process of typing scripts in this language, Darroch embodied the language in HIVE so that visual modules represent programs and their connections specify the chain, effectively translating HIVE module configurations into his chain description language.

In Darroch's version of HIVE, he implemented visual modules for tasks such as searching for specific proteins on the internet and taking the results from such searches and predicting the secondary structure of the proteins. It is often the case the outputs of different bioinformatics programs are in a format that is not compatible with the required input of other programs. To address this, Darroch also implemented visual *translation* modules. The implementation of HIVE in this domain is described in Chapter 5 of Darroch's PhD thesis [Dar04].

By using HIVE, multiple analysis runs can be automated – a process akin to batch runs of hybrid algorithms as demonstrated in Chapter 8. HIVE also provides a uniform environment in which the analyses can be carried out recorded and repeated at later dates. Figure 10.4 shows one of Darroch's chains consisting of a series of visual modules. Darroch added some new interesting functionality to his modules. The "Annotation" button visible at the bottom of each new module allows the user to see (or enter) a description of the module and how it could be used. Ports have also been embellished by adding a drop-down list that the user can query to see what other visual modules would be compatible with the particular data type on that port. By selecting one of the list entries, the corresponding module is loaded and automatically linked to the selected port.

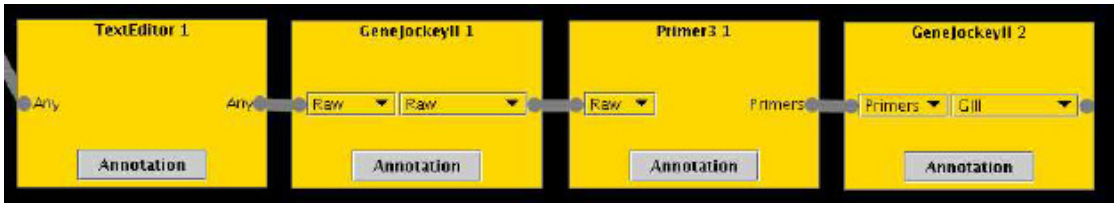


Figure 10.4 A chain of visual modules representing bioinformatics applications in Darroch's Chain Description Tool – a customised version of HIVE.

Darroch carried out an evaluation of his customised version of HIVE with biologists and concluded that the application chains were capable of capturing real-world analyses. The visual programming of data-flows for this domain appear to be intuitive and useful.

10.2.4 HIVE for psychological profiling

Researchers in the Centre for Investigative Psychology (CIP) at the University of Liverpool have been using dimension reduction in behavioural analysis for decades. Dimensions considered in their analysis range from background characteristics of people such as education and occupation, to action variables (or modus operandi) observed at crime scenes. Recall from Sections 4.3.3 and 7.5.3 that multidimensional scaling can be used to aid feature selection.

This is indeed one of the main roles that dimension reduction plays at CIP. In obtaining layouts of relationships among variables, researchers can theorise about the facets of human behaviour that characterise specific types of crime. When these facets (sets of salient variables) have been identified more traditional MDS can be applied to gain layouts of the population of crimes or offenders. Essentially dimension reduction is used to form and test hypotheses in psychological profiling.

SSA (Section 4.3.3) is the most popular type of dimension reduction routine used by CIP. While it has been shown that it has been very effective in its application [CF98], it exhibits a high time complexity and is therefore limited in the amount of data to which it can be applied. Furthermore, CIP's SSA routine is implemented in a tool developed several years ago and while being a tremendous improvement over the old command line FORTRAN implementations that some of the senior psychologists are accustomed to, it still lacks flexibility and the richness of interaction that modern visualisation techniques can afford.

HIVE was brought to the attention of CIP researchers by word of mouth and in late 2004 the author was invited to have dinner with the head of CIP, Professor David Canter, while he was on a business trip in Glasgow. The discussion that followed sparked a mutual interest in the application of HIVE in CIP's work and since then the author has been working closely with CIP researchers, aiding them in their enthusiastic adoption of the software. This collaboration has also resulted in the authors' current employment with the Kelvin Institute, implementing dimension reduction tools in a project funded by the Metropolitan Police Service for an Interactive Offender Profiling System (IOPS).

The author has visited CIP several times now, and the discussions and observations that ensued have provided a wealth of feedback on how the researchers have been using HIVE and where useful improvements could be made. Suggested improvements included constraining the 1:1 aspect ratio of scatterplots so as not to distort the relative inter-point distances; make scatterplot points larger; allow users to define scatterplot background colour; provide surface plots based on binary variable frequencies; provide for user-defined dictionaries to be used in text analysis; be able to import proximity data in the form of a lower triangle matrix. The latest modification suggested by CIP was to be able to freeze an application after it had been visually programmed in HIVE – in effect disallowing other users from modifying the structure of visual modules and data-flows, although still being able to specify the input data.

In addressing the suggested improvements to the software, the development of HIVE entered an iterative design cycle. Each time, after modifying the software, an informal meeting was arranged to provide a demonstration and prompt further discussion. Current users of HIVE were also given updated versions and, after several of these development iterations,

the author carried out informal user testing of HIVE to gain more feedback on its long term use. Two people at CIP who had been using HIVE over several months for quite different work agreed to take part in the study. Since the author did not intend to collect statistical data and was more concerned with gaining feedback from real users, this low number of participants was acceptable.

Rather than inventing tasks for the users to carry out, the author instead asked them to perform some *real* tasks for which they currently use HIVE. This allowed the author to see how deeply the users delved into HIVE and to note which parts were more useful. The evaluation was based upon the think aloud protocol – users were asked to comment on their actions and describe any difficulties encountered as they used the software. The users' interactions with HIVE and their voices were recorded for future reference by screen capture software.

A number of observations were gained from this evaluation. The first user is a psycholinguist and used the text analysis capabilities of HIVE to analyse a set of suicide notes. In this task the user wanted to find out about differences in writings between male and female authors. The user first hooked together HIVE modules to gain a layout (scatterplot) of the notes. In this layout each point represents a note and their mutual proximities should reflect the similarity between them according to the terms used. The user then proceeded to annotate each point in the layout with the names of the notes stating that it would be useful if HIVE would do this automatically. Upon completing this, the user observed that the notes were grouped in contiguous regions of the layout according to whether they were written by a male or female. This observation was aided by linking a text viewer module to the layout so that when layout points were selected, the text of the corresponding notes was displayed. Noting that this did indeed suggest differences in writing style, the user then transposed the data and obtained a second layout, this time depicting the words used in the notes. The purpose of this was to identify words that contribute most to the differentiation of the notes. In this view, words that co-occur frequently would be closer together in the layout than those that rarely co-occur. At this point the user stated that it would be useful if the term frequencies were displayed next to points on the layout. The main difficulty the user encountered with HIVE was with panning and zooming in the layouts. This is achieved via the middle mouse button for zooming and the right button for panning. It appears that the view updates were too sensitive to the user's actions resulting in the user panning or zooming too much and subsequently losing orientation.

The second user embarked on a different task with HIVE. In this case the user wanted to analyse data on a set of burglaries. In this data each datum represented a burglary and the

variables consisted of the actions observed from the corresponding crime scene. For example one variable was whether a window was broken to gain entry, another regards whether an untidy search was carried out. The data are therefore binary. In contrast with the first user who initially obtained a layout of the population (where each point represents a datum), the second user first transposed the data to obtain a layout of the variables. The second user stated that she wanted to look for variables of high and low frequencies and see where they were in relation to each other. She used SSA for dimension reduction and then applied a frequency surface to the layout so that darker areas would correspond to low frequency variables. In this layout the higher the correlation between a pair of variables then the closer they are in the view. While the user visually programmed this application the author observed one difficulty. It is easy to miss the ports when dragging data-flows between modules, resulting in either a module being dragged or the link being lost. A possible solution to this would be to increase the size of the ports.

After exploring the layout of variables, the user then decided to produce a layout of the population of burglaries. The user stated that she wanted to be able to select variables and then see in which burglaries they occurred. Conversely the user also wished to be able to select some burglaries and observe the actions that occurred. This was achieved by inserting a *selection* link between the two layouts to accommodate view coordination.

Both users exhibited familiarity with HIVE by quickly selecting and linking the modules they needed without much difficulty. Only the second user employed the algorithmic profiling capabilities of HIVE. At one point she viewed the co-efficient of alienation and stress to assess the output of an SSA routine. It was also seen that the scatterplot and text views were the only visualisation components they employed, however, when more than one view was present at any time the users were quick to coordinate them and explore their data from another perspective. An interesting observation was that both users obtained a layout of the variables in the data as well as a layout of the population. Upon linking these two types of view, the users began exploring one view, selecting points and then going to the second view to explore what was subsequently highlighted before making further selections and referring back to the first view. This process continued in a cycle and appears to reflect the users' hypothesis generation, testing and refinement as they explored their data.

It should be mentioned that neither user implemented any hybrid algorithms. They simply used individual modules for carrying out MDS. One explanation for this is that these users have limited knowledge about how the algorithms actually work. They are primarily concerned with the interpretation of the output and accordingly treat the algorithms as *black boxes*. Also, the data sets studied were not large and therefore the running time of the solo

algorithms was not high, thus evading the need for more efficient hybrid solutions. In general, the behavioural data sets analysed at CIP tend to be reasonably small – usually between 10 and 10,000 items. If CIP researchers begin to look at larger sets, then the use of hybrid algorithms would be necessary to improve performance. That aside, the users did link together several modules and views in exactly the same way that hybrid algorithms are implemented in HIVE. This indicates that hybrid algorithms were not being avoided due to any problems with visual programming.

In general the feedback from CIP has been excellent and those who are using HIVE continue to express its effectiveness and potential for their work.

10.3 Conclusions

It was the author's intention to gain feedback on how users have adopted HIVE; to determine how they used it and what they used it for and also to see how it could be improved. It has been shown that this has been achieved by having users fill in a questionnaire and to observe their work with HIVE through informal meetings, discussions and user testing.

There were a number of distinctions between the way people used HIVE, for example Williams' and Darroch's use of HIVE (Section 10.2) is more code-intensive than that of Morrison. While Williams and Darroch modified more of HIVE's core source code to build more specialised visualisation systems, Morrison mainly created new visual modules. Interestingly Darroch's use of HIVE for visual programming was at a higher level of abstraction – rather than have visual modules represent algorithmic components, they represent entire applications such as web services. It was also found that HIVE had two modes of use: one taken typically by computer science researchers was to extend HIVE by building new visual modules. The other mode was taken by the researchers at CIP who used HIVE solely for exploring their data. This is understandable since these researchers had no traditional programming expertise.

The range of feedback generated has been positive, indicating that the software is useful for both the creation of algorithms for visualisation and for exploring data that are transformed by such algorithms. The third research question in Section 1.3.2 asks: *As well as facilitating the creation and evaluation of hybrid algorithms, can the system be effective in allowing the exploration of the data they transform?* The feedback and observations of HIVE's use suggest that the answer is yes. Some of the users who are computer scientists have successfully developed and published new visualisation techniques and algorithms built with HIVE while some psychologists have found value in using HIVE for exploring their data. The fourth

research question asks: *Is visualisation good for creating new visualisations?* Observations of the CIP researchers performing real tasks showed that they had quickly become accustomed to visual programming of data-flows and view coordination. This method of building and using an application is a visualisation itself and it appears to be intuitive enough for users to pick it up with little difficulty. This would imply that the answer to the fourth research question, in this case is yes.

11. Conclusions

This chapter will summarise the work described in this thesis. From the development of hybrid algorithms to the provision of HIVE, several design implications have been realised and are discussed. The work documented in this thesis also prompts the author to suggest several directions for future effort, especially for potential future work with psychologists at the University of Liverpool. Prospective general improvements to the HIVE framework will also be described, some of which have been prompted by feedback from its users.

11.1 Summary

The masses of data that are prevalent across diverse domains presents a challenge for information visualisation. As more data are continuously gathered and stored, the task of transforming them into information and affording their exploration calls for continuous progress in the field. Since the human visual system has the broadest bandwidth for conveying information, graphical representations of data must be found to convey pertinent information as quickly as possible. When data are given a graphical representation on a spatial substrate, potentially interesting patterns might become apparent. However, abstract data – those data that do not have any physical derivation – present the biggest challenge because they do not easily lend themselves to the spatial mappings necessary for their rendering. The challenge is further heightened when such data are numerous and multidimensional.

One of the most scalable graphical techniques for presenting data is the scatterplot. If data are represented by points on a 2-d scatterplot, then two of the data dimensions can be mapped onto the plot's axes and another one or two dimensions may be mapped to retinal variables such as the colour and size of the points. A scatterplot can depict many points simultaneously and can often pronounce clusters and trends. However, as the dimensionality of the data increases, it becomes increasingly difficult to map them to spatial structures and retinal variables. Similarly as the cardinality of the data rises, the plot becomes more cluttered. Although a scatterplot can potentially represent many thousands of items, their distribution and ultimately their frequency can cause occlusion and elide detail. To address these limitations, not just with respect to scatterplots but to all graph types, complex data must be simplified. By *simplifying*, the author means to reduce the cardinality and dimensionality of data in such a way that as much as possible of the original information is maintained.

Cardinality reduction can be obtained by techniques including sampling and clustering. On the other hand, dimension reduction can be achieved by algorithms such as PCA and force-directed placement. These orthogonal reductions can make data easier to represent visually and therefore clarify information.

There has been a great amount of work by researchers in the development of clustering and dimension reduction algorithms, some of which has been surveyed in Chapters 3 and 4 of this thesis. However, it has been shown that in being tailored to different types of data and focussing on specific challenges therein, individual algorithms exhibit different strengths and weaknesses in various situations. For example, some dimension reduction algorithms, generally the non-linear techniques, might produce good layouts of data but take too long to run on sets of high cardinality. Modern information visualisation techniques afford users dynamic and rapid interaction with the views of their data. Hence faster algorithms are required. On the other hand, linear projection techniques tend to provide fast solutions at the expense of poorer layouts. This is a frustrating drawback because it is commonly the case that it is harder to find interesting patterns as data sets grow in size, while the applicability of the non-linear algorithms that have more potential in finding such structure diminishes because of their time complexity. In addressing these observations, the author has investigated the diligent combination of individual clustering and dimension reduction algorithms. The reason was to balance their strengths and weaknesses, producing efficient hybrids that provide quality solutions. Chapter 5 provides an account of the author's work in this area and the results show that this hybrid approach works.

The first two research questions posed at the beginning of this thesis regard decisions about which algorithmic components should be combined and when they should be used. In search of an answer, the author developed a framework for the combinatorial hybrid approach to algorithm development. This framework is discussed in Chapter 7. It is mainly based upon a classification of data complexity according to their cardinality and dimensionality. When data are transformed by consecutive stages of a visualisation application, their representation changes and therefore so do their complexity. By matching the complexity of algorithmic components to the complexity of data as they are transformed, the appropriate algorithmic stages can be applied to increase efficiency without being detrimental to the final output.

This combinatorial hybrid approach to algorithmic development has been embodied in the HIVE system. The author's intention in HIVE was to provide a way of easily prototyping and experimenting with hybrid algorithms and be able to use them to explore data. Before designing HIVE, the author investigated contemporary visualisation environments and picked out paradigms such as the data-flow model, visual programming, multiple coordinated views,

as well as the concept of the information workspace. This review, which served as a requirements gathering phase for HIVE is given in Chapter 6, and its outcome – the development of the software – is described in Chapter 7.

HIVE has become advanced, allowing richer interaction with the hybrid algorithms created in it and with the data they transform. Since it is the intention of the author to provide a framework for creating hybrid algorithms and also use them to explore multidimensional data, the tools developed facilitate both of these activities. For example, the intermediate stages of hybrid clustering and dimension reduction algorithms provide multiple views of data, while the profiling modules described in Chapter 8 let the user monitor and visualise algorithmic performance. The discussion of the feedback from HIVE's users in Chapter 10 indicates that the system has been a success. People have not only being using HIVE to explore their data – such as by using the text-mining capabilities discussed in Chapter 9 – they have also been using it to create novel algorithms.

11.2 Ongoing work with CIP

The author's current employment at the Kelvin Institute in Glasgow is involved in the application of MDS routines to the profiling of criminal behaviour. As discussed in the previous chapter, researchers in the Centre of Investigative Psychology (CIP) at the University of Liverpool are involved in this work and have been using HIVE. From numerous meetings and interviews, several possible modifications have been suggested. Since the feedback given by CIP has already been discussed in Section 10.2.4, a summary of the resulting future work is given below.

constrain scatterplot aspect ratio: In HIVE all visual modules can be resized both horizontally and vertically. When scatterplots are resized the relative inter-point distances are modified due to changes in the horizontal and vertical scaling. Professor David Canter stated that users might be tempted to resize scatterplots so that they are rectangular and therefore distort the layout to some extent.

user-defined dictionaries for text analysis: While the weighting and filtering of terms in unstructured text (as discussed in Chapter 9) provides good results for the analysis of text, it would be interesting to allow users to provide a set of terms that take precedence in the analysis.

Freeze module configuration and simplify interface: It would be useful to allow one to freeze an application in HIVE, i.e. fix the positions of visual modules so that other users, such as students cannot modify the structure of the application. At the same time, interface mechanisms such as the module toolbar and any inappropriate menu items could be removed to simplify the interface, in effect tailoring HIVE to a particular application.

The head of CIP, Professor David Canter, hopes that HIVE can embody these modifications as part of a new research project centred on HIVE. Plans for this project and its funding are currently underway.

11.3 Aggregation of flow networks

One of the disadvantages of the data-flow model employed in HIVE is that there can be quite a few visual modules required for a reasonably complex application or algorithm. This can result in a cluttered view. One way around this challenge is to wrap several connected modules into a single derived module. This would allow the organisation of data-flow networks into cohesive units, facilitating greater understanding of the application and freeing up screen space. Another advantage of this would be to save effort – aggregate modules could be saved and incorporated into future applications circumventing the need to *re-wire* commonly used sets of modules.

Module aggregation could also have an impact on the combinatorial hybrid approach for algorithms as discussed in Section 7.2. Recall that HIVE can assist the user by automatically loading a hybrid algorithm when given the user's choice of input data. It was shown that this algorithmic path through a space of data representation states can lead to the desired view of the data. In Figure 7.2 the blue arrows represent a single visual module in changing the representative state of the data in terms of dimensionality and cardinality. With module aggregation, a single visual module could represent a whole algorithm. If a single module represents an entire hybrid algorithm then the algorithmic 'cookbook' could be extended to include hybrid-hybrid algorithms.

11.4 Automatic routing of sub-layouts

Recall from Section 8.5.3 that a Shepard plot can be used in HIVE to interactively detect areas of a layout that could benefit from further processing. For example, consider a tetrahedron;

this is the 3-d simplex and cannot be laid out in two dimensions without distorting some of the inter-vertex distances. The distorted distances would be represented by points in the Shepard plot that deviate most from the 45 degree diagonal. This notion generalises to real data of dimensionality greater than two. Since it is possible to detect the inter-object distances that are most inaccurate and extract the appropriate points for subsequent dimension reduction without the influence of the remainder of the data set, one can obtain a more accurate depiction of a local region of the data. The author demonstrated this, using HIVE, in a recent paper [RMC04]. It would be interesting, however, if there was some automatic mechanism whereby such areas could be extracted and fed into other processes. Figure 11.1 shows how this might look.

The analysis could begin by clustering a layout using the Voronoi clustering algorithm described in Section 5.5. A *cluster-picker* module could then be used to extract the clusters and pass them to other processes such as spring models for independent analysis without the influence of the remainder of the data. The configuration of modules shown in Figure 11.1 has been implemented in HIVE and is fully functional but if it could be created automatically when given the input data, then this might be very powerful. Of course, this could cause an explosion in the number of visual modules. It is therefore suggested that the module aggregation described in the previous subsection would have to be in place to help avoid this.

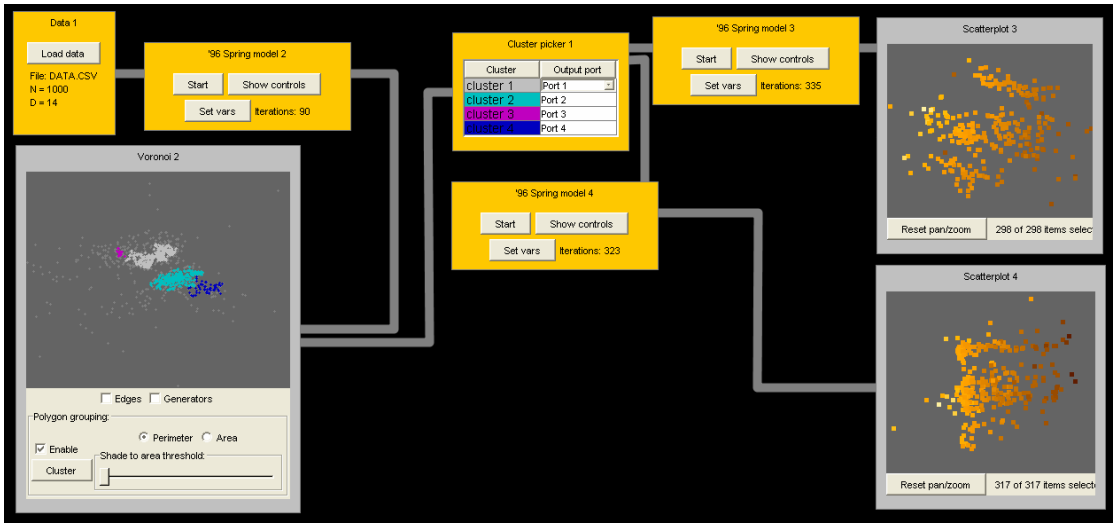


Figure 11.1 A layout of a data set is clustered (bottom-left module). Each cluster feeds into a *cluster-picker* (top-centre module) which can route each cluster to another process for subsequent analysis. In this case spring models are used to layout each of the two largest clusters independently.

11.5 Usability studies

There have numerous calls for more formal evaluation of visualisation systems [GL85, GP96, CC00, Pla04] and recent publications have provided examples and even meta-analyses [CY00]. While Chapter 10 provided evidence that HIVE has been found to be successful in its application to the creation of flexible visualisation applications and hybrid algorithms, it is felt that a formal analysis of its usability would be advantageous. The nature of such analyses is not clear at present but it is suspected that some statistical rigour would be called for to establish the significance of particular facets of HIVE's interface and their use.

Such studies might answer different questions regarding HIVE and the finer details of its user interface. It is anticipated that work in conjunction with CIP, as described in Section 11.2, will prompt this type of evaluation.

11.6 Contribution and thesis statement

The work described in this thesis has contributed to the field of information visualisation in several ways. The early development of hybrid algorithms for dimension reduction produced solutions that were the most efficient at the time of their publication. Such algorithms, being more scalable, have been shown to facilitate the visual exploration of large data sets – sets that were unfeasibly large for the application of earlier dimension reduction solutions. The author also developed a framework for building hybrid algorithms. Knowing more about the types of algorithmic components to combine and when to use them could help in the creation of an algorithmic cookbook where each recipe can be applied to different types of data in various situations. Since this framework has been built into the HIVE system numerous new visualisation techniques and ideas have emerged. In HIVE, hybrid algorithms provide multiple intermediate views of data as they are transformed; algorithms can be profiled at run time; researchers can (and have) extended HIVE to include new visual modules for the development of novel algorithms and they have also used it to explore their data.

At the beginning of this thesis, the author put forward his thesis statement claiming that an algorithmic development environment can be used to build effective dimension reduction solutions. Evidence of this being true is provided in Chapter 10 which provides accounts of how users such as Morrison and Williams have built new algorithms in HIVE. The novel algorithms illustrated in Chapter 5 have also been built and evaluated with the software.

Ultimately, the flexibility of HIVE allows it to be easily extended and new hybrid algorithms to be rapidly prototyped and evaluated.

The thesis statement also claimed that such an environment might support the exploration of abstract multidimensional data. This is supported by findings discussed throughout the thesis. One example is described in Section 7.5.2 where HIVE was used to analyse a data set consisting of environmental measurements gathered from a frozen lake in Antarctica. When HIVE produced a hybrid dimension reduction algorithm and applied it, a final layout was obtained almost instantly in which it was possible to see a clear cluster. Upon further investigation in HIVE, it turned out that this cluster represented erroneous measurements. The ability to view the results at intermediate algorithmic stages has also been demonstrated in this context, and shows potential in allowing greater insight into data. The exploratory power of HIVE is apparent through examples such as its text-mining capabilities and by observations of long term users in the Centre for Investigative Psychology.

The thesis statement finally claimed that building algorithms for visualisation via the use of visualisation methods helps people understand the algorithms better as well as the data that they subsequently transform. Visualisation and data analysis is often a complex task and, as such, is itself a potential application area for InfoVis tools. The author's research developed and explored responses to this, applying analysis techniques to the components, processes and parameters of a visualisation system. An example is provided by the profiling modules. Given demonstrations of their use by Morrison in Section 10.2.1 and in a recent paper [RMC04], they provide a promising way to afford better understanding and control of the visualisation system and, in turn, deepen insight into the visualised information itself.

11.6.1 Novel algorithms

During the course of the author's research several new hybrid algorithms have been developed and have been documented in Chapter 5. The first two were non-linear dimension reduction solutions based upon Chalmers' spring model [Cha96]. One algorithm employed stochastic sampling to obtain an initial reduced representation of the input data before applying further transformations. The other algorithm used k-means clustering in the first stage instead of stochastic sampling.

The author also developed a faster version of Shepard's non-metric MDS algorithm. The increase in speed was obtained by having only two output dimensions and by having both algorithmic stages incorporate the neighbour and sample strategy that was first adopted in Chalmers' spring model.

A novel clustering algorithm was also created. Its purpose was to help highlight potential areas of interest in the output of dimension reduction algorithms. This algorithm is based upon the Voronoi tessellation of a 2-dimensional layout of the input data and is comprised of two stages. The first stage identifies small areas of similar density in the layout while the second stage progressively grows clusters from these density hotspots.

11.7 Research questions

At the beginning of this thesis the author posed four research questions (Section 1.3.2). During the work in this thesis, each has been answered to some extent. The first two questions regard HIVE's algorithmic 'cookbook' for semi-automatically building hybrid algorithms. The second two questions enquire as to their use within HIVE as an environment where algorithm creation and evaluation is integrated with, and potentially enhances, the visual exploration of data. Answering these questions has provided evidence, especially from the observations at CIP Liverpool, that the HIVE framework can enhance hypothesis formation, experimentation and analysis – a fundamental cycle in visual information-seeking.

The answer to the first research question (*Which algorithmic components should be combined?*) was derived from experiments with hybrid algorithms and the development of the hybrid algorithmic framework in HIVE (Chapters 5 and 7). The framework suggests that algorithmic stages should be matched to the complexity of the data as they are transformed by them, successively refining and improving their representation for visualisation. However, it should be noted that some of the visual modules in HIVE, such as the Voronoi clustering module, are self-contained hybrid algorithms. This implies that hierarchical algorithms are possible. Hence the question as to which components to combine may become more concerned with higher level tasks – not just how to dimensionally reduce a data set, but how to use the results in combination with other processes and the overall job in hand.

The second research question (*When should the different types of algorithms be used?*), also regards the hybrid algorithmic framework. The order of successive algorithmic stages is critical to the outcome of a hybrid algorithm and depends upon the components used and the data. The novel dimension reduction algorithms described in Chapter 5 suggest that an algorithm of low time complexity (such as K-means clustering) should first be used to reduce data cardinality before a more expensive algorithm (such as NMDS or a spring model) works to reduce the data dimensionality – further reducing the data to promote their visualisation. In later stages, the representative cardinality can be restored by, for example, fast interpolation.

The third research question in Section 1.3.2 asks: *As well as facilitating the creation and evaluation of hybrid algorithms, can the system be effective in allowing the exploration of the data they transform?* The feedback and observations of HIVE's use, given in Chapter 10, suggest yes. Some computer science researchers have successfully developed and published new visualisation techniques and algorithms built with HIVE. Also, researchers at Liverpool CIP have found value in using HIVE for exploring their data.

The fourth research question asks: *Is visualisation good for creating new visualisations?* Observations of the CIP researchers performing real tasks showed that they had quickly become accustomed to visual programming of data-flows and view coordination. This provides a novel combination of the ability to steer data-flows between processes, and the affordance of interactive coordination across multiple views of data. Also, users did not appear to be hampered by the extensive palette of visual modules. This visual and interactive technique of building and using an application is a visualisation itself and it appears to be intuitive enough for users to learn with little difficulty. Hence the answer to the fourth research question suggests that, yes, it is apparent that visualisations are good for creating new visualisations.

11.8 Reflection and design implications

The author's research has raised design implications for the HIVE software and for hybrid algorithms in general. The course of the author's work spans from early experimentation with novel hybrid dimension reduction algorithms to the design and implementation of HIVE. This progression was made on the basis of a realisation that hybrid algorithms essentially transform data through consecutive – though sometimes parallel – routines until a desired representative state is obtained. Within the context of this thesis, that is the wider field of information visualisation, the general case is that the desired representation state of data is one which lends itself to human visual perception. When it is possible to quickly turn abstract data into information, one is in a better position to pose questions about those data – to form and test hypotheses to obtain actionable knowledge.

In simplifying the representation of data, for example by reducing cardinality and dimensionality, the challenge of mapping their constituent elements to graphical structure is also simplified. Furthermore, if such a reduced representation of the data is obtained where the information that is salient to a user remains encapsulated, then it is more likely that a graphical portrayal will make that information almost immediately apparent.

This provided the motivation for the hybrid framework discussed in Section 7.2, as an abstraction and generalisation of hybrid algorithms. The representation of data according to their quantitative combination of dimensionality and cardinality allows the transition between representations to be thought of as a data-flow through algorithmic components. However, the set of algorithms that have been implemented and reported in this thesis only specify a few of the possible paths through the framework (see Figure 7.2). Although it has been demonstrated how hybrid algorithms such as that in Section 5.3 fit nicely into the framework, there are potentially many more algorithms that the model might fit. There therefore remain lots of algorithmic paths to explore.

This approach clarifies hybrid algorithms by segmenting their components into cohesive units. Treating algorithms this way also provides opportunities for opening them for multiple and intermediate views of data as they are transformed, as demonstrated in Section 7.5.2. It also opens them up for inspection with respect to performance measures such as layout stress and running time. An example of this is given by the profiling modules and case studies discussed in Chapter 8.

The boundaries of the algorithmic units serve as a classification according to suitability of the input and output data. The author has shown how one can take advantage of this to provide a basis for semi-automatically generating algorithms when given input data and a desired output state. In the future it might be worthwhile to see if this would also provide a good basis for explicitly warning the user of excessively time consuming routines due to algorithmic and data complexities, and therefore suggest alternatives. For example, the user might specify that a spring model is to be used to visualise a large body of data. The system could then calculate the estimated time required to run and if it were deemed excessive it could suggest a more efficient algorithm for the job, or perhaps tweak some of the algorithm's parameters. The provision of algorithms according to the framework has been likened to taking recipes from an algorithmic 'cookbook'. However, it is possible that the approach might be extended to not only suggesting algorithmic recipes, but also to suggest recipes for profiling performance. Given a hybrid algorithm, profiling stages might be automatically applied to the appropriate algorithmic stages. Providing this sort of assistance might remove some of the burden from the user when carrying out analyses of data and/or algorithms.

The hybrid algorithmic framework inspired the development of HIVE. Graphically representing algorithmic components and visualisations as well as the flow of data and interactions between them makes them more intuitive. The author took the approach of using visualisation to help create other visualisations, and the feedback from users, as discussed in the previous chapter, indicates that this is indeed useful. As well as providing an almost

tangible grasp of the routines used for visualising data, the same metaphor of computational building blocks applies equally well to profiling modules and other developments such as facilitating batch runs of algorithms. The approach also seems to have benefited tasks such as the analysis of text by providing flexibility in applying queries and viewing the results. However, the author still needs to address issues such as screen space and the potential complexity that accompanies the provision of a wide range of analysis tools. Similar to the assistance HIVE provides with respect to generating hybrid algorithms, it is suggested that assistance might also be given by exposing only the interface widgets that are relevant to the user's current task. This would possibly include module aggregation to make better use of screen space. To achieve this, one would need to define the triggers for prompting such assistance. The user's task at hand could entail the use of several visualisations along with interactions such as brushing and linking views, and this has implications on the use of other tools that might subsequently be used to test hypotheses. Patterns of interaction for different tasks – whether for creating hybrid algorithms or for using them in analysing data – could be used as a basis for dynamically streamlining the user interface, making the more useful widgets and tools closer to hand. This might be achieved by logging user activity with HIVE and subsequently modelling tasks.

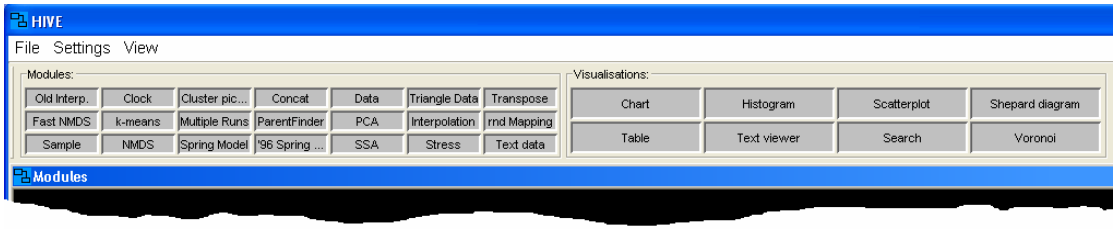
There are several modifications proposed to enhance HIVE and promote its utility. Work prompted by CIP appears to be the most fruitful and will possibly embody all of the suggested future work in this chapter. The potential work discussed here is by no means exhaustive. There are many improvements that can be made. Examples that have not been mentioned include improving Darroch's approach for providing assistance in choosing visual modules [Dar04] and allowing users to create and view annotations pertaining to their use (Section 10.2.3). Also, more hybrid routines must be found that will fall into place in HIVE's 'cookbook' for semi-automatically creating algorithms (Section 7.2). It is hoped that the number of HIVE users will grow and therefore prompt continuing effort in its improvement.

The author's research has produced new algorithms for dimension reduction and clustering, demonstrating the efficacy of a hybrid approach to algorithm development. A framework for this approach has been developed and embodied in a software system allowing the novel combination of algorithm development, evaluation and data analysis via integrated data- and interaction-flows. Early work raised several research questions, which were eventually answered, and as the work progressed, several novel developments occurred and have been published (a list of contributing publications is provided at the start of the thesis).

One of the most interesting aspects of the research is how visualisation techniques are showing potential for helping users understand not only their data but also the tools for exploring them. HIVE has been developed with this in mind, allowing the visualisation of the processes that, in turn, allow the visualisation of data. Using visualisation for visualisation is in keeping with the author's belief that effective tools and techniques should be turned upon themselves whenever possible, to embrace appropriation and hasten their evolution.

Appendix A: List of HIVE modules

In HIVE each toolbar button represents a visual module. Visual modules are the algorithmic or visualisation components that the user drags into the drawing area and connects together to build hybrid algorithms and visualisation applications.



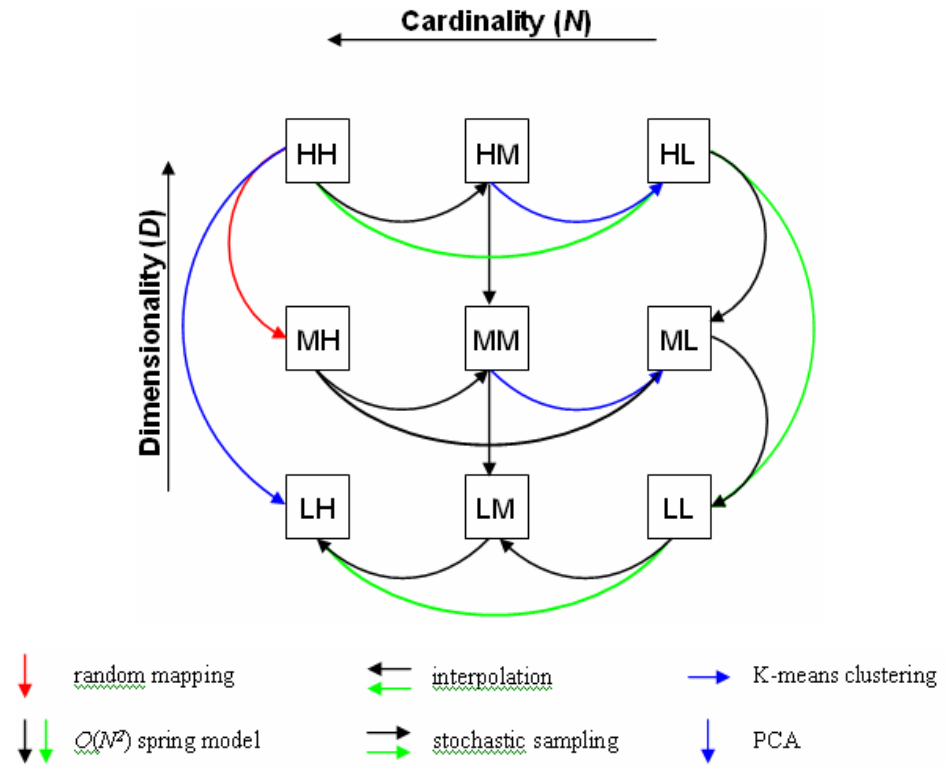
Visual modules can be compiled independently of HIVE and are located in the 'atb\alg' folder of the software's installation directory. User-created visual modules can be imported into HIVE at run time simply by putting them in this directory. A description of the existing set of modules, referring to the above figure, is as follows.

Module name	Description
'96 Spring model	an implementation of Chalmers' spring model
Chart	plots algorithmic profiling measurements such as layout stress against time
Clock	used in algorithmic profiling to measure how long it takes for certain processes
Cluster picker	used in conjunction with the Voronoi module or K-means to allow the user to allocate clusters of data to other processes
Concatenation	takes algorithmic profiling data such as run times, data set sizes and stress measurements to create a HIVE-compatible data source. These data can subsequently be processed in HIVE for visualisation etc.
Data source	allows the user to load in CSV (comma separated value) formatted data. Example data sets are in the 'data in\CSV' directory.
Fast NMDS	a faster implementation of Shepard's non-metric multidimensional scaling
Histogram	allows the user to visualise the distribution of values for a particular quantitative variable. Double-ended sliders allow data selection for coordination with other visualisation modules.
Interpolation	used as part of a hybrid dimension-reduction algorithm to interpolate data onto a 2-d layout
K-means	an iterative, centroid-based clustering algorithm
Multiple runs	used for setting up batch runs of algorithm executions
NMDS	Shepard's original non-metric multidimensional scaling

Module name	Description
Old interp.	an implementation of Brodbeck and Girardin's original interpolation routine [BG98]
ParentFinder	a method developed to speed up data interpolation
PCA	Artificial Neural Network (ANN) implementation of Principal Component Analysis.
Random mapping	implementation of a data projection technique where a data matrix is multiplied by a matrix of random numbers to reduce dimensionality.
Sample	takes a random sample of data
Scatterplot	displays the 2-d output of dimension-reduction algorithms
Search	lets the user search for and highlight text documents. Works in conjunction with 'text data' and 'text viewer'.
Shepard diagram	used for analysing the output of a dimension-reduction process by plotting the high dimensional (ideal) distances to the low dimensional (layout) distances. When the distances are exactly preserved, the plot shows a perfect 45 degree line. View-coordination makes this diagram useful for finding areas of a layout than might be better refined.
Spring model	a customised spring model
SSA	a type of MDS routine called *similarity structure analysis*
Stress	measures stress of a layout configuration
Table	displays a table of data. Combines fisheye focusing and bar chart
Text data	imports a text data index for applying dimension reduction and visualisation of collections of text documents
Text viewer	allows the user to read the contents of documents that are represented in a layout. Used in conjunction with the 'text data' module (see "Text data").
Transpose	transposes a data set, i.e. each row of values becomes a column. This is used in creating layouts of variables.
Triangle data	a data source for lower triangle data matrices
Voronoi	implementation of a clustering algorithm for partitioning a layout

Appendix B: Algorithmic 'cookbook'

The diagram shown below represents the hybrid algorithmic cookbook that has been implemented in HIVE for semi-automatically generating algorithms. Each block represents a data-state or categorisation in terms of dimensionality and cardinality where, for example, *HM* means that the data are of high dimensionality and medium cardinality. Quantitative values for these states are given in Table 7.1. Given data of high dimensionality and high cardinality (*HH*), a user's goal might be to obtain an *LH* data-state, that is a low-dimensional layout of all of the data. This is illustrated by the path of green arrows in the data-state space shown below, and is how the novel algorithms described in Section 5.3 operate.



Each arrow in the diagram represents a change in the data-state which is achieved by a visual module in HIVE. Vertical arrows show dimension reduction and horizontal arrows represent either a reduction in cardinality (pointing right) or an increase in cardinality (pointing left). A path, therefore, constitutes a hybrid algorithm and can be semi-automatically generated by HIVE given the state of the input data and the desired output state.

Appendix C: HIVE user questionnaire

HIVE questionnaire

Greg Ross
Department of Computing Science
University of Glasgow
Scotland
United Kingdom

As part of my PhD research I developed the HIVE software; I would very much appreciate it if you would provide me with some feedback by answering the questions on this page. Your anonymity will be maintained.

It is the aim of this questionnaire to determine whether HIVE has been of use to you, as well as how you have used it.
Please indicate how strongly you agree or disagree with the following statements:

1)	This software is easy to use.	○ Strongly Agree	○ Agree	○ Neither Agree nor Disagree	○ Disagree	○ Strongly Disagree
2)	This software is flexible.	○ Strongly Agree	○ Agree	○ Neither Agree nor Disagree	○ Disagree	○ Strongly Disagree
3)	It is easy to make the software do exactly what I want.	○ Strongly Agree	○ Agree	○ Neither Agree nor Disagree	○ Disagree	○ Strongly Disagree
4)	This software is satisfying to use.	○ Strongly Agree	○ Agree	○ Neither Agree nor Disagree	○ Disagree	○ Strongly Disagree
5)	Hive's design made modification of the source code easy.	○ Strongly Agree	○ Agree	○ Neither Agree nor Disagree	○ Disagree	○ Strongly Disagree

When you load data into a data-source module and then directly connect a scatter plot to it, HIVE automatically loads an algorithm to apply the appropriate transformations for plotting the data. The type of algorithm loaded depends upon the cardinality (number of items) and dimensionality (number of variables) of the data. The following questions pertain to this algorithm-generation and the interaction techniques adopted in HIVE:

6)	HIVE's ability to semi-automatically create hybrid algorithms is useful.	○ Strongly Agree	○ Agree	○ Neither Agree nor Disagree	○ Disagree	○ Strongly Disagree
7)	The automatic generation of algorithms is confusing.	○ Strongly Agree	○ Agree	○ Neither Agree nor Disagree	○ Disagree	○ Strongly Disagree
8)	The interactive creation of data-flows (links between modules) is intuitive.	○ Strongly Agree	○ Agree	○ Neither Agree nor Disagree	○ Disagree	○ Strongly Disagree
9)	The interactive view co-ordination (e.g. coupling between a scatterplot and a histogram) is useful.	○ Strongly Agree	○ Agree	○ Neither Agree nor Disagree	○ Disagree	○ Strongly Disagree
10)	The visual construction of algorithms is advantageous.	○ Strongly Agree	○ Agree	○ Neither Agree nor Disagree	○ Disagree	○ Strongly Disagree

Please answer the following questions about how you used HIVE.

11)	Do you happen to have any previous programming experience?	○ None	○ Less than one year	◉ More than one year but less than five	○ More than five years
12)	Did you ever modify the source code?	○ No	○ Yes		
13)	Did you create any new visual modules in HIVE? (e.g. a new visualisation component)	○ No	○ Yes		
14)	Did you discover any unforeseen patterns or interesting structure in your data when using HIVE?	○ No	○ Yes		

15)	Did HIVE make apparent any patterns in your data that you were already aware of?	○ No	○ Yes
16)	Would you use HIVE again?	○ No	○ Yes

The few remaining questions require you to answer in your own words.

17) What particular aspects of HIVE do you *dislike*?

18) What particular aspects of HIVE do you *like*?

19) What did you use HIVE for?

20) What other software would you use instead of HIVE for the same tasks?

21) Any other comments?

22) Please enter your name. This will not be disclosed with the results.

Thank you very much for taking the time to help me. If you have any queries or would like a newer version of the HIVE software please get in touch with me at: gr@dcs.gla.ac.uk

Submit Reset

Bibliography

[ABK98] Ankerst M., Berchtold S., Keim D. A.: Similarity Clustering of Dimensions for an Enhanced Visualization of Multidimensional Data. *Proceedings of the IEEE Symposium on Information Visualization, InfoVis'98* (1998), 52–60.

[ABKS99] Ankerst M., Breunig M. M., Kriegel H.-P., Sander J.: OPTICS: Ordering Points to Identify the Clustering Structure. *Proceedings of SIGMOD'99, ACM SIGMOD International Conference on Management of Data* (1999), 49–60.

[AC98] Andre, A. D., Cutler H. A.: Displaying Uncertainty in Advanced Navigation Systems. *Proceedings of the Human Factors and Ergonomics Society 42nd Annual Meeting* (October 1998), 31–35.

[Ach01] Achlioptas D.: Database-Friendly Random Projections. *Proceedings of the ACM Symposium on Principles of Database Systems* (2001), 274–281.

[Ado04] Adobe Acrobat Reader 5.0. http://www.adobe.com/acrobat (2004).

[AKS*02] Andrews K., Kienreich W., Sabol V., Becker J., Kappe F., Droschl G., Granitzer M., Auer P., Tochtermann K.: The InfoSky Visual Explorer: Exploiting Hierarchical Structure and Document Similarities. *Information Visualization 1*, (2002), 166–181.

[AS94a] Ahlberg C., Shneiderman B.: Visual Information Seeking using the FilmFinder. *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems 2*, (1994), 433.

[AS94b] Ahlberg C., Shneiderman B.: Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays. *Proceedings of CHI'94, ACM Conference on Human Factors in Computing Systems* (1994), 313–317.

[Asi85] Asimov D.: The Grand Tour: A Tool for Viewing Multidimensional Data. *SIAM Journal on Scientific and Statistical Computing 6*, 1 (1985), 128–143.

[AT95a] Abram G., Treinish L.: An Extended Data-Flow Architecture for Data Analysis and Visualization. *Computer Graphics 29*, 3 (May 1995). 17–21.

[AT95b] Ahlberg C., Truvé S.: Tight Coupling: Guiding User Actions in a Direct Manipulation Retrieval System. *Proceedings of HCI'95*, *Engineering for Human-Computer Interaction* (1995), 305–321.

[ATS82] Apperley M. D., Tzavaras I., Spence R.: A Bifocal Display Technique for Data Presentation. *Proceedings of Eurographics'82, Conference of the European Association for Computer Graphics* (1982), 27–43.

[Aur91] Aurenhammer F.: Voronoi Diagrams: A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys, 23*, 3 (1991), 345–405.

[AW95] Ahlberg C., Wistrand E.: IVEE: An Information Visualization and Exploration Environment. *Proceedings of InfoVis '95, IEEE Symposium on Information Visualization* (October 1995), 66–73.

[Bas00] Basalaj W.: *Proximity Visualisation of Abstract Data*. PhD thesis, University of Cambridge Computer Laboratory (2000).

[Bau75] Baumgart B.: A Polyhedron Representation for Computer Vision. *National Computer Conference* (1975), 589–596.

[BBB*93] Brodlie K., Brankin L., Banecki G., Gay A., Poon A., Wright H.: Grasparc–A Problem Solving Environment Integrating Computation and Visualization. *Proceedings of IEEE Visualization '93, IEEE Computer Society Press* (1993), 102–109.

[BC87] Becker R., Cleveland W.: Brushing Scatterplots. *Technometrics 29*, 2 (1987), 127–142.

[BCLC97] Brodbeck D., Chalmers M., Lunzer A., Cotture P.: Domesticating Bead: Adapting an Information Visualization System to a Financial Institution. *Proceedings of InfoVis '97, IEEE Symposium on Information Visualization* (1997), 73–80.

[BCS96] Buja A., Cook D., Swayne D. F.: Interactive High-Dimensional Data Visualization. *Journal of Computational and Graphical Statistics 5* (1996), 78–99.

[BF98] Bradley P. S., Fayyad U. M.: Refining Initial Points for K-Means Clustering. *Proceedings of the Fifteenth International Conference on Machine Learning* (1998). 91–99.

[BG05] Brodbeck D., Girardin L.: Macrofocus. http://www.macrofocus.com (January 2005).

[BG97] Borg I., Groenen P. J. F.: *Modern Multidimensional Scaling Theory and Applications*. Springer-Verlag, New York, 1997.

[BG98] Brodbeck D., Girardin L.: Combining Topological Clustering and Multidimensional Scaling for Visualising Large Data Sets. Unpublished paper (accepted for, but not published in, *Proceedings of InfoVis'98, IEEE Information Visualization 1998*).

[BH94] Bederson B. B., Hollan J. D.: Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. *Proceedings of UIST'94, ACM Symposium on User Interface Software and Technology* (1994), 17–26.

[BKR98] Bookstein A., Klein S. T., Raita T.: Clumping Properties of Content-Bearing Words. *Journal of the American Society for Information Science 49*, 2 (1998), 102–114.

[BM01] Bingham E., Mannila H.: Random Projection in Dimensionality Reduction: Applications to Image and Text Data. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (2001), 245-250.

[BS02] Buja A., Swayne D. F.: Visualization Methodology for Multidimensional Scaling. *Journal of Classification 19* (2002), Springer-Verlag , New York , 7–43.

[BSLD98] Buja A., Swayne D. F., Littman M., Dean N.: XGvis: Interactive Data Visualization with Multidimensional Scaling. *under review Journal of Computational and Graphical Statistics* (1998).

[BST*94] Brachman R. J., P. Selfridge G., Terveen L. G., Altman B., Borgida A., Halper F., Kirk T., Lazar A., McGuinness D. L., Resnick L. A.: Integrated Support for Data Archaeology. *Proceedings of AAAI-93 Knowledge Discovery in Databases Workshop* (Aug 1994), 197–211.

[Bur00] Burns C. M.: Putting it All Together: Improving Integration in Ecological Displays. *Human Factors 42* (2000), 226–241.

[BWK00] Baldonado M., Woodruff A., Kuchinsky A.: Guidelines for Using Multiple Views in Information Visualization. *Proc. ACM Advanced Visual Interfaces* (May 2000), 110–119.

[Can85] Canter D. (ed.): *Facet Theory: Approaches to Social Research*. Springer-Verlag, 1985.

[CBCH95] Cook D., Buja A., Cabrera J., Hurley H.: Grand Tour and Projection Pursuit. *Journal of Computational and Graphical Statistics 2*, 3 (1995), 225–250.

[CC00] Chen C., Czerwinski M. P.: Empirical Evaluation of Information Visualizations: An Introduction. *International Journal for human-Computer Studies 53*, (2000), 631–635.

[CDH92] Croft A., Davidson R. Hargreaves M.: *Engineering Mathematics: A Modern Foundation for Electronic, Electrical and Control Engineers*. Addison Wesley, 1992.

[CF98] Canter D. Fritzon K.: Differentiating Arsonists: A Model of Firesetting Actions and Characteristics. *Legal and Criminological Psychology 3*, (1998), 73–96.

[Cha93] Chalmers M.: Using a Landscape Metaphor to Represent a Corpus of Documents. *Proceedings of COSIT '93,European Conference on Spatial Information Theory* (1993), 377–390.

[Cha96] Chalmers M.: A Linear Iteration Time Layout Algorithm for Visualising High-Dimensional Data. *Proceedings of IEEE Visualization* (1996), 127–132.

[CIP96] Chalmers M., Ingram R., Pfranger C.; Adding Imageability Features to Information Displays. *Proceedings ACM UIST' 96, ACM Press* (1996), 33–39.

[CKBR97] Chi E. H., Konstan J., Barry P., Riedl J.: A Spreadsheet Approach to Information Visualization. *Proceedings of ACM User Interface System Technologies* (October 1997), 79–80.

[CKPT92] Cutting D. R., Karger D. R., Pederson J. O. Tukey J. W.: Scatter/Gather: A Cluster-Based Approach to Browsing Large Document Collections. *Proceedings of SIGIR'92, ACM Conference on Research and Development in Information Retrieval* (1992), 318–329.

[CL02] Estivill-Castro, V., Lee I..: Argument Free Clustering for Large Spatial Point Data Sets Via Boundary Extraction from Delaunay Diagram. *Computers, Environment and urban systems 26* (2002), 315–334.

[CM97] Card S. K., Mackinlay J. D.: The Structure of the Information Visualization Design Space. *Proceedings of InfoVis'97, IEEE Symposium on Information Visualization* (1997), 92–99.

[CMS99] Card S. K., Mackinlay J. D., Shneiderman B.: *Information Visualisation – Using Vision to Think*. Morgan Kaufmann, 1999.

[Coh97] Cohen J. D.: Drawing Graphs to Convey Proximity: An Incremental Arrangement Method. *ACM Transactions on Computer-Human Interaction (TOCHI) 4*, 3 (1997), 197–229.

[CRB98] Chalmers M., Rodden K., Brodbeck D.: The Order of Things: Activity-Centred Information Access. *Proc. WWW7*, Brisbane (April 1998), 359–367.

[CRM91] Card S. K., Robertson G. G., Mackinlay J. D.: The Information Visualizer, an Information Workspace. *Proceedings of CHI `91, ACM Human Factors in Computing Systems Conference* (1991), 181–188.

[CT98] I. F. Cruz and R. Tamassia. Graph Drawing Tutorial. http://www.cs.brown.edu/people/rt/papers/gd-tutorial/gd-constraints.pdf (1998).

[CY00] Chen C., Yu Y.: Empirical Studies of Information Visualization: a Meta-Analysis. *International Journal of Human-Computer Studies 53*, 5 (November 2000), 851 – 866.

[Dar04] Darroch I. D. G.: *An End-User Mechanism for Describing Software Component Chains*. PhD Thesis, Department of Computing Science, University of Glasgow, Scotland (2004).

[Das00] Dasgupta S.: Experiments with Random Projection. *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence* (2000), 143–151.

[DBM89] DeFanti T. A., Brown M. D., McCormick B. H.: Visualization: Expanding Scientific and Engineering Research Opportunities. *IEEE Computer 22*, 8 (1989), 12–25.

[DDF*90] Deerwester S., Dumais S. T., Furnas G. W., Landauer T. K., Harshman R. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science, 41*, 6 (1990), 391–407.

[DELS99] Dourish P., Edwards W. K., LaMarca A., Salisbury M.: Using Properties for Uniform Interaction in the Presto Document System. *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology* (November 1999), 55–64.

[DGH94] Duyckaerts C., Godefroy G., Hauw J-J.: Evaluation of Neuronal Density by Dirichlet Tessellation, *Journal of Neuroscience Methods 52*, 1 (1994), 47–69.

[DLR77] Dempster A. P., Laird, N. M., Rubin, D. B. Maximum Likelihood for Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society, Series B*, *39* (1977), 341–353.

[DP95] Dumas J., Parsons P.: Discovering the Way Programmers Think about New Programming Environments. *Communications of the ACM 38*, 6 (June 1995), 45–56.

[Ead84] Eades P.: A Heuristic for Graph Drawing. *Congressus Numerantium 42* (1984), 149–160.

[ED91] Everitt B. S. Dunn G.: *Applied Multivariate Data Analysis*. Arnold, 1991.

[EH97] Eldershaw C. Hegland M.: Cluster Analysis Using Triangulation. *Computational Techniques and Applications* (1997), 201–208.

[Eic94a] Eick S.: Graphically Displaying Text. *Journal of Computational and Graphical Statistics 3*, 2 (1994). 127–142.

[Eic94b] Eick S. :Data Visualization Sliders. *Proceedings of UIST '94, ACM Symposium on User Interface Software and Technology*, (November 1994), 119–120.

[Ekm54] Ekman G. Dimensions of Color Vision. *Journal of Psychology 38* (1954), 467–474.

[EKSX96] Ester M., Kriegel H.-P., Sander J., Xu X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining* (1996), 226–231.

[ESK03] Ertöz L., Steinbach M., Kumar V.: Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data. *Proceedings of the 3rd SIAM International Conference on Data Mining* (2003).

[EW95] Eick S. G.: Wills G. J.: High Interaction Graphics. *European Journal of Operational Research 84* (1995), 445–459.

[Fas99] Fasulo D. An Analysis of Recent Work on Clustering Algorithms. http://citeseer.nj.nec.com/fasulo99analysi.html (April 1999).

[FB03] Fern X., Brodley C.: Random Projection for High Dimensional Data: A Cluster Ensemble Approach. *Proceedings of the 20th International Conference on Machine Learning* (2003), 186–193.

[Fek04] Fekete J, -D.: The InfoVis Toolkit. *Proceedings of the 10th IEEE Symposium on Information Visualization* (2004), 167-174.

[FL95] Faloutsos C. Lin K.-I.: FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data* (1995), 163–174.

[FMK*99] Fitzpatrick G., Mansfield T., Kaplan S., Arnold D., Phelps T., Segal B.: Instrumenting and Augmenting the Workaday World with a Generic Notification Service called Elvin. *Proceedings of the 6th European Conference on Computer Supported Cooperative Work* (1999), 431–451.

[FP99] Fekete J.-D., Plaisant C.: Excentric labeling: Dynamic neighborhood labeling for data visualization. *Proceedings of the International Conference on Human Factors in Computing Systems* (1999), 512–519.

[FR91] Fruchterman T. M. J., Reingold E. M.: Graph-Drawing by Force-Directed Placement. *Software – Practise and Experience 21*, 11 (1991), 1129–1164.

[Fre02] Fred A.L.N.: Finding Consistent Clusters in Data Partitions. *Proceedings of the 3rd International Workshop on Multiple Classifier Systems. Eds. F. Roli, J. Kittler* (2002), 309–318.

[Fri94] Friedman J. H.: An Overview of Predictive Learning and Function Approximation. *In V. Cherkassky, J.H. Friedman, and H. Wechsler, eds.* From Statistics to Neural Networks, Proc. NATO/ASI Workshop (1994), 1–61. Springer Verlag.

[FS95] Fishkin K., Stone M. C.: Enhanced Dynamic Queries via Movable Filters. *Proceedings of ACM Conference on Human Factors in Computing Systems* (May 1995), 415–420.

[FTM*98] Flach J. M., Tanabe F., Monta K., Vicente K. J., Rasmussen J.: An Ecological Approach to Interface Design. *Proceeding of the Human Factors and Ergonomics Society 42nd Annual Meeting* (1998), 295–299.

[Fur86] Furnas G. W.: Generalized Fisheye Views. *Proceedings of the ACM Conference on Human Factors in Computing Systems* (1986), 16–23.

[Gav92] Gaver W. W.: The Affordances of Media Spaces for Collaboration. *Proceedings of CSCW '92* (November 1992), 17–24.

[GHJV95] Gamma E., Helm R., Johnson R., Vlissides J.: *Design Patterns*. Addison-Wesley Professional. 1st edition (January 1995).

[Gib79] Gibson J. J., *The Ecological Approach to Visual Perception.* Houghton-Mifflin (1979).

[GL85] Gould J. D., Lewis C., Designing for Usability: Key Principles and What Designers Think. *Communications of the ACM 28*, 3 (March 1985). 300–311.

[GMPS00] Greene S., Marchionini G., Plaisant C., Shneiderman B.: Previews and Overviews in Digital Libraries: Designing Surrogates to Support Visual Information Seeking. *Journal of the American Society for Information Science 51*, 4 (2000), 380–393.

[GP96] Green T.R.G., Petre M.: Usability Analysis of Visual Programming Environments: A `Cognitive Dimensions' Framework. *Journal of Visual Languages and Computing 7*, 2 (1996), 131–174.

[Gre89] Green T. R. G., Cognitive Dimensions of Notations. *Proceeding of the HCI '89 Conference on People and Computers* (1989), 443–460.

[Guo03] Guo D.: Coordinating Computational and Visual Approaches for Interactive Feature Selection and Multivariate Clustering. *Information Visualization, 2* 2003, 232–246.

[Gut68] Guttman L.: A General Nonmetric Technique for Finding the Smallest Coordinate Space for a Configuration of Points. *Psychometrika 33* (1968), 469–506.

[Hae88] Haeberli P.: ConMan: A Visual Programming Language for Interactive Graphics. *Computer Graphics 22*, 4 (1988), 103–111.

[HCL05] Heer J., Card S. K., Landay J. A.: Prefuse: a Toolkit for Interactive Information Visualization. *Proceedings of the SIGCHI conference on Human factors in computing systems* (2005). 421–430.

[HD96] Harrison S., Dourish P.: Re-place-ing Space: The Roles of Place and Space in Collaborative Systems. *Proceedings of the ACM Conference on Computer Supported Cooperative Work CSCW '96* (1996), 67–76.

[HH99] Hendry D.G., Harper D. J.: An Informal Information-Seeking Environment. *Journal of the American Society for Information Science. 48*, 11 (1999), 1036–1048.

[HHK00] Hollan J., Hutchins E., Kirsh D.: Distributed Cognition: Towards a New Foundation for Human-Computer Interaction Research. *ACM Transactions on Computer-Human Interaction 7*, 2 (June 2000), 174-196.

[HHWM92] Hill W. C., Hollan J. D., Wroblewski D., McCandless T.: Edit Wear and Read Wear. *Proceedings of CHI '92, ACM Conference on Human Factors in Computing Systems* (May 1992), 3–9.

[HK97] Hearst M., Karadi C.: Cat-a-Cone: An Interactive Interface for Specifying Searches and Viewing Retrieval Results using a Large Category Hierarchy. *Proceedings of 20th Annual International ACM/SIGIR Conference* (July 1997), 246–255.

[HK98] Hinneburg A., Keim D. A.: An Efficient Approach to Clustering in Large Multimedia Databases with Noise. *Proceedings of KDD'98, The Fourth International Conference on Knowledge Discovery and Data Mining* (1998), 58–65.

[HKLK96] Honkela T., Kaski S., Lagus K., Kohonen T.: Exploration of Full-Text Databases with Self-Organizing Maps. *Proceedings ICNN96, International Conference on Neural Networks* (1996), 56–61.

[HKLK97] Honkela T., Kaski S., Lagus K., Kohonen T.: WEBSOM–Self-Organizing Maps of Document Collections. *Proceedings of WSOM'97, Workshop on Self-Organizing Maps* (1997), 310–315.

[Hon04] Honeycomb. http://www.hivegroup.com (2004).

[Hub85] Huber P.J.: Projection Pursuit. *The Annals of Statistics 13,* 2 (1985), 435–475.

[HW96] Heskes T., Wiegerinck W.: A Theoretical Comparison of Batch-Mode, On-Line, Cyclic, and Almost Cyclic Learning. *IEEE Transactions on Neural Networks 7*, 4 (1996), 919–925.

[HWR03] Huang S., Ward M. O., Rundensteiner E. A.: Exploration of Dimensionality Reduction for Text Visualization. Technical Report TR-03-14, Worcester Polytechnic Institute, Computer Science Department. Available from http://citeseer.ist.psu.edu/huang03exploration.html (2003).

[IB95a] Ingram, R., Benford S.: Improving the Legibility of Virtual Environments. *Proceedings of the 2nd Eurographics Conference on Virtual Environments* (1995), 211–223.

[IB95b] Ingram R., Benford S. Legibility Enhancement for Information Visualization. *Proceedings of IEEE Visualization* (1995), 209 –216.

[Ins85] Inselberg A.: The Plane with Parallel Coordinates. *The Visual Computer 1*, 2 (1985), 69–91.

[Jak04] Jakarta project, Lucene, http://jakarta.apache.org/lucene/docs/index.html (August 2004).

[JL84] Johnson W.B., Lindenstrauss J.: Extensions of Lipschitz Mapping into Hilbert Space. *Proceedings of Conference in Modern Analysis and Probability Vol. 26 of Contemporary Mathematics* (1984), 189–206.

[JMF99] Jain A. K., Murty M. N., Flynn P. J.: Data Clustering: A Review. *ACM Computing Surveys 31*, 3 (September 1999), 264–323.

[JP73] Jarvis R. A., Patrick E. A.; Clustering Using a Similarity Measure Based on Shared Nearest Neighbors. *IEEE Transactions on Computers 22*, 11 (1973), 1025–1034.

[JS91] Johnson B., Shneiderman B.: Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures. *Proceedings of IEEE Visualization '91 Conference* (1991), 284–291.

[JS94] Jain V., Shneiderman B.: Data Structures for Dynamic Queries: An Analytical and Experimental Evaluation. *Proceedings of Conference on Advanced Visual Interfaces `94* (1994), 1–11.

[Kas98] Kaski S.: Dimensionality Reduction by Random Mapping: Fast Similarity Computation for Clustering. *Proceedings of IJCNN'98, 1998 IEEE International Joint Conference on Neural Networks* (1998), 4–9.

[KC03] Koren Y., Carmel L.: Visualization of Labeled Data Using Linear Transformations. *Proceedings of InfoVis'03, the 9th IEEE Symposium in Information Visualisation* (2003), 121–128.

[KHDM98] Kittler J., Hatef M., Duin R. P. W., Matas J.: On Combining Classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*, 3 (1998), 226–239.

[KHK99] Karypis G., Han E.-H., Kumar V.: CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modelling. *IEEE Computer 32*, 8 (1999), 68–75.

[KKL*00] Kohonen T., Kaski S., Lagus K., Salojrvi J., Paatero V., Saarela A.: Self Organization of a Massive Document Collection. *IEEE Transactions on Neural Networks 11*, 3 (2000), 574–585.

[Kor91] Korfhage R. R.: To See, or Not to See – Is *that* the Query? *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1991), 131–141.

[KP88] Krasner G., Pope S.: A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 system. *Journal of Object Oriented Programming 1, 3* (1988), 26–49.

[Krus64] Kruskal J. B., Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis. *Psychometrika 29*, 1 (1964), 1–27.

[KS97] Kandogan E., Shneiderman B.: Elastic Windows: Evaluation of Multi-Window Operations. *Proc. CHI '97* (1997), 250–257.

[Lam00] Lam L.: Classifier Combinations: Implementations and Theoretical Issues. *In Kittler and Roli, eds. Multiple Classifier Systems, vol. 1847 of Lecture Notes in Computer Science* (2000), 78–86.

[LBWR94] Lohse G. L., Biolsi K., Walker N., Rueter H. H.: A Classification of Visual Representations. *Communications of the ACM Vol. 37* (December 1994), 36–49.

[LG03] Lin J., Gunopulos D.: Dimensionality Reduction by Random Projection and Latent Semantic Indexing. *Proceedings of the Text Mining Workshop at the 3rd SIAM International Conference on Data Mining* (2003), 1–3.

[LGH02] Larkin S., Grant A. J., Hewitt W. T.: Visualization in Parallel (VIPAR). Manchester Visualization Centre, University of Manchester, Funded as part of the Portable Software Tools for Parallel Architectures (PSTPA) Project, (EPSRC Research Grant Reference Number: GR/K40390) (2002).

[LHKK96] Lagus K., Honkela T., Kaski S., Kohonen T.: Self-Organizing Maps of Document Collections: A New Approach to Interactive Exploration. *Proceedings of the International Conference on Knowledge Discovery and Data Mining* (1996), 238–243.

[LRB*97] Livny M., Raghu R., Beyer K. S., Chen G., Donjerkovic D., Lawande S., Myllymaki J., Wenger R. K.: DEVise: Integrated Querying and Visual Exploration of Large Datasets. *Proceedings of ACM SIGMOD* (May 1997), 301–312.

[LRP95] Lamping J., Rao R., Pirolli P.: A Focus+Context Technique Based Upon Hyperbolic Geometry for Visualizing Large Hierarchies. *Proceedings of CHI'95, ACM Conference on Human Factors in Computing Systems* (1995), 401–408.

[LS97a] Luger G. F., Stubblefield, W. A.: *Artificial Intelligence – Structures & Strategies for Complex Problem Solving*. (3rd edition), Addison-Wesley (1997).

[LS97b] Lam L., Suen C. Y.: Application of Majority Voting to Pattern Recognition: An Analysis of its Behaviour and Performance. *IEEE Transactions on Systems, Man and Cybernetics–Part A: Systems and Humans 27*, 5 (1997), 553–568.

[LSM91] Lin X., Soergel, D., Marchionini, G : A Self-Organizing Semantic Map for Information Retrieval. *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1991), 262–269.

[Lun99] Lunzer A.: Choice And Comparison Where The User Wants Them: Subjunctive Interfaces For Computer-Supported Exploration. *Proceedings of INTERACT '99, the 7th IFIP Conference on Human-Computer Interaction* (1999), 474–482.

[Lyn60] Lynch K.: *The Image of the City*. M.I.T. Press, 1960.

[Mac67] MacQueen J.: Some Methods for Classification and Analysis of Multivariate Observations. *Proc. 5th Berkeley Symposium* (1967), 281–297.

[MC04] Morrison A., Chalmers M.: Improving Hybrid MDS with Pivot-Based Searching. *Information Visualization 3*, 2 (2004), 109–122.

[MDB87] McCormick B. H., DeFanti T. A., Brown M. D., eds.: Visualization in Scientific Computing. *Computer Graphics 21*, 6 (November 1987).

[MLST03] Manoranjan D., Liu H., Scheuermann P., Tan K. L.: Fast Hierarchical Clustering and its Validation. *Data and Knowledge Engineering 44* (2003), 109–138.

[Mor04] Morrison A.: *Fast Multidimensional Scaling Through Hybrid Nonlinear Algorithms*. PhD Thesis, Department of Computing Science, University of Glasgow, Scotland (2004).

[MRC02] Morrison A., Ross G., Chalmers M.: A Hybrid Layout Algorithm for Sub-Quadratic Multidimensional Scaling. *Proceedings of InfoVis'02, IEEE Symposium on Information Visualisation* (October 2002), 152–158.

[MRC03] Morrison A., Ross G., Chalmers M.: Fast Multidimensional Scaling Through Sampling, Springs and Interpolation. *Information Visualization 2*, 1 (2003), 68–77.

[MRC91] Mackinlay J. D., Robertson G. G., Card S. K.: The Perspective Wall: Detail and Context Smoothly Integrated. *Proceedings of CHI'91, ACM Conference on Human Factors in Computing Systems* (1991), 173–179.

[MS90] McDonald J.A., Stuetzle W.: Painting Multiple Card Views of Complex Objects. *Proceedings of the ECOOP/OOPSLA'90 European Conference on Object Oriented Programming* (October 1990), 245–257.

[Nic04] Nickleby HFE Ltd. NicklebyKIT. http://www.nickleby.com (2004).

[Nor01] North C.: Multiple Views and Tight Coupling in Visualization: A Language, Taxonomy, and System. *Proc. CSREA CISST 2001 Workshop of Fundamental Issues in Visualization* (2001), 626–632.

[Nor88] Norman D. A.: *The Psychology of Everyday Things*. Basic Books, 1988.

[NS00a] North C., Shneiderman B.: Snap-Together Visualization: Can Users Construct and Operate Coordinated Visualizations? *International Journal of Human-Computer Studies 53*, (2000), 715–739.

[NS00b] North C., Shneiderman B.: Snap-Together Visualization: A User Interface for Coordinating Visualizations Via Relational Schemata. *Proceedings of Advanced Visual Interfaces* (2000), 128–135.

[NS01] North C., Shneiderman B.: Component-Based, User-Constructed, Multiple-View Visualization. *Proc. ACM CHI 2001* (2001), 201–202.

[NSP96] C. North. B. Shneiderman and C. Plaisant. User Controlled Overviews of an Image Library: A Case Study of the Visible Human. *Proceedings of DL'96, ACM Conference on Digital Libraries*. 74–82. 1996.

[NTMS91] Nierstrasz O., Tsichritzis D., de Mey V., Stadelmann M.: Objects + Scripts = Applications, *Proceedings of ESPRIT Conference 1991. Kluwer Academic Publishers* (1991), 534–552.

[OBSC00] Okabe A., Boots B., Sugihara K., Chiu S.N.: *Spatial Tessellations. Concepts and Applications of Voronoi Diagrams*. Wiley, Chichester, second edition (2000).

[Oja82] Oja E.: A Simplified Neuron Model as a Principal Component Analyzer. *Journal of Mathematical Biology 15* (1982), 267–273.

[Pag74] Page R. L.: A Minimal Spanning Tree Clustering Method. *Communications of the ACM 17*, 6 (1974), 321–323.

[Pla04] Plaisant C.: The Challenge of Information Visualization Evaluation. *Proceedings of the Working Conference on Advanced Visual Interfaces* (2004). 109–116.

[PMR*96] Plaisant C., Milash B., Rose A., Widoff S., Shneiderman B.: LifeLines: Visualizing Personal Histories. *Proceedings of CHI'96, ACM Conference on Human Factors in Computing Systems* (1996), 221–227.

[Por80] Porter M. F.: An Algorithm for Suffix Stripping, *Program 14*, 3 (1980), 130–137.

[Pri57] Prim R. C.: Shortest Connection Matrix Network and Some Generalizations. *Bell System Technical Journal 36* (1957), 1389–1401.

[PRR*01] Petre M., Roast C., Roe C., Wong A., Young R. M.: Cognitive Dimensions of Notations: Design Tools for Cognitive Technology. *Cognitive Technology* (2001), 325–341.

[PSHD96] Pirolli P., Schank P., Hearst M., Diehl C.: Scatter/Gather Browsing Communicates the Topic Structure of a Very Large Text Collection. *Conference on Human Factors in Computing Systems* (April 1996), 213–220.

[RB99] Ribeiro-Neto B., Baeza-Yates R.: *Modern Information Retrieval*. ACM Press Series, Addison-Wesley, 1999.

[RBSW01] Rodden K., Basalaj W., Sinclair D., Wood K.: Does Organisation by Similarity Assist Image Browsing? *Proceedings of the SIGCHI on Human Factors in Computing Systems* (2001), 190–197.

[RC03a] Ross G., Chalmers M.: A Visual Workspace for Hybrid Multidimensional Scaling Algorithms. *Proceedings of InfoVis'03, IEEE Symposium on Information Visualization* (2003), 91–96.

[RC03b] Ross G., Chalmers M.: A Visual Workspace for Constructing Hybrid MDS Algorithms and Coordinating Multiple Views. *Information Visualization 2*, 4 (2003), 247–257.

[RCK*97] Roth S. F., Chuah M. C., Kerpedjiev S., Kolojejchick J., Lucas P.: Towards an Information Visualization Workspace: Combining Multiple Means of Expression. *Human-Computer Interaction Journal 12*, 1 (1997), 131–185.

[RCM89] Robertson G. G., Card S. K., Mackinlay J. D.: The Cognitive Coprocessor Architecture for Interactive User Interfaces. *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology* (1989), 10–18.

[Ren94] Rennison E.: Galaxy of News: An Approach to Visualizing and Understanding Expansive News Landscapes. *Proceedings of UIST'94, ACM Symposium on User Interface Software and Technology* (1994), 3–12.

[RK02] Roli F., Kittler J.: Fusion of Multiple Classifiers. *Information Fusion 3* (2002) , 243.

[RLS*96] Roth S. F., Lucas P., Senn J. A., Gomberg C. C., Burks M. B., Stroffolino P. J., Kolojejchick J. A., Dunmire C.: Visage: A User Interface Environment for Exploring Information. *Proceedings of IEEE Information Visualization* (October 1996), 3–12.

[RMC04] Ross G., Morrison A. J. Chalmers M.: Coordinating Views for Data Visualisation and Algorithmic Profiling. *In Proceedings of CMV'04, the IEEE International Conference on Coordinated and Multiple Views in Exploratory Visualization* (July 2004), 3–14.

[RMC05] Ross G., Morrison A. J. Chalmers M.: Visualisation Techniques for Users and Designers of Layout Algorithms. *In Proceedings of IV'05, the 9th International Conference on Information Visualisation* (July 2005), 579–586.

[RMC91] Robertson G. G., Mackinlay J. D. Card S. K.: Cone Trees: Animated 3D Visualizations of Hierarchical Information. *Proceedings of CHI'91, ACM Conference on Human Factors in Computing Systems* (1991), 189–194.

[RMS92] Ritter H., Martinetz T., Schulten K.: *Neural Computation and Self-Organizing Maps*. Addison Wesley. 1992.

[Rob98] Roberts J. C.: On Encouraging Multiple Views for Visualization. *IEEE Conference Information Visualization IV '98* (July 1998), 8–14.

[Rob99] Roberts J. C.: Display Models for Visualization. *Proceedings of the International Conference on Information Visualization* (July 1999), 200–206.

[Rom01] Rome E.: Simulating Perceptual Clustering by Gestalt Principles. *25th Workshop of the Austrian Association for Pattern Recognition ÖAGM / AAPR* (June 2001), 191–198.

[RRBW03] Rosario G. E., Rundensteiner E. A., Brown D. C., Ward M. O.: Mapping Nominal Values to Numbers for Effective Visualization. *Proceedings of InfoVis'03, IEEE Symposium on Information Visualization* (2003), 113–120.

[RS00] Roweis S. T., Saul L. K.: Nonlinear Dimensionality Reduction by Locally Linear Embedding, *Science* 290 (2000), 2323–2326.

[RS98] Reising D. V. C., Sanderson P. M.: Designing Displays Under Ecological Interface Design: Towards Operationalizing Semantic Mapping. *Proceeding of the Human Factors and Ergonomics Society 42nd Annual Meeting* (October 1998), 372–376.

[RSPC93] Russell D. M., Stefik M. J., Pirolli P., Card S. K.: The Cost Structure of Sensemaking. *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems* (1993), 269–276.

[SA82] Spence R., Apperley M. D.: Data Base Navigation: An Office Environment for the Professional. *Behaviour and Information Technology 1*, 1 (1982), 43–54.

[Sal71] Salton G. ed.: *The SMART Retrieval System–Experiments in Automatic Document Processing*. Prentice Hall, 1971.

[SB82] Sudman S., Bradburn N. M.: *Asking Questions: A Practical Guide to Questionnaire Design*. Jossey-Bass, 1982.

[SCB98] Swayne D. F., Cook D., Buja A.: XGobi: Interactive Dynamic Data Visualization in the X Window System. *Journal of Computational and Graphical Statistics 7*, (1998), 113–130.

[SC00] Su M.-C., Chang H.-T.: Fast Self-Organizing Feature Map Algorithm. *IEEE Transactions on Neural Networks 11*, 3 (2000), 721–723.

[Sch96] Schikuta E.: Grid Clustering: An Efficient Hierarchical Clustering Method for Very Large Data Sets. *Proceedings of the 13th Conference on Pattern Recognition*, Vol. 2 (1996), 101–105.

[SDB01] Savaresi S. M., Boley D. L.: On the Performance of Bisecting K-Means and PDDP. *First SIAM International Conference on Data Mining* (April 2001), 1–14.

[SEKX98] Sander J., Ester M., Kriegel H.-P., Xu X.: *Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications*. *Data Mining and Knowledge Discovery 2*, 2 (June 1998), 169–194.

[She62] Shepard R. N.: The analysis of proximities: Multidimensional scaling with an unknown distance function, parts I and II, *Psychometrika 27* (1962), 125–140 and 219–246.

[Shn83] Shneiderman B.: Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer 16*, 8 (1983), 57–68.

[Shn94] Shneiderman B.: Dynamic Queries for Visual Information Seeking. *IEEE Software 11*, 6 (1994), 70–77.

[Shn96] Shneiderman B.: The Eyes Have it: A Task by Data Type Taxonomy for Information Visualization. *Proceedings of IEEE Workshop on Visual Languages* (1996), 336–343.

[Sib73] Sibson R.: SLINK: an Optimally Efficient Algorithm for the Single-Link Cluster Method. *The Computer Journal 16,* 1 (1973), 30–34.

[SK02] Schroeder M., Katopodis G.: Can Hierarchical Clustering Improve the Efficiency of Non-Linear Dimension Reduction with Spring Embedding? *Proceedings of VDM'02, 2nd International Workshop on Visual Data Mining* (August 2002).

[SKK00] Steinbach M., Karypis G., Kumar V.: A Comparison of Document Clustering Techniques. *KDD-2000 Workshop on Text Mining* (2000), 109–110.

[SML96] Schroeder W. J., Martin K. M., Lorensen W. E.: The design and implementation of an object-oriented toolkit for 3D graphics and visualization. *Proceedings of the 7th conference on Visualization* (1996), 93–100.

[Sow00] Sowa J. F.: *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole, 2000.

[Spi00] Spiegler I.: Knowledge Management: A New Idea or a Recycled Concept. *Communications of the Association for Information Systems 3*, 1 (June 2000). Article 14.

[Spo93] Spoerri A.: InfoCrystal: A Visual Tool for Information Retrieval and Management. *Proceedings of CIKM Conference* (1993), 11–20.

[Str02] Strehl A.: Cluster Ensembles–A Knowledge Reuse Framework for Combining Multiple Partitions. *Journal of Machine Learning Research 3*, (2002), 583–617.

[Sun02] Sun Microsystems Inc., Fundamentals of JFC/Swing: Part II, http//:developer.java.sun/developer/online Training/GUI/Swing2/index.htm (August 2002).

[Sun03] Sun Microsystems, Java API. http://java.sun.com/api/ (September 2003).

[TG02] Takatsuka M., Gahegan M.: GeoVISTA Studio: A codeless visual programming environment for geoscientific data analysis and visualization. *Computers and Geosciences 28*, 10(2002), 1131–1144.

[Tor52] Torgerson W. S.: Multidimensional Scaling: I. Theory and Method. *Psychometrika 17*, (1952), 401–419.

[TSDS96] Tweedie L., Spence R., Dawkes H., Su H.: Externalising Abstract Mathematical Models. *Proceedings of ACM Conference on Human Factors in Computing Systems* (1996), 406–412.

[UFK*89] Upson. C., Faulhaber T. Jr., Kamens D., Laidlaw D., Schlegel D., Vroom J., Gurwitz R., van Dam A.: The Application Visualization System: A Computational Environment for Scientific Visualization. *IEEE Computer Graphics and Applications* (July 1989), 30–42.

[vRij79] van Rijsbergen C.J.: *Information Retrieval*. 2nd edition, Butterworths, 1979.

[WAM01] Weber M., Alexa M., Mueller W.: Visualizing Time Series on Spirals. *Proceedings of InfoVis'01, IEEE Symposium on Information Visualization* (2001), 7–14.

[Wis99] Wise J. A.: The Ecological Approach to Text Visualization. *Journal of the American Society for Information Science, Special Issue on Integrating Multiple Overlapping Metadata Standards 50*, 13 (November 1999), 1224–1233.

[WM04] Williams M., Munzner T.: Steerable, Progressive Multidimensional Scaling. *Proceedings of InfoVis'04, IEEE Symposium on Information Visualization* (2004), 57–64.

[WTP*95] Wise J., Thomas J., Pennock K., Lantrip D., Pottier M., Schur A., Crow V.: Visualizing the Non-Visual: Spatial Analysis and Interaction with Information from Text Documents. *Proceedings of InfoVis'95, IEEE Symposium on Information Visualization* (1995). 51–58.

[WYM97] Wang W., Yang J., Muntz R.: STING: A Statistical Information Grid Approach to Spatial Data Mining. *Proceedings of 23rd International Conference on Very Large Data Bases* (1997), 186–195.

[XOD02] Xu Y., Olman V. and Xu D.: Minimum Spanning Trees for Gene Expression Data Clustering. *Bioinformatics 18*, (2002), 536–545.

[YH38] Young G., Householder A. S.: Discussion of a Set of Points in Terms of their Mutual Distances. *Psychometrika 3*, 1 (1938), 19–22.

[YPWR03] Yang J., Peng W., Ward M. O., Rundensteiner E. A.: Interactive Hierarchical Dimension Ordering, Spacing and Filtering for Exploration of High Dimensional Datasets. *Proceedings of InfoVis'03, IEEE Symposium on Information Visualization* (2003), 105–112.

[YS93] Young D., Shneiderman B.: A Graphical Filter/Flow Model for Boolean Queries. *Journal of the American Society for Information Science 44*, 1(1993), 327–339.

[YWR02] Yang J., Ward M. O., Rundensteiner E. A.: InterRing: An Interactive Tool for Visually Navigating and Manipulating Hierarchical Structures. *Proceedings of InfoVis'02, IEEE Symposium on Information Visualization* (2002). 77–84.

[Zah71] Zahn C. T.: Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. *IEEE Transaction of Computers 20*, 1 (1971), 68–86.