A Visual Workspace for Constructing Hybrid MDS Algorithms and Coordinating Multiple Views

Greg Ross[*] Matthew Chalmers[†]

Department of Computing Science,
University of Glasgow,
Glasgow,
United Kingdom

Abstract

Data can be distinguished according to volume, variable types and distribution, and each of these characteristics imposes constraints upon the choice of applicable algorithms for their visualisation. This has led to an abundance of often disparate algorithmic techniques. Previous work has shown that a hybrid algorithmic approach can be successful in addressing the impact of data volume on the feasibility of multidimensional scaling (MDS). This paper presents a system and framework in which a user can easily explore algorithms as well as their hybrid conjunctions and the data flowing through them. Visual programming and a novel algorithmic architecture let the user semi–automatically define data flows and the co-ordination of multiple views of algorithmic and visualisation components. We propose that our approach has two main benefits: significant improvements in run times of MDS algorithms can be achieved, and intermediate views of the data and the visualisation program structure can provide greater insight and control over the visualisation process.

CR Categories: I.5.3 [Pattern recognition]: Clustering – Algorithms; E.1 [Data Structures]: Graphs and networks; D.1.7 [Programming Techniques]: Visual Programming; I.3.6 [Computer Graphics]: Methodology and Techniques – Interaction techniques;

Keywords: Data-flow, visual programming, multidimensional scaling, multiple views, hybrid algorithms, complexity

1 Introduction

There is a multitude of algorithms available for clustering and laying out abstract data. The different algorithmic approaches seem to be tailored to specific types of data. Some algorithms perform well with data sets of low cardinality and dimensionality, such as the basic spring model [Eades 1984]. Other algorithms work best with high cardinality data, an example of which is the *self–organising map* or SOM [Kohonen et al. 2000]. In training, a substantial training set allows the SOM to reveal complex non-linear structure in a very large body of data.

Other features of the data set also affect the applicability of algorithms, such as data distribution. For example, K-means clustering [MacQueen 1967] is most effective when the data is distributed in spherical Gaussian clusters [Bradley and Fayyad 1998].

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 [Morrison et al. 2002]. 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.

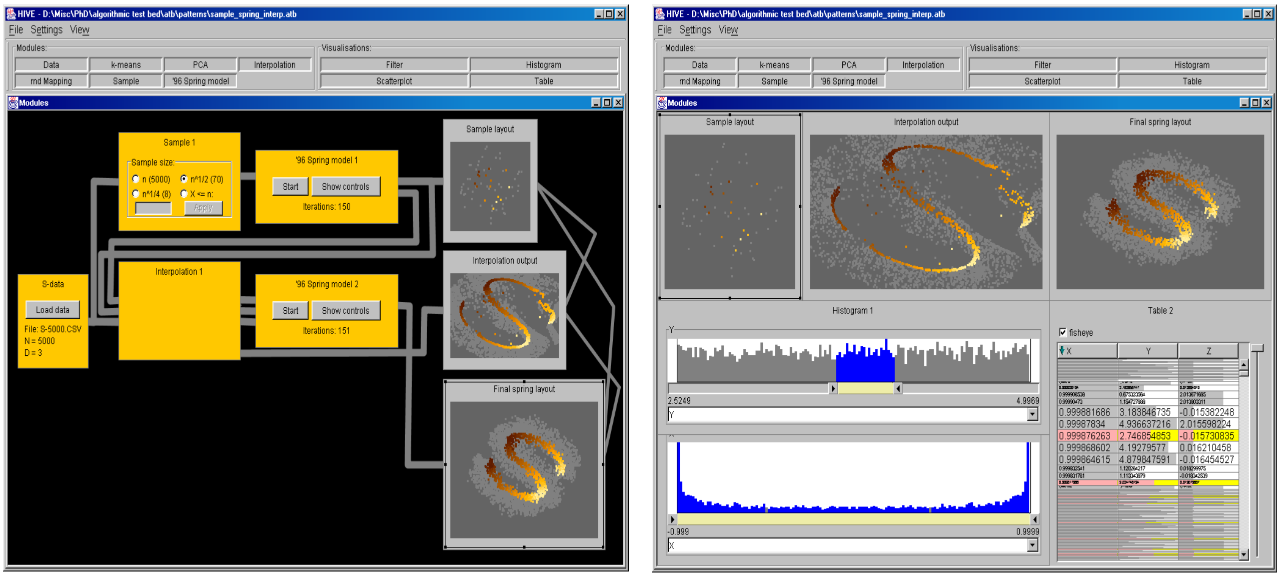
We present an implemented system and framework called HIVE (Hybrid Information Visualisation Environment) that utilises direct manipulation to allow users to interactively create and explore hybrid MDS algorithms. Figure 1 shows screen-shots of the system. Visual programming and a novel algorithmic architecture are proposed as a means to let the user semi–automatically co-ordinate multiple views and interactively steer data flows.

This paper expands on a shorter version [Ross and Chalmers 2003] in which we provided an account of earlier work on HIVE. Within the following pages we shall give a more detailed account of HIVE's implementation of the data-flow model, visual programming techniques and system architecture and also discuss recent developments such as dynamic run-time loading of algorithmic and visualisation components (Section 4.1.1).

The paper has seven sections. Section two describes related work including multidimensional scaling, the data-flow model and visual programming. Section three illustrates the hybrid algorithmic framework, upon which the system is built. Section four describes the HIVE architecture and implementation. Early experience of using HIVE is discussed in section five. Finally, sections six and seven present future work and conclusions respectively.

----

[*] e-mail: gr@dcs.gla.ac.uk
[†] e-mail: matthew@dcs.gla.ac.uk



X	Y	Z
0.999906538	0.875323584	2.013671685
0.99990473	1.154727888	2.013803011
0.999881686	3.183846735	-0.015382248
0.99987834	4.936637216	2.015598224
0.999876263	2.746854853	-0.015730835
0.999868602	4.19279577	0.016210458
0.999864615	4.879847591	-0.016454527
0.999832541	1.120264217	0.018299975
0.999801761	1.113043879	-0.018042509

Figure 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√N)* hybrid MDS algorithm of [Morrison et al. 2002]. 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 consists of 5000 points sampled from a 3D 'S' shaped distribution.

2 Related work

The HIVE system permits users to easily create and experiment with hybrid algorithms for generating visualisations of their data. This process is a visual one in that algorithms and visualisations are represented by visual components that afford direct manipulation. The following sub-sections describe topics in the literature that have influenced HIVE's development.

2.1 Multidimensional scaling

In the application of MDS we are primarily concerned with a lower dimensional representation of multivariate proximity data. Such data are composed of elements or observations that have three or more variables (multidimensional) and where the similarity between one datum and any other can be quantified. The appeal in reducing dimensionality is simple: to cater for visualisation. For example, scatterplots are extremely useful graphical tools, but they are restricted to a 2D spatial representation – they only have two real axes.

MDS has a relatively long history and there are many techniques. In [Buja et al. 1998], two categorisations of MDS methods are devised that are based upon the underlying mechanics of early work in [Torgerson 1952], [Shepard 1962] and [Kruskal 1964]. Torgerson's work is derived from the Eckart-Young SVD approximation theorem [Eckart and Young 1936]. We feel that the abundance of MDS techniques justifies our system, HIVE, as a visual workspace in which it is possible to implement, combine and compare them.

2.2 Visual programming

At around the time when scientific visualisation was being established, the concept of *visual programming* was also becoming prominent [Haeberli 1988; Upson et al. 1989]. Conventional programming languages, whether high level or low level, tend to be built around a vocabulary where the 'words' consist of primitives (characters). Visual programming languages are at a higher level of abstraction than conventional languages. Haeberli [1988] states that a visual programming environment is any system that has adopted a graphical 2D notation for the creation of programs. The visual primitives 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 greater freedom for externalising the plans and thoughts of the user [Hendry and Harper 1999].

Using visual programming for constructing InfoVis algorithms reinforces our commitment to and interest in graphical interaction in computing. With regard to the means-end relationship, the *means* are a visual process and the *end* result is a tool that produces the visual information originally sought after—visualisations are useful for producing other visualisations.

2.3 Data-flow model

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, *"Instead of the user driving an application, the user is often driven and constrained by the application."*

Visual programming addressed a number of these problems, moving away from these monolithic and static applications and providing integrated environments where a user without programming expertise could customise his or her applications. Visual programming in the application design cycle takes the form of a data–flow architecture. In this architecture, users are presented with a library of modules—application components—with 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. This quick and easy process meant that scientists and engineers could concentrate on the problems being studied instead of dealing with the overhead of re-coding and configuring monolithic applications.

2.4 Multiple-view co-ordination

Multiple view co–ordination allows two or more related views of data to run concurrently, with views evolving as data flows into them from some common ancestor in the data flow graph, or as the user interacts with one of them. A well–known example of this is brushing and linking [Becker and Cleveland 1987]. 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 lets the user focus on specific parts of the data set, and see them within the context of other views.

In evaluating their snap-together visualisation system, North and Shneiderman have found that this enhances user-performance in data analysis tasks [North and Shneiderman 2000]. Co-ordination of activity across multiple views gives the user greater control over the visual representations of the data. This ultimately nurtures discovery. In [Buja and Swayne 1996] it is described as linking *"…a graphical query to a graphical response"*, and in [Eick and Wills 1995] it is stated that it gives users the impression that they are *touching* the data.

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 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 [Morrison et al. 2003], 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 1 (left) uses a spring model of a sample of the full data set, to gain an initial small-scale 2D layout. In the left frame of Figure 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 third and final scatterplot, shown in to the right of the frame. 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 3D set. If we then use brushing to select a range of rows in the table or a region in a histogram, we highlight the corresponding points in the scatterplots and reveal more of the structure of the data. An extension of the work presented in [Ross and Chalmers 2003] allows for the neatly tiled layout of visualisation components, as in the right hand frame of the figure.

3 Hybrid algorithmic architecture

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) [Upson et al. 1989] and North and Shneiderman's snap-together system [North and Shneiderman 2000] 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.

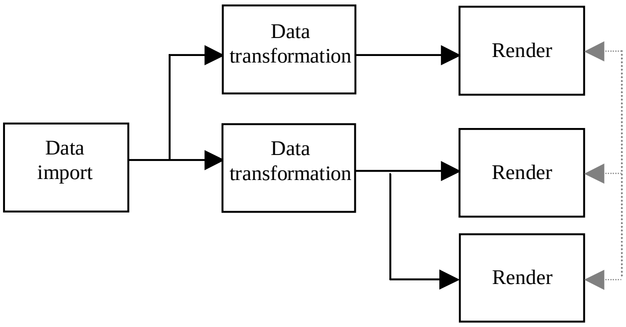


Figure 2. Data-flow and view co-ordination combined. Solid arrows represent data-flow between visual modules and dashed arrows depict co-ordination links between multiple views in HIVE.

HIVE borrows from the data-flow model of AVS to be flexible in creating efficient algorithms for the visualisations, effectively opening the algorithmic 'black box' and allowing the user to interactively steer the flow of data. 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 (see Figure 2). 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. As exhibited by the left hand frame of Figure 1, the modules used in only one hybrid algorithm can potentially use up much of the display space making it difficult to run more than one algorithm concurrently. 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-flow 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. We suggest that these choices can be made incrementally, so that the user employs intermediate representations as they work with and explore their data. We also suggest 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 us an incremental and combinatorial approach to the creation of efficient and informative hybrid visualisations.

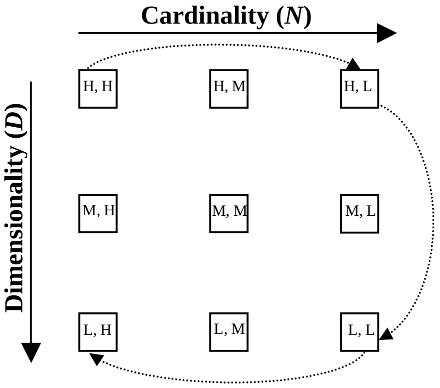


Figure 3. Data input to components in a hybrid algorithmic architecture can be categorised by the ranges of dimensionality and cardinality they are best suited for—high, medium or low. Each component transforms the data, effectively moving across the 3x3 grid. Our hybrid layout algorithm produces a low-dimensional 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*.

Our work has focused on data set cardinality, *N*, and the dimensionality or number of variables associated with each object: *D*. We roughly categorise *D* and *N* using an ordinal range (high, medium and low), and then we can categorise an algorithmic component with values of *D* and *N* for 'good' inputs and for the component's outputs, effectively stating our opinion that the component is best suited to such combinations of *D* and *N*. For example, we consider that the input to K-means clustering should be medium to high in *D* and *N*, whereas a canonical $O(N^2)$ spring model algorithm can only handle low *N* and low to medium *D*.

As shown in Figure 3, the choice of components and how they are connected allows one to solve familiar problems in new ways. The hybrid algorithm of [Morrison et al. 2003] 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.

It should be noted that in the previous example the representative goal state of the data is equivalent to *(L, H)* in figure 3 – a low-dimensional representation of each and every datum. However, this might not always be the best representation. In cases where there are a large number of elements in a data set it might be better to aggregate groups of objects to avoid occlusion and save screen space. For example, it might be wise to plot only cluster centroids in a scatterplot and have another view updated with the cluster members each time a centroid is selected. This representation could be equivalent to the *(L, M)* state in Figure 3.

Tentative default values for these ordinal categories of data are as follows. We derived these values from our own experience of constructing hybrid algorithms, however, HIVE allows the user to tailor them:

Low $D < 3$

$3 <=$ Moderate $D <= 100$

High $D > 100$

Low $N < 1000$

$1000 <=$ Moderate $N <= 25000$

High $N > 25000$

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, Chalmers' 1996 spring model [Chalmers 1996], radial interpolation [Morrison et al. 2002], K-means [MacQueen 1967], neural PCA [Oja 1982], stochastic sampling, scatterplot, histogram and fisheye table. 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 3. Examples are discussed in Section 5, following the next section's discussion of HIVE's internal structure.

4. Implementation

The software has been implemented in Java SDK 1.4. The system architecture, see Figure 4, has been designed to let users compose visualisation tools using modular components for importing data, algorithmic processing and graphical rendering. 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. A hybrid algorithm generator allows HIVE to semi-automatically load and connect algorithmic components.

4.1 Graph manager

The graph manager allows the user to incrementally create executable networks of components. It employs a scripting/composition model [Nierstrasz et al. 1991] to impose constraints upon which modules can be connected and through which 'ports', depending upon factors such as the categorisation of data type mentioned in Section 3, as well as graph structure and port polarity (input only, output only, two-way). A user can manually connect together components, but be warned of potentially unsuitable or inefficient connections. Another mode offers an automatically generated default path through the grid of Figure 3, instantiating components based on the system's classification of the input data set.

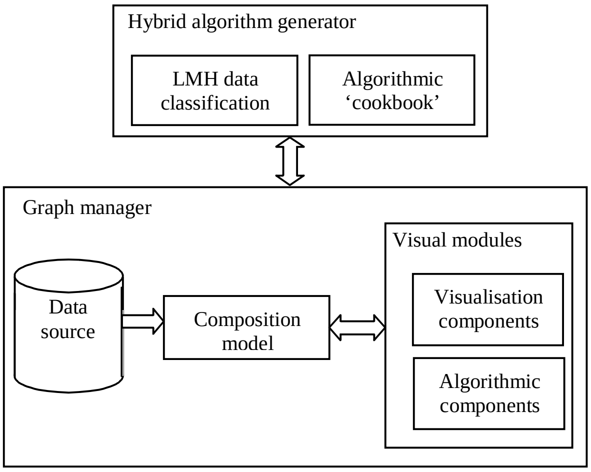


Figure 4. The system architecture of HIVE.

4.1.1 Visual modules

The graph manager defines three types of components to support the construction of hybrid visualisations. These are (1) data source components to allow the import of disparate data sets and perform 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 is 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, e.g. brushing.

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/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.

In another extension of the work described in [Ross and Chalmers 2003], the Java Reflection API [http://java.sun.com/api/] 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 of Glasgow University, 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.

4.1.2 Ports

Visual components 'listen' to each other by way of their ports. When a programmer writes a component, he or she must declare the ports that are necessary for the functioning and communication of the component. Ports operate by extending the Java 'Observable' class and implementing the 'Observer' interface [http://java.sun.com/api/], so that when a link is made between two components, the ports at each end of the link register with each other. This simple approach means that a component can send a message to another connected component by sending data through one of its (observable) output ports to the (observer) input port of the other component.

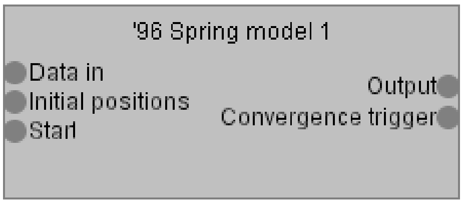


Figure 5. When HIVE is in link mode, all Swing components are hidden while port representations are rendered.

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 2D 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 module – rather like control constructs in a conventional programming language. Selection ports pass integer arrays of selected datum indices between visualisation components.

4.1.3 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 these 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 as follows:

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 only be linked to one output port

fan-out – one output port can be linked to many input ports

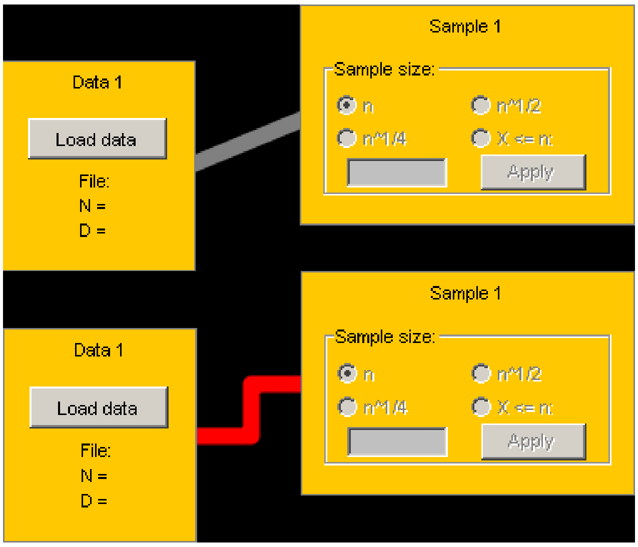
data-structure compatibility – data-in and data-out ports can only be connected when they are declared to handle the same data structure.

data-variable compatibility – data-in and data-out ports can only be connected when they are declared to handle the same variable types.

Here we provide more detail that was not presented in [Ross and Chalmers 2003]. 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 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 off of the edges of modules, it was felt that this would complicate the placement of port labels and 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 could be made from the selected port. If so, the 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 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.



Sample size:
● n ○ n^1/2
○ n^1/4 ○ X <= n:

Sample size:
● n ○ n^1/2
○ n^1/4 ○ X <= n:

Figure 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 the user has selected and bent it.

4.2 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 2D 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 3. 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 3, and are inserted between the two components that the user had originally connected, thus restoring adherence to the data-structure compatibility rule described above.

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

This is an area of the system that will benefit from aggregation. Currently when a hybrid algorithm is loaded, all of the constituent modules and their links are displayed. This might be confusing for a target user who is interested only in the data and resultant visualisations, not in the underlying algorithm. This area is

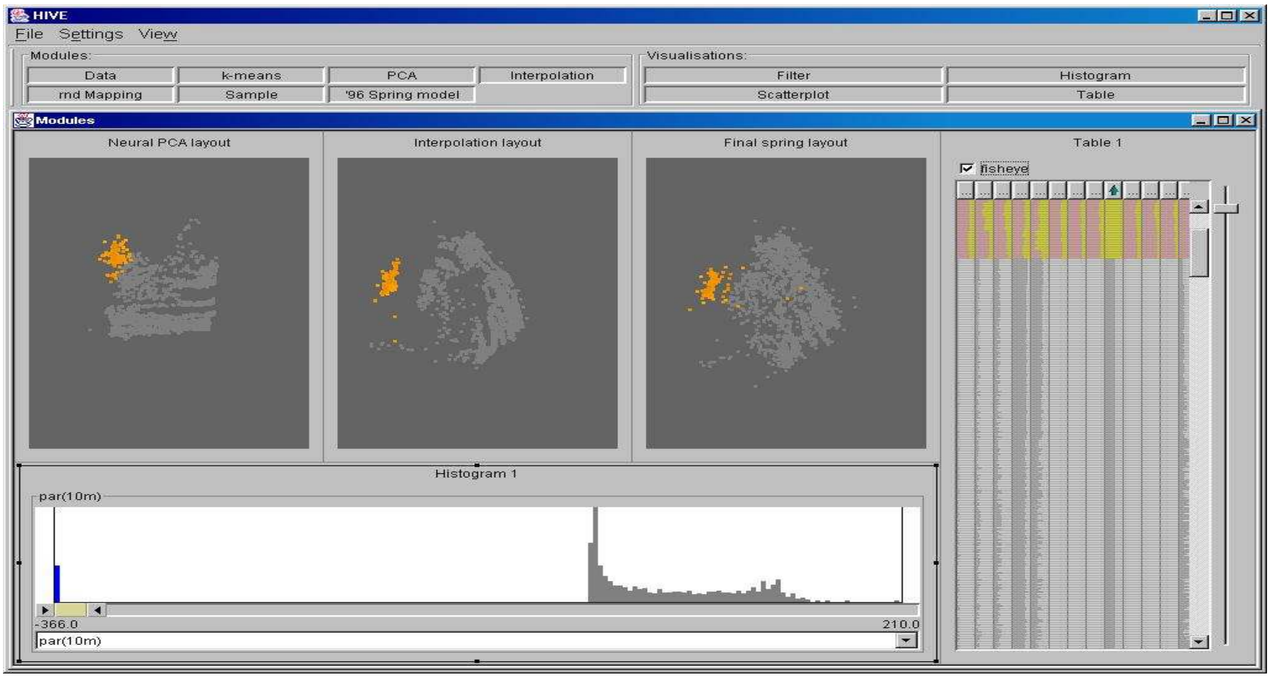


Figure 7. The leftmost scatterplot shows the output of neural PCA. The middle scatterplot shows the data after interpolation around the K-means centroids while the right scatterplot illustrates the output of the final spring model component. The highlighted cluster is a small subset of erroneous PAR 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.

currently being investigated. We are experimenting with the Piccolo toolkit for creating Zoomable User Interfaces (ZUIs) [http://www.cs.umd.edu/hcil/jazz/]. Piccolo utilises the notion of smooth semantic and logical zooming based on work in [Perlin and Fox 1993] and [Bederson and Hollan 1994]. It provides a library of classes and an API for integration with Java applications to make particular elements of the target application's user interface zoomable.

Zooming is of interest in our application of module aggregation because although we wish to condense the data flow graphs, we still require details of constituent modules to be available on demand. If several algorithmic modules are aggregated into one module, this module could consist of a zoomable canvas – a view - in which the aggregated sub-graph remains visible, albeit at a much smaller scale and level of detail. In preserving the spatial relationships of the aggregated modules, the user might find it easier to zoom into a particular module and thus gain access to any parameter controls contained. Or, during the process of aggregation a specific module, perhaps containing controls that govern the start and stop conditions of the whole algorithm, could be selected and be displayed by default at the highest zoom level. It is envisaged that this would be most useful when HIVE automatically generates an algorithm. A module containing critical controls would be foremost in the compound surrogate and be immediately available to the user. The user would still be able to zoom out of the default module and explore other components in the hybrid algorithm.

The above also suggests that hybrid algorithms *per se* might span intermediate stages of the space in Figure 3, i.e. aggregated components could be interconnected to form hybrid-hybrid algorithms. However, these algorithmic nesting and aggregation issues are the focus of ongoing development and shall be subject to user evaluation.

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 is in the same LMH categories and same type of visualisation is requested.

5 Preliminary experience using HIVE

Early experience of the HIVE system was gained when exploring a data set gathered from an eScience project within the Equator Interdisciplinary Research Collaboration (www.equator.ac.uk). The eScience team has 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 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).

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 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 our case, the low-D representation provided by the scatter plot (and underlying hybrid algorithm) immediately caught out 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 3. 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 dimensions. 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 8 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 8.

This exercise demonstrated the fact that some algorithms can be more effective than others when employed in MDS. If PCA had been used alone, the anomalous data might have been overlooked, whereas the hybrid spring model made the cluster immediately apparent. Also, the value of the intermediate view after interpolation boosted the cluster's separation and made it more visible.

6 Ongoing and Future Work

Our ongoing work is focused on implementing further visual modules to be included in the cookbook of hybrid algorithms that will span the simple 3x3 space represented in Figure 3. Algorithms considered include SOMs [Kohonen et al. 2000] and Random Mapping [Kaski 1998]. We are also experimenting with new algorithmic components such as Morrison and Chalmers' $O(N^{5/4})$ hybrid algorithm [Morrison and Chalmers 2003]. These algorithms are being analysed with respect to the data types they can handle, their complexity in time and space, whether or not they produce visualisations as useful intermediate representations, and the order in which they should be applied in a hybrid conjunction. We will also investigate aggregation of visual modules, as described in Section 4.2, as a means of increasing abstraction and therefore simplifying visual programming. Given a larger 'palette' of components and a means of aggregation, we will then carry out user trials of the workspace and the framework concentrating on finding usability concerns (and opportunities). We shall strive to acquire test-participants from task domains in which HIVE would be useful, such as the environmental scientists studying the Antarctica data featured in Section 5. One of our aims for empirical evaluation is to find a small project team that would benefit from visualisations of multidimensional data and whose use of the tool in anger would provide useful feedback.

One boundary issue that could impact on the implementation and usage of the proposed HIVE framework relates to applicable data formats. There are several well-established standards for encoding and handling data including the hierarchical data format (HDF) and others such as the common data format (CDF). For the HIVE framework to be adopted as a feasible information visualisation workspace in a non-experimental setting, the formats of data that it should be capable of importing, modifying, and possibly exporting, should employ these standards.

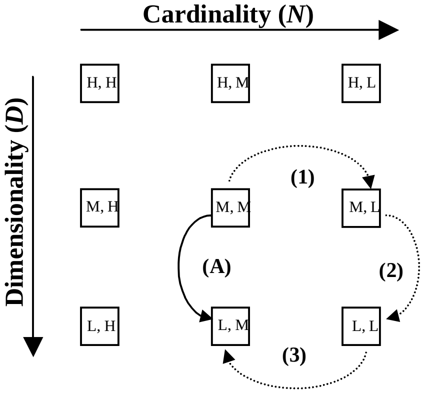


Figure 8. Dashed arrows represent the HIVE-generated hybrid algorithm spanning the space from M, M to L, M via (1) K-means (2) Chalmers' spring model and (3) Interpolation. The solid arrow represents the manually instantiated PCA module.

7 Conclusion

We have developed a framework based upon a novel hybrid algorithmic architecture and visual programming to provide users with interactive steering of data flows and multiple visualisations.

This framework has been embodied in our HIVE software (Hybrid Information Visualisation Environment). From early experience with our prototype, we suggest that the hybrid approach has two-fold benefits: significant improvements in run times of MDS algorithms can be achieved, and intermediate views of the data and the visualisation program structure can provide greater insight and control over the visualisation process. In the near future, we intend to carry out user trials to test this opinion, and to derive system improvements and new design ideas.

Work on HIVE has progressed since reporting its development in a companion paper [Ross and Chalmers 2003]. We have provided more detail on its implementation of ports, visual modules and composition model etc. Interface improvements such as in reflection and view co-ordination have been outlined and the direction our work is taking us, such as in algorithmic view aggregation, new algorithms and user trials, has been illustrated.

The abundance and variety of algorithms for visualising multidimensional data compound the need for assistance in exploring them. In visualising not only the data but also the computational devices for their transformation, we suggest that it is easier to experiment, compare and learn good approaches to gleaning more information from our data.

Acknowledgements

We thank Luc Girardin and Dominique Brodbeck (Macrofocus) for the 'S' data set used in Figure 1, Alistair Morrison (U. Glasgow) for earlier work on the hybrid algorithms in Figures 1 and 3, and Stefan Rennick Egglestone and Chris Greenhalgh (U. Nottingham) for the Antarctic data.

References

Bradley, P. S., Fayyad, U. M. 1998. Refining Initial Points for K-Means Clustering. *Proceedings of the Fifteenth International Conference on Machine Learning 1998*. 91-99.

Buja, A., Cook, D., Swayne, D. F. 1996. Interactive high-dimensional data visualization. *Journal of Computational and Graphical Statistics* 1996, 78-99.

Buja, A., Swayne, D. F. Littman, M. Dean, N. 1998. XGvis: Interactive data visualization with multidimensional scaling. Submitted to *Journal of Computational and Graphical Statistics*.

Becker, R., Cleveland, W. 1987. Brushing scatterplots. *Technometrics 29*, 2, 127-142.

Bederson, B. B., Hollan, J. D. 1994. Pad++: A zooming graphical interface for exploring alternate interface physics. *Proceedings of the ACM Symposium on User Interface and Software Technology (UIST'94)*. 17-26.

Chalmers, M. 1996. A Linear Iteration Time Layout Algorithm for Visualising High-Dimensional Data. *Proceedings of IEEE Visualization* 1996, San Francisco, 127-132.

Eades, P. A. 1984. A heuristic for graph drawing. *Congressus Numerantium 42*.

Eckart, C., Young, G. 1936. The approximation of one matrix by another of lower rank. *Psychometrika*, 1, 3, 211-218.

Eick, S. G., Wills G. J. 1995. High Interaction Graphics. *European Journal of Operational Research 84*, 445-459.

Haeberli, P. E., 1988. ConMan: a visual programming language or interactive graphics. *Computer Graphics*, 22, 4, 103-111.

Hendry, D.G., Harper, D. J., 1999. An informal information--seeking environment. *Journal of the American Society for Information Science*, 48, 11, 1036-1048.

Kaski, S. 1998. Dimensionality reduction by random mapping: Fast similarity computation for clustering. *Proceedings International Joint Conference on Neural Networks 1*, 413-418.

Kruskal, J. 1964, Multidimensional scaling by optimising goodness of fit to a non-metric hypothesis. *Psychometrika*, 29, 1-27.

Kohonen, T., Kaski, S., Lagus, K., Salojrvi, J., Paatero, V., Saarela, A. 2000. Self Organization of a massive document collection. *IEEE Transaction Neural Networks*, 11, 3, 574-585.

MacQueen, J., 1967. Some methods for classification and analysis of multivariate observations. *Proceedings of 5th Berkeley Symposium*, 281-297.

Morrison, A., Chalmers, M. 2003. Improving Hybrid MDS with Pivot-Based Searching. *To appear in Proceedings of the IEEE Symposium on Information Visualization 2003*.

Morrison, A., Ross, G., Chalmers, M. 2002. A hybrid layout algorithm for sub-quadratic multidimensional scaling. *Proceedings of the IEEE Symposium on Information Visualization*. 152-158.

Morrison, A., Ross, G., Chalmers, M. 2003. Fast Multidimensional Scaling through Sampling, Springs and Interpolation. *Information Visualization 2*, 1. 68-77.

Nierstrasz, O., Tsichritzis D., Vicki de Mey, Stadelmann, M. 1991. Objects + Scripts = Applications. *Proceedings of ESPRIT Conference*. Kluwer Academic Publishers, 534-552

North, C., Shneiderman, B. 2000. Snap-together visualization: can users construct and operate coordinated visualizations? *International Journal of Human-Computer Studies 53*, 715-739.

Oja, E., 1982. A Simplified Neuron Model as a Principal Component Analyzer, *Journal of Mathematical Biology 15*, 267--273.

Perlin, K., Fox, D. 1993. Pad – An alternative approach to the computer interface. *Proceedings of the ACM SIGGRAPH*. 57-64.

Ross, G., Chalmers, M. 2003. A visual workspace for hybrid multidimensional scaling algorithms. *To appear in Proceedings of the IEEE Symposium on Information Visualization 2003.*

Shepard, R. N. 1962. The analysis of proximities: multidimensional scaling with an unknown distance function. *Psychometrika*, 27, 2, 125-140.

Torgerson, W. S. 1952. Multidimensional scaling: I. Theory and method. *Psychometrika*, 17, 401-419

Upson, C., Faulhaber Jr, T., Kamens, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R., Van Dam, A., 1989. The application visualization system: a computational environment for scientific visualization. *IEEE Computer Graphics and Applications*. 30-42.