Fast Multidimensional Scaling through Sampling, Springs and Interpolation

Alistair Morrison, Greg Ross and Matthew Chalmers
Department of Computing Science, University of Glasgow
http://www.dcs.gla.ac.uk
{morrisaj, gr, matthew}@dcs.gla.ac.uk

Abstract

The term 'proximity data' refers to data sets within which it is possible to assess the similarity of pairs of objects. Multidimensional scaling (MDS) is applied to such data and attempts to map high-dimensional objects onto low-dimensional space through the preservation of these similarity relationships. Standard MDS techniques have in the past suffered from high computational complexity and, as such, could not feasibly be applied to data sets over a few thousand objects in size. Through a novel hybrid approach based upon stochastic sampling, interpolation and spring models, we have designed an algorithm running in O(N N). Using Chalmers' 1996 $O(N^2)$ spring model as a benchmark for the evaluation of our technique, we compare layout quality and run times using data sets of synthetic and real data. Our algorithm executes significantly faster than Chalmers' 1996 algorithm, whilst producing superior layouts. In reducing complexity and run time, we allow the visualisation of data sets of previously infeasible size. Our results indicate that our method is a solid foundation for interactive and visual exploration of data.

1. Introduction

The visualisation of multivariate abstract data is a fundamental task in many fields. From bioinformatics to the financial sector, there is a great deal of interest in data that have no inherent mapping to a 2D or 3D space. Graphical means of conveying such information are subsequently relied upon to provide insight into patterns and relationships.

A critical requirement of the production of such a representation is the means to generate layouts of the multivariate data in a lower dimensional space. The created visualisation should preserve relationships existing within the data and should be comprehensible enough to allow the user to perceive such patterns.

Multidimensional scaling (MDS) is one means of mapping a data set onto a smaller number of dimensions, so that it may be visualised in a more manageable form. The resulting presentation does not contain the q-dimensional Cartesian space directly, but rather a p-dimensional embedding (where $p < q$) of N objects where high-dimensional inter-object relationships are approximated in the low-dimensional space. Our work focuses on creating 2-dimensional representations.

Although effective in generating layouts, standard MDS operates by means of eigenvector analysis of an N x N matrix, producing a layout based on a linear combination of dimensions. This results in an $O(N^3)$ procedure for producing layouts. As well as this cubic complexity, it should be noted that the computation would need to be performed again in its entirety if the data set were even slightly altered [7]. Iterative techniques overcome these difficulties. It is possible to calculate a measure of the quality of a layout: how well the visual representation conveys relationships present in the initial data. This can be treated as a loss or error function, which is to be iteratively minimised to gain an optimal arrangement. In 1996, Chalmers [6] presented an iterative MDS algorithm capable of producing a representative layout in time proportional to $O(N^2)$. Additionally, by removing the necessity of creating a layout based on a linear combination of dimensions, the system is freer to find an optimum layout.

We describe work on the combination of several iterative techniques that generates a layout in sub-quadratic time. An example of such a layout, and our tool for interacting with it, is shown in Figure 1 (below). This paper will not focus on the tool in terms of the interaction with the data, but on new layout algorithms. The following sections describe MDS in more detail before discussing spring models—the general approach to iterative layout algorithms that we have been following. A later section outlines the model we have been working on, and then we report the results of experiments comparing the new technique with Chalmers' 1996 algorithm. As we reflect on these results, we find a number of avenues of future research open to us. We outline some of these before concluding the paper.

2. Multidimensional scaling

If it is possible to quantify the similarity between individual elements of a data set, the set can be referred to as *proximity data*. Proximity data is abundant in many fields such as the social sciences, information retrieval, geology and archaeology; any source, in fact, that produces data that can be analysed so as to determine similarity between one datum and any other.

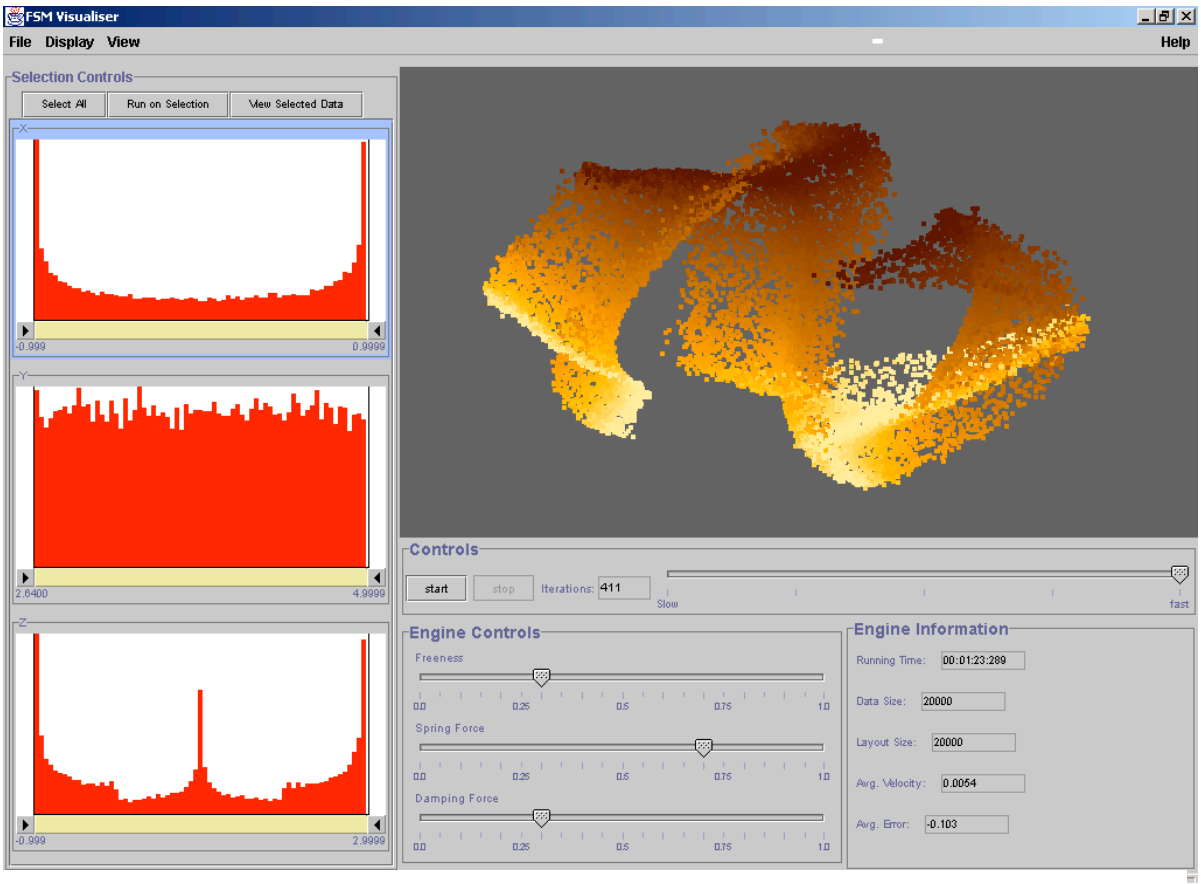

Figure 1. **An example layout close to completion in our *FSMvis* visualisation tool is shown top right. Other components of the tool offer control over spring model parameters (bottom right) and histograms (left) of individual dimensions or attributes allow filtering and selection. The layout is of 20000 points sampled from a 3D 'S' shape: one of the test sets described in Section 5. The histograms, top to bottom, thus represent the input X, Y and Z coordinates of the S shape. Points in the layout are coloured according to their X–coordinates in the original 3D shape. Although the late stages of processing may resolve some of the folds and distortions, the set was chosen because it is inherently impossible to lay out perfectly in 2D.**

As an example, consider an information retrieval system such as a computerised library catalogue. A user looking for books on a specific topic might submit a query to the system comprised of keywords thought to characterise the subject of the books sought. By determining the similarity between the query and book descriptions contained in the catalogue, the most relevant (similar) book descriptions can be revealed.

Extending the above example, suppose that the library catalogue presents a graphical display (a visualisation) of all the books in the library in the form of a scatter plot. Each point on the plot would represent a book, and the proximity of each pair of points would correspond to the books' relative similarity. In this view, clusters might be easily recognised and interpreted as groups of books that share a theme. On submission of a query, the user would be shown a point on the plot representing his or her request. The nearest points or clusters to this point, those that represent the books most similar to the query, would be immediately apparent.

The creation of such a scatter plot, however, is not a trivial task. A simple scatterplot has only two axes on which to plot each data element, yet these data may be composed of a large number of dimensions. In our example, each distinct keyword describing a book could be considered as a dimension. It would be impossible to directly map each of these to an axis of the plot, as there are simply insufficient axes available for a one-to-one relationship.

In statistical analysis, data consisting of more than three variables are termed *multivariate* or *hypervariate* data 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 [1, 5, 6, 16, 17, 18, 21, 24, 25]. As illustrated by the previous example, it is not always possible to directly map multidimensional objects onto a set of visually perceptible orthogonal axes. Methods for overcoming this have included using a matrix of scatter plots for pair-wise comparison of dimensions [9], or the

visualisation and exploration of nested sets of dimensions [11]. Both of these techniques suffer from a lack of general overview of a data set, and both become less feasible with data sets of higher dimensionality. The scatterplot matrix approach in particular uses an amount of screen space quadratic with respect to the number of dimensions. The MDS approach illustrated above is therefore extremely useful as a tool for the analysis of such multidimensional proximity data. Through calculations based on high-dimensional relationships, the same 2D scatterplot may be used as a visualisation tool, regardless of data dimensionality.

The purpose of MDS is to produce visualisations for the exploration of data. Such visualisations allow users to detect interesting latent structure such as Gestalt properties within data, to make comparisons, or to explicitly query the data [4, 21]. An incomplete list of application areas for MDS presented in [5] includes experimental psychology, where human subjects' comparative judgement of stimuli is studied; archaeology, for determining the similarity of digging sites; graph drawing; and chemistry, for modelling molecular conformation.

As stated, MDS is applied to proximity data. There must therefore exist some means of assessing the similarity of a pair of objects. The typical approach in MDS is to measure the distance between objects within high-dimensional space, and therefore the relationship modelled would more accurately be described as *dissimilarity*. The most common distance measures come from the family of metrics known as the Minkowski (or Lebesgue) metrics [15], which operate on the vector representation of each object. The general form of this family of metrics is as follows:

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

where L is a parameter that takes on some value in the interval [1,]. The most commonly used instance of this metric is Euclidean distance, where L = 2. We have adopted Euclidean distance as a measure of distance in our work because it is widely used in measuring distances in lower dimensional space (such as our physical environment), and intuitively generalises to higher dimensional spaces. In order to use this Minkowski metric, we constrain or transform the source data to be continuous numerical vectors.

3. Spring models

Several types of MDS algorithm have been devised, with *metric distance* MDS, or the *spring model* being among the simplest. A general description of the definitive characteristics can be found in [5,7]. We have adopted the spring model for MDS instead of statistical dimensional reduction techniques such as principal component analysis (PCA) [see e.g. 7] for two reasons. Firstly, PCA derives the lower dimensional representation of the data via a linear combination of the original dimensions. This is an additional constraint on the system and renders it less free to converge to an optimal solution with regard to stress. For example, if the data has non-linear clusters, PCA will not do them justice in its representation [5]. A second benefit of spring model MDS over PCA arises from its iterative nature. Such a system permits the observation of the MDS process, allowing a user to abandon an obviously useless layout without the necessity to complete the entire computation. This obviously affords extra exploratory work and encourages more experimentation with different layout conditions. Also, as previously stated, a completed spring model could also accommodate a small subset of additional data by means of a few additional iterations, whereas PCA would require to be re-run in its entirety.

The name 'spring model' arises from work by Eades on a heuristic for graph drawing [10]. Eades described a technique for the aesthetic layout of general graphs through the physical analogy of a system of steel rings connected by springs. A graph, $G = (V, E)$, where V represents the vertices and E the edges, may be used to represent a mechanical system by replacing vertices by steel rings and edges by springs. Each relaxed spring length or 'rest distance' is set to be the ideal proximity of the two vertices; that is, their high-dimensional distance or dissimilarity. 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 equilibrium, and hopefully therefore to a state of minimal energy. The algorithm is iterative, with each iteration refining the layout of the graph.

This technique was found to produce good results with regard to aesthetics of symmetry and uniform edge lengths in graph drawing and has since been applied in MDS where the layout of high-dimensional objects in a low-dimensional space is the goal. Because the analogy of force is explicit, this type of algorithm is often described as being *force-directed.*

When applied to the low-dimensional embedding of a set of abstract data, this model can be considered as a combinatorial optimisation algorithm. The forces in the system are computed as being proportional to the difference between the high-dimensional (desired) distance and the low-dimensional (current) layout distance, and the system strives to minimise the discrepancy between these two. A loss function can be derived, commonly known as *stress*, which indicates the amount of energy in the system and is calculated through a measure of the sum-of-squared errors of inter-object distances. Stress may be defined as below [6], where d_{ij} denotes the desired, high dimensional distance between objects i and j, and g_{ij} denotes the low–dimensional or layout distance:

$$Stress = \frac{\sum_{i<j} \left(d_{ij} - g_{ij} \right)^2}{\sum_{i<j} g_{ij}^2}$$

Indirectly, the objective of running a spring model is to iteratively minimise this stress. The system maintains three properties for each object, namely *position, velocity* and *force*. At each iteration, a force calculation must be performed on each object. The magnitude of the force exacted on an object *i* by another object *j* at any time during the run will be proportional to:

$| highDimensionalDistance(i,j) - layoutDistance(i,j) |$

This calculation must be performed for $1 \leq j \leq N (j \neq i)$ in order to produce the overall force acting on *i*. Object *i*'s force is then used to update its velocity, which in turn is used in updating the object's position in the layout.

Note that $(N-1)$ force calculations must be performed in each iteration of the spring model for each of *N* objects. The number of iterations required to produce a stable layout is commonly proportional to the size of the data set, resulting in an algorithm that is $O(N^3)$ overall. The $N(N-1)$ pairwise interactions at each step are an obvious area for improvement. This is analogous to the well-known *N-body problem* in computational physics.

3.1 Chalmers' 1996 Algorithm

The technique presented by Chalmers in 1996 [6] employs caching and stochastic sampling to perform each iteration of a spring model in linear time, thus permitting the construction of a stable layout in $O(N^2)$ time overall.

This is achieved by reducing the number of force calculations performed for each object during an iteration. Two distinct sets are used for each object *i*. The first set *V* is stored as a list of 'neighbours' of *i*, i.e. the objects so far found to have lowest high–dimensional distance, and thus expected to be laid out nearby in 2D space. The second set, *S*, is reconstructed in each iteration, and contains a random selection of objects not already in the neighbour set. Random objects are selected and each is tested to determine whether it has a high–dimensional distance lower than one or more of the current neighbours. If this is the case, the new object is swapped in to the neighbour set. If not, the object is added to *S*. In this manner, the neighbour set becomes more representative of the most similar objects to *i* over successive iterations. Once both sets are constructed, forces are calculated only between object *i* and each of the members of the two sets.

The number of force calculations required during one iteration of the algorithm was therefore reduced from $N(N-1)$ to $N(Vmax + Smax)$ where $Vmax$ is the maximum size of set *V* and $Smax$ is the maximum size of *S*. As the two set sizes are bounded by constants, the computational cost of an iteration is linear with respect to *N*. Evaluation of this technique indicated that layout quality is still good despite the reduction in force calculations. Indeed, even constant values as low as 5 and 10 for $Vmax$ and $Smax$ respectively yielded favourable results.

In terms of computational time, this is currently the best model using only springs, and is therefore the algorithm that we will use as the basis of comparison with new techniques. It should be noted however, that despite the improvements offered, such a model could not be practically used on data sets over a few thousand objects in size.

4. Hybrid methods of clustering and layout

One clustering algorithm may effectively tackle areas in which others are weaker. A number of researchers have explored combinations of algorithms with a view to maximising the benefits of different approaches while diminishing the impact of any shortcomings.

For example, Kohonen's self-organising feature map (SOM) [16] is an unsupervised learning algorithm applied to the classification of information. SOMs partition a data set into a grid that can be useful in clustering or visualisation applications, but can be quite time-consuming in construction. Su et al. [23] claim that the well-known iterative centroid–based divisive clustering algorithm, K-means [19], has a lower time complexity than a SOM, and therefore employ K–means to gather representative classes or clusters from the data set. These representative centroids are then organised into a discrete *N* by *N* (or more accurately a *k* by *k)* grid, and a SOM is used to fine-tune. Su et al. suggest that this variant SOM approach is much faster than the traditional on-line SOM.

In another example of a hybrid approach, Brodbeck and Girardin [3] used a SOM as the *initial* phase in the creation of a layout. Although the SOM was shown to be the computational bottleneck in the previous example, it does exhibit less complexity than the spring model. Consisting of a discrete grid of cells, SOMs cannot show as much topological structure or detail as a spring model layout, but are often quicker to make and scale to larger data sets.

It was on the basis of this comparison that Brodbeck and Girardin used a SOM to find representative clusters or neurons, and then used a spring model to lay out these neurons without the distortion imposed by the discrete grid of the SOM. In effect, this process produced a set of cluster centroids that were arranged in such a way as to preserve high-dimensional relationships. Although useful in itself, this layout was then used as the template for placement of the entire data set via an original interpolation algorithm.

The accuracy of the interpolation was largely determined by a set of constants used to govern the process, with higher values resulting in longer run-times but more accurate placements. Brodbeck and Girardin evaluated their model through the comparison of run time with the standard canonical $O(N^3)$ spring model. Generated layouts were also compared to those produced using the standard model, both by means of subjective interpretation and a measure of the preservation of inter-object distances (a stress value). Tests were conducted using a 5 dimensional data set of 3840 elements. It was stipulated that such a size had been selected in order to examine performance on as

large a set as possible while limiting the size to such that spring model execution would still be feasible.

Their experiments indicated that both models produced layouts of similar appearance and stress, and, although the results of the time trials were not extensively documented, it was stated that their technique offered an improvement in that it took in the order of hours rather than days.

4.1 K-means as pre-process

Some of our earlier work [20] developed Brodbeck and Girardin's notion of subset layout and interpolation, although we employed K-means clustering rather than a SOM as the initial stage. The main advantage of K-means is its reasonable space and time complexity [15]. The time complexity is *O(nkl),* where *l* is the maximum number of iterations, and the space complexity is *O(n + k)*. One drawback of the K-means algorithm is that it assumes that the clusters it is trying to find lie in a spherical Gaussian distribution [2]. This means that although the algorithm will always converge [19], it may only converge to a local minimum. This is compounded by the fact that clusters may vary in size and that K-means is very sensitive to the initial choice of centroids [15]. This initial decision can determine how well the algorithm converges in terms of local or global minima.

Our tests determined, however, that the K-means phase was taking a prohibitively long time. Although this hybrid model did run more quickly than Chalmers' 96 algorithm, the saving in time was not as substantial as we believed should be possible using an interpolation algorithm. The following section provides a detailed description of our improved hybrid algorithm.

5. A novel hybrid approach to MDS

This section outlines an original method of generating layouts of high-dimensional data in 2-dimensional space. We show that through the combination of sampling and spring techniques, layout construction is possible in sub-quadratic time.

This technique also builds upon the interpolation strategy of Brodbeck and Girardin described in the previous section. As an initial step, however, we take a simple *N* sample (*S)* of the data set, rather than running a SOM. A layout of the subset *S* is then made using Chalmers' spring model [6]. This model will run in *O(N. N)*, i.e. *O(N)*.

A choice of measures exists for determining when to halt spring model execution. The two main termination criteria we use are the difference in velocity in the system between iterations, and the difference in system stress. In practice, we terminate this first stage when the difference in velocity falls below a scalar threshold.

To complete the layout of the entire data set, we use a modified version of Brodbeck and Girardin's original interpolation strategy. This interpolation process is described below and illustrated in Figure 2.

For each object *i*

1. Find the object *x*, which is of least high-dimensional distance from *i* in the original subset *S*.
2. Define a circle round *x* of radius *r,* where *r* is proportional to the high–dimensional distance between the two objects.
3. By comparing differences between actual layout distances and desired distances, determine which quadrant of the circle is likely to be the most satisfactory for positioning of *i*.
4. Perform a binary search on this quadrant to determine *i*'s best location, i_c , and place *i* there.
5. Select a random sample *s* of the original subset (*S)* on which to base the following calculations.
6. Determine the aggregate force vector between *i* and the members of *s*.
7. Add the vector to *i*'s position.
8. Repeat steps 6 and 7 a constant number of times to refine the placement.

We have found that this strategy improves upon Brodbeck and Girardin's original model with respect to position placement. The quadrant comparison and binary search replace the original method of comparing a constant number of positions on the circumference. We also simplified the vector addition at step 7, as we found that the previous strategy of selecting the best position from a number of random locations along the vector was not reliably better than adding the unscaled force vector.

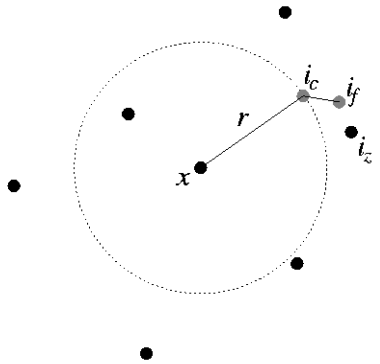

Figure 2. **The placement of the object *i* begins with finding the most similar member of the initial layout, *x,* and then finding the best position i_c on the circumference of the circle of radius *r* around *x*. The position is then refined by iteratively adding aggregate forces from a subset of *S*, moving the object through positions such as i_f until it reaches its final location i_z.**

We found that both these changes contributed to far more accurate object placement, resulting in more representative layouts. The extra time spent performing our version of the interpolation was negligible in comparison with the saving in post–interpolation spring model refinement.

As the sizes of all the random samples in Brodbeck and Girardin's original interpolation strategy were kept as constants, the interpolation could be achieved with a complexity linear with respect to N. This is also the case with our variant, although an extra routine is required (step 1 in the above outline). In Brodbeck and Girardin's method, the initial SOM stage partitions the data set into clusters and the spring model lays out cluster centroids. When interpolating an object i, therefore, they did not have to determine which of the original subset should be used as the basis of calculations (x in Figure 2).

We have no information as to which objects belong to which 'clusters', so an initial pre-processing stage is required. Each of the $(N- N)$ objects to be interpolated is compared to each of the N samples. A best match for all points is consequently calculated in $O(N N)$ time. This pre-processing stage is the dominant factor, making our layout $O(N N)$ overall.

As a final stage in our MDS technique, we run a constant number of iterations of Chalmers' 1996 spring model on the full data set to refine placement. Although our interpolation scheme is very accurate, it is based on the initial layout of the N sample. It may be the case that the sample was not perfectly representative of the full data set, or that the spring model has terminated within a local minimum.

During tests on synthetic data of a known structure, we would often see that a section of the expected shape was out of place; the layout looked rather like a jigsaw puzzle with one piece lying askew. The individual points had interpolated correctly around their 'parent' from the original subset, but this parent had been misplaced in the initial layout. We found that 10 of these full data set spring model iterations were sufficient to considerably lower stress, and to visually slot any errant sections into place.

As previously discussed, Chalmers' 1996 model is linear per iteration. As a constant number of these iterations are run, the complexity of this final phase is also linear with respect to N, and our algorithm remains $O(N N)$ overall.

6. Experimental results

In this section we offer a number of comparisons of our new layout algorithm with Chalmers' 1996 algorithm, currently the best spring model algorithm with respect to computational complexity. We evaluate and compare layout models based both on the subjective quality when explored in interactive use on–screen and on objective quantities such as stress.

Stress is calculated for these experiments as defined in section 3. It should be noted that this metric is used with caution, as stress itself is not necessarily a perfect indication of the perceived quality of the final clustering layout. While it may serve as a rule of thumb, two layouts may have comparable stress but the layouts themselves may be very different; lower stress does not necessarily mean a better, more interpretable layout in a particular context of work. The visualisation tool used to perform these experiments was written in Java SDK 2.1 version 1.3. Tests were run on a PC with an Intel Pentium 3 731MHz and 256MB RAM running Microsoft Windows 2000 Professional. This tool is available for download via the homepages of the authors.

Two distinct collections of data were selected for the experiments. The first collection was synthetically created by sampling points from a 3D structure - a band curving in an 'S' shape through three dimensions. Reconstructing this shape should be possible for a good layout algorithm, although, as may be seen from Figure 1, the 'S' structure is forced to fold in on itself in certain areas as it is impossible to exactly represent all the inter-object distances when one less dimension is available. By sampling at different frequencies, sets of 10 different cardinalities were created from this collection, from 5000 to 50000 elements. The second collection used was a data set of 13–dimensional financial data containing historical performance and volatility information on investment funds. Here 12 sets were used, this time from 2000 to 24000.

We decided to use a synthetic data set as part of our evaluation strategy so that we could compare generated layouts and layout processes easily. We were able to clearly see the well-recognised structure forming, and were able to subjectively measure the quality of layout produced.

Figures 3 and 4, below, compare our technique with Chalmers' 1996 algorithm in terms of layout time and stress. It can be seen that our hybrid approach is by far the quicker: up to three times faster on this data. It is also worthy of note that the time taken to run the hybrid algorithm appears to be increasing linearly, even for data sets of 50000 elements. This may seem unusual, as the $O(N N)$ 'parent finding' step is the most computationally complex phase of our method. In practice, however, for the size of data sets tested, the $O(N)$ interpolation phase is the most time–consuming step. As Figure 4 illustrates, stress is also lower for the hybrid model. This is consistent with what we observed from examining the resultant layouts. The interpolation phase resulted in more accurate positioning, and the layout looked to be more regularly spaced, resulting in a much smoother S-shape. The solo spring algorithm produced a less even structure, characterised by rough edges, tight clusters and gaps.

Similar reductions in computational time and stress over the spring algorithm time can be observed for the second data set (Figures 5 and 6), with the hybrid approach again achieving a lower stress in less time.

To illustrate the degree of improvement offered by our methods over standard MDS techniques, two experiments were performed where the full $O(N^3)$ spring model was run on sets of the 3D 'S' data of size 2000 and 5000 objects. The smaller of these data sets was laid out in 577 seconds (almost 10 minutes) compared to 9 seconds for our hybrid method. The data set of 5000 objects took 3642 seconds (over an hour) to converge, as compared to the 24 seconds average over

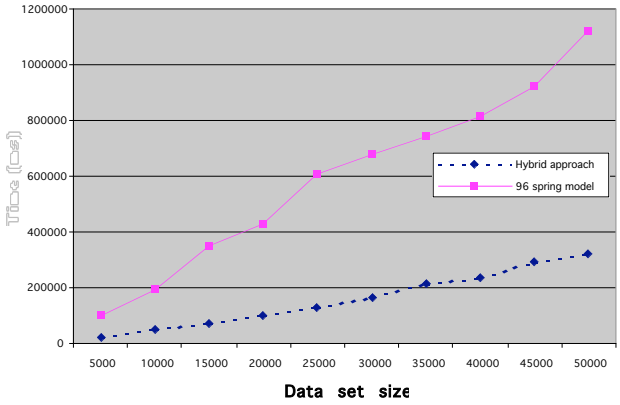

Figure 3. **Run time to completion for different sizes of 3D 'S' data.**

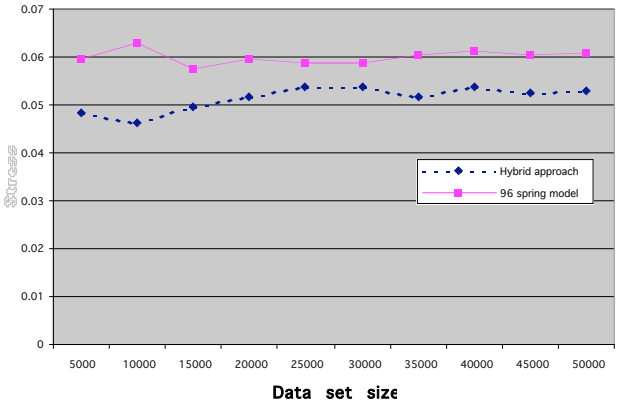

Figure 4. **Stress of completed layout over different sizes of 3D 'S' data.**

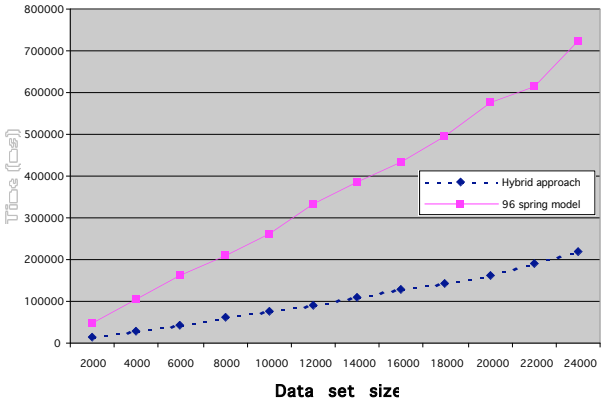

Figure 5. **Run time to completion for different sizes of 13D financial data.**

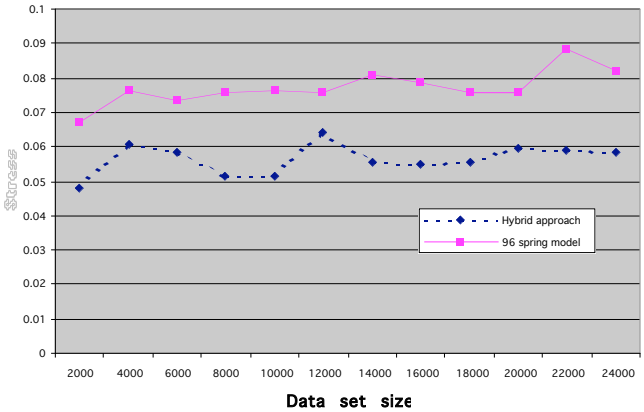

Figure 6. **Stress of completed layout over different sizes of 13D financial data.**

the 10 runs using our approach. It is also interesting to note that stress was much higher in the cubic time model (e.g. 0.2) compared to our interpolation model that finished with stresses of roughly 0.06. It seems as if the velocity threshold was reached before a good layout was made, perhaps because the higher number of springs made the model much 'stiffer' overall.

7. Future work

Our experiments have suggested a variety of possible areas of future work to us. In this section we outline a number of these and their possible benefits.

7.1 Hashing

In the hybrid model that we have presented, the bottleneck in terms of computational complexity is the assignment of remaining data points to a 'parent' sample. This was a precursor to interpolation using the layout of samples. This currently requires $O(N\sqrt{N})$ time (worst-case) because of a brute–force linear search for a parent. This is an example of a nearest-neighbour search, and it may be possible to employ a hashing function at this stage to reduce complexity. Several attempts have been made to use hashing functions to perform similarity searching in high dimensions. Indyk et al. proposed a technique of *locality-sensitive hashing* (LSH) [14] to aid the retrieval of a data element's *approximate* nearest neighbours. This approach is based upon the assumption that the computation required to determine the absolute nearest neighbour is often unnecessary if a good approximation will suffice and if such a value can be found at a fraction of the cost of the full search.

Using such a method would reduce lookup to sub–linear time, but a pre–processing phase is required to place all the n points into each of l hash tables, which would require nl operations. In our favour, this is being performed on the sample rather than the full data set, so $n = \sqrt{N}$. Also, it has been shown [13] that a constant number of hash tables (regardless of data size) can result in high probability of finding very close neighbours. We therefore would have an $O(\sqrt{N})$ pre-processing stage and a lookup technique bounded by l, a constant, resulting in an interpolation algorithm requiring $(l\sqrt{N}) + l(N–\sqrt{N})$ operations i.e. $O(N)$ overall.

This is an area that certainly seems worth exploring. If our results should indicate that a good approximation of a nearest neighbour could be found in sub-linear time, the process could possibly be applied to other areas. For example, it would perhaps be possible to

select an object of interest from an unordered space and be presented with similar elements from the data set, or the techniques could be included within the spring model domain to help identify neighbour sets. This could lead us to fundamentally rethink the spring model algorithm.

7.2 Pivots

This family of algorithms are also predominantly used in nearest neighbour searches and in indexing applications. The idea behind them is to select a number of points (pivots) in the dataset and store the distance from each of these points to every other point in the set. Then a discarding rule can be applied so that the number of distance calculations to find close objects to the query is reduced.

The idea has been described as follows [8]: given a point x in the data set and a pivot p, we can store the distance between these two points as $d(x, p)$. Now, given a query q, we can define the distance to the pivot $d(q, p)$. It is now possible to use the triangular inequality to discard the distance calculation from the query to a point where $|d(q, p) - d(x, p)| > r$. Here r is the predefined maximum distance from q to which an object may be considered close.

Again, this could be used to speed up the operation of building the neighbour sets used in the force calculations. Also, it can be used as a first stage of interpolation. This latter improvement has reduced the complexity of our algorithm to $O(N^{5/4})$, and we are beginning a series of experimental runs and evaluations.

7.3 Dynamically Resizing V+S

We stated earlier that it is possible to create good layouts using values of 5 and 10 respectively for the sizes of the neighbour and random sample sets: V and S. To minimise iteration time, it is obviously advisable to set these values as low as possible without compromising layout quality. We suggest that under certain conditions it may be wise to alter the size of the sets dynamically during program execution. For instance, if an analysis of stress values indicated little change over a certain number of iterations, it could be the case that either the layout has converged to its stable state, or that it has become stuck at a locally optimal layout. By increasing the size of the set of neighbours, each data point will receive greater force pulling it towards its rightful position. This will increase the probability of the layout breaking out of this state and moving towards an overall minimum.

7.4 Proximity Grid

In a recent paper [21], a grid structure is used to determine whether the topological layout of images is beneficial to browsing. The algorithms for creating the grid structure are proposed by Basalaj [1]. In essence the algorithms have an MDS routine as their basis and then transform the continuous layout (inter-object distances) into a discrete topology similar to the SOM-like array in Su et al. [23]. It is thought that this discrete layout could be implemented in such a way as to use the output of our algorithm to create an alternative SOM where the topological ordering of the layout is near–optimal, thus providing a better interface for browsing. We propose that where the data set is too large for one grid, a series of nested grids could be used (with the top-level being the layout of cluster centroids) to present the user with a semantic zooming function.

8. Conclusion

This paper has presented a novel method of performing multidimensional scaling through a combination of sampling, interpolation and spring models. The use of our modified version of Brodbeck and Girardin's interpolation scheme coupled with an original combination of techniques offers sub-quadratic run times of $O(N\ N)$ and layouts of low stress. We have shown that this improvement in complexity over Chalmers' benchmark 1996 algorithm is reflected in significantly faster run times. In reducing complexity and run-time in this manner, we are effectively increasing the size of data sets upon which such MDS layout techniques may be performed.

A significant proportion of this paper is dedicated to a number of further avenues for research, partly to show that this area of visualisation offers many promising lines of work. Techniques such as hashing suggest that future spring model algorithms may run in linear time overall and be applicable to large and complex data sets, but significant development and testing is required before we can say whether such potential can be realised.

9. Acknowledgements

We thank Luc Girardin and Dominique Brodbeck for openness and help with their algorithm and data sets, and Andrew Didsbury for early work on the hybrid algorithm.

10. References

1. Basalaj, W., "Proximity Visualisation of Abstract Data", PhD thesis, University of Cambridge Computer Laboratory (2000).
2. Bradley, P. S., U. M. Fayyad, "Refining Initial Points for K-Means Clustering", *Proceedings of the Fifteenth International Conf. on Machine Learning,* (1998).
3. Brodbeck, D., L. Girardin, "Combining Topological Clustering and Multidimensional Scaling for Visualising Large Data Sets", Unpublished paper (accepted for, but not published in, *Proc. IEEE Information Visualization 1998*).
4. Buja, A., D. Cook, D. F. Swayne, "Interactive high-dimensional data visualization", *Journal of Computational and Graphical Statistics*, 5(1), pp. 78-99 (1996).

5. Buja, A., D. F. Swayne, M. L. Littman, N. Dean, “XGvis: Interactive data visualization with multidimensional scaling”, *under review Journal of Computational and Graphical Statistics* (1998).
6. Chalmers, M., “A Linear Iteration Time Layout Algorithm for Visualising High–Dimensional Data”, *Proc. IEEE Visualization `96*, San Francisco, pp. 127-132 (1996).
7. Chatfield, C., A. J. Collins, *Introduction to Multivariate Analysis*, Chapman & Hall, London (1980).
8. Chávez, E., J. L. Marroquín, G. Navarro, “Fixed Queries Array: A Fast and Economical Data Structure for Proximity Searching”, *Multimedia Tools and Applications (MTAP)*, 14(2), pp. 113-135 (2001).
9. Cleveland, W.S., R. McGill. “The many faces of the scatterplot”, *Journal of the American Statistical Association*, Vol 79, pp. 807-822 (1984).
10. Eades, P., "A heuristic for graph drawing", *Congressus Numerantium*, 42 (1984).
11. Feiner, S. & C. Beshers, “Worlds within Worlds: Metaphors for Exploring n-Dimensional Virtual Worlds”, P*roc. ACM UIST 9*0, 76-83 (1990).
12. Fruchterman, T., E. Reingold. “Graph drawing by force-directed placement”, Software—Practice and Experience, 21(11), pp. 1129-1164 (1991).
13. Gionis, A., P Indyk, R., Motwani, “Similarity Search in High Dimensions via Hashing”, *Proc. 25th International Conference on Very Large Data Bases,* pp. 518-529 (1999).
14. Indyk P., R. Motwani, “Approximate Nearest Neighbours—Towards Removing the Curse of Dimensionality”, *Proc. SIGMOD ’98*, pp. 307-318 (1998).
15. Jain, A. K., M. N. Murty, P. J. Flynn, “Data clustering: A review”, *ACM Computing Surveys*, Vol. 31, No. 3 (September 1999).
16. Kohonen, T., S. Kaski, K. Lagus, J. Salojärvi, J. Honkela, V. Paatero, A. Saarela “Self-organization of a Massive Document Collection”, *IEEE Transactions on Neural Networks*, 11(3), pp. 574-585, (2000).
17. Lin, X., “A self-organizing semantic map for information retrieval,” *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 262-269 (1991).
18. Littman, M., D. F. Swayne, N. Dean, A. Buja, “Visualizing the embedding of objects in Euclidean space”, *Computing Science and Statistics: Proc. of the 24th Symp. on the Interface*, Fairfax Station, VA: Interface Foundation of North America, Inc., pp. 208-217 (1992).
19. MacQueen, J., “Some Methods for Classification and Analysis of Multivariate Observations”, *Proc. 5th Berkeley Symposium on Mathematics and Probability*, pp. 281-297 (1967).
20. Morrison, A., G. Ross, M. Chalmers, “Combining and comparing clustering and layout algorithms”, Technical Report, Department of Computing Science, University of Glasgow, TR-2002-125 (2002).
21. Rodden, K., W. Basalaj, D. Sinclair, K. Wood, “Does Organisation by Similarity Assist Image Browsing?”, *Proc. ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 190–197 (2001).
22. Selim, S.Z., M.A. Ismail (1984). “K-means-type algorithms: a generalized convergence theorem and characterization of local optimality”. IEEE Trans. On Pattern Analysis and Machine Intelligence, vol.6, n.1, pp.81-86.
23. Su, M.-C., H.-T. Chang, “Fast Self-Organizing Feature Map Algorithm”, *IEEE Transactions on Neural Networks*, Vol. 11, No. 3, p. 721 (2000).
24. Tweedie, L., R. Spence, H. Dawkes, H. Su, "Externalising abstract mathematical models", *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI),* (1996).
25. Wise, J., J. Thomas, K. Pennock, D. Lantrip, M. Pottier, A. Schur, V. Crow, "Visualizing the Non-Visual: Spatial analysis and Interaction with Information from text Documents", *Proc. IEEE Symposium on Information Visualization*, pp. 51-58 (1995).