A Hybrid Layout Algorithm for Sub-Quadratic Multidimensional Scaling

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

Many clustering and layout techniques have been used for structuring and visualising complex data. This paper is inspired by a number of such contemporary techniques and presents a novel hybrid approach based upon stochastic sampling, interpolation and spring models. We use Chalmers' 1996 $O(N^2)$ spring model as a benchmark when evaluating our technique, comparing layout quality and run times using data sets of synthetic and real data. Our algorithm runs in $O(N\sqrt{N})$ and 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 have to be performed again in its entirety if the data set was even slightly altered [4]. 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 [3] 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 section describes 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. Spring models

Spring models or force–directed placement [6] techniques are amongst the simplest MDS algorithms. Since the goal of MDS is to create a representation that preserves relationships within a set of objects, spring models determine where a point is laid out based on inter–object similarities. A high–dimensional dissimilarity is calculated for pairs of objects, and then approximated as closely as possible in the lower–dimensional space of the layout. The latter is usually measured as Euclidean distance.

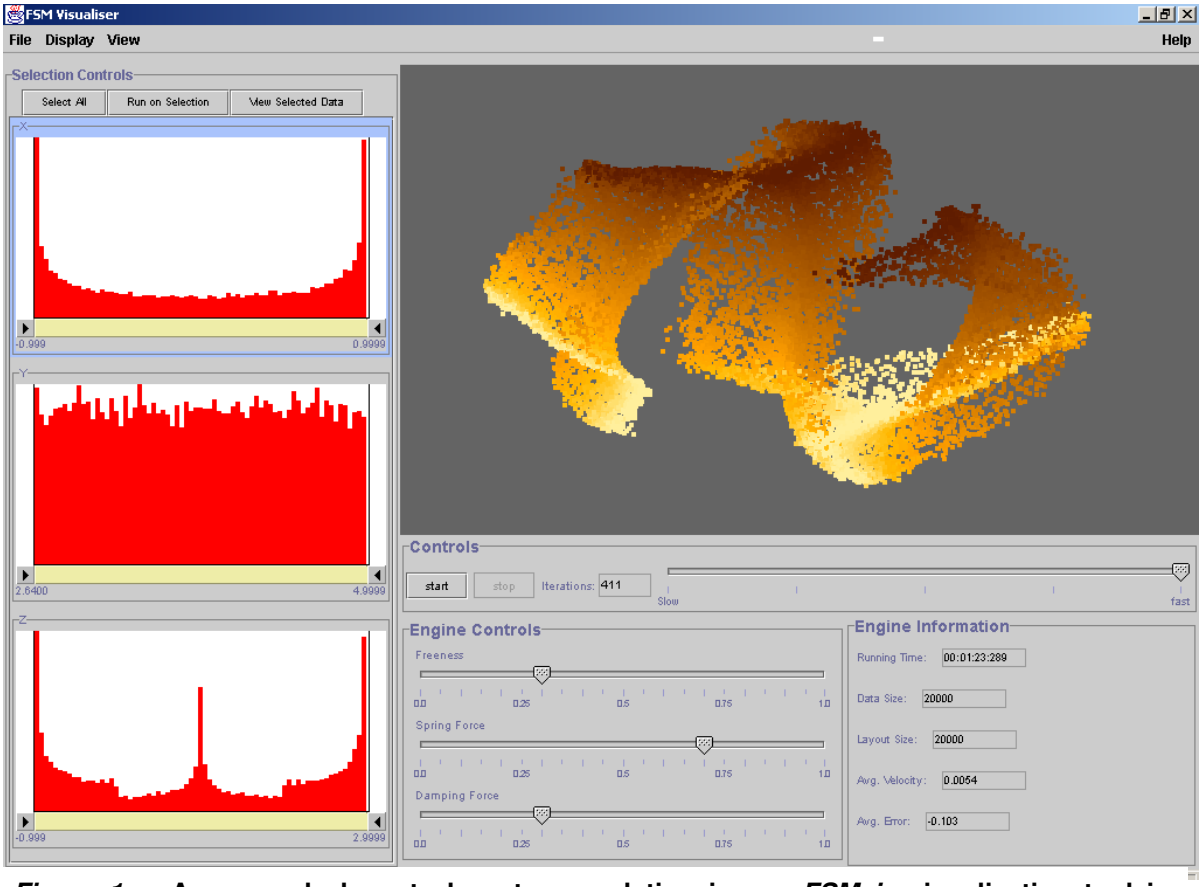

Figure 1. **An example layout close to completion in our *FSMvis* visualisation tool is shown top right. Other components of the tool offer control over spring model parameters (bottom right) and histograms (left) of individual dimensions or attributes allow filtering and selection. The layout is of 20000 points sampled from a 3D 'S' shape: one of the test sets described in Section 5. 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.**

Simulations of physical forces are used to drive the layout process. Each pair of objects is considered to have a spring, the ends of which are attached to the two points. The relaxed spring length or 'rest distance' is the ideal proximity of the two objects, i.e. their high-dimensional distance or dissimilarity. Similar objects too far apart are pulled together, and dissimilar objects too close together are pushed apart. The final layout produced by the system will reflect the spring system in equilibrium. Since a spring is simulated between each pair of objects, N^2-N springs are considered.

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 $(N\text{-}1)$ 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 $N(N\text{-}1)$ pairwise interactions at each step are an obvious area for improvement. This is analogous to the well-known *N-body problem* in computational physics.

2.1 Chalmers' 1996 Algorithm

The technique presented by Chalmers in 1996 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 low 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 by Chalmers, such a model could not be practically used on data sets over a few thousand objects in size.

3. 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) [9] is an unsupervised learning algorithm applied to the classification of information. SOMs partition a data set into a grid which can be useful in clustering or visualisation applications, but can be quite time-consuming in construction. Su et al [12] claim that K-means (a well-known iterative centroid–based divisive clustering algorithm [10]) has a lower time complexity than a SOM, and therefore employ this 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 $\sqrt{k}$ by $\sqrt{k}$) 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.

Conversely, in another example of a hybrid approach, Brodbeck and Girardin [2] 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 nature 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. Example figures showed that layouts could be produced that were strikingly similar to those generated by a full spring model. The time taken to achieve this was described as being in the order of hours rather than days.

4. 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.

Our techniques build on the interpolation strategy of Brodbeck and Girardin described in the previous section. As an initial step, however, we take a simple $\sqrt{N}$ sample (*S)* of the data set, rather than running a SOM. A layout of the subset *S* is then made using Chalmers' spring model [3]. This model will run in $O(\sqrt{N}.\sqrt{N})$, 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. Stress is based on the sum-of-squared errors of inter-object distances and may be defined as below [3], where d_{ij} denotes the desired, high dimensional distance between objects *i* and *j*, and g_{ij} denotes the low–dimensional or layout distance:

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

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.

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 $(N-\sqrt{N})$ objects to be interpolated is compared to each of the $\sqrt{N}$ samples. A best match for all points is consequently calculated in $O(N\sqrt{N})$ time. This pre-processing stage is the dominant factor, making our layout $O(N\sqrt{N})$ overall.

As a final stage in our MDS technique, we run a constant number of iterations of Chalmers' 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 $\sqrt{N}$ 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' 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 $O(N\sqrt{N})$ overall.

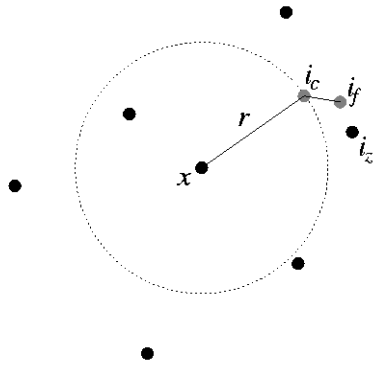

Figure 2. **The placement of the object *i* begins with finding the most similar member of the initial layout, *x,* and then finding the best position i_c on the circumference of the circle of radius *r* around *x*. The position is then refined by iteratively adding aggregate forces from a subset of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

5. Experimental results

In this section we offer a number of comparisons of our new layout algorithm with Chalmers' 1996 algorithm (the current 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 4. 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 carry out 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 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 increasingly linearly, even for

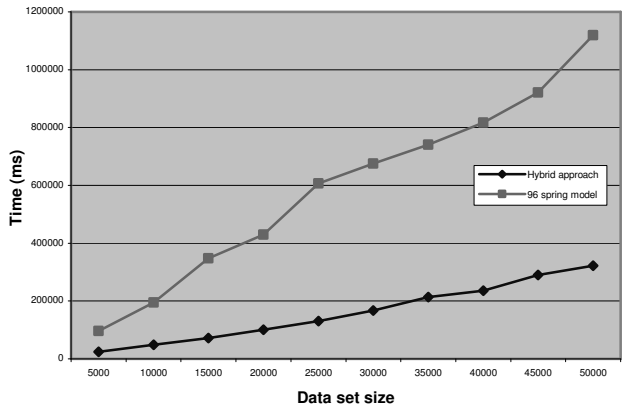

Figure 3. **Run time to completion for different sizes of 3D 'S' data.**

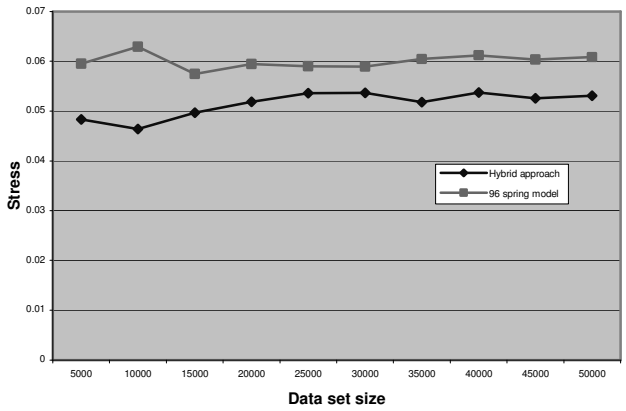

Figure 4. **Stress of completed layout over different sizes of 3D 'S' data.**

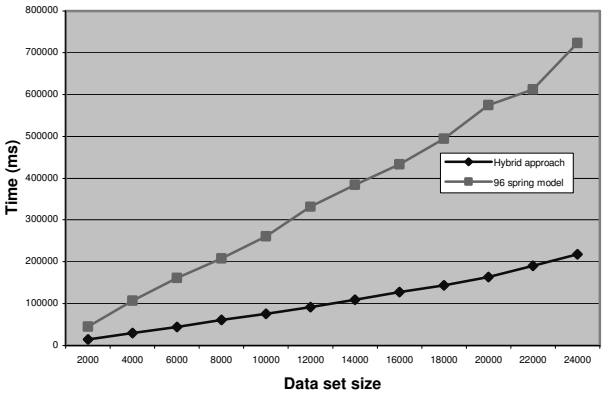

Figure 5. **Run time to completion for different sizes of 13D financial data.**

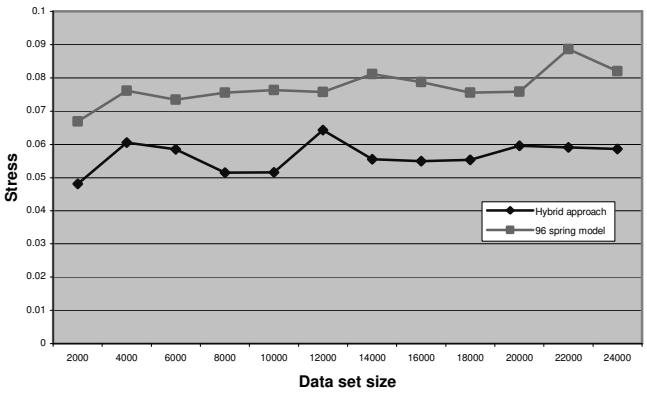

Figure 6. **Stress of completed layout over different sizes of 13D financial data.**

data sets of 50000 elements. This may seem unusual, as the $O(N\sqrt{N})$ '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 (see 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 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.

6. 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.

6.1 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) [8] 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 [7] 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.

6.2 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, using the triangular inequality, 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 [5]: 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, or as a parent-finding operation.

6.3 Dynamically Resizing V+S

We asserted earlier that it was possible to create good layouts using values of 5 and 10 respectively for the sizes of the neighbour and random sample sets. To minimise iteration time, it is obviously advisable to set these values as low as possible. However, we theorise 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 period 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.

6.4 Proximity Grid

In a recent paper [11], 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 [12]. 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.

7. 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 $O(N\sqrt{N})$ 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.

8. 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.

9. References

1. Basalaj, W., "Proximity Visualisation of Abstract Data", PhD thesis, University of Cambridge Computer Laboratory (2000).
2. 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*).
3. Chalmers, M., "A Linear Iteration Time Layout Algorithm for Visualising High–Dimensional Data", *Proc IEEE Visualization '96*, San Francisco, pp. 127-132 (1996).
4. Chatfield, C., A. J. Collins, *Introduction to Multivariate Analysis*, Chapman & Hall, London (1980).
5. 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).
6. Fruchterman, T., E. Reingold. "Graph drawing by force-directed placement", Software—Practice and Experience, 21(11), pp. 1129-1164 (1991).
7. Gionis, A., P Indyk, R., Motwani, "Similarity Search in High Dimensions via Hashing", *Proceedings of 25th International Conference on Very Large Data Bases,* pp. 518-529 (1999).
8. Indyk, P., R. Motwani, "Approximate Nearest Neighbors – Towards Removing the Curse of Dimensionality", *Proceedings of SIGMOD '98* , pp. 307-318 (1998).
9. 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).
10. MacQueen, J., "Some Methods for Classification and Analysis of Multivariate Observations", *Proc. 5th Berkeley Symposium on Mathematics and Probability*, pp. 281-297 (1967).
11. Rodden, K., W. Basalaj, D. Sinclair, K. Wood, "Does Organisation by Similarity Assist Image Browsing?", *Proceedings of the SIGCHI on Human Factors in Computing Systems*, pp. 190–197 (2001).
12. Su, M.-C., H.-T. Chang, "Fast Self-Organizing Feature Map Algorithm", *IEEE Transactions on Neural Networks*, Vol. 11, No. 3, p. 721 (2000).